INSTITUT FÜR INFORMATIK
Machine Learning

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Natural Language Processing for Discussion Platforms

INAUGURAL-DISSERTATION

zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Maike Katrin Behrendt

geboren in Krefeld Uerdingen

Düsseldorf, 29.04.2026

aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Berichterstatter:

1. Prof. Dr. Stefan Harmeling
2. Prof. Dr. Stefan Conrad

Tag der mündlichen Prüfung: 02.09.2026

Abstract

Online participation platforms and digital discussion spaces have become central arenas for public discourse, particularly in times of crisis such as the COVID-19 pandemic. While these environments expand opportunities for civic engagement, they are frequently characterized by incivility, misinformation, and unstructured exchanges. Drawing on the concept of deliberation, defined by the core dimensions of rationality, civility, interactivity and equality, this thesis investigates how Natural Language Processing (NLP) methods can be leveraged to foster more respectful, reasoned, and inclusive online discussions.

The dissertation addresses five central aspects: (i) the current approaches in applying NLP to online discussion platforms, (ii) the optimization of smaller language models for efficient real-time deployment, (iii) the integration of NLP-based interventions into discussion platforms, (iv) user perceptions of Artificial Intelligence (AI)-supported moderation and assistance, and (v) the measurable effects of such interventions on discussion quality.

First, a systematic review provides a comprehensive overview of existing NLP applications in political and civic online discussions, identifying established approaches and tasks that could potentially be used to support discussions (see Chapter 4). Second, the thesis develops and evaluates methods to improve the efficiency and effectiveness of small Transformer-based models, particularly Bidirectional Encoder Representations from Transformers (BERT), for tasks relevant to online deliberation. This includes adapting pre-training strategies for argument similarity detection (ArgueBERT, see Chapter 5) and optimizing fine-tuning procedures for classification performance (MaxPoolBERT, see Chapter 6). In addition, the Additive deliberative Quality score with Adapters (AQuA) is introduced to assess the deliberative quality of user contributions (see Chapter 7). This model, along with a comment recommendation module based on stance detection, has been integrated into an online discussion platform and tested in a large-scale, real-time experiment (see Chapter 8). The findings provide insights into the technical integration of AI-supported tools into online discussion platforms, as well as into users' perceptions of such systems and their observed effects in a specific participatory context. By combining perspectives from computer science, political science, and communication science, this interdisciplinary work contributes to the development of AI-supported tools that aim to support deliberative processes in digital public spheres.

Zusammenfassung

Online-Beteiligungsplattformen und digitale Diskussionsräume sind zu zentralen Schauplätzen des öffentlichen Diskurses geworden, insbesondere in Krisenzeiten wie der COVID-19-Pandemie. Diese Umgebungen erweitern zwar die Möglichkeiten für bürgerschaftliches Engagement, sind jedoch häufig von Unhöflichkeit, Fehlinformationen und unstrukturiertem Austausch geprägt. Ausgehend vom Konzept der Deliberation, das durch die Kerndimensionen Rationalität, Zivilität, Interaktivität und Gleichheit definiert ist, untersucht diese Dissertation, wie Methoden des Natural Language Processings (NLP) genutzt werden können, um respektvollere, fundiertere und inklusivere Online-Diskussionen zu fördern.

Die Dissertation befasst sich mit fünf zentralen Aspekten: (i) den aktuellen Ansätzen bei der Anwendung von NLP auf Online-Diskussionsplattformen, (ii) der Optimierung kleinerer Sprachmodelle für einen effizienten Echtzeit-Einsatz, (iii) der Integration von NLP-basierten Interventionen in Diskussionsplattformen, (iv) der Wahrnehmung von KI-gestützter Moderation und Unterstützung durch die Nutzer sowie (v) den messbaren Auswirkungen solcher Interventionen auf die Diskussionsqualität.

Erstens bietet eine systematische Literaturübersicht einen umfassenden Überblick über bestehende NLP-Anwendungen in politischen und gesellschaftlichen Online-Diskussionen und identifiziert etablierte Ansätze und Aufgaben, die potenziell zur Unterstützung von Diskussionen genutzt werden könnten (siehe Kapitel 4). Zweitens entwickelt und evaluiert die Arbeit Methoden zur Verbesserung der Effizienz und Effektivität kleiner Transformer-basierter Modelle, insbesondere von BERT, für Aufgaben, die für die Online-Deliberation relevant sind. Dazu gehören die Anpassung von Pre-Training Strategien zur Erkennung von Argumentähnlichkeiten (ArgueBERT, siehe Kapitel 5) und die Optimierung des Fine-Tunings für die Klassifikationsleistung (MaxPoolBERT, siehe Kapitel 6). Darüber hinaus wird das AQuA-Modell, zum Bewerten der deliberativen Qualität von Nutzerbeiträgen, eingeführt (siehe Kapitel 7). Dieses Modell wurde zusammen mit einem Stance Detection Modul zur Empfehlung von Kommentaren in eine Online-Diskussionsplattform integriert und in einem groß angelegten Experiment getestet (siehe Kapitel 8). Die Ergebnisse liefern Einblicke in die technische Integration KI-gestützter Tools in Online-Diskussionsplattformen sowie die Wahrnehmung solcher Systeme durch die Nutzer und deren beobachtbare Auswirkungen in einem bestimmten partizipativen Kontext. Durch die Kombination von Perspektiven aus der Informatik, der Politikwissenschaft und der Kommunikationswissenschaft trägt diese interdisziplinäre Arbeit zur Weiterentwicklung von KI-gestützten Werkzeugen bei, die darauf abzielen, deliberative Prozesse in digitalen öffentlichen Bereichen zu stärken.

Erklärung

Ich versichere an Eides statt, dass die Dissertation von mir selbständig und ohne unzulässige fremde Hilfe unter Beachtung der „Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Heinrich-Heine-Universität Düsseldorf“ erstellt worden ist.

Des Weiteren erkläre ich, dass ich eine Dissertation in der vorliegenden oder in ähnlicher Form noch bei keiner anderen Institution eingereicht habe. Ich habe keinerlei andere Promotionsversuche unternommen.

Düsseldorf, 29.04.2026 Maike Behrendt

LLM Usage

During the preparation of this manuscript, AI-based writing assistance tools, including ChatGPT (OpenAI) and DeepL Write, were used to assist with language formulation, grammar correction, and stylistic improvement of text passages. No large language models (LLMs) were used for code generation in the development of the methods and systems presented in this thesis. All conceptual development, research design, interpretation of results, and final editing were performed by the author, who assumes full responsibility for the content of this work.

Düsseldorf, 29.04.2026 Maike Behrendt

Contents

List of Figures

List of Tables

List of Abbreviations

AFS	Argument Facet Similarity
AI	Artificial Intelligence
ANOVA	Analysis of Variance
AQuA	Additive deliberative Quality score with Adapters
BART	Bidirectional and Auto-Regressive Transformers
BERT	Bidirectional Encoder Representations from Transformers
BoW	bag-of-words
CNN	Convolutional Neural Network
CR	Comment Recommendation
DL	Deep Learning
DQ	Deliberative Quality
DQI	Deliberative Quality Index
DRI	Deliberative Reason Index
GCR-NN	Gated Convolutional Recurrent-Neural Network
GDPR	General Data Protection Regulation
GloVe	Global Vectors for Word Representation
GLUE	General Language Understanding Evaluation
GPT	Generative Pre-Trained Transformer
IAC	Internet Argument Corpus
IHTM	Incremental Hierarchical Topic Modeling
KI	Künstliche Intelligenz
LDA	Latent Dirichlet Allocation
LIWC	Linguistic Inquiry and Word Count
LLM	Large Language Model
LQI	Listening Quality Index
LSTM	long short-term memory
MCC	Matthews correlation coefficient
MHA	multi-head attention
ML	Machine Learning

MLM	Masked Language Modeling
MRPC	Microsoft Research Paraphrase Corpus
MSE	Mean Squared Error
NLI	natural language inference
NLP	Natural Language Processing
NLU	natural language understanding
NSP	Next Sentence Prediction
PAWS	Paraphrase Adversaries from Word Scrambling
PCA	Principal Component Analysis
QQP	Quora Question Pairs
ReLU	rectified linear unit
RNN	Recurrent Neural Network
TF-IDF	Term-Frequency Inverse Document-Frequency
SAM	Similar Argument Mining
SBERT	Sentence BERT
SOCC	SFU opinion and comment corpus
SVM	Support Vector Machine
UPEKI	Unterstützung Politischer Entscheidungen mit Hilfe von KI

Acknowledgments

I would like to express my sincere gratitude to everyone who accompanied me on the journey of writing this thesis. My very special thanks go to my two PhD supervisors Stefan Harmeling and Stefan Conrad, who always had an open ear and invaluable advice. Thank you for your continuous feedback, encouragement and support throughout this research. I am equally grateful to my supervisors at the Heinrich Heine University, Martin Mauve and Tobias Escher for their guidance, motivation, and trust. I always felt listened to by each one of you, which made working together both productive and enjoyable.

Many thanks go to Tobias Uelwer, who acted as a mentor when I started my PhD and provided guidance during an important early phase of my doctoral journey. I would also like to thank all my colleagues and fellow researchers from both the computer science and the social science departments: Sebastian, Eric, Jan, Marc, Stefan, Lukas, Carina, Viviana, Katharina, Nicole and Niklas for the inspiring work atmosphere and for making both work and time outside the office so enjoyable. A special thanks to Jan for your last-minute proofreading. I am very grateful to have gotten to know each one of you.

My sincere thanks go to the whole UPEKI team - Dennis, Carina, Marike, Markus, Marc, Ole, Tim, Lena, Maike, Jana, Florian Meißner, Katharina, Charlotte, Florian Sauer, Mira, Stefan Wagner, Stefan Harmeling, Stefan Marschall and especially Gerhard Vowe for this great project, which constitutes the foundation of this thesis. Working in such an interdisciplinary project has been a great privilege and taught me patience, openness, and the value of different perspectives. I would also like to acknowledge the Jürgen Manchot Stiftung for the financial support that enabled this research.

I am deeply thankful for my friends, who have constantly supported me throughout the years. In particular, thank you Chrissi for sending motivating Pedro Pascal reels almost every day, they truly kept me going. Thank you Odile for countless nightly calls and endless conversations. Thank you Ellen for many spaghetti ice cream dates. Thank you Markus, Laurenz, Elli, Caro, John and all other close friends for your encouragement, understanding, and support both during challenging and joyful times. I could not have done this without you.

Last but not least, I am very grateful to my family, my parents and my sister, without whom this journey would not have been possible. Your unconditional support and love mean more to me than words can express. Lastly, thank you, Flo, for holding my hand during the final stretch. I love you.

> For small creatures such as we
> the vastness is bearable only
> through love.
>
> *Carl Sagan*

1 Introduction

Online participation platforms and online discussion spaces, such as social media platforms or comment sections on news outlets, enable citizens to engage with socially relevant issues and discuss topics they care about. These digital environments have become an important venue for democratic communication. During the COVID-19 pandemic, for example, many participatory processes shifted to online environments, further increasing the importance of online discussion spaces for civic engagement and political participation (Hofstra et al., 2023).

Engaging in political discussions can have positive effects on democratic participation. Research shows that exchanging political opinions online is associated with increased participation in activities such as voting or protesting (Vaccari and Valeriani, 2018). Online discussions also provide opportunities for citizens to encounter diverse perspectives and arguments, which can broaden their political understanding and encourage them to reflect on their own views (Min, 2007). Encountering different points of view is widely regarded as an important prerequisite for democratic societies, which depend on informed citizens who are willing to engage with competing ideas and perspectives.

Despite these opportunities, online discussions often face substantial challenges. Written exchanges in online spaces are frequently characterized by a lack of structure, disrespectful behavior, and the spread of misinformation (Arana-Catania et al., 2021a; Anastasiou et al., 2023; Muhammed T and Mathew, 2022). Incivility, personal attacks, and destructive interactions can discourage users from participating and undermine the quality of public discourse. As a result, many online discussions fail to realize their potential as spaces for productive democratic exchange.

The concept of *deliberation* is particularly valuable in addressing these challenges. Deliberation refers to the respectful and reasoned exchange of opinions aimed at collective opinion formation and decision-making. It encompasses four core dimensions: *rationality*, referring to the argumentative exchange of opinions, *civility*, which entails politeness and respect, *interactivity*, which is characterized by responsiveness and active listening, and *equality*, which refers to participants' equal opportunity to express their views and have their arguments considered in the discussion (Bächtiger et al., 2009b; Esau et al., 2021; Graham, 2010). Deliberation is often considered an ideal form of communication, and the aforementioned dimensions of deliberation serve as indicators for evaluating the quality of online discussions (Steenbergen et al., 2003). Research suggests that communication following deliberative standards can positively influence discussion dynamics. Providing reasons supported by evidence, maintaining a polite tone, and demonstrating openness to compromise can encourage more constructive exchanges and foster positive spillover effects within discussions (Heide-Jørgensen et al., 2025). Similarly, users are

more receptive to opposing political viewpoints when discussions are based on deliberative norms such as justification and civility (Zúñiga et al., 2018). Promoting deliberative standards is therefore an important goal for the design of online discussion spaces.

Moderation plays a central role in achieving this goal. By enforcing discussion rules and guiding interactions, moderation can reduce incivility and promote constructive participation. Empirical studies consistently show that moderation improves the quality of online discussions and encourages more respectful exchanges among participants (Friess et al., 2021; Schluger et al., 2022). However, the scale of today's online communication makes it difficult to rely solely on human moderators. Large participation platforms must process vast amounts of user-generated content, making comprehensive human moderation both costly and impractical.

These limitations motivate the use of AI, particularly NLP methods, to support moderation and enhance deliberation in online discourse (Argyle et al., 2023). NLP systems can assist both participants and organizers in creating more structured, respectful, and engaging online environments by automatically analyzing discussion content (Falk et al., 2021), detecting problematic behavior (Alkomah and Ma, 2022), and providing feedback that encourages more constructive engagement (Argyle et al., 2023). Since the introduction of Transformer models (Vaswani et al., 2017), many NLP tasks, including sentiment analysis, hate speech detection, and text generation, can often be performed with high accuracy. However, further research is needed to determine whether such models can support or complement human moderation effectively in real-world discussion settings. Beyond technical performance, users' trust and perceived fairness are crucial for the successful integration of AI in online discussions. At the same time, these social considerations are accompanied by practical deployment challenges, particularly regarding efficiency, scalability, and data protection.

To deploy such Transformer-based models in real-time discussion environments, they must be as accurate and efficient as possible (Bonechi, 2024). Online participation platforms often process large volumes of user-generated content, making computationally expensive models impractical in many settings. Furthermore, privacy considerations and institutional constraints often require that systems run on local servers rather than cloud-based infrastructure. For these reasons, this thesis also examines how smaller language models such as BERT (Devlin et al., 2019) can be optimized for efficient use in discussion platforms.

This thesis contributes to the field of research by investigating how NLP techniques can be applied in discussion platforms and by evaluating their potential and limitations for supporting deliberative processes. It reviews the current state of research, develops methods to improve smaller language models for effective use in this context, and implements and evaluates NLP-based interventions.

The work presented in this thesis emerges from an interdisciplinary research project at Heinrich Heine University Düsseldorf, involving communication science, political science, and computer science. Through close collaboration between these disciplines, this

thesis examines not only technical aspects but also the theoretical foundations and social dynamics of online discussions.

1.1 Research Questions

The challenges outlined above highlight the need for interdisciplinary research at the intersection of computational methods and social science. While recent advances in NLP enable automated analysis and generation of text, several open questions remain regarding their effective and responsible use in online deliberation. In particular, it remains unclear which NLP approaches are most suitable for supporting discussions, how such systems can be implemented efficiently in real-world platforms, and how users perceive and respond to AI-based interventions.

To address these issues, this thesis investigates the following research questions:

1. **What approaches currently exist for applying NLP to online discussion platforms and participatory processes?**
2. **How can smaller language models be refined to achieve efficient and effective performance in real-time discussion contexts?**
3. **How can NLP methods be embedded into discussion platforms to foster deliberative and respectful online discourse?**
4. **How do users perceive AI-based assistance in online discussions?**
5. **What measurable effects do AI-based interventions have on discussion quality and user interaction?**

Taken together, these research questions address both the technical and social aspects of AI-supported online discussions. They guide the investigation of how NLP techniques can be developed, integrated into discussion platforms, and evaluated with respect to their impact on deliberative communication.

1.2 Contributions

To address these research questions, this thesis makes several contributions at the intersection of NLP and the study of online deliberation. These contributions range from a systematic overview of existing research to the development of improved NLP models, and the implementation and empirical evaluation of AI-supported discussion tools in a real-world platform. The main contributions of this thesis can be summarized as follows:

1. **We provide a comprehensive overview of the current state of research, relevant projects, and NLP methods used to support online discussions.** We conducted

a systematic review of the literature to identify which tasks can be addressed in online discussions with the support of AI and which solutions have been developed and examined so far (See Chapter 4). With this work, we aim to provide a recent overview of the literature and show where substantial work has been done and where research gaps remain. This contribution was published or is under review as:

> Maike Behrendt, Stefan Sylvius Wagner, Carina Weinmann, Marike Bormann, Mira Warne & Stefan Harmeling (2026). *Machine Learning for Enhancing Deliberation in Online Political Discussions and Participatory Processes: A Survey*.

2. **We investigate ways to improve BERT's performance on the task of predicting argument similarity by altering its pre-training process.** Models designed to support online discussions must be efficient, as they often need to react in real time and process large volumes of content produced by participants. In Chapter 5, we experiment with adapting the pre-training process of BERT for argument similarity detection. This contribution was published as:

   > Maike Behrendt & Stefan Harmeling (2021). *ArgueBERT: How to Improve BERT Embeddings for Measuring the Similarity of Arguments*.

3. **We introduce a simple yet effective extension to BERT's fine-tuning that uses information from both layers and tokens.** We use max-pooling to enrich the `[CLS]` token for classification and show that this approach is especially useful in low-resource settings in Chapter 6. This contribution was published or is under review as:

   > Maike Behrendt, Stefan Sylvius Wagner & Stefan Harmeling (2025). *MaxPoolBERT: Enhancing BERT Classification via Layer-and Token-Wise Aggregation.*

4. **We develop a method to automatically measure the deliberative quality of user contributions in online discussions.** Based on a total of twenty different dimensions of deliberative quality, we train adapter models and define an additive score from the individual predictions. This contribution was published as:

   > Maike Behrendt, Stefan Sylvius Wagner, Marc Ziegele, Lena Wilms, Anke Stoll, Dominique Heinbach & Stefan Harmeling (2024). *AQuA–Combining Experts' and Non-Experts' Views To Assess Deliberation Quality in Online Discussions Using LLMs*.

5. **We develop concrete NLP models, integrate them into a discussion platform, and examine their effects in real-time discussions.** We expand the open-source

platform adhocracy+[1] with two different AI-supported debate modules to examine their effects in discussions on political topics. We conduct a large-scale user study and report the measured effects. This contribution was published as:

> Maike Behrendt, Stefan Sylvius Wagner, Mira Warne, Jana Leonie Peters, Marc Ziegele & Stefan Harmeling (2025). *Supporting Online Discussions: Integrating AI Into the adhocracy+ Participation Platform To Enhance Deliberation*.

Overall, these contributions provide new insights into how NLP methods can be designed, applied, and evaluated in the context of deliberative online political discussion.

1.3 Project

The publications that form the basis of this thesis were written and published as part of the Unterstützung Politischer Entscheidungen mit Hilfe von KI (UPEKI) project. The UPEKI project was part of the Manchot research group "Decision-making with the help of artificial intelligence methods" (2019-2025) at the Heinrich Heine University Düsseldorf. The project was funded by the Jürgen Manchot Foundation and consisted of two funding phases with four use cases: health, economics, politics and law.

The political use case called UPEKI (short for: supporting political decisions through AI) is an interdisciplinary sub-project involving researchers from political science, communication science, and computer science. The main objective of the project was to examine how AI can support citizens to make political decisions. The interdisciplinary nature of the project significantly influenced the research presented in this thesis.

The project aimed to answer the following questions:

1. Can the use of AI tools reduce problems in online discussions?
2. Does the citizens' satisfaction with the discussion process and outcomes of the discussions increase?
3. Can the quality of the discussion process and outcomes be demonstrably improved?

During the first funding period, the focus was placed on how *individuals* form opinions in online discussions when presented with arguments by an AI algorithm that either align with or contradict their views. To tackle this research question, the online discussion platform Discuss! (Brenneis and Mauve, 2020) was developed, and field experiments were conducted to examine the impact of AI on individuals' opinion formation. The results show that opinion-conforming online environments have an impact on how confident

[1] `https://adhocracy.plus/`

people are in their political decisions and how willing they are to become politically active (Kelm et al., 2023).

Building on these results, the second funding period focused on AI support for political decision-making processes *in groups*. The central research question was: What influence do AI tools have on the joint decision-making of citizens in online discussions on controversial political issues? The project examined online participation processes as a specific form of online discussion. In such processes, groups of citizens exchange views on political topics. The results and publications from the second funding phase are essential components of this thesis and will be discussed in more detail in the following chapters.

Other joint publications emerged from the project but are not central to this thesis. These include Brenneis et al. (2020), Brenneis et al. (2021), Kelm et al. (2023), and Wagner et al. (2025).

1.4 Thesis Outline

This thesis is structured into three main parts, covering theoretical foundations, methodological contributions, and practical applications.

First, Chapters 2 and 3 provide the theoretical background for the concepts central to this thesis, drawing on both the social sciences and computer science, such as online political discussions, deliberation, NLP, Machine Learning, and the Transformer-based models used in this work. An overview of the literature on NLP in political online discussions is provided in Chapter 4.

The second part of the thesis focuses on methodological contributions. Chapter 5 introduces ArgueBERT (Behrendt and Harmeling, 2021), a model for predicting argument similarity. Chapter 6 describes MaxPoolBERT, an optimized BERT model for small-scale datasets. Chapter 7 describes AQuA (Behrendt et al., 2024), a model that measures the deliberative quality of user comments.

The final part of the thesis focuses on application and integration. Chapter 8 describes a platform for online discussions equipped with AI features, including a deliberative quality module based on the AQuA score and a comment recommendation module.

2 Relevant Concepts of the Social Sciences

Theoretical concepts from both disciplines, the social sciences and computer science, are essential to this thesis. This chapter introduces the central concepts and technical foundations required to understand the contributions of this work. We begin by discussing online political discussions as a form of digital participation and outlining their importance for democratic processes in Section 2.1. Subsequently, the concept of deliberation is presented as theoretical framework to evaluate the quality of such discussions in Section 2.2. The next chapter introduces the basics of Machine Learning (ML), especially Natural Language Processing (NLP), followed by an overview of Transformer-based language models, with a particular focus on architectures relevant to this thesis (see Section 3).

2.1 Online Political Discussions

Political discussions are a cornerstone of democratic societies (Habermas, 1991). Through these discussions, "members of society come to clarify their own views, learn about the opinions of others, and discover what major problems face the collective" (Stromer-Galley and Wichowski, 2011). In this regard, political discourse is not only a way of expressing preferences but also a mechanism for opinion formation and collective reasoning. Political discussions can be broadly defined as conversations about political topics, events, or decisions, ranging from informal exchanges between individuals to moderated discourse aimed at collectively addressing societal challenges (Stromer-Galley and Wichowski, 2011).

With the rise of the internet, such discussions increasingly take place in online environments. Digital platforms enable citizens to participate in political discourse independent of time and location, thereby lowering barriers to participation (Hofstra et al., 2023). Key characteristics of online discussions include their *interactivity*, *scalability*, and the *possibility of anonymity*, which allows users to express opinions that they might hesitate to share in face-to-face settings (Suler, 2004). Subsequently, online discussions have become an important component of today's active political participation, which refers to voluntary activities that aim to influence public policy, such as voting, donating money, petitioning and protesting (Uhlaner, 2015). Studies have found a positive relationship between informal political discussions online and political participation (Valenzuela et al., 2011), which once again underscores the importance for democracy.

Online political discussions take place in a variety of settings that differ in their structure and objectives. On the one hand, platforms such as social media enable large-scale, informal exchanges that are often loosely moderated (Kushin and Kitchener, 2009). On the other hand, structured participation platforms are specifically designed to facilitate goal-oriented discussions, for example in the context of policy-making or civic engagement processes (Small et al., 2021). These environments usually include moderation, predefined topics and interaction rules, all of which encourage more considered communication (Esau et al., 2021).

Ideally, online conversations about socially relevant topics require participants who are open to hearing other perspectives without judgment and who can argue their own views by referring to sources and evidence (Stromer-Galley and Wichowski, 2011). This ideal exchange of opinions is rooted in deliberation theory, which is described in more detail in Section 2.2.

In practice, however, online communication often falls short of these standards. Discussions are frequently characterized by incivility, personal attacks, and the spread of misinformation (Arana-Catania et al., 2021a; Anastasiou et al., 2023; Muhammed T and Mathew, 2022). Anonymity, which can be seen as lowering the barrier and encouraging the expression of opinions, also leads to a lack of accountability (Shortall et al., 2022). A more detailed overview of issues and existing computational approaches to address them is provided in Section 4.

Importantly, the behavior of discussion participants, and thus the quality of online discourse, is primarily shaped by the design of platforms and participation processes in which discussions take place (Esau et al., 2017; Dryzek, 2005). Features such as moderation mechanisms, interface design, and discussion structure can significantly influence how users interact with one another. In this context, digital tools and innovations are increasingly explored as means to support more constructive and inclusive discussions. In particular, methods from AI, and NLP in particular, offer promising opportunities to analyze, structure, and guide online discourse at scale (Mikhaylovskaya, 2024).

2.2 Deliberation

Building on the role of online discussions, we introduce *deliberation* as a normative framework for evaluating their quality. Deliberation can be understood as "mutual communication that involves weighing and reflecting on preferences, values and interests regarding matters of common concern" (Bächtiger et al., 2018).

In political theory, deliberation is often considered as the ideal form of communication in order to come to a meaningful, well-considered decision on a socially relevant topic. Theories of deliberative democracy emphasize the importance of reason-giving, inclusion, and mutual respect in democratic processes (Bohman and Rehg, 1997; Dryzek, 2002; Fishkin and Laslett, 2008). While some argue that deliberation should ideally lead to rationally motivated consensus (Cohen, 1989), others highlight that compromise represents

a more realistic and equally legitimate outcome of democratic decision-making (Mansbridge et al., 2010). In line with this latter perspective, this thesis takes a functional approach to deliberation, focusing on enhancing the quality of collective decision-making rather than aiming for complete consensus.

A widely accepted conceptualization of deliberation includes the following four core dimensions: *rationality*, *civility*, *interactivity* and *equality* (Friess and Eilders, 2015). *Rationality* refers to the exchange of well-formulated arguments to underpin one's point of view. The dimension of *civility* emphasizes polite, respectful interaction with other participants in the discussion. *Interactivity* relates to the principle of engaging with and responding to other arguments to weigh them up against one's own views, and *equality* as one of the main principles of democracy, involves ensuring that all parties involved have equal opportunities to share their views and participate in a decision.

It is important to note, that deliberation represents only one normative framework for analyzing communication, and other perspectives may emphasize other aspects, such as personal expression, group identity, or conflict (Freelon, 2010). However, the deliberative model is widely used in the analysis of political communication, particularly in democratic contexts.

The ideals of deliberation are especially relevant in the context of online political discussions, which have become a central environment for public discourse. However, they are not feasible to maintain in a mass participation setting (Landemore, 2024), where discussions often suffer from fragmentation, incivility, and information overload.

In this thesis we want to have a closer look at how online political discussions, which in the best case resemble a deliberative process, can be supported through NLP methods. Recent work on digital deliberative innovations suggests that online platforms can, in principle, scale deliberative processes, but require additional structuring and support mechanisms (Mikhaylovskaya, 2024). In this context, AI-based methods, and in particular NLP, offer promising tools to analyze, structure, and improve online discourse. This thesis therefore investigates how NLP techniques can be used to support deliberative qualities in online political discussions.

3 Basics of Machine Learning

Artificial Intelligence (AI) is primarily rooted in computer science and mathematics and refers to the study and development of systems capable of solving problems that typically require human intelligence. Machine Learning (ML), a subfield of AI, focuses on designing algorithms that can learn patterns from data and use these patterns to make predictions or decisions. Rather than being explicitly programmed with predefined rules, ML models improve their performance through experience derived from data.

In the following sections, we first introduce the basic concepts of NLP and ML, with particular emphasis on Deep Learning (DL). Building on these foundations, we then describe Transformer-based (Vaswani et al., 2017) models and their extensions, which form the basis of most modern NLP models, including those developed in this thesis.

3.1 Natural Language Processing

This thesis examines the use of NLP methods to support online discussions, particularly in the political domain. Online political discourse plays a central role in shaping public opinion, but it is often characterized by polarization, misinformation, and a lack of constructive engagement (Arana-Catania et al., 2021a; Anastasiou et al., 2023). NLP provides tools to automatically analyze large-scale textual interactions, making it possible to detect patterns such as sentiment, stance, or toxicity, and thereby support more informed and constructive discussions.

NLP is a subfield of computer science and computational linguistics that focuses on enabling machines to understand, interpret, generate and interact with human language. It encompasses a wide range of tasks, including semantic text analysis, sentiment detection, morphological segmentation, machine translation and text summarization.

Traditional approaches to NLP have primarily focused on analyzing the linguistic structure of language and documenting rules and vocabularies of human language (Hirschberg and Manning, 2015). For this purpose, statistical methods that depict linguistic relationships were developed, such as the bag-of-words (BoW) model, which counts word frequencies to represent a piece of text. Term-Frequency Inverse Document-Frequency (TF-IDF) follows a similar approach, but reduces the importance of frequently occurring words. While these methods provide simple and interpretable representations, they are limited in their ability to capture context and complex linguistic structure.

Modern approaches still rely on numerical representations that capture statistical properties of language. However, due to advances in computational power and the re-emergence of neural networks in the context of deep learning, these representations have

become significantly more complex. Current language models, which are mostly based on the Transformer architecture (Vaswani et al., 2017), are remarkably good at solving NLP tasks and generating language, both spoken and written. In addition, they enable the analysis and classification of large volumes of textual data.

While this section introduced the domain of NLP, the underlying methods rely on general principles of ML. To understand how such systems learn from data and make predictions, we now shift the focus to the core concepts of ML, which provide the foundation for modern NLP models.

3.2 Machine Learning

ML focuses on designing algorithms that learn patterns from data in order to perform tasks such as classification or regression. Rather than relying on explicitly defined rules, these models learn patterns and relationships from data and use them to improve their predictions over time.

At a high level, an ML application consists of three core components: (i) a model, (ii) data, and (iii) an optimization procedure, commonly referred to as training. In the following, we introduce these components step by step.

Learning Paradigms. Depending on the structure of the data and the type of feedback available, different learning paradigms can be distinguished:

- *Supervised learning* aims to learn a mapping from inputs to outputs based on labeled data, i.e., examples that are assigned target values corresponding to the task. Tasks typically include classification, in which an algorithm assigns inputs to discrete categories, and regression, which models relationships between variables to predict continuous values.
- *Unsupervised learning* aims to identify patterns and structure in data without relying on labeled examples. Instead, the model analyzes the input to discover underlying regularities, similarities, or groupings, e.g, through clustering.
- *Reinforcement learning* involves learning through interaction with an environment, where an agent improves its behavior based on feedback in the form of rewards.

Among these paradigms, supervised learning is particularly relevant to many NLP tasks considered in this work. Therefore, we focus on supervised learning to introduce the core concepts of ML.

Model. The goal of supervised machine learning is to approximate an unknown target function $f^\star : X \to Y$, which maps elements from an input space X to an output space Y.

Since $f^\star$ is not accessible, we instead learn a parameterized function f_θ with parameters θ, which serves as an approximation to $f^\star$. These parameters determine how input features influence the model's predictions. The quality of this approximation depends on both the chosen model and the data used for training. Models can range from classical approaches, such as a Support Vector Machine (SVM), to complex neural networks consisting of many layers. Neural networks are the primary models used in DL, which will be discussed in more detail later.

Data. To learn the parameters of the model, we need a dataset consisting of input-output pairs

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ \ldots, \ (x_n, y_n)\}, \tag{1}$$

- where each x_i is an instance of the input space X (e.g., a feature vector, an image, or a piece of text),
- $y_i \in Y$ represents the corresponding label,
- and n is the number of samples in the dataset.

Note that we use x to denote a generic input, while x_i refers to the i-th example from the dataset $\mathcal{D}$.

Since ML models operate on numerical data, inputs must be represented in a suitable vector form. Images can, e.g., be represented by pixel values, while text data requires encoding methods, which will be discussed in later sections. Having introduced the model and the data, the next step is to determine how the model's parameters can be learned from data. This is achieved through the training process.

Training. Given a model f_θ and a dataset $\mathcal{D}$, the goal of training is to learn parameters θ such that the model produces accurate predictions. This is achieved by defining a *loss function* $L(\theta)$ that measures the discrepancy between the true labels and the model predictions. In many classification settings, models are designed to estimate probabilities over possible classes. In this case, a common choice is the cross-entropy loss:

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \log p_\theta(y_i \,|\, x_i), \tag{2}$$

where $p_\theta(y_i \,|\, x_i)$ denotes the predicted probability assigned to the correct class. The loss is calculated over the whole training dataset with n samples. Intuitively, the loss evaluates how much probability the model assigns to the true labels. If the model predicts a high probability for the correct class, the loss is small. Conversely, if the predicted probability is low, the loss becomes large. This means that the loss function not only reflects whether a prediction is correct, but also how confident the model is in its prediction.

The optimal parameters are obtained by minimizing the loss

$$\theta^* = \arg\min_\theta L(\theta), \tag{3}$$

which is equivalent to maximizing the likelihood of the observed data under the model.

In practice, this optimization is solved using iterative methods such as *gradient descent*, which gradually adjusts the parameters to improve the model's predictions. This iterative process allows the model to improve its predictions by aligning them with the observed data.

3.3 Deep Learning

The framework described above is general and applies to a wide range of models. In recent years, however, neural networks have emerged as the dominant approach for many applications, particularly in NLP. These models, which consist of multiple layers of non-linear transformations, are the focus of *deep learning*, introduced in the following section.

Neural Networks. A *neural network* is a parameterized function f_θ that maps an input vector to an output by applying a sequence of non-linear transformations. Neural networks are inspired by the structure of the human brain, where simple computational units, so-called *neurons*, form a network, in which each layer transforms its input representation into a more abstract representation (Shalev-Shwartz and Ben-David, 2014). In the context of text data, early layers may capture syntactic patterns, while deeper layers encode more semantic or task-specific information (Rogers et al., 2020).

Traditional machine learning approaches often rely on manually engineered features, such as word counts or linguistic patterns, where domain knowledge is used to design input representations that are relevant for the task. A key advantage of neural networks is their ability to automatically learn useful features from data. Instead of relying on manually engineered features, the model learns representations that are optimized for the task during training. The learned representations can then be used for, e.g., regression, classification or clustering tasks.

Formally, let $h_0 = x \in \mathbb{R}^d$ denote the input vector of dimension d, which is also the first layer of the network. A feed-forward network with $L+1$ layers and no cyclic connections computes a sequence of representations

$$h_l = f_l(h_{l-1}), \quad l = 1, \ldots, L, \tag{4}$$

where each f_l denotes a transformation applied at layer l, and $h_l \in \mathbb{R}^{d_l}$ is the resulting representation. These transformations may correspond to different types of layers, such as fully-connected, convolutional, or other operations, depending on the architecture.

In the case of a fully-connected layer, also referred to as dense layer, every neuron of one layer is connected to every neuron of the next. Formally, an affine transformation is applied to the input, followed by a non-linear activation function:

$$h_l = \sigma\big(W_l h_{l-1} + b_l\big), \quad l = 1, \ldots, L, \tag{5}$$

where $h_l \in \mathbb{R}^{d_l}$ is an intermediate representation, $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ is a learnable weight matrix, $b_l \in \mathbb{R}^{d_l}$ is a learnable bias vector, and σ is a non-linear activation function applied element-wise. Here d_l denotes the dimensionality of layer l. The weight matrices determine how input features are combined, while the bias vectors allow shifting the transformation independently of the input. The activation function plays a crucial role: without it, each layer would perform only an affine transformation, and the composition of multiple layers would still be equivalent to a single affine function. The non-linearity enables the model to capture complex relationships in the data.

A widely used activation function is the rectified linear unit (ReLU) (Nair and Hinton, 2010), which is also employed in the Transformer architecture. It is defined as

$$\mathrm{ReLU}(u) = \max(0, u), \tag{6}$$

and is applied element-wise. This means that a scalar pre-activation value u is passed through unchanged if $u > 0$ and set to 0 otherwise.

Each layer transforms the representation, enabling the network to progressively extract more informative and task-relevant features. In a classification setting with a finite set of classes C, the final layer of a network usually outputs a vector of real-valued scores, known as *logits*, which we denote by $\ell \in \mathbb{R}^{|C|}$. To obtain a probability distribution over all classes, the softmax function is applied to these outputs:

$$q = \mathrm{softmax}(\ell) \tag{7}$$

where $q \in \mathbb{R}^{|C|}$ is a vector whose components satisfy $q_c \in [0, 1]$ for all $c \in C$ and $\sum_{c \in C} q_c = 1$. The individual components are given by:

$$q_c = \frac{e^{\ell_c}}{\sum_{j=1}^{|C|} e^{\ell_j}} \quad \text{for } c = 1, \ldots, |C|. \tag{8}$$

Thus, each component q_c corresponds to the conditional probability $p_\theta(y = c \,|\, x)$. Given this distribution, the model predicts a label by choosing the class with the highest probability

$$\hat{y} = \arg\max_{c \in C} q_c \in \{1, \ldots, |C|\}. \tag{9}$$

During training, the model's predictions are compared to the true labels using a loss function, which measures how well the model performs. The network's parameters are then adjusted to reduce this error, allowing the model to gradually improve its predictions. This process of forward computation, loss evaluation, and parameter updates is repeated for many training examples.

Gradient Descent. To learn the parameters θ, we minimize the loss function $L(\theta)$ defined in the previous section. Since the loss function is typically high-dimensional, iterative optimization methods are used. The most common approach is *gradient descent*,

which updates the parameters in the direction that most strongly decreases the loss. This direction is given by the negative gradient of the loss function with respect to the parameters, denoted by $-\nabla_\theta L(\theta) \in \mathbb{R}^{|\theta|}$.

Starting from an initial parameter vector θ_0, the parameters are updated iteratively according to:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta_k), \tag{10}$$

where $\alpha > 0$ is the *learning rate*, controlling the step size of each update. By repeatedly adjusting the parameters in this way, the model gradually reduces the discrepancy between predicted and true labels. In practice, variants such as stochastic gradient descent are commonly used, where the gradient is approximated using subsets of the data. This makes the optimization computationally efficient, especially for large datasets.

Backpropagation. To apply gradient descent, we need to compute the gradient of the loss function with respect to all model parameters, i.e., $\nabla_\theta L(\theta)$. For neural networks, this is achieved using *backpropagation*, an efficient algorithm based on the chain rule of calculus.

Since a neural network is a composition of functions, the loss depends on the parameters of all layers. Let θ be a parameter in the l-th layer. Then the chain rule allows us to decompose the gradient of the loss L with respect to θ into a sequence of local derivatives:

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial h_L} \cdot \frac{\partial h_L}{\partial h_{L-1}} \cdot \ldots \cdot \frac{\partial h_l}{\partial \theta}, \tag{11}$$

where the factors on the right-hand-side are Jacobian matrices.

Backpropagation computes these derivatives efficiently by propagating an error signal from the output layer back through the network. Starting from the loss, gradients are calculated layer by layer using intermediate results from the forward pass. Intuitively, each layer receives information about how much it contributed to the overall error and adjusts its parameters accordingly.

Having established how neural networks are trained, we now introduce several architectural components that are commonly used in modern deep learning models. These building blocks form the basis of more complex architectures, such as the Transformer (Vaswani et al., 2017), which will be discussed in detail in a later section.

Residual Connections. Residual connections, also known as skip connections, are used to stabilize training and improve gradient flow in deep networks. Instead of learning a direct transformation, a layer learns a residual function by skipping some layers through element-wise addition of x:

$$h_l = \mathcal{F}(x) + x, \tag{12}$$

where $x \in \mathbb{R}^d$ denotes the layer input, and $\mathcal{F}$ denotes a subnetwork consisting of one or multiple layers, such as feed-forward modules (see Figure 1). This allows the model to preserve information from earlier layers and mitigates issues such as vanishing gradients.

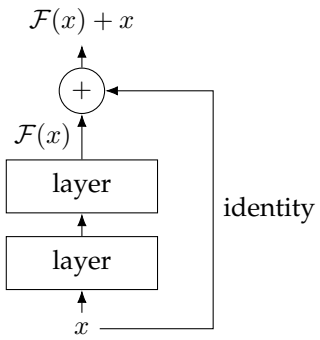


Figure 1: **Residual connection** in which the input is propagated across two intermediate layers and added to their output.

Layer Normalization. Layer normalization is applied to stabilize the activations within a layer and improve training dynamics by shifting the activations of the layer to have mean 0 and a variance of 1, followed by a learned affine transformation. Given a layer representation $h_l \in \mathbb{R}^{d_l}$, it normalizes the activations across its dimensions as follows:

$$\text{LayerNorm}(h_l) = \gamma \odot \frac{h_l - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} + \beta, \tag{13}$$

- $\gamma, \beta \in \mathbb{R}^{d_l}$ are learnable parameters,
- $\mu_l, \sigma_l \in \mathbb{R}$ denote the mean and standard deviation of $\mathbf{h}_l$,
- ϵ is a small constant for numerical stability,
- $\odot$ denotes element-wise multiplication, and addition and subtraction are also applied element-wise, with μ_l and σ_l broadcast across the feature dimensions.

This stabilizes the distribution of activations during training, which helps to reduce internal covariate shift and leads to more stable and efficient optimization.

In the Transformer architecture, these components are combined such that each sublayer is followed by a residual connection and layer normalization. We will introduce the network's architecture and models based on it in the following section.

3.4 Transformer

Introduced by Vaswani et al. (2017), the Transformer architecture forms the foundation of most modern NLP models, including BERT (Devlin et al., 2019) and Generative Pre-Trained Transformer (GPT) (Radford et al., 2018). It is designed to process sequences of text using attention mechanisms, enabling efficient modeling of long-range dependencies. We will discuss the details of the Transformer architecture in the following.

Encoder-Decoder Networks. The Transformer follows an encoder-decoder architecture, as illustrated in Figure 2. Such architectures are widely used in NLP tasks, e.g., in neural machine translation (Cho et al., 2014; Sutskever et al., 2014). These networks

consist of two sub-networks: an *encoder* and a *decoder*. The encoder aims to extract features from an input sequence and create a vector representation from it. A representation is a numerical encoding of the input that captures relevant information for the task. The decoder network then takes this vector representation as input and incrementally generates an output from it. In the case of neural machine translation, the encoder input is the original sentence and the decoder output is the translated sentence.

Input Representations. Before a text can be processed by a Transformer model, it must be converted into a numerical form. As a first step, the sequence is split into discrete units, called *tokens*, in a process known as *tokenization*. Tokens may correspond to words, subwords, or even individual characters, depending on the chosen tokenization scheme. Modern NLP models such as BERT (Devlin et al., 2019) typically use subword tokenization methods (Wu et al., 2016), which decompose rare or unknown words into smaller, more frequent units. For example the word "unbelievable" might be split into tokens like "un", "believ", and "able". This enables the model to handle a large vocabulary efficiently while still capturing meaningful linguistic structure. Formally, a text input x is transformed into a sequence of tokens:

$$x \mapsto (w_1, w_2, \ldots, w_T), \tag{14}$$

where each w_t represents a token in the sequence of length T. Each token is then mapped to an integer index, referred to as its *input ID*, based on a fixed vocabulary:

$$(w_1, w_2, \ldots, w_T) \mapsto (id_1, id_2, \ldots, id_T). \tag{15}$$

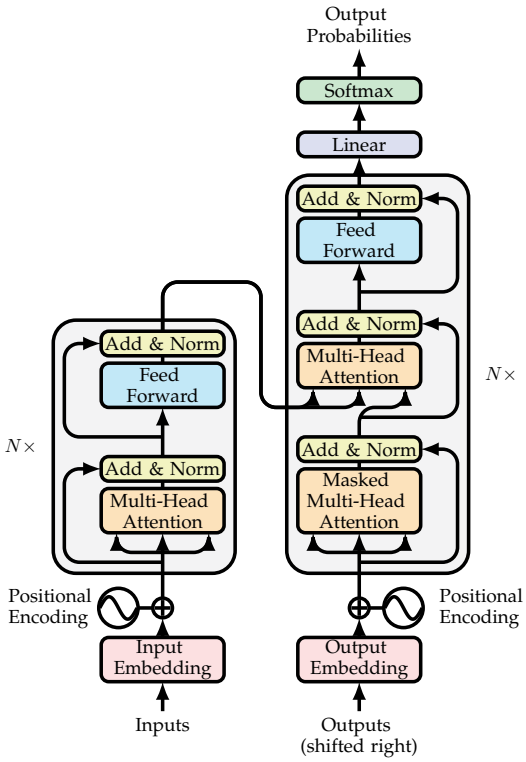


Figure 2: **Transformer architecture** (adapted from Negrinho (2020)). (Left) The encoder maps the input sequence to contextualized representations. (Right) The decoder generates the output sequence by combining masked self-attention with attention over the encoder outputs.

These token IDs are subsequently transformed into dense vector representations through an embedding layer. Each input ID is mapped to a token embedding $e_t \in \mathbb{R}^{1 \times d_{\text{model}}}$, where d_{model} denotes the embedding dimension (512 in the original Transformer). The parameters of the embedding layer are learned during training.

Since the Transformer processes all token embeddings in parallel, it does not encode any information about their order. Therefore, positional information must be incorporated

explicitly. For this purpose, a positional encoding $pos_t \in \mathbb{R}^{1\times d_{\text{model}}}$ (Gehring et al., 2017) is added to each token embedding:

$$i_t = e_t + pos_t. \tag{16}$$

The resulting sequence of vectors $(i_1, \ldots, i_T)$ serves as the input to the Transformer model. Equivalently, these vectors can be stacked into an input matrix $X \in \mathbb{R}^{T\times d_{\text{model}}}$.

Attention Mechanism. Traditional sequence models, such as Recurrent Neural Networks (RNNs), process text input token by token and compress the entire sequence into a single fixed-size vector representation. This representation is then used by the decoder to produce, e.g., a translated version of the input. However, this approach can limit the ability to capture long-range dependencies. Attention mechanisms (Bahdanau et al., 2014) address this limitation by allowing the decoder to access all encoder representations directly. Instead of relying on a single fixed representation, the decoder computes a weighted combination of the encoder outputs at each time step when generating the next output. The resulting attention weights reflect the relevance of each representation for the current prediction. Each encoder representation, denoted by $i_t \in \mathbb{R}^{1\times d_{\text{model}}}$, is linearly transformed into three vectors: a *query* $q_t \in \mathbb{R}^{1\times d_k}$, a *key* $k_t \in \mathbb{R}^{1\times d_k}$, and a *value* $v_t \in \mathbb{R}^{1\times d_v}$:

$$q_t = i_t W_Q, \quad k_t = i_t W_K, \quad v_t = i_t W_V, \tag{17}$$

where $W_Q, W_K \in \mathbb{R}^{d_{\text{model}}\times d_k}$ and $W_V \in \mathbb{R}^{d_{\text{model}}\times d_v}$ are learnable weight matrices. The query represents the information the current position is seeking, the keys encode what each element in the sequence offers, and the values contain the information that is aggregated based on the attention weights.

Stacking the individual query, key, and value vectors for all positions yields matrices Q, $K \in \mathbb{R}^{T\times d_k}$, and $V \in \mathbb{R}^{T\times d_v}$, allowing the attention mechanism to be computed efficiently for the entire sequence in parallel:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \tag{18}$$

The matrix product $QK^\top$ computes similarity scores between all pairs of token representations, which are scaled down by dividing it by $\sqrt{d_k}$ to avoid excessively high values. The softmax function normalizes these scores into a probability distribution, which determines how strongly each embedding attends to the others. The resulting weighted sum of value vectors yields a context-aware representation of each embedding. Intuitively, this can be understood as the model shifting its *attention* at each time step.

Self-Attention. A special case of the attention mechanism used in the Transformer is *self-attention*, in which each token representation is updated as a weighted combination of all other token representations in the same sequence. Intuitively, self-attention can be

understood as a mechanism that allows the model to *focus* on the most relevant parts of a sentence when computing the representation of a given element. When analyzing the meaning of a word in a sentence, not all other words are equally important. Self-attention assigns higher weights to those token embeddings that are most informative for the current context, while less relevant embeddings receive lower weights. By assigning higher weights to informative tokens and lower weights to less relevant ones, self-attention enables the model to capture dependencies between words regardless of their distance in the sequence. In the decoder, a variant known as *masked self-attention* is used to prevent a position from attending to future positions. This is necessary during training, where the entire target sequence is available, but the model should only use information from preceding positions. This is achieved by adding a mask $M \in \mathbb{R}^{T\times T}$ to the attention scores before applying the softmax function:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top + M}{\sqrt{d_k}}\right)V, \tag{19}$$

where M is an upper triangular matrix with entries

$$M_{ij} = \begin{cases} -\infty & \text{if } j > i, \\ 0 & \text{otherwise,} \end{cases}$$

and i, j denote the row and column indices of the masking matrix. This ensures that each position can only attend to itself and preceding positions.

Multi-Head Attention. Instead of computing a single attention distribution, the Transformer employs *multi-head attention*, where multiple attention mechanisms are applied in parallel. Given an input sequence $X \in \mathbb{R}^{T\times d_{\text{model}}}$, each head computes attention using separate learned projections of the input:

$$Q_i = XW_Q^{(i)}, \quad K_i = XW_K^{(i)}, \quad V_i = XW_V^{(i)}, \tag{20}$$

with projection matrices $W_Q^{(i)}, W_K^{(i)} \in \mathbb{R}^{d_{\text{model}}\times d_k}$ and $W_V^{(i)} \in \mathbb{R}^{d_{\text{model}}\times d_v}$. This yields $Q_i, K_i \in \mathbb{R}^{T\times d_k}$ and $V_i \in \mathbb{R}^{T\times d_v}$. The output of head i is then given by

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i), \tag{21}$$

where $\text{head}_i \in \mathbb{R}^{T\times d_v}$.

The outputs of all heads A heads are concatenated and projected back to the model dimension:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_A)W_O, \tag{22}$$

where $W_O \in \mathbb{R}^{Ad_v\times d_{\text{model}}}$ is a learned output projection matrix. The resulting representation has dimension $\mathbb{R}^{T\times d_{\text{model}}}$. Multi-head attention allows the model to capture different types of relationships between tokens simultaneously.

Model Architecture. The Transformer consists of $N = 6$ stacked encoder and decoder blocks, as illustrated in Figure 2. Each encoder block comprises a multi-head self-attention layer, and a position-wise feed-forward network. Both components are combined with residual connections (He et al., 2016) and layer normalization (Ba et al., 2016), which stabilize training and improve gradient flow. The decoder has a similar structure but includes an additional attention layer over the encoder representations. During training, the Transformer processes entire sequences in parallel, enabling efficient computation. During inference, however, the decoder generates tokens sequentially in an auto-regressive manner.

3.5 BERT Model

The Transformer model was a groundbreaking innovation in the field of NLP. Building on this foundation, many other models with different task-specific objectives have been proposed. One such model is the language representation model BERT, introduced by Devlin et al. (2019). While the original Transformer architecture can be used for both generative and discriminative tasks, BERT adopts an encoder-only design for language understanding rather than text generation. In the following, we describe the architecture, training procedure, and application of BERT.

Architecture. BERT is based on the encoder part of the Transformer architecture and consists of a stack of Transformer encoder layers based on self-attention. Two standard configurations are commonly used: $\text{BERT}_{\text{Base}}$, with $N = 12$ layers, hidden dimension $d_{\text{model}} = 768$, and $A = 12$ attention heads, and $\text{BERT}_{\text{Large}}$, with $N = 24$, $d_{\text{model}} = 1024$, and $A = 16$. The maximum sequence length for both models is $T_{\max} = 512$.

Pre-Training and Fine-Tuning Paradigm. A key idea underlying BERT is the pre-training and fine-tuning paradigm. Through self-attention, the encoder produces context-aware representations, where each representation is computed with respect to all other representations in the input sequence. Building on this capability, BERT is first trained on large-scale unlabeled text corpora to learn general-purpose language representations that capture syntactic and semantic features of language. These representations are subsequently adapted to specific downstream tasks using comparatively small amounts of labeled data. As a result, the model can be efficiently applied to a wide range of natural language understanding (NLU) tasks such as question answering, sentiment analysis, or paraphrase identification.

Input Representations. As an encoder-only model, BERT operates on input sequences that are processed similarly to those in the Transformer architecture. However, while the Transformer is a sequence-to-sequence model that maps an input sequence to an output sequence, the structure of BERT's input depends on the task. Some tasks, such as question

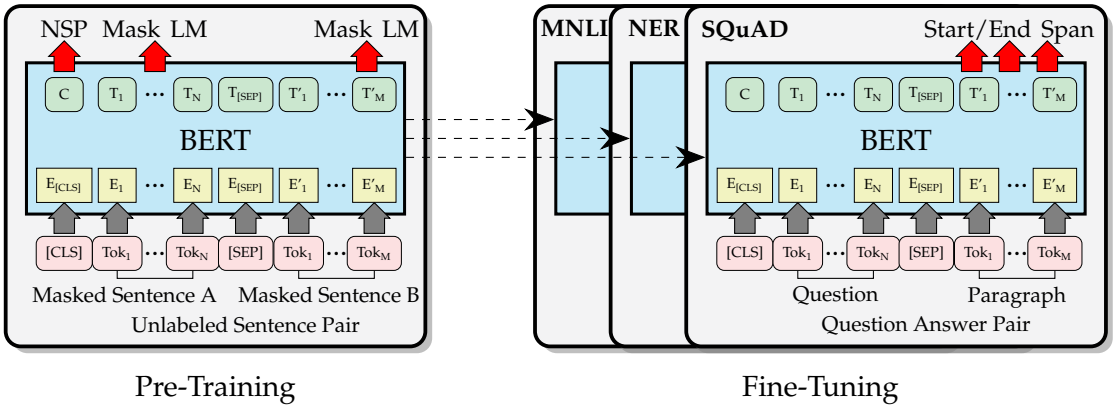


Figure 3: **BERT's pre-training and fine-tuning approach.** During pre-training, the model solves the Next Sentence Prediction and Masked Language Modeling tasks to learn basic language understanding. During fine-tuning, the pre-trained model is trained on labeled data to solve a specific downstream task. For example, in question answering, the model learns to predict the start and end span within a paragraph in order to answer a given question. (Figure: own work, inspired by Devlin et al. (2019)).

answering, require a sentence pair as input (e.g., a question and a text paragraph that contains the answer), while others, such as sentiment analysis, require a single sentence input.

The input sequence is first tokenized and mapped to token embeddings, to which positional encodings are added to capture word order. In addition, segment embeddings are introduced to distinguish between different parts of the input, such as two sentences. Furthermore, BERT uses special tokens to structure the sequence. Each input is prepended with a classification token `[CLS]`, and sentence pairs are separated by a `[SEP]` token. These enriched input representations are then processed by the stacked encoder layers.

Pre-Training. BERT is pre-trained on large unlabeled text corpora (Zhu et al., 2015) to learn general-purpose representations of text. The model is trained on two different tasks, namely Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

In the MLM task, approximately 15% of the tokens in the input sequence are randomly selected and replaced by a special `[MASK]` token or a random token from the vocabulary. The model is trained to predict the original tokens given their context, which resembles a Cloze test (Taylor, 1953). Consider the input sentence:

"The movie was absolutely fantastic."

After masking, the input becomes:

"The movie was absolutely [MASK]."

Let $\mathcal{M}$ denote the set of masked positions. The training objective is to minimize the negative log-likelihood of the original tokens at these positions:

$$L(\theta)_{\text{MLM}} = -\sum_{t \in \mathcal{M}} \log p_\theta(w_t \,|\, w_1, \ldots, w_{t-1}, \texttt{[MASK]}, w_{t+1}, \ldots, w_T), \tag{23}$$

where w_t denotes the original token (e.g., "fantastic"). This objective enables the model to learn bidirectional contextual representations, since each token is predicted using both left and right context.

Complementing this, NSP is a binary classification task that operates on pairs of sentences. The model is trained to predict whether the two input sentences are consecutive in the original text or whether the second sentence is randomly sampled. Consider the following sentence pair:

s_1: "The weather was beautiful today."
s_2: "We decided to go for a walk."

If s_2 follows s_1 in the original document, the label is *IsNext*; otherwise, it is *NotNext*. For instance, a negative example could be:

s_1: "The weather was beautiful today."
s_2: "Artificial intelligence is transforming many industries."

Formally, given two sentences s_1 and s_2, the objective is defined as:

$$L(\theta)_{\text{NSP}} = -\log p_\theta(y \,|\, s_1, s_2), \quad y \in \{\text{IsNext}, \text{NotNext}\}. \tag{24}$$

This objective encourages the model to capture relationships between sentences. Both objectives are optimized jointly during pre-training, allowing the model to learn both token-level and sentence-level representations.

We modified the pre-training process of BERT for the task of argument similarity, which is described further in Chapter 5.

Fine-Tuning. After pre-training, the model is adapted to specific downstream tasks through fine-tuning. The goal of fine-tuning the model is to refine its weights for specific tasks, without retraining the model from scratch. The model's weights are initialized with the values learned during pre-training, and they are refined for the task at hand using labeled data. For this purpose, a task-specific output layer, often referred to as a classification head, is added to the pre-trained model. This head can, for example, be implemented as a linear classifier. In a binary classification task, the classification head takes the final sequence representation of the special [CLS] token as input and outputs

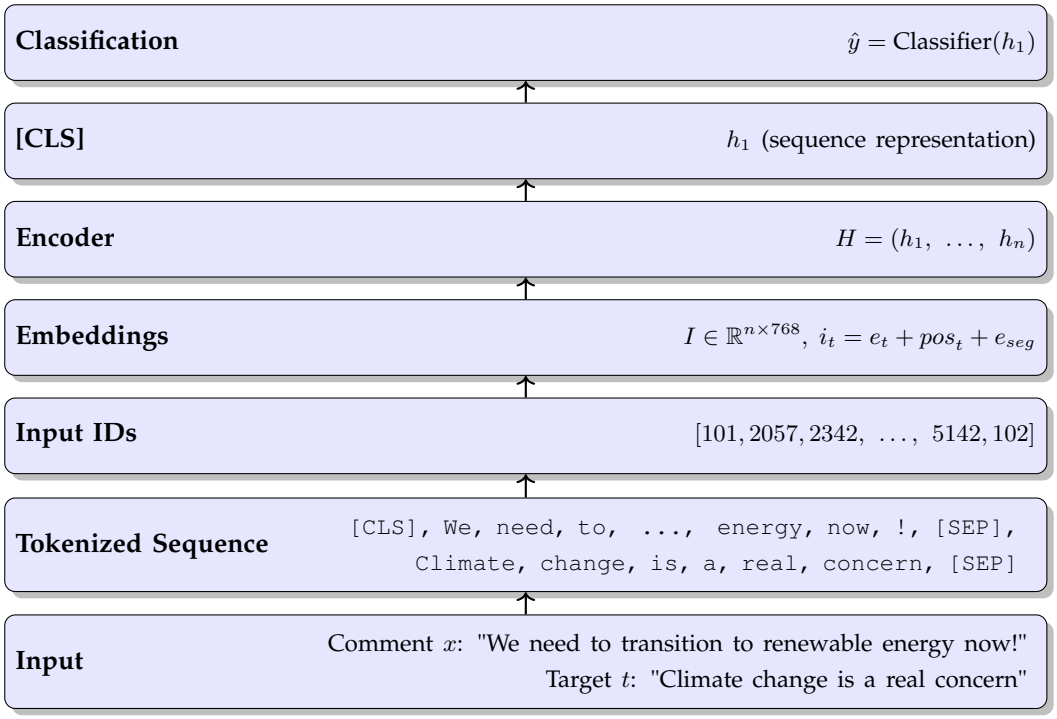


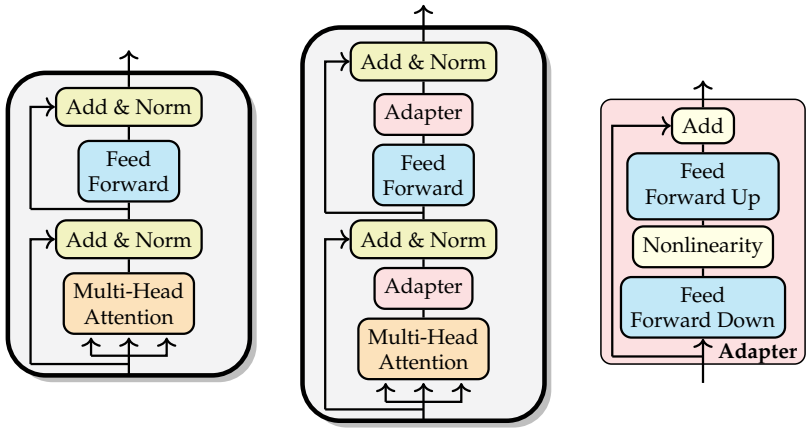
Figure 4: **Processing pipeline for stance detection using BERT.** The input is transformed into embeddings, contextualized by the encoder, and classified using the representation of the `[CLS]` token.

a task-specific prediction. The representation of the special `[CLS]` token serves as an aggregate representation of the entire input sequence, enabling efficient classification. We experimented with this input representation in Chapter 6.

Figure 3 summarizes the overall training procedure of BERT, where the pre-trained model serves as a starting point for fine-tuning on downstream tasks. The BERT model forms the basis for ArgueBERT and MaxPoolBERT (see Chapters 5 and 6).

3.5.1 Example: Stance Detection with BERT

To illustrate how BERT processes inputs in practice, we give an example of stance detection. Figure 4 shows the complete processing pipeline. The input consists of a user comment x and a target topic t, which are combined into a single token sequence using the special tokens `[CLS]` and `[SEP]`. This sequence is tokenized and mapped to input IDs, which are then transformed into dense vector representations via the embedding layer. The resulting embedding sequence $I \in \mathbb{R}^{T \times d_{\text{model}}}$ is passed through the BERT encoder, producing contextualized representations $H = (h_1, \ldots, h_T)$, where each vector h_i captures the meaning of a token in its full context. For classification, the representa-



(a) Encoder layer without adapter modules. (b) Encoder layer with adapter modules. (c) Single adapter module.

Figure 5: **Adapter modules in Transformer encoder layers**. Comparison of a Transformer encoder layer (a) without adapters and (b) with adapter modules inserted after the attention and feed-forward sublayers following Houlsby et al. (2019). (c) The internal structure of a single adapter module consisting of a down-projection, a non-linearity, and an up-projection with a residual connection.

tion corresponding to the `[CLS]` token, h_1, is used as an aggregate representation of the sequence and fed into the classification head to predict the stance label $\hat{y} \in C$.

3.6 Adapters

While the pre-training and fine-tuning paradigm has proven highly effective, updating all parameters of pre-trained language models such as BERT is computationally expensive and requires storing a separate model instance for each downstream task. Moreover, full fine-tuning can lead to catastrophic forgetting of previously learned representations (Xu et al., 2020).

To address these limitations, Rebuffi et al. (2017) proposed a parameter-efficient transfer learning approach in the context of computer vision. The key idea is to keep the pre-trained model parameters θ_0 fixed and introduce small task-specific modules with additional parameters Φ, which are trained for each new task. This allows the model to *adapt* to new applications, while only updating a small fraction of its parameters.

Adapter Module. The idea of adding additional modules into a fixed pre-trained model was later transferred to NLP by Houlsby et al. (2019), who introduced *adapter modules* for Transformer-based models such as BERT. An adapter module is a small bottleneck network inserted into each Transformer layer. Given an intermediate representation $h_l \in$

$\mathbb{R}^{T \times d_{\text{model}}}$, an adapter is defined as:

$$\text{Adapter}(h_l) = h_l + \sigma(h_l W_{\text{down}}) W_{\text{up}}, \tag{25}$$

where $W_{\text{down}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{bottleneck}}}$ projects the input to a lower-dimensional space, $W_{\text{up}} \in \mathbb{R}^{d_{\text{bottleneck}} \times d_{\text{model}}}$ projects it back, and σ is a non-linear activation function as depicted in Figure 5 on the right-hand side. For readability, bias terms are omitted from the notation. The residual connection ensures that the adapter modifies the representation without overwriting the original information.

In practice, adapter modules are inserted into each encoder layer, typically after the attention sublayer and the feed-forward sublayer, following Houlsby et al. (2019). This architecture is illustrated in Figure 5. They act as lightweight bottleneck transformations that allow task-specific adaptation without modifying the original model parameters. In experiments on the GLUE benchmark (Wang et al., 2018), the authors demonstrated that while training only a small fraction of the parameters (approximately 3.6%), their model achieved a similar overall score across the different tasks.

In this thesis, we build on this approach and employ the variant proposed by Pfeiffer et al. (2021) for the AQuA score (see Chapter 7).

4 Machine Learning for Enhancing Deliberation in Online Political Discussions and Participatory Processes: A Survey

Corresponding Publication. This chapter is based on the work:

Machine Learning for Enhancing Deliberation in Online Political Discussions and Participatory Processes: A Survey
Maike Behrendt, Stefan Sylvius Wagner, Carina Weinmann, Marike Bormann, Mira Warne & Stefan Harmeling

Personal Contributions. This work is a collaboration between computer scientists and social scientists as part of the UPEKI project. It served as the theoretical basis for the development of our own methods for use in online discussions. Mira Warne conducted the initial literature review, which included a literature search and classification within a theoretical framework. Carina Weinmann and Marike Bormann developed the theoretical framework, which is not part of the survey presented in this thesis but served as the basis for the publication by Friess et al. (2025). Maike Behrendt was primarily responsible for drafting the paper, which focuses on issues in political online discussions, tasks that can be addressed by AI, and applications and platforms that integrate AI features. The draft was revised and finalized with the help of all co-authors Stefan Sylvius Wagner, Carina Weinmann, Marike Bormann, Mira Warne, and Stefan Harmeling. After the review process, Maike Behrendt revised the paper, incorporated the reviewers' comments, and conducted an expanded literature search to identify and include recent work.

Abstract. Political online participation in the form of discussing political issues and exchanging opinions among citizens is gaining importance with more and more formats being held digitally. To come to a decision, a thorough discussion and consideration of opinions and a civil exchange of arguments, which is defined as the act of *deliberation*, is desirable. The quality of discussions and participation processes in terms of their deliberativeness highly depends on the design of platforms and processes. To facilitate online communication for both participants and initiators, machine learning methods offer a lot of potential. In this work we want to showcase which issues occur in political online discussions and how machine learning can be used to counteract these issues and enhance

deliberation. We conduct a literature review to (i) identify tasks that could potentially be solved by Artificial Intelligence algorithms to enhance individual aspects of deliberation in political online discussions, (ii) provide an overview on existing tools and platforms that are equipped with AI support and (iii) assess how well AI support currently works and where challenges remain.

Research Problem & Motivation. In this work, we conduct a systematic literature review on the use of NLP to enhance deliberation in online political discussions. Understanding the current state of research and the findings of previous projects was essential for developing and applying our own models. The literature review served as the starting point for our project, with the primary goal of identifying gaps in research, particularly in areas where less research has been conducted to date. We summarize known issues in online participation processes and other forms of online discussions and identify tasks that AI can address to support such discussions. We categorize these tasks within a deliberation-oriented framework and emphasize which aspects of deliberation could be promoted. We also discuss studies examining the effectiveness of AI in online discussions. Additionally, we list open-source platforms that use AI, as well as datasets that can be used to train models for the aforementioned tasks.

Main Results. We identified several challenges in online participation processes and related forms of online discussions. Overall, we found 36 tasks, organized into five subcategories, that can be supported by AI. Our research revealed that there is a particular focus on the automatic detection of offensive language to assist human moderators. Another clear finding was the importance of human-AI collaboration. While AI can provide valuable support for discussions and help scale them to large numbers of participants, final decisions should rest with humans.

Publication Status. Under Review. The survey is currently under review for publication in ACM Computing Surveys. The manuscript is available as a preprint (Behrendt et al., 2026).

4.1 Preamble

To establish a systematic foundation for this thesis, we present a survey of NLP applications in political online discussions and participatory processes. This publication addresses Research Question 1 by reviewing the current state of research, identifying relevant tasks, organizing them within a deliberation-oriented framework, and summarizing empirical evidence regarding their effectiveness.

4.2 Introduction

Over the last two decades, the amount of research has grown continuously, particularly with the ongoing emergence of new platforms that allow for exchange online. Besides simple exchange of opinions, online participation platforms are also effective instruments to make use of collective intelligence to find responses to severe challenges like pandemics (Mouter et al., 2021) or climate change (Introne et al., 2011).

Online political participation comes in many different forms. While some definitions include both passive (e.g., reading news articles) and active (e.g., voting) political behavior, this review focuses exclusively on active engagement through written opinion exchange, e.g., in online citizen councils, referendums and discussions on political topics (Ruess et al., 2023). This also includes informal exchanges on social media or news websites, which have been shown to stimulate citizens' political engagement (Conroy et al., 2012; Durotoye et al., 2025). From a deliberative perspective, participation in such discussions can be seen as an essential, if not central aspect of citizens' democratic involvement (Barrett and Brunton-Smith, 2014; Carpini et al., 2004; Torell, 2006).

The theoretical foundation for this perspective is deliberative democracy, which has become one of the most influential conceptions of democracy in political theory and research (Curato et al., 2022). It represents a normative approach, under which basically a multitude of deliberative theories can be subsumed, each with a different emphasis (Bohman and Rehg, 1997; Hamlin and Pettit, 1991; Dryzek, 2002; Fishkin and Laslett, 2008; Gutmann and Thompson, 2004). However, they agree that democracy should consist of a discursive exchange among equals with the participation of all members of society. This exchange should be characterized by rational arguments as well as the willingness to respectfully acknowledge and understand the viewpoint and arguments of the other side, to reconsider and, if necessary, to change one's own viewpoint. The goal of this exchange is to jointly find solutions to social problems (Chambers, 2003; Hamlin and Pettit, 1991; Habermas, 1994).

Deliberation can be characterized by the following four core dimensions (Friess and Eilders, 2015):

- *Rationality*: argumentative exchange of opinions.
- *Interactivity*: being responsive and listening to each other.
- *Equality*: same opportunities for anyone.
- *Civility*: respectful interaction between participants.

The degree to which online discussions fulfill these deliberative criteria depends strongly on both platform design (Esau et al., 2017) and the presence of effective moderation. Human moderators can substantially improve discussion quality (Friess et al., 2021; Schluger et al., 2022), yet moderation is a resource-intensive task that is difficult to scale across large online platforms (Kuo et al., 2023). Recent advances in AI, and particular

in Machine Learning, and Natural Language Processing, offer promising opportunities to support and enhance online deliberative processes in online discussions (Argyle et al., 2023). ML methods can help structure large-scale exchanges, identify patterns in argumentation or sentiment, and foster more civil and constructive dialogue. NLP, as a subfield of ML and computational linguistics, provides tools for analyzing and generating human language, thereby enabling automated facilitation and moderation of online discussions. The technical terms such as ML and NLP are defined in detail in Section 4.4.1.

Scope and Focus of This Review. The overarching goal of this paper is to explore how ML and NLP can contribute to enhancing deliberation in online political discussions. Specifically, we aim to:

- provide a comprehensive overview of common issues in online discussions and participation processes,
- identify and describe ML and NLP tasks that could enhance deliberative quality in online discussions,
- summarize existing studies examining the effects of ML based interventions,
- review current ML and NLP techniques applied in online discussion contexts,
- and outline open challenges and directions for future research.

From these objectives, we derive the following research questions:

1. What are the main issues that occur in political online discussions and participatory processes?
2. Which tasks can potentially be addressed with ML or NLP algorithms?
3. How can ML and NLP contribute to enhancing deliberation?
4. How well do algorithms perform at the moment and what challenges remain?

This review concentrates on political online discussions and participation processes. We exclude applications in online education and collaborative learning platforms, where objectives and user dynamics differ substantially (for an overview, see Ouyang and Zhang (2024)). Similarly, tools designed for group meetings or seminars, such as the discussion board system by Sasaki et al. (2021), fall outside our scope. Given the particular importance of transparency and accessibility for democratic participation, our review focuses on open-source or non-profit platforms that are freely available to the public. Overall, we identified 36 tasks within five sub-categories that could be solved by ML methods to help to enhance deliberation in online discussions. We found 23 software tools that have

been developed or are still being developed that already incorporate NLP to enhance online discussions. Together, these findings provide a systematic overview of the state of research and technology in this emerging field.

Our paper is structured as follows: first, we describe the methodology employed for conducting the extensive literature review in Section 4.3. We then define the most important terms in Section 4.4.1 and discuss known issues in both online participation processes as well as general online discussions in Sections 4.4.2 and 4.4.3. Next, we provide an overview of the tasks that can be solved by ML methods to enhance deliberation in online discussions in Section 4.5. Subsequently, we present software tools that already make use of NLP methods in practice in Section 4.6. Finally, we outline key challenges and open problems in Section 4.8. Section 4.9 concludes the paper with a summary of our findings and an outlook on future research directions.

4.3 Methodology

We employed an iterative literature review process, which we describe below. A comprehensive search was conducted across relevant databases in NLP, with academic search engines such as Google Scholar serving as a starting point for a systematic, forward and backward snowballing process, as outlined in (Wohlin, 2014). Snowballing, also known as citation chaining, is the process of identifying relevant works and examining the cited literature, as well as the literature that has cited the relevant works, to locate additional significant papers. To ensure the timeliness of the research, the study exclusively considered publications subsequent to 2014. We did not restrict our search to a certain research field, as the topic is highly interdisciplinary. The criteria established for potential tasks to solve in order to enhance online deliberation were rather broad. In the case of discussion and political participation platforms we excluded platforms that were not (1) open source, or non-profit, (2) supported by any kind of AI, and (3) developed for online discussions, deliberation and participation processes, without a special focus on, e.g., online education and business meetings. Additionally, the Democracy Technologies Database[2] was utilized to identify discussion tools and platforms. In total, we identified 36 tasks, which we assigned to 5 different categories, as well as 23 software tools for online discussions or their evaluation, and 22 datasets for training and evaluating ML models on discussion data.

"AI" / "NLP" / "Machine Learning"
+ "citizen councils"
+ "citizen participation"
+ "decision making"
+ "e-participation"
+ "group decision making"
+ "group discussions"
+ "online deliberation"
+ "online discussions"
+ "moderation"
+ "social media"
+ "social sciences"
"facilitation of online discussions"
"issues in online discussions"

Table 1: Used search terms for the conducted literature review.

[2] https://democracy-technologies.org/database/

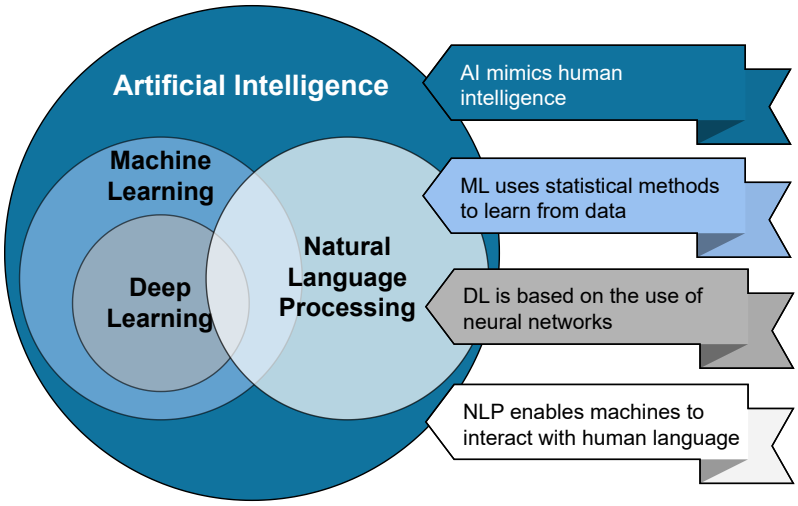


Figure 6: Classification of Artificial Intelligence related terms.

Search Terms. Table 1 lists search terms we used in different combinations to gain a broad overview on the topic of ML and NLP to support political online discussions and online participation processes.

4.4 Background

Before discussing tasks that can be solved by NLP to support participants and initiators of online participation processes and online discussions in general, we want to first provide the reader with a definition of AI, ML, NLP, Large Language Models and deliberation, and afterwards shed light on challenges in both contexts to better understand how ML and NLP can be deployed.

4.4.1 Terminology

This review is aimed at computer scientists, as well as social and political scientists, as the reviewed literature comes from a highly interdisciplinary field where technology and the theory of deliberation converge. In order to define the technical terms used in this paper, they are explained in detail below.

Artificial Intelligence. AI is the study and design of algorithms that enable machines to perform tasks that normally require human intelligence, such as learning, reasoning, and problem-solving.

Machine Learning. ML is a branch of AI that enables computers to automatically learn from data and improve their performance without explicit programming. ML bridges computer science, mathematics, and statistics, and provides the foundation for many applications such as pattern recognition, predictive modeling, and NLP.

Deep Learning. DL is a branch of ML. Traditional ML methods often rely on humans to select features that are fed into algorithms such as decision trees. In contrast, deep learning is based on artificial neural networks that learn features directly from the data (Janiesch et al., 2021). Deep learning uses layered neural networks which are modeled on the structures of the human brain. These networks can automatically discover important patterns in large amounts of data, e.g., learning to recognize objects in images without being told which features to look at. This makes deep learning effective for tasks such as speech recognition, language understanding, and image classification.

Natural Language Processing. NLP is a subfield of computer science and computational linguistics that develops methods to enable machines to process and interact with human language. The field covers a broad spectrum of tasks, including semantic text analysis, sentiment detection, morphological segmentation, machine translation, and text summarization. In the context of online discussions, text contributions are the main data, which is why NLP is relevant here. Figure 6 illustrates the relation between AI, ML, DL and NLP.

Large Language Models. Large Language Models (LLMs) are a recent development in NLP, based on DL. LLMs are deep neural networks that are trained on extensive collections of text. They learn to statistically represent linguistic structures and meanings. Consequently, LLMs can generate and interpret language in a remarkably flexible and human-like way. Because they can generate content, LLMs are considered *generative AI*. Building upon advances in m ML, LLMs have become central to applications such as chatbots, text summarization, and automated writing assistance.

Deliberation. Deliberation can be understood as a communicative process in which participants respectfully and thoughtfully exchange arguments with the goal of forming a collective judgment. It is commonly described along four dimensions: *rationality*, which refers to the logical and evidence-based justification of positions; *civility*, which encompasses respect and politeness, *interactivity*, which highlights engagement and responsiveness in dialogue; and *equality*, which ensures that every participant has the same opportunity to participate (Bächtiger et al., 2009b; Esau et al., 2021; Graham, 2010).

4.4.2 Issues in Online Participation Processes

We will take a closer look at issues that have been reported for online participation processes in the following. The first challenge that initiators of participation processes have to deal with are *low rates of participation*, due to various reasons such as lack of motivation, lack of awareness about the process, or lack of online access (Shortall et al., 2022; Davies and Procter, 2020). Promoting public participation projects and ensuring their success is, however, a task that is out of scope for NLP methods, as it takes place before the actual process of participation. Another phenomenon in participation processes, in which citizens can make proposals or share ideas, is the large number of gathered posts. Too many contributions cause what is referred to as *information overload*, which is denoted as one of the key challenges for digital mass participation by Arana-Catania et al. (2021a). One effect of information overload is the *redundancy of contributions* which makes it difficult for participants to find proposals of interest to support or comment (Davies and Procter, 2020; Lago et al., 2019). Being confronted with a multitude of postings on a platform makes it impossible for participants to gain an overview of ideas that have been added and leads to *a lack of interaction* as they solely share their own ideas without noticing others. As the number of postings grows, it also becomes difficult for local authorities to *evaluate* the process afterwards (Lago et al., 2019).

Simonofski et al. (2021) analyze requirements of both participants and public servants for a participation process in Belgium and elaborate guidelines for the development and implementation of participation platforms. One guideline includes the consideration of NLP support to cluster and summarize submitted ideas. Balta et al. (2019) examine how the application of NLP in online citizen participation could be standardized. They point out the importance of high-quality data for training models in order to make adequate predictions. They also state that human based analysis is still preferable for some tasks as humans perform much better on the explored NLP tasks, while additionally the decisions of a ML method are often not explainable.

Tsai et al. (2024) present a framework for what to consider and what guiding principles to follow when integrating generative AI into pro-democratic citizen participation platforms. They see great potential in augmenting processes with AI to allow for more informed processes in which participants have equal opportunities to participate, provided that the user's agency and autonomy are not compromised. However, they also point out fundamental risks, such as potential over-censorship by AI agents, the misuse of AI to generate deepfakes, and over-reliance on AI.

4.4.3 Issues in Other Forms of Online Discussions

Several reports and studies have examined issues and drawbacks in online discussions on social media platforms in general. Since these issues can likely be transferred to political contexts, we will summarize them in the following.

The reported issues include *disorganized content*, *redundancy in posts*, *many contributions of low quality*, and *polarization* (Klein, 2011). Other issues are *formation of small groups* that share the same opinion, which is connected to the emergence of filter bubbles and echo chambers, *poor argumentation*, which leads to *biases* and a *lack of references* to the contributions of other participants (Klein, 2015; Hadfi and Ito, 2022; Gao et al., 2013), and incivility toward other users (Anderson et al., 2014). Many of the mentioned problems in online discussions are related to one or multiple dimensions of deliberation. Human facilitators are responsible for managing discussions and preventing the aforementioned issues. However, the growing number of participants and accompanying increase in posts make it difficult to maintain an overview (Ito et al., 2014). NLP methods could support and simplify the work of human facilitators and initiators of political online discussions. Selecting the appropriate tools and facilitation mechanisms is a crucial part of digital online participation in order to conduct a successful process and rule out possible negative side effects (Simon et al., 2017).

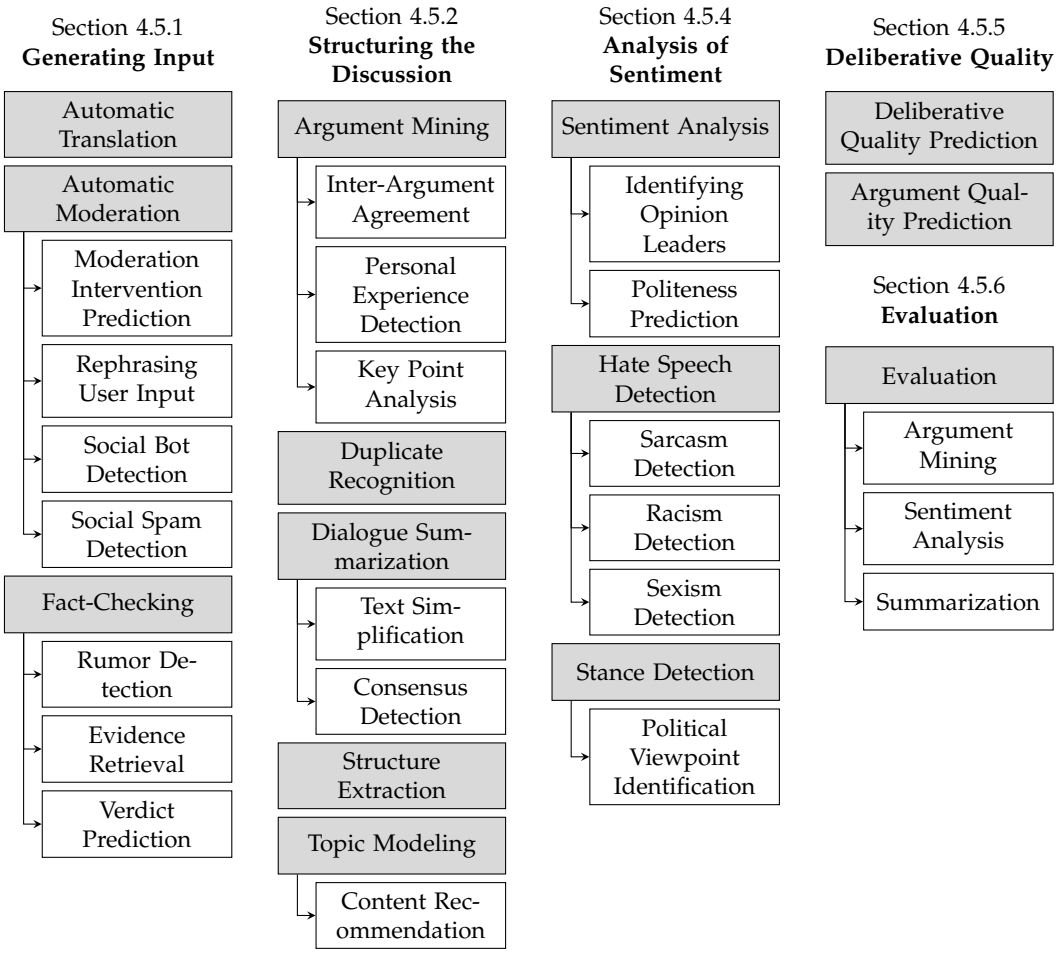


Figure 7: Identified NLP tasks that could potentially support and improve online discussions.

4.5 Tasks to Solve in Online Discussions

During the review of literature on the topic of NLP and ML to support online discussions, we identified a list of tasks with great potential for facilitating and supporting online discussions. In the following, we explain each task and list important literature. Please note that, since a large body of literature exists for each task, we only discuss significant work or refer to recent overviews summarizing the literature on specific topics. If available, we also provide evidence showing effectiveness of these tasks when applied to online discussions. However, it is important to note that the usefulness of an AI tool applied in an online discussion depends on multiple factors. While the performance of the tool is important, the design of the platform is equally relevant (Esau et al., 2017).

Many of the tasks discussed in this section were initially considered classification tasks that relied on annotated data to train supervised classification models. With the rise and improvement of large generative language models such as GPT-5[3], a new paradigm has established and much research evaluates the capacities of these generative models to solve NLP tasks to enhance deliberation (Wachsmuth et al., 2024) and to generate and augment data to train models (Wagner et al., 2025).

We categorize the identified tasks (see Figure 7 for an overview) into the following categories: (i) tasks that generate input, (ii) tasks to structure the discussion, (iii) tasks to analyze the sentiment, (iv) tasks that measure discourse quality and (v) tasks that help evaluate the discussion. Some tasks appear multiple times, as, e.g., sentiment analysis and argument mining can be applied during a discussion, as well as during the evaluation phase.

4.5.1 Generating Input

Probably one of the most direct ways NLP can be used in discussions is to let a model generate input for the discussion. In the following, we take a look at NLP tasks that enhance the discussion by generating input in some way. This can range from simple automatic translations to more complex tasks like automated moderation.

Automatic Translation. Automatic translation of texts holds great potential for online deliberation, as it allows for more inclusiveness and thereby *equality* of participants, by breaking down language barriers. If high-quality translations can be provided in real time for any language, more people with different cultural backgrounds can engage in decisions and share their opinion. With the emergence of the Transformer model by Vaswani et al. (2017), automatically generated translations have become fairly reliable and are currently mostly researched for low resource languages. Anastasiou (2022) for example proposes a BERT-based (Devlin et al., 2019) language model for Luxembourgish that will be applied in public administration as a chatbot for automatic translation

[3] https://openai.com/index/introducing-gpt-5/

and question answering. Stahlberg (2020) provides a recent overview on methods and applications for automatic translation.

Automatic Moderation. Moderation of online discussions is a crucial task that often determines the quality of held discussions (Friess et al., 2021). It is a time and resource consuming task, that could be facilitated or even replaced by artificial moderators. As an initial approach to support human moderators, Falk et al. (2021) train models on the task of moderation intervention prediction on the Regulation Room Corpus (Park et al., 2012). They aim to classify whether a comment needs intervention, which can then be suggested to a human moderator. In their study, Horta Ribeiro et al. (2023) examine the effects of fully automated content moderation. They apply a system that automatically hides or deletes comments that break community guidelines. They find that automatic deletion of comments has a positive effect on the posting behavior of users. Only hiding content did not obtain the same effect. Similar effects have been found when intervening even before posting harmful content (Katsaros et al., 2022). Argyle et al. (2023) use a GPT-3-based (Brown et al., 2020) chat assistant in one-on-one discussions about gun control that suggests rephrased messages in real time. Their study shows that the quality of conversations and the feeling of being understood by their counterpart could be improved through the application of the ML assistant. Checking user comments for their tone and suggesting reformulations can increase *civility* of discussions, creating a better atmosphere in which more people tend to express their opinions. Klein (2025) suggests to let LLMs generate more precise user contributions in cases where they lack context or use complex terms that not everyone understands which contributes to the *equality* of each participant within the discussion and could also improve *interactivity* between users.

There are many other examples of recent developed chatbots, e.g., to structure the conversation (Kim et al., 2021), to identify harmful language (Clever et al., 2022; Bonechi, 2024) and spam (Bonechi, 2024), or to interact with participants by posting messages (Ito et al., 2022).

The technology behind supportive automated moderators is the same as that used by social bots. The difference between the two is their purpose. Social bots appear as human profiles on social media, however, they are used to spam, spread misinformation, and manipulate users on social media platforms, such as Twitter (Hagen et al., 2022). Therefore, an important task for participation processes and social media discussions is detecting social bots (Fazil et al., 2021) and social spam (Rao et al., 2021).

Recent studies show that automated content moderation has a great potential in reducing harmful content on online discussion platforms, which enhances deliberative values, especially *civility*. AI-assisted moderation was found to motivate participants to write more constructive comments and follow the given rules on online platforms (Yu et al., 2024), enhancing both *interactivity* and *rationality*. Nevertheless, the use of AI models should be transparent, and users should be informed why their comment was deleted. LLMs can perform this reasoning process (Kolla et al., 2024), but human moderators currently make the most comprehensible decisions and can therefore be supported by AI, without

being completely replaced. For a recent systematic literature overview on the effects of automatic content moderation see the work of Yu et al. (2024).

Fact-Checking. Fact-checking is a crucial task in online discussions, as the spread of misinformation online is a severe problem (Rocha et al., 2021). Some attempts have been made to create fact-checking pipelines with neural network based models. Guo et al. (2022) give an extensive overview of the tasks that need to be solved in the fact-checking pipeline and datasets used to train models for these tasks. An important part of fact-checking is claim detection, i.e., identifying content that should be checked because it contains some kind of claim. Rumor detection, i.e., the detection of unverified claims about any event, has been employed for this purpose, as, e.g., by Guo et al. (2018). Another important aspect of fact-checking is evidence retrieval. Recently, so called neural retrievers have emerged, which are models that, retrieve corresponding evidence given a query (Maillard et al., 2021). Arguably, the most crucial part of fact-checking is verdict prediction, i.e., labeling the fact checked content. Simple models are binary classifiers which label a given claim as either true or false (Potthast et al., 2018). However, a more fine-graded labeling is commonly used, e.g., *false, barely-true, mostly true*, etc. (Wang, 2017). The final step in the fact-checking pipeline is to generate a justification for the given prediction, as explainability plays an important role when some fact is predicted as false. For this purpose models are trained to produce a textual justification (Atanasova et al., 2020). Chen and Shu (2024b) examine the opportunities and challenges of LLMs in tackling the task of fact-checking. LLMs have the ability to generate fake news on a large scale, which, according to the latest findings, is very difficult to distinguish from real news (Chen and Shu, 2024a). Nevertheless, the great advantage of using LLMs for fact-checking is that external knowledge can be fed into them. Automated web searches can also help ensure that the models have access to up-to-date information.

Discourse that is automatically fact-checked is more *rational*, as the accuracy of claims increases and participants might be motivated to verify their positions. Current systems are effective at identifying and verifying check-worthy claims, especially if they have access to up-to-date information. However, it should not be forgotten that LLMs are capable of generating and spreading misinformation that sounds valid.

4.5.2 Structuring the Discussion

Another group of NLP tasks deals with using the information contained in discussions to provide a clearer, more structured overview.

Argument Mining. The aim of argument mining is to automatically recognize and classify argumentative structures in text. Boltužić and Šnajder (2014) initially proposed the task of argument recognition and come up with an approach for argument-based opinion mining. In a follow-up work, Boltužić and Šnajder (2015) extended the task of argument

recognition by suggesting an unsupervised method to identify the most prominent argument in an online discussion. Harly and Girsang (2022) use a Convolutional Neural Network (CNN)-BERT architecture to measure inter-argument agreement in online discussions. Falk and Lapesa (2022a) try to recognize the utilization of personal experience in argumentation. One sub-task of argument mining is to select a set of keypoint arguments from a collection of given arguments (Friedman et al., 2021; Alshomary et al., 2021). By extracting and displaying the key arguments of a discussion to participants, they can inform themselves about the discussion, keep an overview and form their own opinion. This contributes to a more informed, thus more *rational* discussion.

Duplicate Recognition. Duplicate contributions, i.e., posts in discussions that share the same idea, but use a different wording, constitute a significant challenge for participants and initiators of large-scale online participation processes.

Yang et al. (2006) tackle the task of recognition of near duplicate entries for e-rulemaking and propose an algorithm called DURIAN. There is not much recent work on identifying duplicate posts in online discussions and participation processes. However, the task of identifying semantically similar texts, which is also known as paraphrase identification in the NLP community, is useful for many areas, reaching from the identification of duplicate questions in online forums (Prabowo and Budi Herwanto, 2019) to detection of plagiarism (Pokharana and Garg, 2023). Zhou et al. (2025) give a recent overview of methods and datasets for paraphrase identification.

Automatically identifying and marking similar posts in discussions improves clarity and counteracts the problem of information overload, allowing for a more *rational* and structured discussion.

Dialogue Summarization. Automated summarization of written text, or more specifically, dialogue between multiple people, is another task that supports online discussions. It is closely related to argument mining, but deals with other challenges, such as capturing all expressed perspectives without omitting any points of view. Summarization can provide key points for participants that join an ongoing discussion. For example, Anastasiou and De Liddo (2021) found that presenting participants with a report on given arguments can improve users' ability to make sense of the discussion and their perception of its quality. Arana-Catania et al. (2021b) evaluate and compare different ML models to summarize deliberative processes for non-English languages. To solve this task LLMs can be trained and utilized. For this purpose, Fabbri et al. (2021) present a benchmark dataset for conversation summarization called ConvoSumm.[4] Small et al. (2023) use an LLM to generate summaries for Polis (Small et al., 2021). They propose a collaborative approach between LLMs and humans to evaluate automatically generated summaries because they pose the risk to leave out points of view, especially if they incorporate any kind of bias, which is very critical in the context of political and socially relevant dis-

[4] https://github.com/Yale-LILY/ConvoSumm

cussions. When generating summaries, models could focus on areas of agreement or disagreement among participants. We refer to this task as *consensus detection*. Pointing out areas where participants agree can strengthen the discussion. Highlighting areas of disagreement can lead to a focused search for solutions (Bakker et al., 2022; Small et al., 2023).

Another sub-task of text summarization, is the *simplification* of the used language (Al-Thanyyan and Azmi, 2021), enabling more people to understand what the discussion is about.

For a comprehensive overview on recent methods and challenges of dialogue summarization, see the work of Jia et al. (2023). Simple summaries of debates at the touch of a button could increase the *equality* of online deliberation and motivate people to participate.

4.5.3 Structure Extraction.

Extracting the structure of online discussions is beneficial for facilitation tools in order to gain further insight of the ongoing discussion and extract important parts. The task involves identifying the relationship between messages and the flow of the conversation. There are different approaches to structure extraction. It can be solved with graph-based models, since conversations often exhibit tree or graph like structures in which posts are nodes and the edges represent their relationships. The process of structure extraction involves two main steps. First, nodes are classified, and then their relationships are predicted. Suzuki et al. (2021) present a node classification approach using graph attention networks (Veličković et al., 2018).

Gupta and Klein (2025) propose CivicParse, a benchmark and a pipeline for structured online deliberation. The pipeline is two-staged, first extracting distinct points of discussion that are classified with a content type, e.g., if the commenter proposes a solution to the discussion and a stance *(pro, con)*. Babatunde et al. (2025) use an LLM to automatically embed a user's response in a deliberation map. The task for the LLM is to identify questions, proposed solutions and arguments, as well as the stance of a given argument to create deliberation maps that make it easier for participants to collaboratively discuss questions and find solutions for them.

Structure extraction is usually a combination of extraction and classification and contributes to a clearer, more *rational* discussion. The models presented solve this task with high precision.

Topic Modeling. Topic modeling is the automated identification of the topic being discussed in a text. Sun et al. (2023) present a topic model using deep learning methods that is tailored for conversations. Automatically identifying the topic of a written post, e.g., on citizen platforms that gather ideas for improving cities, makes it easier for participants to gain an overview and find posts they are interested in. Furthermore topic modeling

could also help to find similar proposals to merge or suggest to other participants. In their experiments on Polis, Small et al. (2023) show that prompting LLMs with topic modeling is an effective way to support and structure discussions. They use Anthropic's Claude (Askell et al., 2021) to not only categorize the comments but also to produce two levels of topic labels. The authors emphasize that the task can be handled very well by LLMs and they consider the task less critical. Errors in topic assignments do not impact the discussion negatively, but a successful categorization can help to structure large-scale discussions.

In addition to a better structure, assigned topics are also helpful to recommend contents to participants they might be interested in. This saves participants a lot of time and promotes exchange between people who have similar interests and pursue similar goals. This could significantly increase *interactivity* and facilitate participation in general.

4.5.4 Sentiment Analysis.

Sentiment analysis, also referred to as opinion mining, is the automated identification of a users feelings, opinion and other subjective elements in text. While traditional sentiment analysis focuses on categorizing text as *positive*, *negative*, or *neutral* and is often used to assess the overall tone of discussions as an evaluation step, sentiment signals in online discourse can also serve as a strong indicator of hateful or harmful content (Sheth et al., 2023).

Khan et al. (2023) employ a Transformer-based architecture (Vaswani et al., 2017) to address the task of politeness detection. A recent survey by Sharma et al. (2025) provides an extensive overview of sentiment analysis, including its core sub-tasks, benchmark datasets, and unresolved challenges.

Hate Speech Detection. Hate speech, toxic language and incivility often occur when people with opposing opinions discuss controversial topics. Civility is a central aspect of deliberation, as participants should feel heard and equally accepted in a conversation (Friess and Eilders, 2015). Hate speech detection is the task of automatically identifying whether a given piece of text contains hateful or offensive language. A widely used ML-based approach for recognizing toxic language in online discussions is Perspective.[5] Perspective is a free to use application programming interface (API) that employs ML models to compute a toxicity score for user-generated comments. Hate speech detection encompasses several sub-tasks such as sarcasm (Băroiu and Trăușan-Matu, 2022), racism (Lee et al., 2022) and sexism (Schütz et al., 2021) detection. For an overview of recent methods and datasets for textual hate speech detection, we refer to the work of Alkomah and Ma (2022).

[5] https://perspectiveapi.com/

Effectively recognizing and moderating hateful comments has been shown to significantly improve the climate of discourse (Horta Ribeiro et al., 2023) and, consequently, increase overall *civility*.

Stance Detection. Stance detection is a sub-task of sentiment analysis which aims to predict an author's position with regard to a given topic or central question (e.g. distinguishing between *in favor* and *against*). It is, among other applications, commonly used to identify positions in political online discussions (Walker et al., 2012b). Stance detection plays an important role for a variety of downstream tasks such as debate summarization and structuring. A recent in-depth survey on the task of political viewpoint identification is given by Doan and Gulla (2022). Vamvas and Sennrich (2020) present the multilingual X-Stance dataset for stance detection in political contexts.[6]

In the context of online discussions and online participation, the automatic detection of an authors position could be leveraged to structure the debate more effectively. For instance, it enables filtering and grouping of arguments to improve overview, or even to expose participants to opposing viewpoints and encourage exchange of opinions to potentially enhance *interactivity*.

Barriere and Balahur (2023) train transformer models on different stance detection datasets of public discourse. The anticipated effect of stance detection strongly depends on how the predictions are used.

4.5.5 Discourse Quality Measurement

In contrast to filtering out rule violations in discussions, highlighting particularly high-quality contributions is another moderation strategy. Marking posts in discussion that fulfill deliberative standards, motivates participants to write better comments, improving the overall discourse quality (Wang and Diakopoulos, 2022).

ML models that can assess the quality of a contribution could be used not only for this task, but also to evaluate tools and systems that aim to improve deliberation in online context. We provide an overview of previous approaches in the following.

Measuring Deliberative Quality. DelibAnalysis, developed by Fournier-Tombs and Di Marzo Serugendo (2020) is a framework to analyze discussion quality based on the Deliberative Quality Index Steenbergen et al., 2003. The Deliberative Quality Index (DQI) is a popular measure for deliberative quality. By now, other measures have been developed for different factors of quality, e.g., the listening quality index (Scudder, 2022) and the deliberative reasoning index (Ercan et al., 2022). Behrendt et al. (2024) propose a deliberative quality score for comments that is called AQuA, which is composed of 20 individual dimensions, such as justification and references from other users.

[6] https://github.com/ZurichNLP/xstance

Fournier-Tombs and MacKenzie (2021) tackle the task of quality assessment with different ML techniques and predict a developed version of the DQI. They point out that for training these models, annotated datasets of high quality are needed. One dataset that includes contributions of a European-leveled poll annotated with the DQI, is EuroPolis (Gerber et al., 2018). In their work, Falk and Lapesa (2022b) show how to augment data annotated for deliberative quality prediction in order to avoid class imbalances.

Argument Quality Prediction. A subtask of measuring the overall deliberative quality of a discussion is to measure the quality of given arguments. Automatically accessing a score for given arguments can also be used to support human moderators in creating meaningful summaries. Van der Meer et al. (2022) try different approaches based on GPT-3 (Brown et al., 2020) and RoBERTa (Liu et al., 2019) to predict the validity and novelty of arguments.

Exchanging clear and convincing arguments is essential in deliberation, especially on socially relevant topics. Finding well-formulated and logically consistent arguments improves the overall *rationality* of a discussion and helps with the decision-making process. LLMs could be used not only to measure and predict argument quality, but also improving the quality of written arguments (Wachsmuth et al., 2024). Rescala et al. (2024) examine how well LLMs can recognize the quality of arguments and choose persuasive arguments based on demographics of the target group. They find that LLMs perform on a level similar to humans, which turned out to be not particularly high. Nevertheless, the authors see the risk that LLMs could spread misinformation based on particularly convincing arguments.

For a recent overview on the topic, see the work of Wachsmuth et al. (2024).

4.5.6 Evaluation of Participation Processes

In the final phase of an online participation project, municipalities are faced with the challenge to evaluate the data they collected and transform it into political action. Evaluation is a time-consuming task in which NLP methods can make a noticeable difference (Romberg and Escher, 2023). As previously mentioned, many tasks, such as sentiment analysis and text summarization, are used for evaluation, and will not be explained in detail again. In the following, we review previous work on evaluating participation processes using NLP methods.

An important part of the evaluation is gaining insight into citizens' proposed arguments. Liebeck et al. (2016) automatically extract components of arguments from a participation process regarding a German airport. Romberg and Conrad (2021) also follow this approach for a mobility-related urban planning in Germany. They examine different ML models to recognize argumentative structures and classify them. Cernuzzi et al. (2020) use rule-based sentiment analysis with lexicons to calculate sentiment scores (either positive, neutral, or negative) for civic participation comments. Their work establishes the

Tool	Link	Task
All Our Ideas	https://all-our-ideas.citizens.is/	Engagement Tool
bcause	https://bcause.app	Discussion Platform
Civic CrowdAnalytics	https://github.com/ParticipaPY/civic-crowdanalytics	Discussion Evaluation
CollAgree	-	Discussion Platform
CommunityPulse	https://communitypulse.cs.umass.edu/	Discussion Evaluation
CONSUL	https://consuldemocracy.org/	Citizen Participation Platform
ConvoKit	https://github.com/CornellNLP/ConvoKit	Conversation Analysis Toolkit
Coral	https://coralproject.net/	Commenting Tool
Debate Hub	https://debatehub.net/	Discussion Platform
DebateVis	https://debate-vis.xiaohk.vercel.app/	Visualization
Decidim	https://decidim.org/	Citizen Participation Platform
Deliberatorium	https://deliberatorium.org/	Discussion Platform
DIPAS	https://dipas.org/en	Citizen Participation Platform
Discourse	https://www.discourse.org/	Discussion Platform
Go Vocal (former CitizenLab)	https://www.govocal.com/	Discussion Platform
LiteMap	https://litemap.net/	Visualization & Evaluation
Opinion Space	-	Visualization
Policy Synth	https://policy-synth.ai/	Smarter Crowdsourcing
Polis	https://pol.is/home	Discussion Platform
Potluck (Prototype)	-	Discussion Platform
RHETORiC	https://www.rhetoric.tech/user/login	Commenting Tool
ThreadReconstructor	-	Visualization
Your Priorities	https://yrpri.org	Discussion Platform

Table 2: List of discussion platforms mentioned in this work.

core functionality for the Civic CrowdAnalytics tool. Other tools for the automated evaluation of participation processes are described in Section 4.6.3. For a recent overview of NLP methods for the evaluation of public participation projects, see the work of Romberg and Escher (2023).

The AI-supported evaluation of online discussions and participation processes does not directly improve the discussions themselves. Nevertheless, if carried out precisely, taking into account all the opinions gathered and deriving actionable results from them, it is a very important step which can lead to greater trust in administrations. Well-evaluated processes increase citizens' motivation to participate in participatory processes. Ensuring that all voices are heard guarantees *equality* of all participants in retrospect. As with every other aspect concerning the use of AI tools to support discussions, these tools should be adopted to support human actors, and not replace them.

4.6 Overview of NLP in Practice

In the following Section we want to take a closer look on tools and platforms that integrate NLP or more general ML to enhance deliberative processes.

4.6.1 Discussion Platform Software

There is a wide range of different online discussion and participation platforms that include some kind of algorithmic facilitation, both open source and proprietary. In the following, we solely regard *open source*, or at least *free to use* non-profit platforms, as the transparency and accessibility of platforms is especially important for democratic processes. A list of all mentioned platforms with their link, if available, can be found in Table 2. For a list that also includes proprietary platforms, see the Democracy Technologies Database[7].

All Our Ideas is an engagement tool that is used to co-create a ranked list based on public input (a so called wiki survey). The tool enables ideas for a given question to be gathered and voted on. An integrated generative AI can automatically generate suitable graphics for the survey and also suggest answers that participants can vote on. Gambrell (2025) examines how an AI Task Force from the State of New Jersey was able to effectively use All Our Ideas to develop actionable policy recommendations.

bcause is a platform for accessible, structured and decentralized online deliberation, developed by Anastasiou and De Liddo (2026). It includes automatic transcript analysis, argument mining and intelligent feedback aggregation and automated reporting with visualization tools. The authors aim to build a discussion environment with an integrated analytic engine that allows for more qualitative online debates. An experiment conducted by the research group showed that automated created reports can help users to make sense of an ongoing discussion and also improves the overall debate quality (Anastasiou and De Liddo, 2021). Recently, the platform has been expanded by an argument mining tool (Elguendouze et al., 2025).

COLLAGREE (short for collective or collaborative agreement) is an online discussion platform that supports consensus decision-making, developed by Ito et al. (2014) and Ito et al. (2015). COLLAGREE implements functions to support discussion facilitators. These functions include a text-based sentiment analysis, an automated agreement/ disagreement prediction for each post, and an automatic generation of highlighted keywords. Ito et al. (2014) also added an incentive mechanism that rewards participants for the effectiveness of their posts.

CONSUL is another citizen participation tool for debates, proposals, budgeting and polls. Arana-Catania et al. (2021a) develop and evaluate different NLP tools for the platform to counteract the problem of information overload that is often observed in digital mass participation. In their project, the authors mainly focus on tackling information overload with regard to proposals that can be added to the platform by citizens. The modules integrated in Consul categorize proposals, suggest relevant

[7] https://democracy-technologies.org/database/

proposals for users so that they can support them, group citizens by their interests to encourage the exchange among citizens and summarize the comments that were added to one proposal. Davies et al. (2021) use the developed NLP methods in Consul to facilitate participatory budgeting processes in Scotland. In the future, the platform will be expanded to include an open-source LLM-based assistant.

Coral is a tool for large communities, primarily built for journalism. The tool is supported by many features that simplify effective moderation, such as highlighting and filtering comments and give feedback to participants. The developers at Coral rely on algorithmic support for moderation, but firmly believe that human moderators should always make the final decision.

Debate Hub (Quinto et al., 2021) is a tool to share ideas and discuss pro and contra arguments. It has been developed by the Open University's Knowledge Management Institute. The tool, i.a., includes a grouping mechanism to build discussion groups, a moderator toolbar and a visualization dashboard to better structure and understand the ongoing debate.

Decidim is a large-scale digital platform for citizen participation which is continuously being developed by a whole community. The project was initiated by the Decidim Free Software Association. The current version includes services for spam detection to support moderators on the platform and the possibility to incorporate machine translations.

Deliberatorium (Klein, 2011) is a discussion platform, based on large-scale argumentation and argument mapping. Participants can either post an issue, an idea or a pro or contra argument for an idea or another argument. The discussion is structured in a tree-like network. Moderators verify each post and check if they follow the guidelines to ensure that every gathered idea is unique and people can support each others ideas. Currently, LLM support for the platform, including argument mining, answer clustering and comment rephrasing suggestions, is being evaluated, and an extension of the Deliberatorium is being developed (Klein, 2025; Babatunde et al., 2025).

DIPAS (Lieven, 2017) is a digital citizen participation tool, developed by Hamburg's Department of Urban Development and Housing in Hamburg, the State Office for Geoinformation and Surveying and the HafenCity University's CityScienceLab. The tool has two components, an online module and an on-site module. The online component is a participation platform where citizens can submit proposals, questions and criticism, which can be linked to locations in the city via geodata. The on-site component is used for face-to-face meetings where data is gathered on digital touch tables. DIPAS Analytics is an NLP toolbox that supports the evaluation and analysis of text contributions in

DIPAS. The toolbox includes methods to extract key points and categorize them and links text and geodata.

Discourse is a community platform including features for civilized online discussions. The platform has several AI modules build in, e.g., automatic topic recommendation based on semantic similarity, an assistant to write a new topic that proofreads the users' text, suggests titles and translates it to English, a toxicity detection module and a summarization tool that summarizes discussed topics. More features supported through AI are currently planned.

Go Vocal (former CitizenLab) is a citizen participation platform, developed by an eponymous company, founded in Brussels. The software is used worldwide for citizen participation processes and polls. The platform includes an algorithm that is able to automatically analyze proposals of citizens to extract keywords and tag them. Additionally, NLP methods are used to identify posts with similar topics and summarize them. This helps participants to find topics of interest and decrease information overload on the platform.

Policy Synth (Bjarnason et al., 2024) integrates the All Our Ideas and Your Priorities platforms and LLMs, such as GPT-4 (Achiam et al., 2023) for so called Smarter Crowdsourcing. The tool is able to automatically formulate, rank and prioritize both socially relevant issues and potential solutions. It helps to combine recommendations made by experts and solutions proposed by LLMs based on web searches to enhance decision-making and idea gathering processes. A case-study by Gambrell (2025) demonstrates how these tools could be used effectively to support policy-makers come up with recommendations for socially relevant issues.

Polis (Small et al., 2021) is a system for idea gathering and analysis supported by ML, developed by the Computational Democracy Project.[8] Participants can add comments to specific topics, which are randomly suggested to other users who can vote on them. This creates a sparse comment matrix. After the conversation, the data matrix is ready to be exported and analyzed. Polis solely relies on language independent methods calculated on the data matrix like PCA Pearson, 1901, UMAP (McInnes et al., 2018) and different clustering methods. Recently, the potential and risks of integrating LLMs into Polis have been examined by Small et al. (2023). They use Anthropic's Claude (Askell et al., 2021) for tasks such as topic modeling, summarization, and generating textual cues to guide conversations for human facilitators. They conclude that LLMs have the potential to improve Polis discussions effectively. However, they ultimately emphasize the importance of human involvement in human-AI collaboration.

[8] `https://compdemocracy.org/`

Potluck (Lieu et al., 2022), a commenting system for large scale online discussions, aims to allow for more constructive and inclusive online discussions. It is named after a form of dinner where everyone brings and shares their food in a gathering. The systems includes a module for automatic summarization using a pre-trained DistillBART model.[9] For aggregation of similar posts they use SentenceBERT (Reimers and Gurevych, 2019) and the Perspective API (Jigsaw, 2025) to screen inputs for toxicity. Until now, there is only a prototype of the tool available.

RHETORiC (an acronym for Reducing Hate through Editorial Tools for Online Reactions and Comments) by Bossens et al. (2022) is a discussion tool that supports civil participation in news comment sections. The main features of RHETORiC are: real-time feedback on the quality of the written comments, proposal of similar arguments, the possibility to like other comments and a clustered view of all comments based on their similarity. Comments are also automatically classified and sent to pre-moderation, if they appear to be uncivil. Experiments showed that the made design choices had an impact on the comment quality and the overall engagement (Bossens et al., 2022). However, participants did not always agree with the decisions of the ML algorithm. Thus the authors plead to further develop and utilize explainable ML algorithms that are more transparent in their decisions.

Your Priorities is an online platform for idea generation, deliberation and decision making, developed by the Citizens Foundation.[10] The platform includes a detector for inappropriate content, a module for automatic translations and a recommendation algorithm that suggests posts that most likely are interesting to a user. It also implements a fraud detection system and an analysis tool to cluster ideas.

There is another branch of research that develops tools especially for discussion analysis and annotation, see e.g. the work by Fischer et al. (2024), which we did not include in our work.

4.6.2 Visualization Software

Another approach to gather insights on political debates is to visualize them, in order to navigate through conversations and explore different aspects visually. In the following we describe tools that analyze conversation through visualization.

DebateVis (South et al., 2020) is a discussion visualization tool that helps non-expert users to navigate and analyze transcripts of political debates. It can, e.g., be used in preparation for elections to inform voting decisions. Given the transcript for a debate,

[9] https://huggingface.co/sshleifer/distilbart-cnn-12-6
[10] https://citizens.is/

DebateVis provides the user with an interaction graph, marking the mentions of other candidates and topics in the discussion.

LiteMap is a tool that collects, visualizes, analyzes, and summarizes the content of online discussions across different platforms (De Liddo and Strube, 2021). LiteMap helps to gather ideas and pro and contra arguments from public debates. The tool incorporates a booklet function to save web content and a website to create and share argument maps.

Opinion Space (Faridani et al., 2010) is another tool that enables users to collect and visualize opinions online. Comments are depicted as dots on an interactive map, with nearby dots representing similar opinions and distant dots representing dissimilar opinions. The Opinion Space interface should counteract the problems of typical list-based discussion forums, in which users tend to lose track of the discussion.

ThreadReconstructor, presented by El-Assady et al. (2018), is a tool to reconstruct threads and untangle conversations in large online discussions. The tool is specialized on transcripts of discussions in online forums, where no threaded reply structure is given.

4.6.3 Evaluation Software

Besides ML supported platforms, there are also tools to evaluate and analyze citizen participation processes which are presented in the following. These tools are particularly useful for initiators of participation processes.

Civic CrowdAnalytics is a web application, developed by Aitamurto et al. (2016) to analyze data gathered from crowdsourcing processes in the context of policy making. The tool includes topic categorization via concept extraction, sentiment analysis (Cernuzzi et al., 2020) (indicating positive, neutral or negative sentiment) and shows which words occur often in the analyzed text.

CommunityPulse is an ongoing research project to explore comments from community participation processes and visualize them (Jasim et al., 2021). The conduction of expert interviews revealed that for organizers of civic participation it is most important to keep an overview of the gathered data, both on a high level as well as on the individual comment level. Furthermore it is important to grasp the overall sentiment and to understand the main topics that were discussed. Based on these findings, CommunityPulse offers three different views. One for visualizing the discussion, the topics and the sentiment of the participants on a high level. A view to analyze comments in more detail and a view to directly compare contributions.

Machine Learning Method	Reference
Adapter (Pfeiffer et al., 2021)	Falk and Lapesa; Behrendt et al.
BART (Lewis et al., 2020)	Arana-Catania et al.
BERT (Devlin et al., 2019)	Arana-Catania et al.; Anastasiou; Jasim et al.; Falk et al.; Bonechi; Katsaros et al.; Bonechi; Falk and Lapesa; Fournier-Tombs and MacKenzie
BiLSTM (Stab and Gurevych, 2017)	Ito et al.; Guo et al.
BoW	Sun et al.; Fournier-Tombs and MacKenzie
BTM (Yan et al., 2013)	El-Assady et al.
Claude (Askell et al., 2021)	Small et al.
CNN-BERT (Harly and Girsang, 2022)	Harly and Girsang
Decision Trees	Aitamurto et al.
DeepSBD (Fazil et al., 2021)	Fazil et al.
DistillBART	Lieu et al.
DistillBERT (Sanh et al., 2019)	Atanasova et al.
ECGA (Giannakopoulos et al., 2019)	Romberg and Conrad
fastText (Joulin et al., 2017)	Romberg and Conrad
GCR-NN (Lee et al., 2022)	Lee et al.
GloVe Embeddings (Pennington et al., 2014)	Arana-Catania et al.; Prabowo and Budi Herwanto
GPT-3 (Brown et al., 2020)	Van der Meer et al.; Argyle et al.; Kolla et al.; Rescala et al.
GPT-4 (Achiam et al., 2023)	Bjarnason et al.; Babatunde et al.; Rescala et al.
Graph Attention Networks (Veličković et al., 2018)	Suzuki et al.
IHTM (Assady, 2015)	El-Assady et al.
K-Means	Aitamurto et al.
K-nearest-neighbor classification	Liebeck et al.
LDA	Arana-Catania et al.; Jasim et al.; El-Assady et al.; South et al.
Llama 2 (Touvron et al., 2023)	Rescala et al.
Mistral 7B (Jiang et al., 2023)	Rescala et al.
Naive Bayes	Aitamurto et al.
No Language Left Behind (Team et al., 2022)	Bonechi
Non-Negative Matrix Factorization	Arana-Catania et al.
PCA (Pearson, 1901)	Small et al.; Faridani et al.
Perspective API (Jigsaw, 2025)	Horta Ribeiro et al.
PMI-IR (Turney, 2002)	Ito et al.
Random Forest Classifier	Fournier-Tombs and Di Marzo Serugendo; Aitamurto et al.; Falk et al.; Liebeck et al.
RoBERTa (Liu et al., 2019)	Falk et al.; Van der Meer et al.
Sentence-BERT (Reimers and Gurevych, 2019)	Lieu et al.
SWB (Chemudugunta et al., 2006)	El-Assady et al.
SVM	Aitamurto et al.; Liebeck et al.; Romberg and Conrad
TF-IDF	Arana-Catania et al.; Aitamurto et al.
TextRank (Mihalcea and Tarau, 2004)	Arana-Catania et al.
Transformer (Vaswani et al., 2017)	Chang et al.; Sun et al.; Khan et al.; Schütz et al.
UMAP (McInnes et al., 2018)	Small et al.
VADER (Gilbert, 2014)	Aitamurto et al.

Table 3: List of machine learning methods applied in online discussions and participation processes.

ConvoKit (Chang et al., 2020), is a toolkit that combines several methods for text analysis. With the help of ConvoKit, linguistic features and social phenomena can be extracted from conversations, such as politeness, or rather impoliteness of the used language, linguistic diversity and redirection of conversation flow. A model that is trained to forecast the outcomes of conversations is also included.

Table 2 lists all mentioned tools and a link to their website or GitHub Repository, if available.

4.6.4 Used Machine Learning Methods

We gave an extensive overview of ML and NLP methods in existing tools for online debating and participation. Table 3 lists all ML methods and models that have been used throughout literature. While some works have utilized, mostly Transformer based (Vaswani et al., 2017) deep learning models such as BERT (Devlin et al., 2019) or RoBERTa (Liu et al., 2019), most of the listed methods are classical ML methods such as support vector machines (SVMs) and K-Means.

Dataset	Task
args.me (Ajjour et al., 2019)	Argument Search
ArguAna Counterargs (Wachsmuth et al., 2018)	Counterargument Retrieval
Change my View (Tan et al., 2016)	Persuasive Arguments
CivicParse (Gupta and Klein, 2025)	Structure Extraction
CoFE (Barriere et al., 2022b)	Multilingual Multi-target Stance Classification
Conversations Gone Awry (Chang and Danescu-Niculescu-Mizil, 2019)	Conversation Derailment Forecasting
Cornell eRule-making Corpus (Park et al., 2012)	Argument Mining
DEBAGREEMENT (Pougué-Biyong et al., 2021)	(Dis)agreement Detection
Debating Europe (Barriere et al., 2022a)	Multilingual Multi-target Stance Classification
Deliberation Bank (Zhu et al., 2025)	Opinion Summarization
DeliData (Karadzhov et al., 2021)	Multi-Party Problem Solving
Europolis Dataset (Gerber et al., 2018)	Deliberative Quality
Internet Argument Corpus (Walker et al., 2012a)	Several Dialogue and Argument tasks
Factify (Mishra et al., 2022)	Fact-Verification
Factify 2 (Suryavardan et al., 2022)	Fake News & Satire News
Jigsaw Toxic Comment Dataset (Kivlichan et al., 2020)	Multilingual Toxic Comment Classification
PerspectiveMod (Vecchi et al., 2025)	Moderator Intervention Prediction
StoryARG (Falk and Lapesa, 2023b)	Personal Stories in Arguments
Twitter Spam Detection Dataset (Bhidya, 2019)	Social Spam Detection
UMOD Dataset (Falk et al., 2024)	User Moderation
Webis-TLDR-17 Corpus (Völske et al., 2017)	Text Summarization
X-Stance (Vamvas and Sennrich, 2020)	Stance Detection

Table 4: List of annotated political online discussion and participation process datasets.

4.7 Datasets

High-quality datasets annotated by experts form the foundation of ML models that solve the tasks discussed in the given work. The lack of high-quality datasets remains a key challenge for training and utilizing models to facilitate processes in the public sector. However, we found datasets during our literature review that can be used for this purpose. The list of datasets and their respective tasks is provided in Table 4. The results are limited to English-language datasets, or multilingual datasets that include English. Please note that this list is not exhaustive, and there are a lot of data sets for each task area. Here, we have limited ourselves to data sets in which conversations from online discussions or participation processes have been annotated.

4.8 Key Challenges & Open Problems

So far we mainly discussed opportunities for applications of ML in online discussions and participation processes. Still there are some key challenges and risks that need to be addressed.

Lacking Accuracy. The number of tasks that ML systems can solve in order to support municipalities and policy makers is growing. However, methods still need improvement to generate meaningful insights, e.g., in participation process data for evaluation purposes (Aitamurto et al., 2016). For this purpose, annotated datasets of high quality are necessary. The latest research shows that LLMs trained on large amounts of data can perform these tasks very well in some cases (Small et al., 2023). However, using LLMs also brings other risks, which we will discuss below.

Induced Biases. Automated processes that are used in social contexts always pose risks. This is especially true when tracing the transparency, explainability, plausibility, reliability or legitimacy of the applications is not possible. Strauß (2021) points out that it is necessary to keep potential biases in mind when developing methods and also when using them in practice. Precisely in the field of politics and online discussions where data to train models is taken from real conversations, it is likely that the data and subsequently also the models that learn from the data contain political biases, leading to unethical, unfair or even discriminating decisions (Mehrabi et al., 2021).

Manipulation. The use of ML methods in environments with social impact such as political participation always has to be well considered. It has been shown that algorithms can lead to increased polarization, selective exposure (Dylko et al., 2017) and influenced views. ML can help us make sense of big amounts of data and can be a supportive tool, but it should also be used in a transparent and ethical way.

Transparency. We have seen a variety of tasks that can be solved or at least supported through ML methods. Nevertheless, if it is not clear for the participants how methods are used, what they do in the process and how they make decisions, it can have an impact on the process and lead to trust issues (Lago et al., 2019; Gambrell, 2025). Explainable ML is an own branch of research that is studied by many researchers, yet it is not common to use ML methods that are transparent to those who use it.

Lack of Data. In order to train models that support participation processes, up-to-date data on current issues and of high quality is necessary. Communes would profit from sharing their newest data, but data protection policies do not always allow free sharing of discussion data.

Hallucination. With the emergence of LLMs and their potential application in online discussions and participation processes, there is a high risk of models to hallucinate facts (Huang et al., 2025), or invent details when summarizing user opinions (Small et al., 2023). Incorrect facts and errors in summaries are very difficult to identify and can potentially distort a discussion.

Evaluation. There are many approaches from the field of ML to solve potential issues in online discussions and online participation processes. However, not many have been used and examined in practice, yet. It is still an open question whether the application of NLP methods can demonstrably contribute to improve online discussions and if they are perceived as helpful by participants. Controlled experiments could help to systematically evaluate the use of ML in online discussions and provide a better understanding on how these tools could be used in a supporting way (Helbing et al., 2023).

Many of these risks can be mitigated by keeping humans in the loop and leaving the opportunity to override and correct decisions made by ML models.

4.9 Conclusion

Civic participation is an important instrument to democracy to collectively solve problems of public concern, both on a local as well as on a global level. Today, more and more communities and municipalities make use of digital tools to include citizens in decision making. While the digital space holds many opportunities, as it is neither location nor time-bound and enables humans from all over the world to contribute to important decisions, it also implies issues. It is important to never lose track of potential risks. Unfortunately, the use of algorithms is always twofold, as it holds potential for manipulation and abuse.

With our work we hope to showcase the importance of deliberative elements in online discussions, especially in political contexts. If more platforms implement designs that

enhance respectful, inclusive conversations online, citizen councils and other democratic instruments hold great potential to come to thoughtful decisions and collectively fight crises that affect all people. A platform where design choices are gathered and scientifically evaluated is the Prosocial Design Network[11], which is continuously developed further.

ML has the potential to noticeably facilitate collective decision making, but it should always be seen as a supporting tool and not replace the work of humans. Nevertheless, it will be part of our lives as it is already today. Researchers should further investigate the use of ML, but also study side effects and risks that accompany it.

The survey revealed that while numerous NLP tasks have been proposed, there is still a need for efficient, high-performing models that can be deployed in real-time participation contexts. This motivated the technical investigations presented in the following section.

[11] https://www.prosocialdesign.org/

5 ArgueBERT: How to Improve BERT Embeddings for Measuring the Similarity of Arguments

Corresponding Publication. This chapter is based on the work:

ArgueBERT: How to Improve BERT Embeddings for Measuring the Similarity of Arguments
Maike Behrendt & Stefan Harmeling
Proceedings of the 17th Conference on Natural Language Processing (KONVENS 2021)

Personal Contributions. This publication is based on the Master's thesis of Maike Behrendt, which was supervised by Stefan Harmeling and Stefan Conrad. The core idea of modifying BERT's pre-training process to improve performance on argument similarity tasks was developed by Maike Behrendt and further refined in collaboration with Stefan Harmeling. Maike Behrendt was responsible for conducting all experiments, including implementing all model variants, evaluating them, and documenting the results. After completing the thesis, she co-authored the conference paper with Stefan Harmeling. The work was also presented as a poster at the 15th Women in Machine Learning Workshop at NeurIPS 2020. The source code and the pre-trained models are publicly available at `https://github.com/mabehrendt/argueBERT`.

Abstract. Argumentation is an important tool within human interaction, not only in law and politics but also for discussing issues, expressing and exchanging opinions and coming to decisions in our everyday life. Applications for argumentation often require the measurement of the arguments' similarity, to solve tasks like clustering, paraphrase identification or summarization. In our work, BERT embeddings are pre-trained on novel training objectives and afterwards fine-tuned in a siamese architecture, similar to Reimers and Gurevych (2019), to measure the similarity of arguments. The experiments conducted in our work show that a change in BERT's pre-training process can improve the performance on measuring argument similarity.

Research Problem & Motivation. The BERT model (Devlin et al., 2019) is widely used for sentence classification tasks. Its pre-training involves two objectives. The first is Masked Language Modeling (MLM), in which random tokens are replaced with either a special `[MASK]` token or a random token from the vocabulary, and the model must predict the correct token. The second objective is Next Sentence Prediction (NSP), a binary

classification task in which the model must predict whether two input sentences are consecutive in the original text, or if the second sentence is randomly sampled. However, multiple studies have shown that the NSP objective is of limited effectiveness (Yang et al., 2019; Joshi et al., 2020; Liu et al., 2019). In this work, we aimed to improve BERT's performance, especially for argument similarity detection. Thus, we implemented and evaluated three alternative pre-training objectives, resulting in model variants including ArgueBERT.

Main Results. The experimental evaluation demonstrates that replacing BERT's original NSP objective with alternative pre-training tasks tailored to argumentative structures improves performance on our proposed SAM task. In particular, the proposed ArgueBERT variant yields more semantically meaningful embeddings for argumentative text and achieves measurable performance gains compared to the standard BERT baseline. These findings suggest that adapting pre-training objectives to task-specific discourse phenomena can enhance representation quality without increasing model size.

Publication Status. Published. This work was accepted at the 17th Conference on Natural Language Processing (KONVENS 2021) (Behrendt and Harmeling, 2021).

5.1 Preamble

Currently, LLMs play an essential role in research on AI-supported online discussions. They are, e.g., employed as automated moderators (Kolla et al., 2024) and to generate suggestions for more polite or constructive user comments (Argyle et al., 2023).

Despite these advances, efficient models that deliver strong classification results remain highly relevant, particularly in the context of online political discussions and citizen participation processes. The deployment of large-scale models is often constrained by substantial computational requirements and environmental costs (Schwartz et al., 2020; Bender et al., 2021). Moreover, participation platforms frequently operate in public-sector settings where data protection, transparency, and controllability are essential. Regulatory frameworks such as the General Data Protection Regulation (GDPR) emphasize the need for data minimization, local processing, and explainability of AI systems. In such contexts, reliance on externally hosted, computationally intensive proprietary models may be problematic.

It is therefore important to investigate and optimize small models for classification that can be deployed locally while maintaining strong performance on relevant tasks. As Mikhaylovskaya (2024) argues, AI interventions designed to support deliberative processes should align with democratic values such as equality and inclusiveness. Achieving these goals requires models that are not only accurate, but also efficient, interpretable, and adaptable to specific participation contexts.

Recent work on efficient Transformer variants demonstrates that compact models can achieve competitive performance while significantly reducing resource requirements (Sanh et al., 2019). Optimizing smaller language models therefore constitutes both a technically and ethically important research direction for deliberation-related tasks.

In this chapter, we focus on improving the quality of language representations. Specifically, we investigate how modifications to the pre-training objectives of BERT can enhance argument similarity detection.

The contributions presented in this chapter address Research Question 2 by exploring how pre-training can be adapted to better capture argument-specific structures in text. In the subsequent chapter we shift the focus to the optimization of BERT's fine-tuning process.

5.2 Introduction

Since today it is common to share opinions on social media to discuss and argue about all kinds of topics, the interest of research in the field of artificial intelligence in argumentation is constantly rising. Tasks like counter-argument retrieval (Wachsmuth et al., 2018), argument clustering (Reimers et al., 2019) and identifying the most prominent arguments in online debates (Boltužić and Šnajder, 2015) have been examined and automated in the past. Many of these tasks involve measuring the textual similarity of arguments. Transformer-based language models such as the Bidirectional Encoder Representations from Transformers (BERT) by Devlin et al. (2019) are widely used for different Natural Language Processing (NLP) tasks. Nevertheless, for large-scale tasks like finding the most similar sentence in a collection of sentences, BERT's cross-encoding approach is disadvantageous as it creates a huge computational overhead. In our work, we focus on exactly these large-scale tasks. We want to train embeddings of arguments in order to measure their similarity, e.g., to automatically recognize similar user entries in ongoing discussions in online argumentation systems. In this way redundancy can be avoided when collecting arguments. We base our approach on Sentence BERT (SBERT), proposed by Reimers and Gurevych (2019), which is a bi-encoder, fine-tuning the model's parameters to place similar sentences close to one another in the vector space. This approach yields good results on paraphrase identification tasks, but evaluating it on an argument similarity corpus shows a noticeable drop in performance. To improve this method, we propose and evaluate three alternative pre-training tasks that replace the NSP in BERT's pre-training process to optimize SBERT for measuring the similarity of arguments. These proposed tasks are similarity prediction, argument order prediction and argument graph edge validation. Being pre-trained on these tasks and fine-tuned in a siamese SBERT architecture, we call these models argueBERT throughout this work. To examine the models' applicability in practice, we also propose a new evaluation task, which is called SAM. Solving the task of SAM includes recognizing paraphrases (if any are present) in a large set of arguments, e.g., when a user enters a new argument to an ongoing discussion

in some form of argumentation system. In summary our contributions of this paper are the following:

1. We propose and evaluate new pre-training objectives for pre-training argument embeddings for measuring their similarity.
2. We propose a novel evaluation task for argumentation systems called SAM.

5.3 Related Work

Alternative Pre-Training Objectives. The original BERT model uses two different pre-training objectives to train text embeddings that can be used for different NLP tasks. Firstly Masked Language Modeling (MLM) and secondly Next Sentence Prediction (NSP). However, Liu et al. (2019) have shown that BERT's next sentence prediction is not as effective as expected and that solely training on the MLM task can slightly improve the results on downstream tasks. Since then there have been attempts to improve the pre-training of BERT by replacing the training objectives. Lewis et al. (2020) propose, i.a., token deletion, text infilling and sentence permutation as alternative pre-training tasks. Their experiments show that the performance of the different pre-training objectives highly depends on the NLP task it is applied to. Inspired by this we want to explore tasks that perform well on measuring the semantic similarity of arguments. Lan et al. (2020) propose a sentence ordering task instead of the next sentence prediction, which is similar to our argument order prediction. They find that sentence ordering is a more challenging task than predicting if a sentence follows another sentence. Instead of continuous text, we use dialog data from argumentation datasets, as we hope to encode structural features of arguments into our pre-trained embeddings. Clark et al. (2020) use replaced token detection instead of MLM, where they do not mask tokens within the sentence, but replace some with alternative tokens that also fit into the sentence. In this way they implement a contrastive learning approach into BERT's pre-training, by training the model to differentiate between real sentences and negative samples. Their approach outperforms a model pre-trained on MLM on all tasks.

Argument Embeddings. Embeddings of textual input that encode semantic and syntactical features are crucial for NLP tasks. Some research has already been conducted using the BERT model or its embeddings to measure the similarity of arguments. These are described briefly in the following. Reimers et al. (2019) use, inter alia, BERT for argument classification and clustering as part of an open-domain argument search. This task involves firstly classification of arguments concerning their topic, and afterwards clustering the arguments in terms of their similarity. They achieve the best results with a fine-tuned BERT model, when incorporating topic knowledge into the network. In a proximate work Reimers and Gurevych (2019) introduce SBERT which serves a base for our work. They train a BERT model in a siamese architecture to produce embeddings of textual input for tasks like semantic similarity prediction. The model is described in

detail in Section 3.1. Dumani et al. (2020) build upon the work of Reimers et al. (2019) and propose a framework for the retrieval and ranking of arguments, which are both sub-tasks of an argument search engine.

Thakur et al. (2021) present an optimized version of SBERT and publish a new argument similarity corpus, which we also use for evaluation in this work. They expand the training data for the SBERT model through data augmentation, using the original BERT model for labeling sentence pairs. To the best of our knowledge there are currently no contextualized embeddings developed especially for the task of measuring the similarity of arguments.

5.4 Background

In this section the SBERT (Reimers and Gurevych, 2019) architecture, the training procedure and characteristics are explained in detail.

5.4.1 SBERT

We use SBERT, proposed by (Reimers and Gurevych, 2019) to fine-tune the BERT models pre-trained with our novel proposed pre-training tasks.

SBERT is a network architecture that fine-tunes BERT in a siamese or triplet architecture to create embeddings of the input sentences to measure their similarity. Unlike the original BERT model, SBERT is a bi-encoder, which means it processes each input sentence individually, instead of concatenating them. The advantage of bi-encoders is their efficiency. Cross-encoders like BERT generate an enormous computational overhead for tasks such as finding the most similar sentence in a large set of sentences, or clustering these sentences.

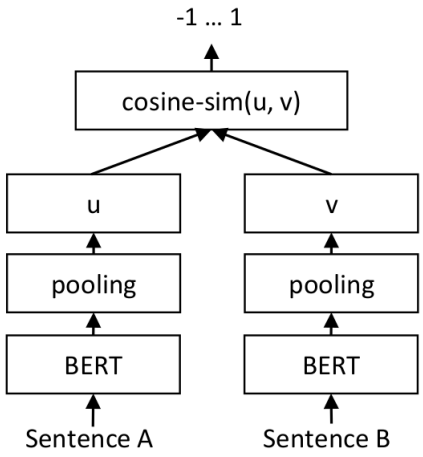


Figure 8: SBERT architecture for measuring sentence similarity.

By connecting both input sequences, handling it as one input, BERT is able to calculate cross-sentence attention. Although this approach performs well on many tasks, it is not always applicable in practice. SBERT is much faster and produces results that outperform other state-of-the-art embedding methods (Reimers and Gurevych, 2019).

To fine-tune the model the authors propose different network structures. For regression tasks, e.g., measuring sentence similarity, they calculate the cosine similarity of two embeddings u and v, as shown in Figure 8, and use a Mean Squared Error (MSE) loss as

objective function. To calculate the fixed sized sentence embeddings from each input, a pooling operation is applied to the output of the BERT model. The authors experiment with three different pooling strategies, finding that taking the *mean* of all output vectors works best for their model.

In the siamese architecture the weights of the models are tied, meaning that they receive the same updates. In this way the BERT model is fine-tuned to create sentence embeddings that map similar sentences nearby in the vector space.

In the original paper, the model is fine-tuned on the SNLI (Bowman et al., 2015) and the Multi-Genre NLI datasets (Williams et al., 2018) to solve multiple semantic textual similarity tasks, which leads to improved performance in comparison to other state-of-the-art embedding methods. However, evaluating the model on the AFS (Misra et al., 2016) dataset shows a significant drop in accuracy. Different than in our work, the authors do not pursue the measurement of argument similarity in the first place, but rather use the model for general textual similarity tasks. The aim of this work is therefore to optimize BERT's pre-training process to generate argument embeddings that lead to better results on this task.

5.5 argueBERT

5.5.1 Pre-Training

We propose and evaluate three new tasks, which should improve the performance of BERT embeddings on measuring the similarity of arguments. The proposed pre-training objectives that are optimized instead of the next sentence prediction are the following:

1. **Similarity prediction:** Given a pair of input sentences s_1 and s_2, predict whether the two sentences have the same semantic meaning. BERT therefore is pre-trained on the Paraphrase Adversaries from Word Scrambling (PAWS) (Zhang et al., 2019) and the Quora Question Pairs (QQP)[12] dataset.

2. **Argument order prediction:** Given an argumentative dialog consisting of a statement and an answer to that statement, predict if the given paragraphs p_1 and p_2 are in the correct order. For this task we train BERT on the Internet Argument Corpus (IAC) 2.0 (Abbott et al., 2016), which contains argumentative dialogues from different online forums. This task is the same as the sentence ordering objective from ALBERT (Lan et al., 2020) but with argument data.

3. **Argument graph edge validation:** Given two arguments a_1 and a_2 from an argument graph, classify if they are adjacent, thus connected through an edge in the graph. For this task we use several argument graph corpora, taken from `http://corpora.aifdb.org/` for pre-training.

[12] `https://www.kaggle.com/c/quora-question-pairs/data`

The pre-training process of argueBERT is the same as for the original BERT, except that we replace the next sentence prediction task. Our novel proposed pre-training objectives are trained as binary classification tasks.

To compare the new pre-training tasks, we train medium sized BERT models with 8 layers and a hidden embedding size of 512 (Turc et al., 2019). We train the models for a total of 100, 000 training steps. To guarantee comparability we also train a model with the original NSP and MLM objectives for $100,000$ steps on the BookCorpus (Zhu et al., 2015). To examine if the pre-training tasks also perform on a larger scale, we additionally train a $\text{BERT}_{\text{BASE}}$ model (12 layers, hidden embedding size 768) on our best performing pre-training task for 1, 000, 000 steps. All hyperparameters we used for pre-training can be found in Table 9 in the Appendix.

5.5.2 Fine-Tuning

For fine-tuning argueBERT, we use SBERT (Reimers and Gurevych, 2019). The model fine-tunes the weights of the pre-trained BERT model in a siamese architecture, such that the distance between embeddings of similar input sentences is minimized in the corresponding vector space. Therefore, $\hat{y}$ is calculated as the cosine similarity between two input embeddings u and v and then the MSE loss

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{26}$$

is optimized. Here n is the batch size and y the true label. We fine-tune each model on every evaluation dataset for a total of five epochs with a batch size of 16 and a learning rate of $2e-5$. All hyperparameters used for fine-tuning can be found in Table 10 in the Appendix.

5.5.3 Similar Argument Mining (SAM)

The main idea of proposing argueBERT as an improved version of SBERT on measuring argument similarity is in particular to use it for identifying and mining similar arguments in online argumentation systems. In order to evaluate language models for this purpose, we propose a new evaluation task which we call SAM. It is defined as follows.

Task definition. Given a query argument q, match the argument against all arguments of an existing set $S = \{a_1, a_2, \ldots, a_n\} \backslash \{q\}$ to predict, if S contains one or more paraphrased versions of q and find the paraphrased sentences in the set.

For the evaluation on SAM, the model is given a set of arguments of which some are paraphrased argument pairs and some are unpaired arguments that are not considered equivalent to any other argument in the set. The model then encodes all arguments into

vector representations and calculates the pairwise cosine similarities. If the highest measured similarity score for an argument exceeds a pre-defined threshold, the argument is classified as being a paraphrase. We calculate the accuracy and the F1 score of the models on this task.

5.6 Experiments

5.6.1 Datasets

We use the following datasets for the evaluation of our embeddings.

- The Microsoft Research Paraphrase Corpus (MRPC)[13] (Dolan and Brockett, 2005), which includes $5,801$ sentence pairs for paraphrase identification with binary labeling (0: *"no paraphrase"*, 1: *"paraphrase"*), automatically extracted from online news clusters.
- The Argument Facet Similarity (AFS) Dataset[14] (Misra et al., 2016), consisting of $6,000$ argument pairs taken from the Internet Argument Corpus on three controversial topics (*death penalty*, *gay marriage* and *gun control*, annotated with an argument facet similarity score from 0 (*"different topic"*) to 5 (*"completely equivalent"*).
- The BWS Argument Similarity Dataset[15] (BWS) (Thakur et al., 2021), which contains $3,400$ annotated argument pairs on 8 controversial topics from a dataset collected from different web sources by Stab et al. (2018b). Labeled via crowd-sourcing with similarity scores between 0 and 1.
- The UKP Argument Aspect Similarity Corpus[16] (UKP) (Reimers et al., 2019) with a total of $3,595$ argument pairs, annotated with four different labels *"Different topic/ can't de- cide"*, *"no similarity"*, *"some similarity"* and *"high similarity"* on a total of 28 topics, which have been identified as arguments by the ArgumenText system (Stab et al., 2018a).

As baselines we use (i) a medium sized SBERT, pre-trained with the standard BERT pre-training procedure, fine-tuned in a siamese architecture, and (ii) average word2vec[17] (Mikolov et al., 2013) vectors with vector-size 300, pre-trained on part of the Google News dataset.

To be able to fine-tune the models, the discrete labels of the AFS and UKP corpus are transformed into similarity scores between 0 and 1. The labels of the AFS corpus, which

[13] https://www.microsoft.com/en-us/download/details.aspx?id=52398
[14] https://nlds.soe.ucsc.edu/node/44
[15] https://tudatalib.ulb.tu-darmstadt.de/handle/tudatalib/2496
[16] https://tudatalib.ulb.tu-darmstadt.de/handle/tudatalib/1998
[17] https://code.google.com/archive/p/word2vec/

range from 0 to 5, are normalized by dividing it through the maximum value of 5. For the UKP corpus, the labels *"different topic/ can't decide"* and *"no similarity"* are assigned the value 0, *"some similarity"* is translated into a similarity score of 0.5 and for all pairs with label *"high similarity"* we assign a similarity score of 1. The labels for the MRPC corpus remain unchanged.

We perform two different evaluations. Firstly on the task of similarity prediction. Therefore we evaluate the models by calculating the Pearson's and Spearman's rank correlation for the predicted cosine similarities. Secondly we calculate the accuracy and F1 score on the novel proposed task of SAM.

For the AFS corpus, which contains arguments for three different controversial topics, we use the same cross-topic evaulation strategy as suggested by Reimers and Gurevych (2019). The models are fine-tuned on two of the three topics and evaluated on the third one, taking the average of all possible cross-topic scenarios as overall model performance score.

The UKP corpus, including arguments on 28 different topics, is evaluated with a 4-fold cross-topic validation as done by Reimers et al. (2019). Out of the 28 topics, 21 are chosen for fine-tuning the model and 7 are used as test set. The evaluation result is the averaged result from all folds.

The BWS argument similarity dataset incorporates 8 different controversial topics. For evaluation we fine-tune the models on a fixed subset ($T_1 - T_5$), validate them on another unseen topic (T_6) and use the remaining two topics as test set ($T_7 - T_8$), as suggested by Thakur et al. (2021).

	MRPC		**UKP**		**AFS**	
Model	r	ρ	r	ρ	r	ρ
average word2vec	18.17	17.96	22.29	17.44	11.25	5.22
SBERT	47.12	44.54	32.04	30.89	38.02	35.92
argueBERT sim. pred. (ours)	**48.33**	**46.34**	**35.33**	**34.77**	37.57	35.83
argueBERT order pred. (ours)	45.08	43.15	28.41	28.11	**38.25**	**36.80**
argueBERT edge val. (ours)	40.88	40.03	28.36	26.64	36.89	34.04

Table 5: Pearson's correlation r and Spearman's rank correlation $\rho \times 100$ on the MRPC, UKP and AFS corpora.

5.7 Results

First of all, we evaluate how well our models can predict the similarity of a given argument pair by calculating the cosine similarity between the two embeddings. Table 5 shows the Pearson correlation r and Spearman's rank correlation ρ on this task for the MRPC, AFS and UKP datasets.

On the MRPC dataset, the model pre-trained with a similarity prediction objective performs slightly better than the baseline that is trained with the next sentence prediction objective. The argueBERT order prediction model only performs a little worse on this

dataset, than the next sentence prediction model, while the model trained on edge validation can not compete with the aforementioned models.

On the UKP dataset the performance increase by the model that used the similarity prediction objective for pre-training is even more significant. It outperforms the traditionally pre-trained SBERT model by 3 points for Pearson correlation and almost 4 points for the Spearman rank correlation.

Surprisingly, the order prediction model is able to outperform the similarity prediction task on the AFS corpus. But it has to be noticed that there is not much difference in the performance of all models on this dataset. Only the averaged word2vec vectors perform notably worse than all other evaluated models.

Out of all evaluated datasets, the recently published BWS corpus is the only one whose similarity values are quantified on a continuous scale. Table 6 shows the evaluation results for all models for three different distance measures. We chose the cosine similarity as default distance measure for evaluation. But in the case of the BWS corpus it is striking that both Manhattan and Euclidean distance result in a higher Pearson correlation as well as Spearman rank correlation. The embeddings of argueBERT pre-trained with a similarity prediction objective achieve the highest correlation for all distance measures. The model outperforms the SBERT model by 4 points. The argument order pre- diction model also performs better than the model pre-trained with next sentence prediction, only the edge validation argueBERT model does not lead to an improvement and even performs worse than the word2vec baseline approach for both Manhattan and Euclidean distance measures.

To see how well the pre-training works for larger models, we also trained an argueBERT$_{\text{BASE}}$ model on the task of similarity prediction for 1, 000, 000 training steps on the PAWS and QQP datasets. The evaluation results for the BWS dataset are shown in Table 7. For comparison we also list the evaluation result of the standard BERT$_{\text{BASE}}$ model on this dataset. Even though argueBERT$_{\text{BASE}}$ was trained on a comparably small dataset, it outperforms SBERT on the BWS argument similarity prediction task and almost reaches the level of the BERT$_{\text{BASE}}$ cross-encoder.

Lastly, Table 8 shows the results on the MRPC dataset on the task of SAM for both the small and large pre-trained models. The small argueBERT model, pre-trained with the similarity prediction objective, by far achieves the highest accuracy, as well as the highest F1 value for a threshold of 0.8. This reflects the evaluation results of the sentence embeddings on this dataset, showing that the similarity prediction argueBERT model is able to recognize paraphrases in the dataset quite well. The second best performing models, which are the argueBERT model trained on the task of edge validation and the baseline, trained on next sentence prediction, are almost more than 16 points behind. This shows the great potential of incorporating similarity prediction in the pre-training process of BERT. Looking at the results for the larger models, the argueBERT$_{\text{BASE}}$ model does not perform as well as the SBERT model on this dataset.

The remaining argument similarity datasets were found to be unsuitable for the task of SAM as they do not only contain dedicated paraphrased argument pairs, but rather present all increments of similarity. This means that very similar arguments are not necessarily matched as argument pairs in the data. Therefore, for future research new datasets that suit the task of SAM are required.

Model	Cosine		Manhattan		Euclidean	
	r	ρ	r	ρ	r	ρ
average word2vec	8.98	3.46	41.67	43.61	41.73	43.54
SBERT	38.55	38.72	42.34	42.02	42.55	42.09
argueBERT sim. pred. (ours)	**43.84**	**43.76**	**46.22**	**46.44**	**45.65**	**46.07**
argueBERT order pred. (ours)	38.97	38.02	44.40	43.29	44.14	43.30
argueBERT edge val. (ours)	33.72	33.22	39.49	38.79	39.74	39.17

Table 6: Pearson's correlation r and Spearman's rank correlation $\rho \times 100$ on the **BWS** Argument Similarity Dataset (Thakur et al., 2021) for three different distance measures.

Model	ρ
average word2vec	43.54
$\text{SBERT}_{\text{BASE}}$ (Thakur et al., 2020)	58.04
argueBERT$_{\textbf{BASE}}$ sim. pred. (ours)	62.44
$\text{BERT}_{\text{BASE}}$ (Thakur et al., 2020)	**65.06**

Table 7: Spearman's rank correlation $\rho \times 100$ on the **BWS** argument similarity dataset.

Model	Acc.	F1
average word2vec	35.49	45.68
SBERT	49.32	52.54
argueBERT sim. pred. (ours)	**64.09**	**69.80**
argueBERT order pred. (ours)	38.14	40.18
argueBERT edge val. (ours)	48.10	49.08
$\text{SBERT}_{\text{BASE}}$	66.88	71.45
argueBERT$_{\textbf{BASE}}$ sim. pred. (ours)	65.92	70.76

Table 8: Accuracy and F1 score on SAM for the **MRPC** corpus for a threshold of 0.8.

5.8 Discussion

Our conducted experiments show that the new proposed pre-training tasks are able to improve the SBERT embeddings on argument similarity measurement, compared to the next sentence prediction objective. Nevertheless, our presented approach has some limitations that should be addressed in the following.

First of all, the proposed models were pre-trained and fine-tuned on a single GPU. Due to the limited resources, a BERT model in medium size was chosen as basis for all pre-trained models. The models were trained only for a total of 100, 000 training steps, which is just a small fraction of the conducted training of the original BERT model. The achieved results have to be regarded as comparative values on how much an adaptation of the pre-training process can improve the performance. However, training a larger model for 1, 000, 000 steps on the task of similarity prediction indicates that the adapted pre-training also works for large models and is able to compete with a pre-trained cross-encoder.

Another point is that the corpora we used for pre-training have quite different characteristics. The IAC (Abbott et al., 2016) for example consists of posts from different online forums. The used language is colloquial and the posts strongly vary in length and linguistic quality. The same applies to the QQP corpus. In contrast, the PAWS dataset consists of paraphrases extracted from Wikipedia articles, implying a formal language without misspellings. Training models on informal datasets can be advantageous, depending on the application of the trained model. In our case the differences of the used datasets rather constitute a disadvantage, as it may affect the comparability of the resulting models.

Additionally to having different characteristics, the few available datasets on paraphrase identification, argument similarity and also the argument graph corpora are relatively small, compared to the corpora the original BERT model is trained on. For the task of argument similarity prediction only the recently published BWS corpus (Thakur et al., 2021) includes argument pairs annotated with continuous scaled similarity scores. It can be said that there is still a lack of high-quality annotated argumentation corpora for this task.

5.9 Conclusion

In our work, we proposed and evaluated different pre-training tasks to improve the performance of SBERT embeddings on the task of argument similarity measurement. We call the new pre-trained model variants argueBERT. Evaluation of the models shows that adapting the pre-training process of BERT has an impact on the resulting embeddings and can improve the models' results. ArgueBERT trained with a similarity prediction objective led to a performance improvement up to 5 points Spearman's rank correlation on the evaluated BWS argument similarity corpus, compared to the model trained with the classic NSP pre-training task and also showed the best results on our new proposed evaluation task SAM on the MRPC corpus.

A larger argueBERT$_{\text{BASE}}$ pre-trained with the similarity prediction task could improve the evaluated embeddings compared to SBERT and almost reaches the results of the cross-encoding BERT$_{\text{BASE}}$ model.

For future research, the new proposed task of SAM can be used to evaluate models on the ability to identify paraphrases from a large collection of sentences. Fields of application are, for example, online argumentation tools, where users can interchange arguments on certain topics. Newly added arguments can be compared to existing posts and duplicate, paraphrased entries can be avoided. A trained model that is good at measuring argument similarity is also advantageous for tasks like argument mining and argument clustering.

BERT model	BERT medium uncased, BERT base uncased
learning_rate	1e-4, 2e-5
do_lower_case	True
max_seq_length	128
max_predictions_per_seq	5
masked_lm_prob	0.15
random_seed	12345
dupe_factor	10

Table 9: Settings for creating the pre-training data.

Appendix

Pre-Training and Fine-Tuning Settings. Table 9 hows the used settings for pre-training all proposed BERT models in this work.

Table 10 shows the settings for fine-tuning SBERT on the evaluated datasets, using the sentence- transformers library[18] published by the UKPLab on GitHub.

learning_rate	2e-5
train_batch_size	16
num_epochs	5
optimizer_class	transformers.AdamW
weight_decay	0.01

Table 10: Settings for fine-tuning.

[18] https://github.com/UKPLab/sentence-transformers

6 MaxPoolBERT: Enhancing BERT Classification via Layer- and Token-Wise Aggregation

Corresponding Publication. This chapter is based on the work:

MaxPoolBERT: Enhancing BERT Classification via Layer- and Token-Wise Aggregation
Maike Behrendt, Stefan Sylvius Wagner & Stefan Harmeling

Personal Contributions. Together with Stefan Harmeling and Stefan Sylvius Wagner, Maike Behrendt developed the idea of enhancing the `[CLS]` token in BERT by pooling information across layers and tokens. Maike Behrendt was primarily responsible for implementing the model and conducting all experiments. The paper was written and revised in collaboration with the co-authors Stefan Harmeling and Stefan Sylvius Wagner, who also provided feedback on the implementation process and the experiments.

Abstract. The `[CLS]` token in BERT is commonly used as a fixed-length representation for classification tasks. However, prior work has shown that both other tokens and intermediate layers encode valuable contextual information, raising the question of whether the standard `[CLS]` embedding fully exploits the representational capacity of the model. In this work, we propose MaxPoolBERT, a lightweight extension to BERT that refines the `[CLS]` representation by aggregating information across layers and tokens. Specifically, we explore three modifications: (i) max-pooling the `[CLS]` token across multiple layers, (ii) enabling the `[CLS]` token to attend over the entire final layer using an additional multi-head attention (MHA) layer, and (iii) combining max-pooling across the full sequence with MHA. Our approach enhances BERT's classification accuracy (especially on low-resource tasks) without requiring pre-training or significantly increasing model size. We evaluate our approach on the GLUE benchmark using multiple random seeds to address both performance and stability. Our results show that MaxPoolBERT consistently achieves a better performance on the standard BERT-base model on low resource tasks of the GLUE benchmark. These findings demonstrate that small, targeted adjustments to the `[CLS]` aggregation mechanism can yield improvements in downstream classification performance.

Research Problem & Motivation. Similar to ArgueBERT (Behrendt and Harmeling, 2021), the goal of MaxPoolBERT was to optimize a small BERT model for downstream

tasks. In online discussions, the automatic classification of content, such as user contributions, is the basis for a variety of applications. These applications include the identification of harmful language in user comments (Risch et al., 2021) and the automatic labeling of topics in user contributions (Romberg and Escher, 2022). In citizen participation processes, where data protection compliance is important and computational resources are limited, small models with full control over the fine-tuning data are crucial.

Main Results. The proposed layer- and token-wise max-pooling technique improves classification performance on small-scale datasets of the GLUE benchmark. The results suggest that aggregating information across multiple layers enhances the robustness of sentence representations during fine-tuning. Importantly, these improvements are achieved without increasing the overall model size, making the approach a promising candidate for resource-constrained participation contexts. The code to fine-tune the model on GLUE is available at: `https://github.com/mabehrendt/MaxPoolBERT/`.

Publication Status. Under Review. This work is currently under review at IEEE Access. The manuscript is available as a pre-print (Behrendt et al., 2025a).

6.1 Preamble

Building on the previous chapter, which investigated improvements to pre-training for argument-aware language representations, we now turn to further enhancing the efficiency and effectiveness of compact Transformer-based models in downstream settings.

The contributions presented in this chapter complement those of the previous chapter by addressing Research Question 2 from a different perspective: In this chapter, we focus on architectural adaptations during fine-tuning to improve classification performance, particularly in low-resource settings.

6.2 Introduction

Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019), is one of the best known Transformer-based (Vaswani et al., 2017) language models. The core principle of BERT is the unsupervised pre-training approach on large corpora, enabling it to learn contextual word representations, which can then be used to solve various downstream tasks. Through fine-tuning, BERT adapts its representations to aggregate the most relevant information required for a given task.

A key component of BERT's architecture is the classification token (abbreviated `[CLS]`), a special token that is prepended to every input sequence. During fine-tuning, the `[CLS]` token serves as the only input to the classification head, which generates predictions for

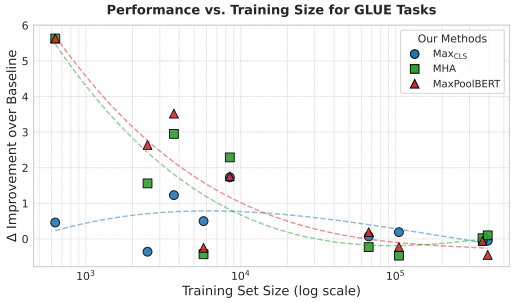


Figure 9: **MaxPoolBERT performs best on low-resource datasets.** We show that our methods, in particular MaxPoolBERT, provide significant improvements for smaller datasets indicating that the model learns a better representation during fine-tuning (top-left).

the task at hand. Through self-attention, the `[CLS]` token is expected to capture the sentence-level information necessary for downstream tasks. In this paper, we ask the question whether we can enrich the `[CLS]` token with information from the layers below the top level.

We know that the last layers of BERT change the most during fine-tuning and encode the most task-specific information (Rogers et al., 2020). This is why the `[CLS]` token embedding from the final layer is conventionally used for classification. However, assuming that only the `[CLS]` token retains meaningful sentence-level information is misleading. Prior studies have shown that all token embeddings in the final layer contain sentence-level information (Rogers et al., 2020), and that using different token positions for classification results in only minor differences in accuracy (Goyal et al., 2020). Goyal et al. (2020) also found that embedding vectors in the final layer exhibit high cosine similarity due to information mixing through self-attention.

Motivated by these findings, we explore incremental modifications to the BERT base architecture for sequence classification, aiming to enhance its performance on downstream tasks. We specifically focus on improving the informativeness of the `[CLS]` token by (i) incorporating more *width* information of the whole sequence, and (ii) incorporating more *depth* information from additional layers. In the end we find that a mixture of these approaches leads to the best results.

Contributions.

1. We introduce MaxPoolBERT, an effective extension to BERT that enriches the `[CLS]` token representation using max-pooling and attention mechanisms across layers and tokens.

2. We systematically evaluate three architectural variants that incorporate *width* (token-level) and *depth* (layer-level) information into the `[CLS]` embedding.

3. We show that our proposed approach improves fine-tuning performance on 7 out of 9 GLUE tasks and achieves an average gain of 1.25 points over the BERT base baseline.

4. We demonstrate that MaxPoolBERT is particularly effective in low-resource scenarios, providing improved stability and accuracy where training data is limited.

All of our models are available under `https://github.com/mabehrendt/MaxPoolBERT`.

6.3 Related Work

Much research has been done dedicated to improving and optimizing BERT's training process through architectural modifications and fine-tuning strategies. Below, we discuss advancements in fine-tuning stability, text representations, model enhancements, and training efficiency. Our work falls within the branch of research aimed at optimizing BERT's representation to enhance downstream classification results, with a particular focus on augmenting the informativeness of the `[CLS]` token.

Stabilized BERT Fine-Tuning. The pre-training and fine-tuning paradigm for language models such as BERT (Devlin et al., 2019) has led to significant improvements across a wide range of NLP tasks while keeping computational costs manageable. However, fine-tuning remains unstable due to challenges like vanishing gradients (Mosbach et al., 2021) and limited dataset sizes (Zhang et al., 2021). Several studies have proposed techniques to address this instability.

Zhang et al. (2021) explore re-initializing BERT layers before fine-tuning, demonstrating that retaining all pre-trained weights is not always beneficial for fine-tuning. They also show that extending fine-tuning beyond three epochs improves performance. Hao et al. (2020) examine how fine-tuning affects BERT's attention, finding that higher layers change significantly while lower layers remain stable. They propose a noise regularization method to enhance stability. Mosbach et al. (2021) identify high learning rates as a key issue that cause fine-tuning instability. They propose using small learning rates with bias correction and increasing training iterations until nearly zero training loss is achieved. Hua et al. (2021) introduce Layer-wise Noise Stability Regularization which further stabilizes fine-tuning through regularization. Xu et al. (2023) propose self-ensemble and self-distillation mechanisms that enhance fine-tuning stability without requiring architectural changes or external data.

Our method, while not explicitly targeting stability, contributes to more robust performance especially on low-resource tasks by enabling the `[CLS]` token to integrate a broader context via pooling and attention. We analyze fine-tuning stability of our variants in Section 6.6.2.

Faster and More Efficient Training. In addition to stabilization, architectural enhancements have been introduced to boost BERT's efficiency and effectiveness. Goyal et al. (2020) propose to eliminate tokens after fine-tuning, to reduce the time of inference. They discovered that the token representations in the highest layer of BERT base carry similar information.

Recently, Warner et al. (2025) introduce ModernBERT, an updated version of BERT with an increased sequence length of 8192. ModernBERT incorporates architectural improvements such as GeGLU activations (Shazeer, 2020), Flash Attention (Dao et al., 2022), and RoPE embeddings (Su et al., 2024).

While other approaches improve the input embedding size of BERT (Nussbaum et al., 2025) or refine the pre-training process for GPU's (Geiping and Goldstein, 2023; Portes et al., 2023; Izsak et al., 2021), our work specifically concentrates on optimizing the `[CLS]` token during fine-tuning, leveraging the information captured in BERT's layers after pre-training.

Improved BERT Fine-Tuning and Representation Learning. Lastly, several approaches refine BERT's classification capability through optimized fine-tuning strategies and enriched sentence representations - areas that align closely with our approach (see also Stankevičius and Lukoševičius (2024) who provide a comprehensive survey of methods for extracting sentence-level embeddings from BERT).

Toshniwal et al. (2020) systematically compared different text span representations using BERT, and found that max-pooling performs quite well across tasks, though its effectiveness varies. Bao et al. (2021) construct sentence representations for classification by selecting meaningful n-grams and combining sub-tokens of a pre-trained BERT model into span representations using a max-pooling approach. In contrast, our method does not require span selection or input modification, and applies pooling and attention directly to hidden states during fine-tuning. s Hu et al. (2024) introduce a flexible BERT architecture with dynamic width and depth that adapts the number of attention heads, hidden size, and number of layers at inference time using knowledge distillation. Our approach does not alter the base architecture, instead we enrich the fixed-size `[CLS]` embedding to boost classification performance.

Chang et al. (2023) introduce Multi-CLS BERT, a framework that modifies pre-training and adds multiple `[CLS]` tokens to the sequence for fine-tuning. We achieve comparable results on the GLUE benchmark without altering the pre-training setup. Chen et al. (2023) present HybridBERT, which incorporates a hybrid pooling network and drop masking during pre-training to accelerate training and improve downstream accuracy. While HybridBERT combines multiple pooling strategies (mean, max, and attention) to replace the `[CLS]` token, we retain the original `[CLS]` embedding and instead enrich it through architectural refinements such as an additional multi-head attention layer and optional sequence-wide pooling. This allows our method to be applied to any pre-trained

BERT-like model without re-training, with particular benefits observed on tasks with limited training data.

Recently, Galal et al. (2024) explore aggregation techniques such as mean pooling and self-attention on output embeddings for Arabic sentiment analysis. They show that freezing BERT during fine-tuning can boost performance. Our method can be combined with such techniques but focuses on improving the [CLS] pathway, especially under low-resource conditions.

Lastly, Lehečka et al. (2020) propose modifying BERT's output pooling strategy to improve large-scale multi-label text classification. Specifically, they replace the [CLS] token with combined mean and max pooling over the final hidden states of all tokens, arguing that this captures richer semantic information for classification. While their method entirely discards the [CLS] embedding, our approach retains it and enhances its contextual richness by integrating sequence-wide information via additional architectural layers during fine-tuning.

6.4 Refining the [CLS] Token

It has been shown that other token representations in the layers of BERT also capture sentence-level representations (Rogers et al., 2020). We investigate whether the informativeness of the [CLS] token embedding can be further enhanced during fine-tuning, to improve downstream classification results. To do this, we include more *depth* information from other BERT layers and also more *width* information from other tokens within the sequence. We study different versions of fine-tuning BERT for sequence classification tasks. All variants are described below.

6.4.1 Preliminaries

Final-Layer [CLS] Representation. As a baseline we use the [CLS] token of the final encoder layer of a fine-tuned vanilla BERT base model (Devlin et al., 2019) for classification (see Figure 10a). Recall that a single layer of BERT can be written as

$$f_i : \mathbb{R}^{T\times d} \to \mathbb{R}^{T\times d}, \tag{27}$$

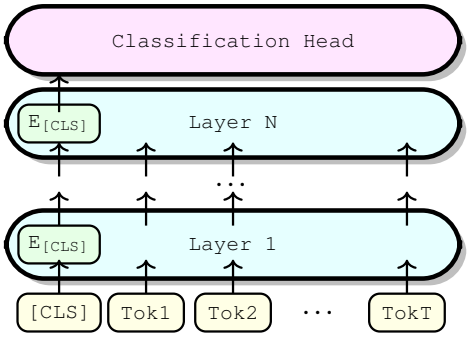
where i indicates the layer number (BERT base has 12 layers), T is the number of tokens, and d is the dimensionality of each token vector. We denote the values of the intermediate layers by $y^{(i)}$:

$$y^{(1)} = f_1(x), \qquad y^{(i+1)} = f_{i+1}(y^{(i)}). \tag{28}$$

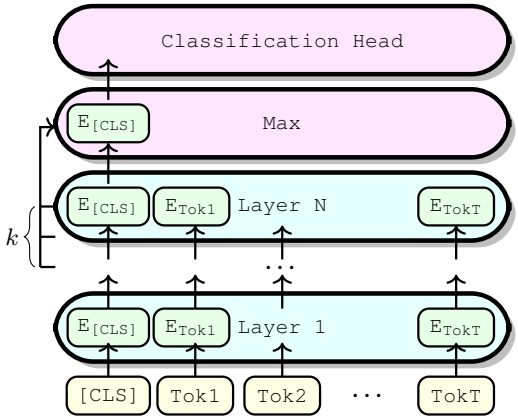
The classification token of each layer is the first token, i.e., for a sequence of tokens $y^{(i)} = [t_{1i}, ..., t_{Ti}]$ in the ith layer,

$$\texttt{[CLS]}_i = t_{1i} \in \mathbb{R}^{1\times d}. \tag{29}$$

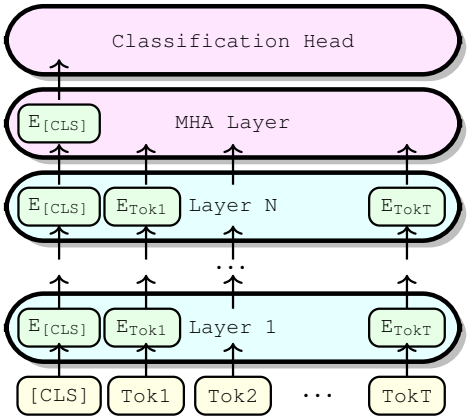
The embedding of the [CLS] token serves as the input for the classification head c, which we have choosen to be a simple linear layer without an activation function, since we are



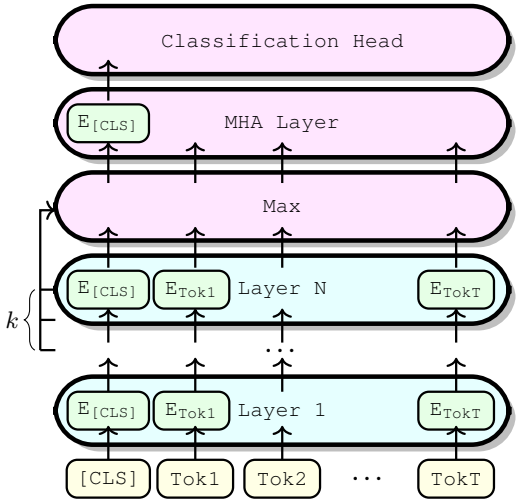
(a) **Baseline.** Plain vanilla BERT for sequence classification, where the embedding of the [CLS] token of the final layer is used as input for the classification head.



(b) **Max$_{\text{CLS}}$.** A max-pooling operation is applied on the [CLS] tokens of the last k layers before classification.



(c) **MHA.** An additional multi-head attention layer allows the [CLS] token to attend to all tokens of the last layer.



(d) **Max$_{\text{Seq}}$ + MHA.** A max-pooling operation on the whole sequence is combined with an additional MHA layer.

Figure 10: **Comparison of four BERT architectures for sequence classification.** *(Left above)* Classical BERT for sequence classification architecture. *(Right above)* Applying max-pooling on the token embeddings of the [CLS] token over the last k layers. *(Left below)* Adding an additional MHA layer before classification. *(Right below)* MaxPoolBERT architecture: After the Nth layer (N = 12 for BERT base), we apply a sequence-wide max-pooling operation over the last k layers (we used $k = 3$). The [CLS] token can then attend to every token after the max-pooling and the resulting [CLS] token embedding is used for classification.

just interested in the plain expressiveness of the refinement (instead of adding tanh as in the original BERT implementation):

$$c : \mathbb{R}^{1\times d} \to \mathbb{R}. \tag{30}$$

Thus, the baseline model for sequence classification can be written as:

$$(c \circ \text{CLS} \circ f_{12} \circ \ldots \circ f_1) : \mathbb{R}^{T\times d} \to \mathbb{R} \tag{31}$$

for BERT base with 12 layers.

Max-pooling operation. The final layers of a BERT model are known to contain the task-specific information. In order to utilize not only the last layer but several layers, we have to define a flexible maximizing operation, that can work with several sequences of vectors. For this, we write $\Theta_t^{(k)} \in \mathbb{R}^{k\times t\times d}$ for the tensor that contains the first t token vectors (each d dimensional) of the last k layers. For instance, $\Theta_1^{(1)} \in \mathbb{R}^{1\times 1\times d}$ is the `[CLS]` token, and $\Theta_1^{(k)} \in \mathbb{R}^{k\times 1\times d}$ collects the token vectors the `[CLS]` token of the last k layers. Similarly, $\Theta_T^{(1)} \in \mathbb{R}^{1\times T\times d}$ contains all token vectors of the last layer, and $\Theta_T^{(k)} \in \mathbb{R}^{k\times T\times d}$ all token vectors of the last k layers.

Next, we define an element-wise max-pooling operation that maximizes over the first dimension, i.e.,

$$\max : \mathbb{R}^{k\times t\times d} \to \mathbb{R}^{t\times d}. \tag{32}$$

Written as Pytorch[19] code, the operation is `torch.max(Theta, dim=1)` for b-sized minibatches of shape $b \times k \times t \times d$.

Mean-pooling operation. Several studies indicate, that max-pooling seems to be a stable choice to aggregate information into a single sentence representation. In the experimental section (Section 6.5), we also consider mean-pooling to challenge these results. For this, we apply an element-wise mean-pooling operation

$$\text{mean} : \mathbb{R}^{k\times t\times d} \to \mathbb{R}^{t\times d} \tag{33}$$

on every vector of our k chosen layers (defined analogously as the max-pooling operation).

6.4.2 Depth-Wise `[CLS]` Pooling (Max_{CLS})

To use the *vertical* information (i.e., more depth) as one possible improvement for BERT's fine-tuning, we take information from the last k layers (instead of only from the last layer): we extract the last k `[CLS]` embeddings $[\texttt{[CLS]}_{12-k+1}, \ldots, \texttt{[CLS]}_{12}]$ which corresponds to $\Theta_1^{(k)} \in \mathbb{R}^{k\times 1\times d}$ (using the notation of the previous paragraph). Then we apply the element-wise max-pooling operation on the extracted tokens (see Figure 10b).

[19] `https://pytorch.org/`

6.4.3 Token-Wise Attention via Additional MHA Layer (MHA)

The orthogonal way to enrich the information in the `[CLS]` token, is to consider *horizontal* information (i.e., more width, see Figure 10c). For this, we include *all* tokens of the last layer. To obtain a single vector, we employ an additional multi-head attention (MHA) layer on the encoder output, but compute the attention only for the `[CLS]` token. We write the MHA as (see Vaswani et al., 2017),

$$\text{MHA(Q,K,V)} = [\text{head}_1, \ldots, \text{head}_h]W_0 \tag{34}$$

where the heads are defined as

$$\text{head}_s = \text{Attention}(QW_s^Q, KW_s^K, VW_s^V). \tag{35}$$

Using the standard BERT base model with 12 layers, we have $Q =$ `[CLS]`$_{12}$ and $K = V = y^{(12)}$. Through the attention mechanism, the `[CLS]` token can attend to all other tokens once more before classification. Note that the additional MHA layer is not part of the pre-training process and is added and initialized before the fine-tuning process. We use the default initialization of the Pytorch[1] multi-head attention implementation which is a Xavier uniform initialization (Glorot and Bengio, 2010). For the number of attention heads we choose $h = 4$.

6.4.4 Sequence-Wide Pooling with MHA (Max_{Seq} + MHA & Mean_{Seq} + MHA)

Finally, we combine the additional *depth* and *width* information of $\textbf{Max}_{\textbf{CLS}}$ and **MHA** by extending the max-pooling operation to the whole sequences of the last k layers by using $\max(\Theta_T^{(k)}) \in \mathbb{R}^{k \times T \times d}$. We call this setup $\textbf{Max}_{\textbf{Seq}}$**+ MHA**, since the maximum is now along the whole sequence and the additional MHA layer aggregates the pooled information. We call this approach **MaxPoolBERT** in the following. As a variant, we replaced max pooling with mean pooling. We report the results for mean pooling with an additional MHA layer as $\textbf{Mean}_{\textbf{Seq}}$**+ MHA**.

Parameter	Value
learning rate	2e-5
epochs	4
batch size	32
warmup ratio	0.1
weight decay	0.01

Table 11: Hyperparameters used for all fine-tuning experiments.

6.5 Experiments

In order to evaluate each previously presented modification of the BERT architecture for sequence classification, we fine-tune each model on different classification tasks of the GLUE benchmark and compare the results. As a baseline, we use a standard BERT base model (Devlin et al., 2019). In addition, we assess the generalizability of our approach by applying it to a BERT variant, namely RoBERTa base (Liu et al., 2019).

6.5.1 Datasets

The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) is a well known benchmark for natural language understanding (NLU) and natural language inference (NLI) tasks. We evaluate on the following 9 tasks:

- **CoLA** (Corpus of Linguistic Acceptability (Warstadt et al., 2019)): 10,657 sentences from linguistic publications, annotated for grammatical acceptability (*acceptable* or *unacceptable*).
- **MRPC** (Microsoft Research Paraphrase Corpus (Dolan and Brockett, 2005)): 5,800 sentence pairs from news source, annotated for paraphrase identification (*equivalent* or *not equivalent*).
- **QNLI** (Question NLI): an NLI dataset derived from the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) containing question paragraph pairs. The task is to predict if the question is answered by the given paragraph (*entailment* or *no entailment*).
- **MNLI** (Multi-Genre NLI (Williams et al., 2018)): includes 433,000 sentence pairs, annotated with three different indicators for entailment (*neutral*, *contradiction* or *entailment*). MNLI includes both matched (in-domain) and mismatched (cross-domain) sections.
- **SST-2** (The Stanford Sentiment Treebank (Socher et al., 2013)): 215,154 phrases annotated for sentiment analysis (*positive* or *negative*).
- **STS-B** (Semantic Textual Similarity Benchmark (Cer et al., 2017)): 8,630 sentence pairs annotated with a textual similarity score (*from zero to five*).
- **RTE** (Recognizing Textual Entailment (Dagan et al., 2006)): 5,770 sentence pairs annotated for entailment recognition (*entailment* or *no entailment*).
- **QQP** (Quora Question Pairs): 795,000 pairs of questions from Quora, annotated for semantical similarity (*duplicate* or *no duplicate*).
- **WNLI** (Winograd NLI Levesque et al., 2012): 852 sentence pairs annotated for textual entailment (*entailment* or *no entailment*).

6.5.2 Experimental Details

All experiments were run on a single NVIDIA A100 GPU. We used the Huggingface transformers and dataset libraries[20] to implement and train all of our models. Each model was fine-tuned three times with three different random seeds for four epochs. We report

[20] https://huggingface.co/

Model	CoLA	MRPC		QNLI	MNLI		SST-2	STS-B	RTE	QQP		WNLI
	MCC	Acc.	F1	Acc.	m	mm	Acc.	Sp.	Acc.	Acc.	F1	Acc.
Train Size	8.5k	3.7k	3.7k	105k	393k	393k	67k	5.75k	2.5k	364k	364k	634
BERT base	53.59	82.43	87.49	90.96	84.27	84.57	92.55	88.47	63.42	90.65	87.40	49.77
Max_{CLS}	55.32	83.66	88.5	**91.15**	84.22	84.55	92.62	**88.97**	63.06	90.59	87.33	50.23
MHA	**55.88**	85.38	89.51	90.49	**84.37**	**84.67**	92.32	88.04	64.98	90.67	**87.45**	**55.4**
Max_{Seq}+MHA	55.35	**85.95**	**89.78**	90.73	83.82	84.24	**92.74**	88.22	66.06	90.59	87.32	**55.4**
Mean_{Seq}+MHA	55.10	85.62	89.66	90.86	83.78	84.2	92.51	87.91	**66.67**	**90.68**	87.41	54.46
Δ	2.29	3.52	2.29	0.19	0.24	0.1	0.19	0.5	3.25	0.03	0.05	5.63

Table 12: **Our proposed variants improve the performance over BERT base on GLUE validation tasks (average of 3 seeds).** The size of the training data set is highlighted in gray. We report Matthews correlation coefficient (MCC) for CoLA, accuracies for matched (m) and mismatched results (mm) for MNLI, and Spearman correlation (Sp.) for STS-B. Below we report the improvement from the best performing variant over the baseline as Δ.

Model	CoLA ↓	MRPC ↓	QNLI ↓	MNLI ↓	SST-2 ↓	STSB ↓	RTE ↓	QQP ↓	WNLI ↓
BERT Base	6.34e-02	2.42e-02	**2.08e-03**	**1.97e-03**	**1.99e-03**	**3.2e-03**	**1.78e-02**	10.8e-04	5.86e-02
Max_{CLS}	4.55e-02	**2.02e-02**	3.89e-03	2.73e-03	3.81e-03	3.8e-03	1.86e-02	9.26e-04	4.61e-02
MHA	4.3e-02	2.1e-02	5.69e-03	2.43e-03	3.63e-03	4.99e-03	1.86e-02	8.09e-04	4.61e-02
Max_{Seq} + MHA	**4.22e-02**	2.18e-02	5.11e-03	4.45e-03	3.64e-03	4.63e-03	1.96e-02	7.87e-04	4.31e-02
Mean_{Seq} + MHA	**4.22e-02**	2.03e-02	4.49e-03	5.04e-03	4.46e-03	4.55e-03	2.12e-02	**7.46e-04**	**3.97e-02**

Table 13: **Standard deviations for three fine-tuning runs with different random seeds.**

the mean of all runs and use the validation sets of all GLUE tasks for evaluation. Experimenting with different values for k (the number of the considered layers), we found that $k = 3$ works best (see Appendix 6.7.1). All others hyperparameters are listed in Table 11.

6.6 Results

We report the results for all model variants in each task and analyze fine-tuning stability by measuring the standard deviation between runs with different seeds.

6.6.1 Performance Across GLUE Tasks

The performance of each of our four variants on the GLUE benchmark tasks is presented in Table 12. For each task, at least one variant achieves higher performance than the BERT baseline, indicating that our proposed methods for enriching the `[CLS]` token representation are effective. However, the magnitude of improvement varies across tasks.

The $\textbf{Max}_{\textbf{CLS}}$ variant, which applies max-pooling over the `[CLS]` token representations from the last k layers, results in marginal to no improvement for most tasks. Notably, this variant achieves the best performance among all variants on QNLI and STS-B, suggesting that layer-wise max-pooling can be beneficial for certain task types. Both tasks incorporate semantic matching between two texts, thus both require nuanced understanding of sentence meaning.

Model	GLUE avg.
BERT Base	79.63
Max_{CLS}	80.02
MHA	80.76
Max_{Seq}+MHA	**80.85**
Mean_{Seq}+MHA	80.75
Δ	1.25

Table 14: **Average performance across all GLUE tasks.** MaxPoolBERT shows a consistent gain over BERT base.

The **MHA** variant introduces an additional MHA layer, allowing the final-layer `[CLS]` token to attend to the full sequence before classification. This variant consistently improves upon the baseline BERT model, indicating that this extra attention step, effectively enhances the model's ability to integrate global context. The biggest improvement is observed on the WNLI dataset, which has the fewest training examples in the GLUE benchmark (634 training examples in total), suggesting that the added attention is particularly helpful in low-resource settings.

The **Max_{Seq}+MHA** variant combines token-wise max-pooling over the sequence with the additional MHA layer. This configuration shows the most consistent improvements, achieving a higher performance than the baseline in 7 out of 9 tasks. As shown in Figure 9, the largest improvements are again seen on datasets with limited training data, such as CoLA, MRPC, RTE and WNLI. These findings suggest that combining sequence-level pooling with attention further enhances robustness in low resource settings.

Model	GLUE avg.
RoBERTa Base	**82.62**
Max_{Seq}+MHA	82.6
Δ	-0.02

Table 15: **Average performance across all GLUE tasks.** MaxPoolBERT shows a gain over RoBERTa base.

Overall, **Mean_{Seq}+MHA** performs similarly well as max-pooling, on RTE it even achieves higher performance than max-pooling. In the end, max-pooling seems to be a better choice as mean-pooling as it works better for most GLUE tasks.

For clarity, Table 14 shows the average performance of each model variant across all GLUE tasks. The **Max_{Seq}+ MHA** variant, which we call **MaxPoolBERT**, achieves the highest overall average. While the average improvement over the baseline is 1.25 points, individual tasks show greater improvements.

Additionally we apply the max-pooling and MHA combination to a BERT variant, namely RoBERTa (Liu et al., 2019). The results are depicted in Table 16. For RoBERTa, our max-pooling + MHA variant shows improvements on 4 out of 9 tasks but on average both models perform equally (see Table 15). RoBERTa is an optimized version of BERT with a similar architecture, however the effects vary on this BERT variant. A significant improvement can only be observed on the CoLA dataset.

Model	CoLA	MRPC		QNLI	MNLI		SST-2	STS-B	RTE	QQP		WNLI
	MCC	Acc.	F1	Acc.	m	mm	Acc.	Sp.	Acc.	Acc.	F1	Acc.
Train Size	8.5k	3.7k	3.7k	105k	393k	393k	67k	5.75k	2.5k	364k	364k	634
RoBERTa Base	54.96	**88.56**	**91.64**	**92.29**	**87.15**	86.97	93.2	89.78	**73.29**	90.84	87.64	**56.34**
Max_{Seq}+MHA	**57.43**	87.58	91.01	92.15	86.99	**87.14**	**93.43**	**90.49**	70.4	**90.93**	**87.85**	55.87
Δ	2.47	-0.98	-0.63	-0.14	-0.16	0.17	0.23	0.71	-2.89	0.09	0.21	-0.47

Table 16: **Performance on GLUE validation tasks for RoBERTa (average of 3 seeds).**

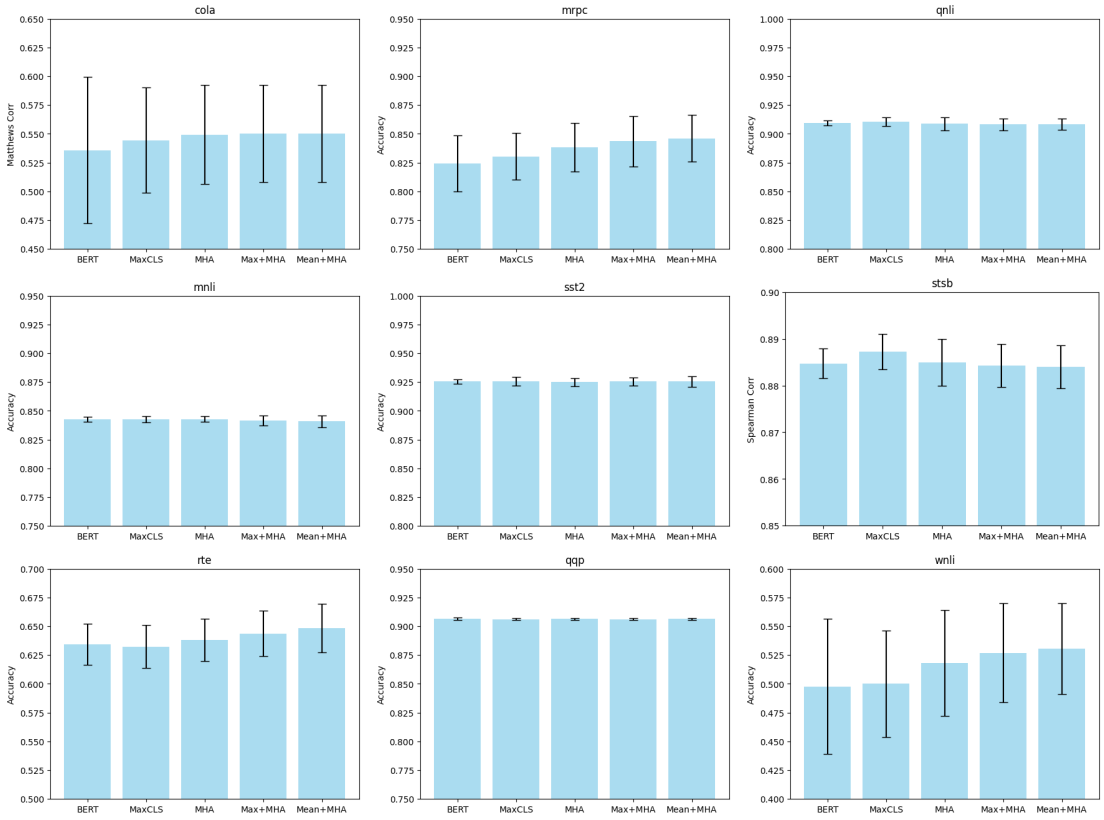


Figure 11: **Accuracies for the GLUE benchmark with error bars.** We show the standard deviation between three fine-tuning runs with three random seeds. Note that the y-axis is shifted but scaled equally across tasks.

6.6.2 Stability on Low-Resource Tasks

To assess fine-tuning stability, which is usually worse for smaller datasets (Devlin et al., 2019; Lee et al., 2020; Dodge et al., 2020), we run all experiments with three different seeds for each GLUE task. We report the mean accuracy across runs (for CoLA we report Matthews correlation coefficient, for STS-B we report Spearman rank correlation), and include error bars showing the standard deviation of these three runs (see Figure 11 and Table 13).

We observe that the stability in fine-tuning remains comparable across model variants for most datasets. However, improvements are observed for datasets with fewer training samples such as CoLA, MRPC, QQP and WNLI, where our variants exhibit reduced variability between runs. These findings suggest that our proposed modifications improve robustness in the low-sample regime.

6.7 Conclusion

We introduced MaxPoolBERT, a lightweight yet effective refinement of BERT's classification pipeline that improves the representational quality of the `[CLS]` token for the BERT base model. Our method leverages max-pooling across layers and tokens, and introduces a multi-head attention layer that allows the `[CLS]` token to re-aggregate contextual information before classification. These modifications require no changes to pre-training and add minimal overhead to fine-tuning.

Empirical results on the GLUE benchmark demonstrate that MaxPoolBERT outperforms standard BERT base across most tasks, with especially strong improvements in low-resource settings. This suggests that BERT's native use of the final-layer `[CLS]` embedding underutilizes available information and that small architectural additions can enhance generalization without sacrificing efficiency.

Limitations

While MaxPoolBERT improves downstream performance, several limitations remain:

- **No task-specific tuning.** Our experiments use shared hyperparameters across tasks. Further gains could be possible with task-specific settings for pooling depth, attention heads, or training schedules.
- **Model size and generalization.** Our work focuses on BERT base. We also examined one BERT variant but were not able to demonstrate an advantage of applying max-pooling + MHA for RoBERTa.
- **Scope of evaluation.** We focus on sentence-level classification tasks in GLUE. The applicability of our approach to other tasks, such as token classification, generation, or cross-lingual transfer, is not yet evaluated.

In the future we aim to further investigate how to optimize the fine-tuning of small BERT models. While larger models often yield better performance, smaller models are crucial in real-time or resource-constrained environments. The BERT training paradigm following pre-training and fine-tuning has been predominant for several years and is widely used, so it is important to study whether further improvements can be made through small changes to this learning paradigm.

Acknowledgments

This research has been funded/supported by the Federal Ministry of Education and Research of Germany and the state of North Rhine-Westphalia as part of the Lamarr Institute for Machine Learning and Artificial Intelligence.

Appendix

6.7.1 Ablations

We conduct ablation studies to test different modifications of our architectures. We describe all experiments and report their results in the following.

Choice of k. We experiment with the choice of k for the max-pooling layer on the smaller GLUE datasets (CoLA, MRPC and RTE) and report the results in the following in Table 17. Because we average over three runs with different random seeds, the choice of k does not have an immense influence on performance, but it is apparent that $k = 3$ is the best choice on the data sets tested.

	CoLA	**MRPC**		**RTE**
	Acc.	Acc.	F1	Acc.
$k = 1$	54.85	83.58	88.44	63.18
$k = 2$	55.76	85.21	89.29	65.34
$k = 3$	55.35	**85.95**	**89.78**	**66.06**
$k = 4$	**56.42**	85.29	89.27	65.34
$k = 6$	55.65	85.13	89.25	65.7
$k = 12$	55.41	85.21	89.17	65.46

Table 17: **Effect of max-pooling depth k on small GLUE tasks.** $k = 3$ generally yields best results.

Time Difference. To evaluate differences in fine-tuning and inference time, we measured the time to fine-tune both standard BERT and MaxPoolBERT for four epochs on the MRPC dataset on a single A100 GPU. We also measured the inference time on the MRPC validation set for both model variants. Fine-tuning BERT took 289.298 seconds (approx. 72.324 seconds per epoch), inference on the validation set took 2.9935 seconds. In contrast, fine-tuning MaxPoolBERT on the MRPC dataset takes: 294.504 seconds (approx. 73.626 seconds per epoch). Inference time on the validation set is 3.0284 seconds. That is a difference of approximately 1.3 seconds per epoch of fine-tuning and 0.035 seconds difference for inference, which is neglectable.

7 AQuA – Combining Experts' and Non-Experts' Views To Assess Deliberation Quality in Online Discussions Using LLMs

Corresponding Publication. This chapter is based on the work:

AQuA – Combining Experts' and Non-Experts' Views To Assess Deliberation Quality in Online Discussions Using LLMs
Maike Behrendt, Stefan Sylvius Wagner, Marc Ziegele, Lena Wilms, Anke Stoll, Dominique Heinbach & Stefan Harmeling
In Proceedings of the First Workshop on Language-driven Deliberation Technology (DELITE)

Personal Contributions. This work is the result of a collaboration between computer scientists and social scientists as part of the UPEKI project. Maike Behrendt, Stefan Sylvius Wagner, Marc Ziegele, Anke Stoll, Lena Wilms, and Stefan Harmeling jointly developed the idea for the AQuA score. Maike Behrendt was responsible for training the adapter models used as the basis for the score. She also conducted all experiments to evaluate the trained models. The training data was collected and annotated by Lena Wilms, Anke Stoll, Dominique Heinbach, and Marc Ziegele. Maike Behrendt wrote the initial draft of the paper, which was finalized in collaboration with Stefan Sylvius Wagner, Marc Ziegele, Lena Wilms, Anke Stoll, and Stefan Harmeling. The code and the fine-tuned adapter models are available at `https://github.com/mabehrendt/AQuA`.

Abstract. Measuring the quality of contributions in political online discussions is crucial in deliberation research and computer science. Research has identified various indicators to assess online discussion quality, and with deep learning advancements, automating these measures has become feasible. While some studies focus on analyzing specific quality indicators, a comprehensive quality score incorporating various deliberative aspects is often preferred. In this work, we introduce AQuA, an additive score that calculates a unified deliberative quality score from multiple indices for each discussion post. Unlike other singular scores, AQuA preserves information on the deliberative aspects present in comments, enhancing model transparency. We develop adapter models for 20 deliberative indices, and calculate correlation coefficients between experts' annotations and the perceived deliberativeness by non-experts to weigh the individual indices

into a single deliberative score. We demonstrate that the AQuA score can be computed easily from pre-trained adapters and aligns well with annotations on other datasets that have not been seen during training. The analysis of experts' vs. non-experts' annotations confirms theoretical findings in the social science literature.

Research Problem & Motivation. The main goal of the UPEKI project was to implement and evaluate AI-based solutions to improve user discussions by enhancing the quality of decision-making processes and their outcomes. The survey conducted in Chapter 4 revealed that much of the existing research focuses on identifying hate speech and other forms of inappropriate comments to support human moderators. One alternative strategy to improve the quality of user comments and thus the quality of the whole discussion is to highlight high-quality content (Wang and Diakopoulos, 2022). In the study by Wang and Diakopoulos (2022), the authors showed positive effects when high-quality comments below newspaper articles were selected and highlighted by editors. We aimed to examine whether the selection of high-quality comments could be automated by an AI model and whether this would have similar effects. To this end, we developed a score based on 20 smaller adapter models that predict features of deliberative quality.

Main Results. We demonstrate how deliberative quality can be operationalized as an additive score derived from multiple automatically predicted features. The individual models achieve reliable performance across several deliberative dimensions, and the aggregated score correlates with human assessments of comment quality. The results indicate that AI systems can approximate both expert and non-expert evaluations of deliberative contributions, thereby enabling scalable assessment of discussion quality in participation platforms.

Publication Status. Published. This paper was accepted for the First Workshop on Language-driven Deliberation Technology (DELITE2024), co-located at the LREC-COLING conference 2024 in Torino, Italy (Behrendt et al., 2024).

7.1 Preamble

While the previous chapters focused on improving the efficiency and performance of compact language models, the question remains how such models can be embedded into real-world online discussion environments and whether they effectively support deliberative communication in practice.

In this chapter, we therefore move from model development to the design of AI-based methods for analyzing online discussions. We introduce the AQuA score, a multi-dimensional framework for automatically assessing the deliberative quality of user comments. The approach combines insights from deliberation theory with NLP techniques to capture different aspects of the quality of user comments.

By providing a structured and scalable measure of discussion quality, this work lays the foundation for the development of targeted AI-based interventions, which are explored in the subsequent chapter.

7.2 Introduction

In the evolving landscape of democratic discourse, the concept of deliberation stands as a cornerstone, embodying the exchange of ideas, critical discussion, and consensus-building among citizens (Dryzek, 2002). Central to the efficacy of these deliberations is their quality, a multifaceted construct traditionally gauged by dimensions such as rationality, civility, reciprocity, and constructiveness (Friess and Eilders, 2015). More recent research has explored various indicators of deliberative quality in online discussions (Steenbergen et al., 2003; Friess and Eilders, 2015; Scudder, 2022). However, most of these approaches require manual annotation of discussion data from trained coders and serve to analyze the discussion in retrospect. As the digital age drives an increasing volume of public conversations onto online platforms, the demand to assess their quality through the previously mentioned dimensions in an automated, scalable manner is growing (Diakopoulos, 2015; Beauchamp, 2020).

Previous efforts have demonstrated the potential of using Natural Language Processing (NLP) and machine learning algorithms to automatically identify features of deliberation such as argumentative structure, emotional tone, and engagement patterns (Lawrence and Reed, 2020; Acheampong et al., 2020; Shin and Rask, 2021). The interest in automating such assessments, with projects like the one implemented by Falk and Lapesa (2023a) in their examination of argument and deliberative quality with adapter models (Houlsby et al., 2019), is growing.

Motivated by this research, this study introduces AQuA, an index to measure the deliberative quality of individual comments in online discussions with a single score. While there is an ongoing debate on the usefulness of aggregating multiple indices of deliberation (Bächtiger et al., 2022), we argue that for some tasks a single value, composed of several theoretically based criteria is favorable. Our approach combines predictions on various dimensions of deliberation with insights gained from both expert and non-expert evaluations, resulting in a single deliberative quality score. We make use of data that has been annotated from both trained experts and crowd annotators, representing the non-experts' view.

We calculate correlation coefficients between the annotated deliberative quality criteria and the perceived deliberativeness of the comments to attribute importance to each individual criterion.

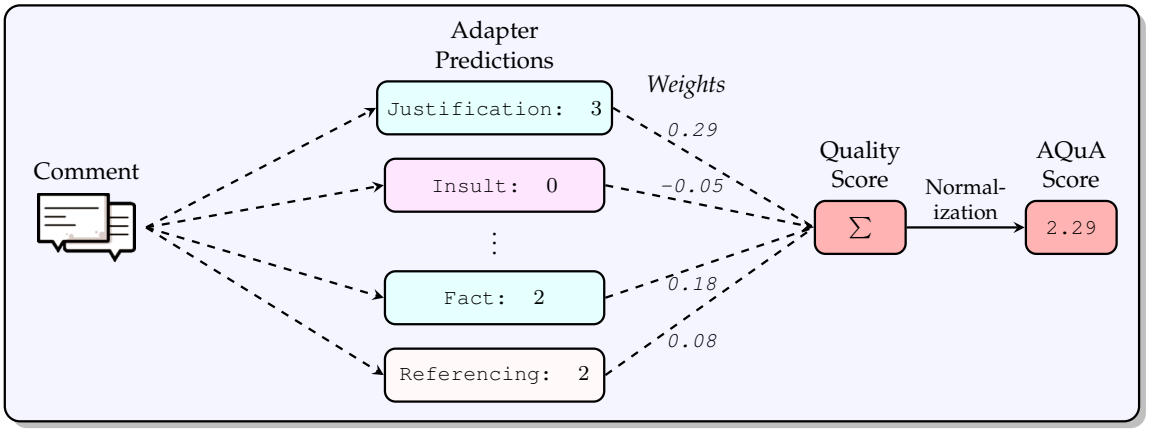


Figure 12: AQuA calculates a single score for deliberativeness from weighted adapter predictions on 20 different deliberative aspects. The adapter predictions are weighted by the correlation coefficients between each deliberative aspect and the perception of crowd workers about whether a comment is deliberative or not. The normalized score can then be used to compare the deliberative quality of individual comments.

Our contributions:

1. We train 20 adapter models on aspects of deliberation to form the basis for a single deliberation score.
2. To combine the automated predictions in a meaningful way, we calculate the correlation coefficients between experts' and non-experts' assessments of deliberative quality.
3. We define a single normalized score using the correlations as weights, hereby, creating an interpretable and explainable measure for deliberative quality.
4. Finally, we show in experiments that our score can automatically assess the deliberative quality of discussion comments.

Our method consists of two components: (1) the utilization of adapters trained on discrete facets of deliberation, and (2) the integration of correlations between annotations from experts and non-experts to establish a normalized score for deliberative quality. In developing this index, we extensively test and evaluate its effectiveness across diverse datasets, demonstrating its utility in real-world applications. By doing so, we aim to contribute to the burgeoning field of computational social science, offering scholars, policymakers, and practitioners a tool to monitor and analyze public dialogues. Our trained adapter weights and the code for calculating AQuA scores are available under `https://github.com/mabehrendt/\acs{AQuA}`.

7.3 Related Work

Before explaining our approach in detail, we give an overview on the previous work to quantify aspects of deliberation in online discussions and the adapter approach to efficiently train language models for downstream tasks.

7.3.1 Deliberative Quality Indices

Various attempts have been made in the literature to conceptualize deliberation aspects to assess the quality of discourse. Here, we provide a summary of key indicators and metrics proposed in this domain.

The *Deliberative Quality Index (DQI)*, introduced by Steenbergen et al. (2003) and further refined by Bächtiger et al. (2022), is a prominent and frequently applied metric for evaluating deliberative quality. The DQI comprises five dimensions: *equality of participation*, *level of justification*, *content of justification*, *respect*, and *constructive politics*. These dimensions are assessed for each contribution and averaged for a single speaker.

Scudder's (2022) *Listening Quality Index (LQI)* emphasizes deliberative listening as a crucial factor in communication quality, organizing elements of existing measures into a hierarchical scale. This scale progresses from minimal listening to a stage where the speaker feels acknowledged, emphasizing the sequential fulfillment of criteria. The LQI differentiates between speakers and listeners, considering not just the contributions to the dialogue but also the participants' behavior and their feeling of being heard.

The *Deliberative Reason Index (DRI)* by Niemeyer et al. (2024) seeks to capture deliberative quality at the group reasoning level rather than evaluating individual contributions. This approach, akin to the LQI, employs surveys conducted before and after discussions to gauge participants' views and preferences on debated topics, calculating agreement scores that are then aggregated to a group score.

Although referred to as indices, the discussed methodologies do not necessarily provide a single index. They often yield multiple metrics rather than a singular measure, demanding a comprehensive evaluation to determine the overall quality of contributions or debates. Friess et al. (2021) suggest aggregating the presence of deliberative qualities — rationality, respect, reciprocity, and civility — and computing their average to establish a quality ratio, treating each criterion with equal importance. We argue, however, that certain aspects may be more important than others to estimate the deliberative quality of a contribution (Chen, 2017).

While the indices presented are valuable for in-depth political debate analysis, their application requires extensive effort from trained coders for annotation and reliability assessments. To streamline the analysis of the deliberative quality of online discussions, several automation proposals have emerged. For instance, Wyss et al. (2015) employ cognitive complexity to analyze Swiss parliamentary debates, using indicators derived from the Linguistic Inquiry and Word Count (LIWC) dictionary (Tausczik and Pennebaker,

2010). Gold et al. (2015) automate the measurement and annotation of features like participation and justification, subsequently employing a visual analytics system for data representation. Fournier-Tombs and Di Marzo Serugendo (2020) introduced DelibAnalysis, a framework for predicting the DQI of online discussion contributions through machine learning, while Shin and Rask (2021) proposed leveraging network and time-series analyzes to assess deliberation criteria automatically.

Our proposed method seeks to bridge the gap between NLP techniques and the theoretical aspects of deliberative quality assessment. We introduce the AQuA score to (i) combine the theoretical underpinnings of deliberation with the comment quality in online debates as perceived by non-experts, and thereby (ii) offering a tool to quantify deliberation aspects through advanced deep learning methods.

7.3.2 Adapters

Adapters, as introduced by Rebuffi et al. (2017) are an efficient approach to customize pre-trained language models like RoBERTa (Liu et al., 2019) for specific tasks. This method involves the integration of additional bottleneck layers into the model for each distinct task, which adds new weights while leaving the original pre-trained weights unaltered.

The concept of adapter layers was first applied to NLP by Houlsby et al. (2019), who adapted the Transformer architecture (Vaswani et al., 2017) to include these layers. The design of the adapter involves compressing the input's dimensionality to a significantly smaller size, applying a non-linear function, and incorporating a skip-connection to circumvent the bottleneck, with task-specific layer normalization parameters also being adjustable.

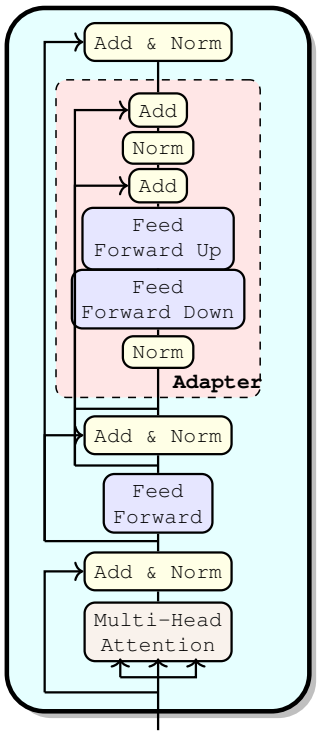


Figure 13: For the individual adapter predictions, we use a Transformer based model with adapter layers inserted after the feed forward layer of the Transformer as proposed by Pfeiffer et al. (2021).

The strategic insertion of adapter layers has been a focus of research, with Houlsby et al. (2019) positioning them subsequent to both the multi-head attention and feed-forward layers within the Transformer architecture. Pfeiffer et al. (2021) found in an extensive search on architectural parameters, that placing only one adapter after the feed forward layer in the Transformer works best throughout all their experiments. We also apply this architecture for our models. The introduction of AdapterHub by Pfeiffer et al. (2020) and the adapters library by Poth et al. (2023) further facilitated the sharing and reuse of pre-trained adapters within the community.

Subsequent studies, such as those by Mendonca et al. (2022), explored the training of individual adapters for dialogue quality estimation, and the use of AdapterFusion (Pfeiffer et al., 2021) to merge features from different adapters. Falk and Lapesa (2023a) trained 20 adapters on features for argument and deliberative quality to examine their dependencies. In our work, we follow a similar path to train adapters to evaluate specific aspects of deliberative quality and subsequently combine them using correlation coefficients between experts' and non-experts' annotations, to create a single deliberative quality metric.

7.4 An Additive Score for Deliberative Quality

With AQuA we propose a metric for assessing the quality of individual comments in online discussions. Our approach combines predictions on various dimensions of deliberation with insights gained from both experts' and non-experts' evaluations, resulting in a single deliberative quality score. Our methodology consists of two components: (1) the utilization of adapters trained on discrete facets of deliberation, and (2) the integration of correlations between experts' and non-experts' annotations to establish a normalized score for deliberative quality. We therefore harness annotations of the same data, once labeled by trained experts for a variety of deliberative qualities, such as the degree of justification, and once labeled by non-experts on their personal assessment of the deliberativeness of a comment. We calculate correlation coefficients between each individual deliberative criterion (experts' labels) and the binary indicator for deliberativeness (non-experts' labels).

The idea of our approach is to aggregate individual scores calculated by adapters in a meaningful way to obtain a single score for each comment, in which some aspects contribute more to the perceived deliberativeness than others. For this reason we call our approach AQuA, an "Additive deliberative Quality score with Adapters".

7.4.1 Datasets

Our analysis is based on three datasets:

1. The KODIE dataset, comprising 13,587 comments that were collected and annotated as part of a scientific study that explored the impact of news organizations' interactive moderation on the deliberative quality of users' political discussions (Heinbach et al., 2022). The comments were posted on the Facebook pages of four German national and regional news outlets with high outreach and diverse audiences. These news outlets delivered data that included all published and deleted/hidden posts and comments on their Facebook pages for a period of 12 weeks per news outlet.

2. The #meinfernsehen2021 (German for my television) dataset (Gerlach and Eilders, 2022) is the result of a large scale citizen participation on the future of public televi-

	Adapter	Description	Weight
Rationality	**Relevance**	**Does the comment have a relevance for the discussed topic?**	**0.20908452**
	Fact	**Is there at least one fact claiming statement in the comment?**	**0.18285757**
	Opinion	**Is there a subjective statement made in the comment?**	**-0.11069402**
	Justification	**Is at least one statement justified in the comment?**	**0.29000763**
	Solution Proposals	**Does the comment contain a proposal how an issue could be solved?**	**0.39535126**
	Additional Knowledge	**Does the comment contain additional knowledge?**	**0.14655912**
	Question	Does the comment include a true, i.e., non-rhetoric question?	-0.07331445
Reciprocity	Referencing Users	Does the comment refer to at least one other user or to all users in the community?	-0.03768367
	Referencing Medium	Does the comment refer to the medium, the editorial team or the moderation team?	0.07019062
	Referencing Contents	Does the comment refer to content, arguments or positions in other comments?	-0.02847408
	Referencing Personal	**Does the comment refer to the person or personal characteristics of other users?**	**0.21126469**
	Referencing Format	Does the comment refer to the tone, language, spelling or other formal criteria other comments?	-0.02674237
Civility	Polite form of Address	Does the comment contain welcome or farewell phrases?	0.01482095
	Respect	Does the comment contain expressions of respect or thankfulness?	0.00732909
	Screaming	Does the comment contain clusters of punctuation or capitalization intended to imply screaming?	-0.01900971
	Vulgar	Does the comment contain language that is inappropriate for civil discourse?	-0.04995486
	Insult	Does the comment contain insults towards one or more people?	-0.05884586
	Sarcasm	**Does the comment contain biting mockery aimed at devaluing the reference object?**	**-0.15170863**
	Discrimination	Does the comment explicitly or implicitly contain unfair treatment of groups or individuals?	0.02934227
	Storytelling	**Does the commenter include personal stories or personal experiences?**	**0.10628146**

Table 18: **Correlation weights w_k of all 20 trained deliberative quality adapters.** The weights are calculated as the correlation coefficients between the experts' annotations and non-experts' ones. The most important indicators for a high quality comment are marked in bold. Note that positive correlations correspond to a positive trait in a high quality comment, while negative correlations correspond to negative traits.

sion in Germany. Overall, 1,714 comments from the participation process have been manually coded as part of a quantitative content analysis to examine the discussion quality.

3. The CrowdAnno project Wilms et al. (2023) collected a non-expert representation of deliberative quality via crowd annotations for a subset of, i.a., both the KODIE and #meinfernsehen datasets.

The annotations from two different perspectives are explained in the following.

KODIE & #meinfernsehen - the Experts' View. The KODIE annotation framework (Heinbach et al., 2022), assigns 23 score-based deliberative and further labels on other aspects to each comment. These annotations were conducted by trained coders with a scientific background, focusing on deliberative criteria such as fact claims, relevance to the discussion topic, and respectful engagement with other users. The deliberative criteria can each be assigned to one of the three main dimensions of deliberation (Bächtiger et al., 2009b; Esau et al., 2021; Graham, 2010; Coe et al., 2014; Papacharissi, 2004):

Rationality, measured by indicators such as reasoning, solution proposals, and provision of additional knowledge.

Reciprocity, measured as mutual references between users within a discussion.

Civility, measured as the presence of a respectful interaction with others and the absence of insults, pejorative speech, and other markers of disrespect.

The following coding scheme was used: all categories were coded on a four-point scale from "clearly not present" to "clearly present". Intercoder reliability was tested on a subset of 130 comments and exceeded the critical threshold of Krippendorff's α of .67 for all categories (Ø = .83). The #meinfernsehen data is annotated with the same scheme as KODIE. For #meinfernsehen intercoder reliability was tested on 159 comments, exceeding the critical threshold of Krippendorff's α of .67 for 20 out of 21 categories (Ø = .74).

We selected 19 out of the 23 deliberative quality criteria to train adapters, since some annotated aspects, e.g., *threat of violence* were not found in the data. In addition to the deliberative quality criteria, we included *storytelling*, which is considered a type II deliberation criterion, according to Bächtiger et al. (2009a), since the description of personal experience when suggesting a solution contributes to the perceived quality of a comment (Falk and Lapesa, 2023b). The 20 deliberative aspects that we use are listed in Table 18. After filtering out data points with missing annotations and coding errors, we were left with a total of 13,069 comments to train our adapter models. In the following we will write

$$s_k(i) \in \{0, 1, 2, 3\} \tag{36}$$

for the k-th score ($1 \leq k \leq 20$) of the i-th comment ($1 \leq i \leq 13,069$).

CrowdAnno - the Non-Experts' View. In the CrowdAnno project, Wilms et al. (2023) gathered data on non-experts' perception of uncivil, deliberative, and fact-claiming communication within German online comments through crowd annotation. The dataset includes 13,677 comments from different news media comment sections and online citizen participation projects, annotated by 681 crowdworkers. For AQuA, we used a subset of 1,742 comments that are identical to the KODIE and #meinfernsehen data. Crowd workers were tasked with evaluating, whether a comment is perceived as enriching and value-adding to the discussion or not, i.e., marking if it contains enriching communication, which could serve as a proxy for deliberative quality. The final score is aggregated from evaluations by 9 different crowd annotators via majority vote. To minimize annotator bias, the crowd workers were sampled to reflect various sociodemographic and educational backgrounds. We will write

$$c(i) \in \{0, 1\} \tag{37}$$

for the binary deliberativeness label of the i-th comment.

7.4.2 Training the Adapters

To automatically predict the various deliberation criteria, we use pre-trained language models, such as BERT (Devlin et al., 2019). We follow the adapter approach: adapters are extra weights θ_k, that are plugged into pre-trained language models and then learned for a specific task k. The adapted language model for the k-th deliberation criterion is written as $f_{\theta_k}(x)$, where x is some text input. Note that while learning these extra weights, we do

not alter the pre-trained model weights. More precisely, we used the adapter architecture proposed by Pfeiffer et al. (2021), which is shown in Figure 13. We trained 20 individual adapters to predict scores $f_{\theta_k}(x)$ for individual indicators for deliberative quality in user comments for the KODIE dataset. For training we perform a 65% (train), 15% (val), 20% (test) split on our dataset, resulting in 8,495 training data points, 1,960 for validation and 2,614 for testing. Each of the 20 adapters for AQuA is trained with a multi-label classification objective, minimizing the cross entropy loss. We train each adapter for 10 epochs and save the model with the best macro F1 score.

7.4.3 Calculating the Weights

Assigning an importance to the individual quality dimensions for the overall quality measurement is not a simple task. Our intuition for weighting the deliberative criteria is to include the perception of people who potentially read and write these comments. For that reason we linked the scientific theory of deliberation to the view of non-scientists by combining the datasets described in detail in Section 7.4.1. More precisely, we obtain the weight for each deliberative criterion k by calculating the correlation coefficient,

$$w_k = \frac{\sum_{i=1}^{N}(s_k(i) - \bar{s}_k)(c(i) - \bar{c})}{\sqrt{\sum_{i=1}^{N}(s_k(i) - \bar{s}_k)^2}\sqrt{\sum_{i=1}^{N}(c(i) - \bar{c})^2}}, \tag{38}$$

between the scientific label $s_k(i)$ (with mean $\bar{s}_k$) for each of the $K = 20$ aspects of deliberation and the perception of crowd workers on the comments deliberativeness $c(i)$ (with mean $\bar{c}$) for all N comments. Note that w_k is a value from the interval between -1 and 1.

7.4.4 Building the AQuA Score

We build an overall quality score $s(x)$ for each comment as the weighted sum of the weights w_k and the predicted score $f_{\theta_k}(x)$ for each of the $K = 20$ quality adapters:

$$s(x) = \sum_{k=1}^{K} w_k f_{\theta_k}(x). \tag{39}$$

The highest and lowest possible scores depend on the number K of criteria and on the range of the predictions $f_{\theta_k}(x)$. Since the labels from KODIE are from the set $\{0, 1, 2, 3\}$, the predictions are also from this set. The highest possible score can be reached by setting all positively weighted criteria to their maximum value (i.e, 3) and all negatively weighted criteria to their minimum value (i.e, 0),

$$s_{\max} = \sum_{k=1}^{K} 3 \cdot w_k \cdot [w_k \geq 0] \approx 4.9893, \tag{40}$$

where $[w_k \geq 0] = 1$ if $w_k \geq 0$ and zero otherwise. Similarly, the smallest possible score is

$$s_{\text{min}} = \sum_{k=1}^{K} 3 \cdot w_k \cdot [w_k \leq 0] \approx -1.6693. \tag{41}$$

To get a more intuitive range of values, we scale $s(x)$ to an interval between 0 and 5:

$$s_{\text{AQuA}}(x) = 5 \cdot \frac{(s(x) - s_{\text{min}})}{(s_{\text{max}} - s_{\text{min}})}, \tag{42}$$

which is the definition of our proposed AQuA score. Figure 12 graphically illustrates, how the AQuA score is calculated for a given input comment.

		German BERT	Multilingual BERT cased	Multilingual BERT uncased
Rationality	Relevance	**0.39**	0.37	0.37
	Fact	**0.58**	0.56	0.54
	Opinion	**0.59**	0.57	0.5
	Justification	**0.7**	0.69	0.67
	Solution Proposals	0.77	**0.79**	0.76
	Additional Knowledge	0.71	**0.78**	0.74
	Question	0.84	**0.87**	**0.87**
Reciprocity	Referencing Users	0.86	**0.88**	0.87
	Referencing Medium	0.92	0.93	**0.94**
	Referencing Contents	0.7	**0.81**	0.8
	Referencing Personal	0.83	**0.92**	**0.92**
	Referencing Format	0.89	**0.96**	**0.96**
Civility	Polite form of Address	0.96	0.97	**0.98**
	Respect	0.81	0.9	**0.91**
	Screaming	0.77	**0.81**	0.79
	Vulgar	0.76	0.74	**0.86**
	Insults	0.87	0.87	0.87
	Sarcasm	**0.48**	**0.48**	0.34
	Discrimination	0.83	**0.88**	0.87
	Storytelling	0.83	0.85	**0.86**
	Ø Total Average (F1-Score)	0.7545	**0.7815**	0.771

Table 19: **Base models.** We analyze the performance of different base models with adapter training on the 20 deliberative aspects. We show the weighted average F1 score. Overall, the multilingual BERT cased model performs best on the KODIE test dataset. We therefore use multilingual BERT as a base model for the AQuA score.

7.4.5 Applying the Score to English Comments

To apply our method to English datasets, we used the `wmt19-en-de-model`[21] (Ng et al., 2019), to automatically translate all comments in the examined dataset from English to German. Another alternative would be to train adapter models on English data. Since the KODIE dataset consists of German Facebook comments on political issues, discussing

[21] `https://huggingface.co/facebook/wmt19-en-de`

German politicians as well, we decided not to translate these comments to train adapter models, but to translate English comments and use the pre-trained German models for evaluation.

7.5 Analysis and Experiments

After defining the AQuA score in the previous sections, we briefly discuss the choice of our base model and then analyze the weights that we calculated for the individual adapter predictions. Finally, we conduct several experiments to show that our model can successfully predict deliberative quality in user comments.

	Label	Frequency			
		0	1	2	3
Rationality	Relevance	130	200	345	1065
	Fact	1155	113	155	317
	Opinion	27	15	13	123
	Justification	1177	78	139	346
	Solution Proposals	932	400	281	127
	Additional Knowledge	1524	76	91	48
	Question	1590	55	45	50
Reciprocity	Referencing Users	1164	128	62	386
	Referencing Medium	173	1	1	3
	Referencing Contents	1142	98	119	381
	Referencing Personal	177	1	0	0
	Referencing Format	177	0	0	1
Civility	Polite form of Address	1725	3	6	6
	Respect	1572	25	100	43
	Screaming	1612	30	53	45
	Vulgar	1654	44	23	19
	Insults	1670	29	21	20
	Sarcasm	1327	115	130	168
	Discrimination	170	2	1	5
	Storytelling	1617	59	46	18

Table 20: **CrowdAnno.** Absolute frequencies of each label in the subset of the CrowAnno dataset, used to calculate the correlation coefficients.

7.5.1 Choice of the Base Model

The correlation coefficients are one important part that affect the composition of AQuA. The other part are the predictions of each of the 20 trained adapters. The adapter weights can be trained with different base architectures. To determine which base model performs best, we examine the performance of different models, namely German BERT Base cased (Chan et al., 2020) and multilingual BERT (Devlin et al., 2019) in the cased and uncased variants, on the KODIE test split. The training procedure is the same as described in Section 7.4.2. The results are shown in Table 19. As the datasets are highly imbalanced, and some deliberative qualities do not occur often in the training data, we report the weighted averaged F1 score, i.e., a global weighted average F1 score for each class. The trained adapter weights with the multilingual BERT model as base model outperform the German BERT model on 15 out of the 20 tasks. In direct comparison, the cased variant of Multilingual BERT performs slightly better than the uncased one. Based on these results we take the multilingual BERT Base cased model[22] as our base model for calculating the AQuA score.

[22] https://huggingface.co/bert-base-multilingual-cased

7.5.2 Insights from the Correlations

The calculated correlation coefficients serve as weights in AQuA to give more importance to some deliberative aspects than others. Besides their values determining the importance for each criterion, the sign of the correlation coefficient reveals if an aspect is positively or negatively associated with comment quality. In the following, we discuss the coefficients and examine whether findings from previous deliberative research are consistent with our results. The coefficients with large absolute values are marked bold in Table 18.

	Adapter	F1 Score
Rationality	Relevance	13.22
	Fact	18.48
	Opinion	42.93
	Justification	29.49
	Solution Proposals	56.04
	Additional Knowledge	38.97
	Question	62.25
Reciprocity	Referencing Users	66.85
	Referencing Medium	69.23
	Referencing Contents	66.28
	Referencing Personal	70.40
	Referencing Format	70.40
Civility	Polite form of Address	69.89
	Respect	69.67
	Screaming	67.96
	Vulgar	65.64
	Insults	70.40
	Sarcasm	66.12
	Discrimination	65.84
	Storytelling	65.33

Table 21: **SOCC.** Adapters that align with toxicity reach a high weighted average F1 score with toxicity levels from the SOCC dataset.

For an overview of the data distribution, Table 20 lists the absolute frequencies of each label for each deliberative quality criteria in the subset of the KODIE and #meinfernsehen datasets that have been annotated using the CrowdAnno framework. These points were used to calculate the correlation coefficients. Note that these are not the frequencies in the dataset used for training the adapters. However, the small subset reflects the class imbalance that is present in the data, indicating that some categories such as vulgar language, insults and even storytelling do not occur often.

It is striking that nearly all indicators for *rationality* are strongly positively correlated with non-experts' perceived deliberative quality of comments. Using well-reasoned arguments that are relevant to the topic has been found to be an important aspect in distinguishing between comments of high and low deliberative quality (Diakopoulos, 2015; Kolhatkar et al., 2020). Unfounded expressions of opinion, on the other hand, are perceived as non-constructive, i.e., negative, in user comments. Our results support that finding, as opinion is highly negatively correlated with the perceived deliberative quality.

Of all the indicators of *reciprocity*, referring to personal characteristics of others has the greatest positive impact on the overall score. This is surprising as deliberative literature primarily highlights engaging with others' positions, not their personal traits, as a quality indicator (e.g., Ziegele et al., 2020).

Within the *civility* criteria, sarcasm stands out with a rather high negative correlation coefficient. Sarcasm, as well as doubting, criticism, and insults have been identified as one form of expressing disrespect towards other participants (Bender et al., 2011). The

large correlation weight for sarcasm is a stable finding, since it is more frequent in the KODIE data, in contrast to insults.

While not being a central aspect of deliberation, storytelling in form of personal anecdotes can foster empathy and mutual understanding between participants and resolve differences (Black, 2008). Thus, it is reasonable that *storytelling* plays an important role in the weighting of AQuA, as well.

7.5.3 Evaluating the Score

Having trained the AQuA score using the KODIE, #meinfernsehen and CrowdAnno datasets, we next show that the learned adapter weights and correlations transfer to other datasets as well and give scores that are qualitatively and also quantitatively convincing.

SFU Opinion and Comments Corpus. We predict AQuA scores on comments of the SFU opinion and comment corpus (SOCC) (Kolhatkar et al., 2020). The dataset includes 1,121 comments on news articles that have been annotated for *constructiveness* (binary annotations) and *toxicity* (four point scale from not toxic to very toxic). According to Kolhatkar et al. (2020), constructive comments are required "to create a civil dialogue through remarks that are relevant to the article and not intended to merely provoke an emotional response".

We calculate AQuA scores and use them to predict the binary constructive label for each comment in the SOCC. Choosing a threshold of 2.3, i.e., inferring $\hat{y}_{\text{constructive}} = 1$, if $s_{\text{AQuA}} \geq 2.3$, we get an F1 score of 81.73. Note that the threshold is a hyperparameter and a value of 2.3 was chosen, because with performed best on the data. As the dataset also comprises labels for toxic comments, we use the individual adapter predictions for *screaming*, *vulgar*, *insults*, *sarcasm*, and *discrimination* to predict the level of toxicity for each comment. Both the SOCC labels y_{toxic} as well as our predictions $s_k(i)$ are numbers from 0 to 3, therefore we simply use the individual predictions of each adapter as an indicator for the toxicity level and calculate the weighted average F1 score.

With 829 comments labeled as not toxic at all (label 0), 172 with label 1, 35 with label 2 and only 7 comments that are marked as clearly toxic (label 3), the distribution is very similar to the one we see in the datasets we used for AQuA. Table 21 shows that we reach good F1 scores for adapters that align with toxicity.

Europolis. For a qualitative analysis of the AQuA score, we apply it to the Europolis dataset (Gerber et al., 2018). Europolis includes transcribed speech contributions of a deliberative poll on migration and climate change, annotated for *interactivity*, *respect*, *storytelling*, *justification* and *common good*. We calculate AQuA scores for each contribution in the dataset and report the top 3 highest and lowest ranked comments in Table 22. For interpretability, we list both the predicted labels of the individual adapters and the original Europolis labels (in both cases only for values greater than 0). While both differ, the

Top 3 Comments from Europolis			
Comment	**Europolis Labels**	**Adapter Predictions**	**Score**
The problem with the whole story is that first of all the cost of living has to be equalized - that includes, of course, wages, or salaries. If that - I assume we are only Poles and Germans here - and an Austrian, excuse me Julian - that we, I think, as I have come to know it - I have just said, we have a twin town in Poland - the cost of living was at least two years ago in Poland much lower than in Germany and then of course higher wages have to be paid here, so that you can buy the piece of bread, which is correspondingly lower in Poland and that's why Frankfurt/Oder to the other side is a constant border traffic. Buying gas in Poland is just much cheaper than in Frankfurt/Oder on the border. So the problem is simply that the cost of living in the individual states is so different that you can't equate it with wages and salaries at all.	*interact.: 2, respect: 1, storytelling: 1, justification: 2*	*rel.: 3, fact: 3, opinion: 3, justification: 3, suggest. sol.: 3, additional know.: 3, storytelling: 3*	4.0005
Financial problems always existed in different countries. If someone wants to live in another country, he can always do so. So if he/she wants to work a few years in some country in order to send the family money that he/she earned, he/she should not be prevented from doing so.	*interact.: 3, respect: 1, justification: 3, common good: 2*	*rel.: 3, fact: 3, justification: 3, suggest. sol.: 3, additional know.: 2, storytelling: 1*	3.9803
Many people are coming to other countries not just because of economic reasons. Often, they are persecuted in their own countries on the religious grounds and they are trying to find asylum in another country. Then, the government should give them political asylum, papers or right of permanent residency and then they can work. For example Germany is rich enough to give jobs for immigrants and integrate them in the society because the society is aging and somebody has to work for the new generation which would like to get future pensions or something like that. Society is aging so they need immigrants. Similar to Poland where the government should legalize immigrants in a similar way. It is hard to say how it actually should look like.	*respect: 2, justification: 2, common good: 1*	*rel.: 3, fact: 3, suggest. sol.: 3, additional know.: 2., justification: 3, discrim.: 3*	3.9666
Lowest 3 Comments from Europolis			
Comment	**Europolis Labels**	**Adapter Predictions**	**Score**
A question for Udo: To what dimension is the problem with the migration of workers growing?	*interact.: 2, respect: 1*	*question: 3, ref. user: 3, ref. content: 3*	0.9393
Thank you very much. Aurore, you also wanted to say something especially before the break but now too?	*interact.: 2, respect: 1, storytelling: 1, justification: 2, common good: 2*	*fact: 1, question: 3, ref. user: 3, ref. content: 3, polite addr.: 2, sarcasm: 1*	0.9849
To tell you the truth, I do not know what is discussed? Are we talking about the quotas – how many people could come here?	*respect: 1, storytelling: 1, justification: 1, common good: 1*	*question: 3, ref. user: 3*	1.0034

Table 22: Europolis. Top 3 comments with the highest and top 3 comments with the lowest calculated AQuA scores. We only show the scores and the predicted labels of the individual adapters where the prediction is larger than zero. The original labels (from Europolis, 5 labels) show that the AQuA score is well aligned with the original labels.

AQuA labels approximately match the original Europolis labels. The top 3 comments are all rated highly with positive deliberative aspects such as storytelling, justification and additional knowledge, while the lowest comments exhibit negative deliberative aspects such as sarcasm and references to other participants. Overall, all of the the lowest scored comments are questions to clarify certain aspects in the discussion, whereas the higher scored comments consist of sophisticated opinions.

When comparing the AQuA predictions to the original Europolis labels, we find that the AQuA score seems consistent with the original labels, while enhancing the prediction since the AQuA score consists of 20 deliberative aspects instead of the 5. This demon-

strates the value of AQuA as a unified score that can be applied to any dataset based on the chosen deliberative aspects.

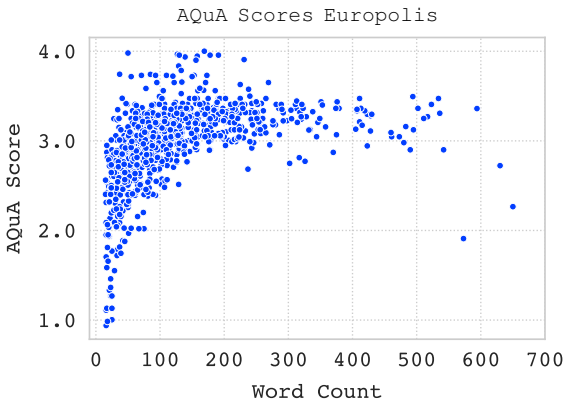


Figure 14: **Europolis.** AQuA scores (y-axis) vs the comment length (x-axis, word count) rule out that comment length alone is a factor for a high AQuA score.

Does comment length matter? An interesting observation is that the lowest ranked comments in the dataset are much shorter than the high ranked ones. To study whether comment length alone is the most important factor that causes our model to predict a large score, we take a closer look at the distribution of scores depending on the length of the comment. Figure 14 displays the AQuA score (y-axis) in comparison to the comment length (x-axis, word count). While it is true that short comments get the lowest scores, which is probably due to the fact that they do not have much content, the visual analysis reveals also that medium length comments get the highest scores. This rules out that comment length is the most relevant factor for our score.

7.6 Conclusion

In this work we introduce AQuA, an approach for an automated deliberative quality score based on large language models and adapters. The score combines annotations of experts and the view of non-experts on real online discussion comments.

We show that the trained adapters are capable of predicting individual scores for different aspects of deliberative quality and that the overall score aggregates these predictions in a meaningful way. The correlation coefficients between experts' and non-experts' annotations reveal the most important positive and negative deliberative aspects, which allows us to confirm theoretical and empirical findings in deliberation literature into AQuA.

Furthermore, we evaluate our score (trained on KODIE and CrowdAnno) on two further datasets (SOCC and Europolis) to show that the predictions of the learned adapters transfer well to unseen datasets. First, we show that the adapter predictions that build the AQuA score are useful for classifying constructive and toxic comments on the SOCC dataset. Then we perform a qualitative analysis of the AQuA score by manual assessing the top 3 and bottom 3 scored comments in the Europolis dataset and show that comments with well formed opinions receive large scores, while comments providing little value to the discussion receive lower scores.

Overall, we show that AQuA can be used successfully to automatically assess deliberative quality while aligning with theoretical and empirical background in deliberation literature.

8 Supporting Online Discussions: Integrating AI Into the adhocracy+ Participation Platform To Enhance Deliberation

Corresponding Publication. This chapter is based on the work:

Supporting Online Discussions: Integrating AI Into the adhocracy+ Participation Platform To Enhance Deliberation
Maike Behrendt, Stefan Sylvius Wagner, Mira Warne, Jana Leonie Peters, Marc Ziegele & Stefan Harmeling
In Proceedings of the Fourth Workshop on Bridging Human-Computer Interaction and Natural Language Processing (HCI + NLP)

Personal Contributions. This work introduces the platform that we expanded with AI features in order to examine their effects in real-time online discussions. Maike Behrendt and Stefan Sylvius Wagner implemented the AI-supported discussion modules, based on the open source platform adhocracy+, and the experimental infrastructure in equal parts. Maike Behrendt was primarily responsible for drafting the initial paper, conducting a literature search for the related work section, and writing the sections on the Deliberative Quality Module. Stefan Sylvius Wagner contributed the figures and wrote the section describing the Comment Recommendation Module of the platform. Marc Ziegele, Jana Leonie Peters and Mira Warne were responsible for carrying out and describing the experiments and evaluating their results. All authors collaborated on the revision and finalization.

Abstract. Online spaces provide individuals with the opportunity to engage in discussions on important topics and make collective decisions, regardless of their geographic location or time zone. However, without adequate support and careful design, such discussions often suffer from a lack of structure and civility in the exchange of opinions. Artificial Intelligence (AI) offers a promising avenue for helping both participants and organizers in managing large-scale online participation processes. This paper introduces an extension of adhocracy+, a large-scale open-source participation platform. Our extension features two AI-supported debate modules designed to improve discussion quality and foster participant interaction. In a large-scale user study we examined the effects and

usability of both modules. We report our findings in this paper. The extended platform is available at `https://github.com/mabehrendt/discuss2.0`.

Research Problem & Motivation. Within the UPEKI project, we examined the impact of two different AI-based methods on online discussions. We developed AQuA (Behrendt et al., 2024), a score for measuring comment quality (see Chapter 7), and a stance detection algorithm (Wagner et al., 2025) to support a simulated citizen participation process. In this work, we present the platform we expanded for our experiments. The platform adhocracy+ already included a discussion feature, which we expanded to include the two AI-based modules. The platform is available as open source and can be used free of charge.

Main Results. The platform study provides empirical evidence of the effects of AI-assisted interventions in a simulated citizen participation process. The integration of the deliberative quality module and a comment recommendation system influenced discussion dynamics and user interaction patterns. The findings offer insights into user perceptions of AI support, including perceived fairness and usefulness, and demonstrate both the potential and limitations of AI-driven moderation and guidance in real-time online discussions. These results contribute to understanding how NLP systems can be responsibly embedded into deliberative participation environments.

Publication Status. Published. This paper was accepted for the fourth Bridging Human-Computer Interaction and Natural Language Processing (HCI+NLP) workshop as part of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP 2025), held in Suzhou, China (Behrendt et al., 2025b).

8.1 Preamble

Building on the previous chapter, which introduced the AQuA score as a framework for assessing deliberative quality, we now turn to the practical integration of AI-based methods into real-world participation systems.

While automatic assessment is a crucial prerequisite, the question remains whether such methods can effectively support users in practice and lead to measurable improvements in online discussions. To address this, we integrate NLP-based intervention modules into an existing online participation platform and evaluate their effects in a large-scale experimental setting.

This chapter therefore focuses on the deployment and empirical evaluation of AI-supported interventions. In addition to measuring their impact on discussion quality, we also examine how users perceive and interact with these systems.

The contributions presented in this chapter address Research Questions 3, 4, and 5 by investigating the practical applicability of NLP-based interventions and evaluating their effects on deliberative communication in real-world settings.

8.2 Introduction

Online discussions and participation platforms enable people to engage in socially relevant issues. However, written exchanges in online spaces are frequently marked by a lack of structure, often leading to *information overload*, making it difficult for both participants and providers to process large volumes of contributions (Arana-Catania et al., 2021a). According to Anastasiou et al. (2023), other key issues include polarization, incivility, toxic behavior, superficial content, and insufficient collaboration among participants. To address these challenges, the concept of deliberation proves particularly valuable. Deliberation is defined as the respectful and argumentative exchange of opinions aimed at making a decision. It encompasses three core dimensions: *rationality*, referring to the argumentative exchange of opinions; *civility*, which entails politeness and respect; and *reciprocity*, characterized by responsiveness and active listening (Bächtiger et al., 2009b; Esau et al., 2021; Graham, 2010).

AI presents a promising opportunity to enhance deliberation, supporting both participants and organizers in creating a more structured, respectful, and engaging environment for meaningful exchange of opinions. In this work, we propose two AI-based solutions to improve online discussions, implemented for adhocracy+, an open-source participation platform.

Our contributions:

1. **Comment Recommendation Module:** To encourage user interaction and expose participants to opposing viewpoints, we developed a comment recommendation module based on a stance detection model.

2. **Deliberative Quality Module:** To enhance user engagement and improve the quality of contributed comments, we implemented a debate module that automatically detects and highlights the most deliberative comments.

3. **Application and Evaluation:** To examine the effects of the proposed modules, we conducted a large-scale panel study (N = 1,356). The results of the user study are presented in detail in the following sections.

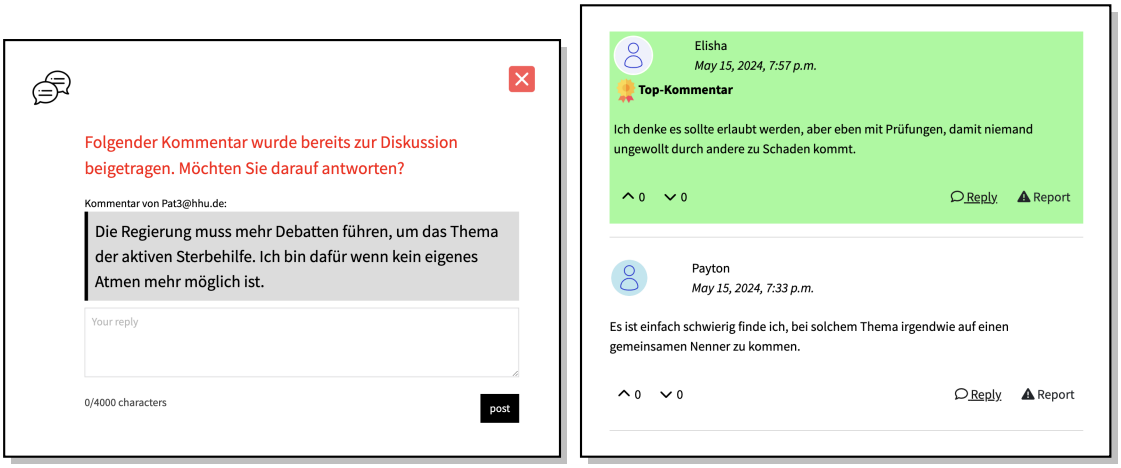


Figure 15: We propose two AI tools that we integrate into adhocracy+. **(Left) Comment Recommendation Module:** Participants are confronted with a comment that contradicts their own opinion and are asked if they want to respond. The AI tool determines the stance of the comments, which is used to propose opposing comments. Translation: The following comment has already been added to the discussion. Do you want to reply to it? **(Right) Deliberative Quality Module:** We predict a deliberative quality score (AQuA score) for each comment. Comments with a high AQuA score are sorted to the top of the discussion and highlighted in bright green and marked as "top comment".

8.3 Related Work

Previous efforts to integrate AI into discussion platforms have often focused on structuring and summarizing discussions. The CONSUL[23] citizen participation tool enables citizens to propose ideas to local politicians on improving their city. These proposals can be supported and discussed by other participants on the platform. To address the issue of *information overload*, Arana-Catania et al. (2021a) improved the platform with several Natural Language Processing (NLP) methods, including tools to summarize existing proposals, automatically categorize them and recommend proposals to participants according to their interests.

In the KOSMO project, an AI-supported moderation dashboard was developed for the adhocracy+ platform to assist moderators during citizen participation processes[24]. Two models were trained to identify uncivil and engaging comments (Risch et al., 2021), which are flagged for moderators, allowing them to decide on appropriate actions, such as blocking uncivil comments.

The BCause platform, created by Anastasiou et al. (2023), supports discussions with an automatic text summarization tool and an argument recommendation system. This system suggests arguments from scientific literature based on the user's stance on the discussed topic. Other examples of open-source discussion tools that incorporate AI features

[23] https://consulproject.nl/en/
[24] https://github.com/liqd/a4-kosmo

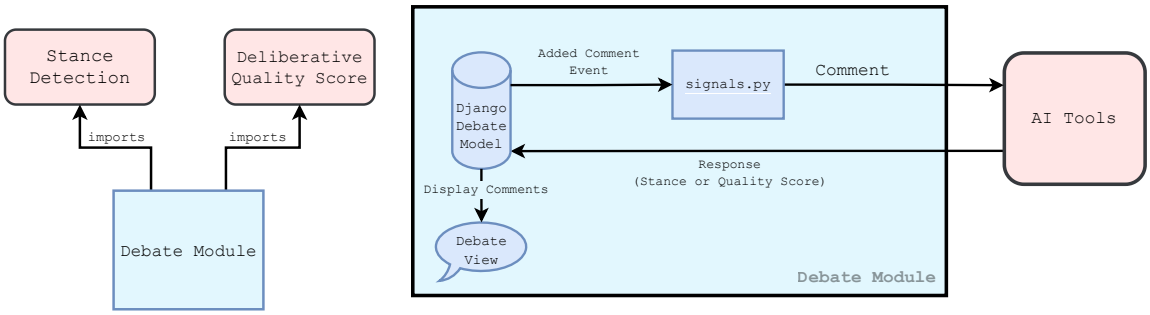


Figure 16: **Overview of the architecture to extend adhocracy+ with our AI tools. (Left)** The *debate module* imports both the *stance detection* and *deliberative quality* AIs as Python modules. **(Right)** The Django database model sends out an event when a new comment is added to the database. The event is handled in signals.py where the new comment is passed either to the stance detection or deliberative quality model. These send a response (either a stance or quality score) back to the database where the corresponding response is stored.

include Discourse[25] and Polis (Small et al., 2021) from the Computational Democracy Project.

Another notable example is CommunityPulse (Jasim et al., 2021), a platform equipped with tools for text analysis and visualization to help civic leaders to make sense of community input. It includes a sentiment analysis of contributions and topic modeling to automatically extract discussed topics.

Beyond civic tech, there is a broader body of research focused on using AI to support discussion in the context of collaborative learning (see, e.g., Kong et al. (2025)).

Similar to our approach, Yeo et al. (2024) also aim to enhance deliberative quality on online discussion platforms. They employ Large Language Models (LLMs) to generate reflective nudges designed to promote users' self-reflection, thereby fostering more thoughtful and deliberative contributions. In our work, we focus on directly enhancing the deliberative quality of discussions by improving their reciprocity and rationality. To achieve this, we introduce two new modules for the adhocracy+ platform: (i) the *Comment Recommendation Module* that suggests comments based on whether participants are *in favor* or *against* the discussed issue, encouraging participants to engage with opposing viewpoints, and (ii) the *Deliberative Quality Module* that automatically identifies and highlights the most deliberative user comments, motivating participants to contribute further high-quality comments.

8.4 Features

In the following, we will discuss the features of the two implemented modules from both a technical and a user perspective.

[25] https://meta.discourse.org/

8.4.1 Enhancing Reciprocity with the Comment Recommendation Module

As previously mentioned, large-scale online discussions often involve a high volume of postings, including redundant, toxic, or uncivil content. Simultaneously, these discussions frequently lack structure, leading participants to experience information overload (Arana-Catania et al., 2021a). Under this condition, participants struggle to follow the discussion and engage with others, which results in a lack of reciprocity within the conversation (Lago et al., 2019). Another consequence of information overload is dysfunctional argumentation (Klein, 2011). This, in turn, fosters the formation of small groups of participants who share similar opinions and avoid interacting with those holding opposing views (Klein, 2015).

To mitigate information overload, enhance reciprocity among participants, and improve the overall quality of discussions, we developed a *Comment Recommendation Module* integrated into the adhocracy+ debate module. This module suggests comments to the participants that reflect a point of view opposite to their own. For instance, if a participant holds an *against* stance on the debate question, the module will recommend a comment from another participant with an *in favor* stance.

Stance Detection. To detect the stance of a comment, we use an uncased German BERT Base model (Chan et al., 2020)[26] fine-tuned on the X-Stance dataset (Vamvas and Sennrich, 2020). This dataset includes 48.6k German comments on 150 political questions, answered by political office candidates in Switzerland. Since the adhocracy+ platform is specifically designed for discussion and decision making on politically relevant issues, the dataset fits our purpose very well. The fine-tuned model operates as a binary classifier, outputting either *in favor* or *against* based on a given debate question and a specific comment.

The complexity and diversity of political and social issues make it challenging to obtain sufficient labeled data for stance detection. To address this, we follow the approach of Wagner et al. (2025), leveraging synthetic data generated by LLMs. We employ Mistral-7B (Jiang et al., 2023) to generate comments reflecting an *in favor* or *against* stance. These synthetic comments are then used to further fine-tune the stance detection model. For existing comments, the synthetic data helps identify real comments that are most challenging for the model to classify. These comments can be manually labeled to further improve the model's performance. For additional details, see Wagner et al. (2025).

Comment Recommendation User Experience. The main purpose of the *Comment Recommendation Module* is to present a comment to the user that opposes their own position on the debate question. Therefore, the stance of every comment, posted in the discussion is predicted and stored into the database. When a user logs into the platform for the first time, they are prompted to indicate their stance on the debate question.

[26] `https://huggingface.co/dbmdz/bert-base-german-uncased`

The user's position, which can be either *in favor* or *against*, is stored in the database. This information is then used to determine suitable comments for recommendation. The system retrieves comments from the database that oppose the user's stance. If multiple opposing comments are available, one is randomly selected from the list. If there is no suitable comment available, a message is displayed to the user, indicating that no comment can be suggested at that time.

The selected comment is displayed to the user in a popup window (see Figure 15, left), where they are given the option to reply. Once the user responds, the popup dialog closes, and the screen automatically scrolls to the suggested comment within the discussion. Additionally, users can reopen the suggestion popup by clicking a designated icon. When reopened, a new opposing comment (if available) is proposed for the user to reply to.

8.4.2 Enhancing Debate Quality and Engagement With the Deliberative Quality Module

In addition to disorganized content and dysfunctional argumentation (which diminishes reciprocity, see the previous section), online discussions face other challenges, including low-quality contributions (Klein, 2011). Addressing this issue is crucial for fostering meaningful and productive conversations. In an observational study, Wang and Diakopoulos (2022) found that manually highlighting high-quality comments in the comment section of the New York Times (referred to as the *New York Times Picks*) increases the overall discussion quality and the user engagement. The authors suggest that highlighting well-written comments is beneficial to the quality of new comments as the picked comments constitute a social feedback mechanism (Wang and Diakopoulos, 2022).

We build on these findings and develop the *Deliberative Quality Module* which aims to promote high quality comments by automatically highlighting them. It remains to be investigated whether the human component, i.e., the selection by a New York Times editor, has a significant impact on the participants' perceptions, or whether simply highlighting the comments has the same effect. To measure the deliberativeness of a user comment, we calculate the AQuA score (Behrendt et al., 2024) for each comment and define a threshold for high quality.

AQuA Score. The AQuA score, proposed by Behrendt et al. (2024), is a weighted sum of the predictions of individual BERT-based adapter models f_{θ_k} (Pfeiffer et al., 2020), fine-tuned for 20 different deliberative quality indicators. These include, i.a., *justification*, *proposing solutions*, *referencing other users* and, as an indicator for low quality, the use of incivility markers, such as *sarcasm*. Each adapter prediction is weighted with a number $w_k \in \mathbb{R}$ that is estimated from data. Some of the weights are positive, indicating a positive correlation between the respective indicator and the overall quality of the comment, and some are negative, indicating a negative correlation. The total score for a comment c is

calculated as

$$s_{\text{AQuA}}(c) = \sum_{k=1}^{20} w_k f_{\theta_k}(c). \tag{43}$$

AQuA scores are normalized to the range between 0 and 5. Note that the individual predictions of AQuA are trained on expert evaluations, which are combined with weights estimated from non-expert assessments, for details see Behrendt et al. (2024). In the *Deliberative Quality Module*, AQuA scores allow us to identify high quality comments.

Deliberative Quality User Experience. The three comments with the highest predicted AQuA scores, which exceed a specified threshold, are automatically identified as top comments. They are prominently displayed above the other comments and highlighted in light green color (see Figure 15 on the right, showing only a single top comment). The other comments are displayed below the top comments in chronological order. The exact threshold depends on the discussion and can be set as a hyperparameter.

8.5 Implementation Details

Adhocracy+ is built on the Django framework[27] and provides a wide range of functionalities and modules to facilitate large-scale online discussions. The platform's debate module features a forum-like structure where a discussion topic is defined and displayed at the top of the page, enabling users to comment on the topic or respond to other participants' comments. Additional details about the platform's features are available on the adhocracy+ website[28].

We extend adhocracy+ by importing the AI tools into the debate module, as shown in Figure 16 (left). A more detailed view is shown in Figure 16 on the right. When a new comment is added by a user, the Django debate model fires an event, which is handled in the `signals.py` file. Here, we import the AI tools to pass the comments to the stance detection or the deliberative quality model. The AI tools then return a response (either a stance or deliberative quality score), which is stored back to the database for the corresponding comment. This stored response is then presented by the corresponding module as shown in Figure 15. For the purposes of this study, they were implemented as distinct debate modules within the adhocracy+ platform in order to enable the separate evaluation of their respective effects. Their integration into a unified module remains a plausible direction for future development. Overall, this architecture is flexible: In our experiments, we ran the AI tools locally on a Linux server. But the AI tools could also be run as services where communication is handled via Rest API.

[27] `https://www.djangoproject.com/`
[28] `https://adhocracy.plus/info/features/`

8.6 Evaluation

In the following, we analyze the effectiveness of our two proposed modules. We start by evaluating both models on existing datasets and measure how well they perform in terms of accuracy and F1 score. Furthermore, we conducted a large-scale user study to evaluate participants' satisfaction when using the modules in a real online discussion as well as to gauge the effects of the modules on other perceptions and behaviors of the participants.

Model	Acc.	F1
BERT Base German Cased	0.7381	0.7426

Table 23: The performance on the test set of the X-Stance dataset (Vamvas and Sennrich, 2020) of the fine-tuned BERT Base German cased model we used for stance prediction.

8.6.1 Model Performance

Comment Recommendation Module. Table 23 displays the performance of the German BERT Base uncased model, which was fine-tuned on the X-Stance dataset Vamvas and Sennrich, 2020. The model reaches an accuracy of 73.81 and an F1 score of 74.26 on the test dataset.

Deliberative Quality Module. A multilingual BERT base uncased model[29] serves as the basis for the trained adapter models that build the AQuA score (Behrendt et al., 2024). Table 24 lists the weighted average F1 scores on the test dataset for each of the 20 trained adapter models on deliberative aspects.

	Deliberative Aspect	MBERT uncased
Rationality	Relevance	0.37
	Fact	0.56
	Opinion	0.57
	Justification	0.69
	Solution Proposals	0.79
	Additional Knowledge	0.78
	Question	0.87
Reciprocity	Referencing Users	0.88
	Referencing Medium	0.93
	Referencing Contents	0.81
	Referencing Personal	0.92
	Referencing Format	0.96
Civility	Polite form of Address	0.97
	Respect	0.9
	Screaming	0.81
	Vulgar	0.74
	Insults	0.87
	Sarcasm	0.48
	Discrimination	0.88
	Storytelling	0.85
	Ø Total Average (F1-Score)	0.7815

Table 24: We show the weighted average F1 score for the 20 different deliberative aspects the AQuA score adapter models are trained on.

8.6.2 User Study

8.6.3 Methodology

To investigate the effects of both AI modules, we conducted a field experiment as part of a three-wave panel survey in July 2024. Participants were recruited from the German population through Bilendi, an online access panel provider and market research company. The final sample consisted of N = 1,356 participants with a mean age of 52 years (47% female; 58% with at least a high school diploma).

[29] https://huggingface.co/google-bert/bert-base-multilingual-cased

#	Survey Question	Scale (1-7)
Q1	On the platform, discussion contributions were suggested to me, to which I could reply.	1 = strongly disagree, 7 = strongly agree
Q2	On the platform, contributions were marked as top comments.	1 = strongly disagree, 7 = strongly agree
Q3	To what extent did you feel that this process was supported by artificial intelligence?	1 = most certainly not, 7 = most certainly yes
Q4*	I enjoyed using discuss20.	1 = strongly disagree, 7 = strongly agree
Q5*	The functions of the discuss20 platform threatened my freedom to choose what I wanted.	1 = strongly disagree, 7 = strongly agree
Q6	All in all, I was satisfied with the discussion.	1 = strongly disagree, 7 = strongly agree
Q7*	The contributions contained arguments and justifications.	1 = strongly disagree, 7 = strongly agree
Q8*	The participants responded to the contributions of others.	1 = strongly disagree, 7 = strongly agree
Q9*	The contributions were discriminating.	1 = strongly disagree, 7 = strongly agree
Q10*	There was a wide range of opinions in the discussion.	1 = strongly disagree, 7 = strongly agree

Table 25: Excerpt from our user study survey questions. Questions that are marked with an asterisk are example questions that are part of a larger index.

Participants joined a simulated citizens' assembly with a 10-day online discussion phase on the extended adhocracy+ platform (internally referred to as *discuss20*). They engaged in small-group discussions on two selected political topics: (1) whether active euthanasia should be legally permitted in Germany, and (2) whether the sale of alcoholic beverages should be more restricted in Germany. These topics were identified in a preliminary survey as the most engaging from a broader selection of issues.

The experimental design consisted of five conditions for each of the two discussed topics, aimed at testing the effects of the AI modules. These included: discussions supported by the *Comment Recommendation Module*, which either (i) recommended comments that contradicted the participant's opinion or (ii) recommended random comments. Discussions supported by the *Deliberative Quality Module*, which either (iii) highlighted three comments with the highest deliberative quality scores as "top comments" or (iv) highlighted three randomly selected comments as "top comments" and (v) discussions without AI support, serving as the control group. Participants and experimental conditions were randomly assigned, resulting in ten distinct experimental groups. Randomization checks showed no significant differences between the groups in terms of age, gender, education, or political interest.

During and after the discussions, the participants completed standardized online questionnaires to evaluate their experiences on the platform. To explore the effects of the AI modules, this user study focuses on four aspects:

1. **Manipulation effectiveness** - the extent to which participants recognized and responded to the implemented AI features.
2. **Quantitative participation** - the extent of the engagement of the participants in the discussions.
3. **Platform evaluation** - users' perceptions of the platforms usability and functions.

	AI CR Module (n = 289)		**Random CR Module** (n = 276)		**Control** (n = 262)		**F**
	M	*SD*	*M*	*SD*	*M*	*SD*	
(1) Manipulation effectiveness							
Discussion contributions were suggested to me, to which I could reply	6.20[a]	1.29	5.88[a]	1.54	3.83[b]	2.22	116.18***
To what extent did you feel that the discussion was supported by artificial intelligence?	4.33[a]	1.65	4.18[a]	1.62	3.80[b]	1.71	7.47***
(2) Quantity of participation							
Average number of comments per user	12.71[a]	12.85	12.98[a]	11.85	9.16[b]	10.58	9.82***
(3) Platform evaluation							
Overall satisfaction with the platform	6.08	1.13	6.11	1.06	6.15	0.97	0.34
Experience of threats to freedom of choice	1.44	0.89	1.51	0.96	1.37	0.80	1.61
(4) Discussion evaluation							
Satisfaction with the discussion	5.98[a]	1.16	5.89[ab]	1.33	5.63[b]	1.49	4.60*
Perception of diversity	6.03[a]	0.94	5.89[ab]	1.01	5.74[b]	1.05	5.50**
Perception of reciprocity	5.64[a]	1.03	5.41[a]	1.13	4.97[b]	1.35	21.00***

n = 827, One-Way ANOVA (Post-Hoc-Test: Bonferroni/Games-Howell), *p<0.05, ** p<0.01, *** p<0.001.
Note: Groups with different code letters (a, b) differ significantly at the 5% level.

Table 26: Results of One-Way Analyses of Variance (ANOVAs) for the Comment Recommendation (CR) Module.

4. **Discussion evaluation** - participants' assessments of the discussion quality, including satisfaction and deliberative characteristics.

An excerpt of the corresponding survey questions is listed in Table 25. The effectiveness of the manipulations was measured through participants' recognition of the module-specific functions (see Q1 and Q2) and their assessment of the AI-support (see Q3). Evaluation of the platform included overall satisfaction with the platform (mean index of 5 items, see, e.g., Q4*, Cronbach's alpha = .86) as well as evaluation of the functions against the backdrop of freedom of choice (perceived autonomy, mean index of 4 items, see, e.g., Q5*, Cronbach's Alpha = .85). Lastly, evaluation of the discussions included overall satisfaction with the discussion (see Q6) and the perceived deliberative quality, evaluated across four dimensions, namely the perception of the rationality (mean index of 4 items, see, e.g., Q7*, Cronbach's alpha = .85), reciprocity (mean index of 3 items, see, e.g., Q8*, Cronbach's alpha = .89), civility (mean index of 4 items, e.g., Q9*, Cronbach's alpha .78) and diversity of the discussions (mean index of 4 items, e.g., Q10*, Cronbach's alpha .88).

As the *Comment Recommendation Module* aims to expose users to diverse viewpoints, it fosters diversity and reciprocity by encouraging interaction with opposing opinions. In contrast, the *Deliberative Quality Module* promotes civility and rationality by highlighting comments that exemplify high deliberative quality, thereby setting a constructive standard for discussion. Consequently, the analysis focuses on diversity and reciprocity for the *Comment Recommendation Module* and on civility and rationality for the *Deliberative Quality Module*, as these dimensions best capture the intended effects of each intervention.

8.6.4 Results

Comment Recommendation Module. We conducted One-way Analyses of Variance (ANOVAs) to investigate group-specific manipulation effectiveness, quantitative participation, platform evaluation, and discussion evaluation. A summary of the results for the *Comment Recommendation Module* is provided in Table 26. We report mean (M) and standard deviation (SD) and F-Values (F). Regarding manipulation effectiveness, participants in the *Comment Recommendation Modules* scored significantly higher on identifying this platform feature and on perceiving AI support compared to the control group. However, the participants' assessment whether the discussion was supported by AI did not significantly differ between the modules with random and AI-based comment recommendation.

Regarding participation, participants in the *Comment Recommendation Modules* wrote an average of approximately three to four more comments per user compared to the control group. Again, it was inconsequential whether the recommended comment was suggested randomly or AI-based. Regarding users' evaluation of the platform, the *Comment Recommendation Modules* did not impair users' satisfaction with the platform due to the module-specific implemented functions. Another positive finding is that the *Comment Recommendation Modules* did not restrict participants' feelings of autonomy. In contrast, regarding the effects on discussion evaluation, especially participants in the AI-supported *Comment Recommendation Module* reported a significantly higher satisfaction with the discussion and higher perception of the deliberative dimension of diversity compared to the control group. Finally, comment recommendation significantly increased participants' perception of reciprocity within the discussion compared to the control group. Regardless of an underlying AI-based recommendation, we found that recommending comments had an overall positive effect on individual participation.

Deliberative Quality Module. Table 27 provides an overview of the ANOVA results for the *Deliberative Quality Module*. Regarding manipulation effectiveness, participants in both the AI *Deliberative Quality* and Random *Deliberative Quality Modules* were significantly more likely to recognize platform contributions marked as top comments and to perceive AI support compared to the control group. However, the participants' assessment of AI support did not significantly differ between the AI *Deliberative Quality* and Random *Deliberative Quality Modules*.

In terms of participation quantity, the average number of comments per user did not differ significantly between the groups, suggesting that neither the AI *Deliberative Quality* nor the Random *Deliberative Quality Module* led to an increase in users' commenting activity. Similarly, for platform evaluation, no significant differences were found in users' overall satisfaction with the platform or their perceptions of autonomy. Participants across all groups reported similarly high satisfaction and did not feel restricted in their freedom to choose actions on the platform. Finally, regarding discussion evaluation, no significant differences were observed between the groups in terms of satisfaction

	AI DQ Module (n = 264)		Random DQ Module (n = 265)		Control (n = 262)		F
	M	*SD*	*M*	*SD*	*M*	*SD*	
(1) Manipulation effectiveness							
On the platform, contributions were marked as top comments	5.81[a]	1.76	5.90[a]	1.61	3.05[b]	2.02	189.17***
To what extent did you feel that the discussion was supported by artificial intelligence?	4.20[a]	1.80	4.50[a]	1.61	3.80[b]	1.71	11.22***
(2) Quantity of participation							
Average number of comments per user	9.40	10.39	9.58	9.69	9.16	10.58	0.12
(3) Platform evaluation							
Overall satisfaction with the platform.	6.16	1.02	6.06	1.13	6.15	0.97	0.74
Experience of threats to freedom of choice	1.42	0.85	1.39	0.91	1.37	0.80	0.21
(4) Discussion evaluation							
Satisfaction with the discussion	5.71	1.51	5.63	1.45	5.63	1.49	0.27
Perception of civility	6.84	0.49	6.83	0.47	6.77	0.66	0.63
Perception of rationality	5.67	1.04	5.49	1.05	5.51	0.97	2.53

n = 791, One-Way ANOVA (Post-Hoc-Test: Bonferroni/Games-Howell), *p<0.05, ** p<0.01, *** p<0.001
Note: Groups with different code letters (a, b) differ significantly at the 5% level.

Table 27: Results of One-Way Analyses of Variance (ANOVAs) for the Deliberative Quality (DQ) Module.

with the discussion, civility, or rationality. While the modules aimed to enhance discussion quality, their implementation did not result in perceptible changes in these specific evaluative dimensions.

In order to compare the actual quality of the discussions across the different groups, content analyses are currently being conducted. Preliminary results suggest that, for the topic of active euthanasia, the quality of discussions was higher in the *Deliberative Quality Module* than in the other modules. Again, however, it appears that it does not seem to make a difference whether the top comments are selected by the AI or at random.

8.7 Conclusion

In this work we present extensions to the adhocracy+ platform for citizen participation. We implemented two additional modules to support more deliberative online discussions. In the *Comment Recommendation Module* participants are confronted with opposing views to encourage user interaction, hence improving the reciprocity in the discussion. The *Deliberative Quality Module* aims to improve the quality of contributed comments by automatically highlighting the most deliberative ones.

In a large-scale user study, we tested the effects of both AI modules. We found that the *Comment Recommendation Module* increased participation on the platform and improved users' perception of the deliberative quality of the discussions while not diminishing their sense of autonomy. The *Deliberative Quality Module*, in contrast, did not significantly improve users' perceptions of the platform or the discussions. Still, there are indications

that both modules had a positive influence on the discussions, albeit independently of whether AI was involved or not.

We see great potential in the features we presented to support human actors in conducting large online discussions. Certainly it remains an open task to improve the AI to a level, where people perceive its selection performance as far superior than random selection. The resulting platform is freely available under an open source license and can hopefully be used for political decision-making in the future.

Future Work. In the future we want to examine how both AI extensions to adhocracy+ can be further improved. This means gathering and annotating additional conversational data to fine-tune and improve both models. To further evaluate effects of both modules on the comment quality within the discussions, content analyses are currently being carried out.

Limitations

While our extensions to adhocracy+ introduce AI-driven enhancements, we must acknowledge several limitations.

Currently, the platform and both AI modules are only available in German. This limits accessibility for non-German speaking users and limits the potential for wider adoption.

Moreover, the effectiveness of both AI modules highly depends on the quality of their training data. They may struggle with nuanced or complex discussions, and incorrect predictions can potentially frustrate participants.

The effects we observed were predominantly very small, which may be due to the design of our study. In field experiments, numerous noise factors can influence the outcomes we measured - such as the perceived quality of discussions. At the same time, our experiments offer high external validity, as they were conducted in a realistic setting rather than under artificial laboratory conditions.

The partly non-significant differences between the AI, random, and control conditions may also be attributed to the statistical procedures employed. We used post hoc tests that apply strict corrections for multiple testing, which makes it more difficult to detect statistically significant effects.

However, when conducting planned contrast analyses, some differences between the AI-supported and Random *Comment Recommendation* and *Deliberative Quality Modules* do reach significance, suggesting that the AI-supported modules were perceived more positively by the participants than those working with random content selection.

Nonetheless, planned contrasts require more specific a priori hypotheses, which could not be formulated within the scope of this exploratory paper. Developing and testing such hypotheses remains a task for future research.

9 Conclusion

Online discussions play an increasingly important role in democratic participation. Social media, comment sections of news outlets, and digital participation processes give citizens the opportunity to discuss political issues and contribute to collective decision-making. However, these environments also face persistent challenges, such as incivility, misinformation, and unstructured debates, which can hinder constructive discourse. In this context, the concept of deliberation provides a normative framework for evaluating and improving the quality of online discussions.

This thesis has investigated the potential and limitations of applying methods from NLP and ML to online discourse in the context of deliberative communication. The work has combined technical advances in NLP with insights from communication science and political science, reflecting the interdisciplinary nature of research on online deliberation.

The first contribution of this thesis has been a systematic survey of NLP applications in online political discussions. The survey has mapped the current research landscape and identified a wide range of tasks addressed by existing work, including argument mining, toxicity detection, stance detection, summarization, and content recommendation. The results show that NLP techniques are increasingly used to analyze and structure discussions, yet many applications remain experimental and have not been deployed in real-world discussion platforms. The survey therefore provides an overview of the current state of research and highlights promising directions for further development.

The thesis has also addressed the technical challenge of developing efficient models suitable for real-time discussion environments. Two approaches for improving the performance of smaller language models have been presented. With ArgueBERT, this thesis has explored alternative pre-training strategies to improve the representation of argument similarity. With MaxPoolBERT, it has introduced a simple yet effective extension to the fine-tuning process by aggregating information across layers and tokens. The experimental results show that targeted modifications to fine-tuning can improve classification performance, particularly in low-resource settings, while maintaining the compact model size required for resource-constrained settings.

Another key contribution has been the development of a method for measuring the deliberative quality of user contributions. The AQuA framework has combined predictions across multiple dimensions of deliberation to produce an overall quality score for discussion comments. By integrating expert and non-expert perspectives, the approach enables a more nuanced assessment of deliberative quality than single-score classifiers. While this form of automated quality assessment can inform feedback mechanisms in online

discussions, the experiments conducted in this thesis suggest that its effects on user behavior may depend on additional design factors.

Finally, the thesis has demonstrated how NLP models can be integrated into real discussion platforms and evaluated in practice. The open-source participation platform adhocracy+ has been extended with AI-supported debate modules, which have been tested in a large-scale user study. The results provide empirical insights into how users perceive AI-supported moderation and assistance and how such interventions influence the dynamics of online discussions. These experiments represent an important step toward understanding the real-world impact of NLP-based support systems.

Taken together, the contributions of this thesis indicate that NLP methods can provide useful support for online deliberation, while also highlighting important limitations and challenges for their deployment in real-world discussion settings. At the same time, the results emphasize that technical performance alone is not sufficient for successful deployment. Rather, AI-assisted discussion systems need to be designed with attention to user perception, fairness, transparency, and the broader social dynamics of online communication.

9.1 Future Work

While the results presented in this thesis demonstrate the potential of NLP-based approaches to support online deliberation, they also point to several directions for future research.

- First, further work is needed to **improve the robustness and generalizability of NLP models used in discussion platforms**. Many models are trained and evaluated on relatively small or domain-specific datasets. Future research should investigate how models can be adapted to different discussion contexts, languages, and cultural settings. In particular, multilingual models and cross-domain evaluation will become increasingly important as participation platforms are used in diverse environments.

- Second, **optimizing efficient language models remains an important research challenge**. Although LLMs have achieved impressive performance on many NLP tasks, their computational requirements often make them unsuitable for real-time deployment in participation platforms. Future work should therefore continue exploring methods for improving smaller models, including parameter-efficient training techniques, distillation approaches, and hybrid architectures that combine efficiency with strong representation capabilities.

- Third, more research is needed on the **integration of NLP systems into real-world discussion platforms**. While this thesis has demonstrated the feasibility of AI-assisted interventions, designing effective interaction mechanisms remains a complex challenge. Future studies could explore different forms of feedback, such as

real-time writing assistance, argument suggestions, or visualizations of discussion dynamics. Understanding how such interventions influence user behavior and discussion quality over longer periods will be crucial for assessing their practical value.

- Fourth, further research should examine the **perception and acceptance of AI-based moderation**. Trust, transparency, and perceived fairness play a central role in determining whether users accept automated support systems. Future work should therefore investigate how AI decisions can be explained to users, how biases in automated moderation can be mitigated, and how users can maintain control over the discussion process.
- Fifth, more attention should be paid to the **meaningful evaluation of tools applied in online discussions.** Developing and applying methods to make online spaces safer is an important task, but from a research perspective, it is equally important to study the effects of these methods very carefully. Currently, standardized procedures for reliably measuring discussion quality and the effects of AI interventions remain limited. Future work should therefore develop evaluation frameworks that account not only for model performance, but also for user experience, discussion dynamics, and deliberative outcomes.

In summary, advancing AI-supported deliberation requires progress at both technical and social levels. By combining efficient NLP models with thoughtful platform design and transparent evaluation, future research conducted through interdisciplinary collaboration can contribute to the development of digital discussion spaces that foster a more constructive, respectful, and inclusive democratic dialogue.

10 Publications of the Author

Publications as First Author

[P1] M. Behrendt & S. Harmeling, (2021). **Arguebert: How to improve bert embeddings for measuring the similarity of arguments.** In Proceedings of the 17th Conference on Natural Language Processing (KONVENS 2021) (pp. 28-36). https://aclanthology.org/2021.konvens-1.3/.

[P2] M. Behrendt, S. S. Wagner, M. Ziegele, L. Wilms, A. Stoll, D. Heinbach & S. Harmeling, (2024). **AQuA – Combining Experts' and Non-Experts' Views To Assess Deliberation Quality in Online Discussions Using LLMs.** In Proceedings of the First Workshop on Language-driven Deliberation Technology (DELITE) @ LREC-COLING 2024, pages 1–12, Torino, Italia. ELRA and ICCL. https://aclanthology.org/2024.delite-1.1/.

[P3] M. Behrendt, S. S. Wagner, M. Warne, J. L. Peters, M. Ziegele & S. Harmeling, (2025). **Supporting Online Discussions: Integrating AI Into the adhocracy+ Participation Platform To Enhance Deliberation.** In Proceedings of the Fourth Workshop on Bridging Human-Computer Interaction and Natural Language Processing (HCI+NLP), pages 7–16, Suzhou, China. Association for Computational Linguistics. https://aclanthology.org/2025.hcinlp-1.2/.

[P4] M. Behrendt, S. S. Wagner & S. Harmeling, (2025). **MaxPoolBERT: Enhancing BERT Classification via Layer-and Token-Wise Aggregation.** arXiv preprint arXiv:2505.15696. https://doi.org/10.48550/arXiv.2505.15696.

[P5] M. Behrendt, S. S. Wagner, C. Weinmann, M. Bormann, M. Warne & S. Harmeling, (2026). **Machine Learning for Enhancing Deliberation in Online Political Discussions and Participatory Processes: A Survey.** arXiv preprint arXiv:2506.02533. https://doi.org/10.48550/arXiv.2506.02533.

[P6] M. Behrendt, V. Warnken, D. Friess, M. Ziegele & T. Escher, (2026). **Using AI to Support Discursive Integration in Online Discussions.** In Proceedings of the Second Workshop on Language-driven Deliberation Technology (DELITE) @ LREC 2026. https://lrec.elra.info/lrec2026-ws-delite-01

Further Publications

[P7] M. Brenneis, M. Behrendt, S. Harmeling, & M. Mauve, (2020). **How Much Do I Argue Like You? Towards a Metric on Weighted Argumentation Graphs.** In SAFA@ COMMA (pp. 2-13). https://safa2020.argumentationcompetition.org/papers/paper_1.pdf.

[P8] M. Brenneis, M. Behrendt & S. Harmeling, (2021). **How Will I Argue? A Dataset for Evaluating Recommender Systems for Argumentations.** In Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue (pp. 360-367). https://aclanthology.org/2021.sigdial-1.38/.

[P9] H. Weiland, M. Behrendt & S. Harmeling, (2023). **Automatic Dictionary Generation: Could Brothers Grimm Create a Dictionary with BERT?** In Proceedings of the 19th Conference on Natural Language Processing (KONVENS 2023) (pp. 102-120). https://aclanthology.org/2023.konvens-main.11/.

[P10] O. Kelm, T. Neumann, M. Behrendt, M. Brenneis, K. Gerl, S. Marschall, F. Meißner, S. Harmeling, G. Vowe & M. Ziegele, (2023). **How algorithmically curated online environments influence users' political polarization: Results from two experiments with panel data.** Computers in Human Behavior Reports, 12, 100343. https://doi.org/10.1016/j.chbr.2023.100343.

[P11] S. S. Wagner, M. Behrendt, M. Ziegele & S. Harmeling, (2025). **The Power of LLM-Generated Synthetic Data for Stance Detection in Online Political Discussions.** In The Thirteenth International Conference on Learning Representations. https://openreview.net/forum?id=ws5phQki00.

[P12] T. Uelwer, J. Robine, S. S. Wagner, M. Höftmann, E. Upschulte, S. Konietzny, M. Behrendt & S. Harmeling, (2025). **A survey on self-supervised methods for visual representation learning.** Machine Learning, 114(4), 111. https://doi.org/10.1007/s10994-024-06708-7.

References

R. Abbott, B. Ecker, P. Anand, and M. Walker (2016). "Internet Argument Corpus 2.0: An SQL schema for Dialogic Social Media and the Corpora to go with it". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Ed. by N. Calzolari, K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, and S. Piperidis. European Language Resources Association (ELRA), pp. 4445–4452.

F. A. Acheampong, C. Wenyu, and H. Nunoo-Mensah (2020). "Text-based emotion detection: Advances, challenges, and opportunities". In: *Engineering Reports* 2.7, e12189. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/eng2.12189`.

J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. (2023). "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774*.

T. Aitamurto, K. Chen, A. Cherif, J. S. Galli, and L. Santana (2016). "Civic CrowdAnalytics: Making Sense of Crowdsourced Civic Input with Big Data Tools". In: *Proceedings of the 20th International Academic Mindtrek Conference*. AcademicMindtrek '16. Association for Computing Machinery, pp. 86–94.

Y. Ajjour, H. Wachsmuth, J. Kiesel, M. Potthast, M. Hagen, and B. Stein (2019). "Data Acquisition for Argument Search: The args.me corpus". In: *42nd German Conference on Artificial Intelligence (KI 2019)*. Ed. by C. Benzmüller and H. Stuckenschmidt. Springer, pp. 48–59.

F. Alkomah and X. Ma (2022). "A Literature Review of Textual Hate Speech Detection Methods and Datasets". In: *Information* 13.6.

M. Alshomary, T. Gurcke, S. Syed, P. Heinisch, M. Spliethöver, P. Cimiano, M. Potthast, and H. Wachsmuth (2021). "Key Point Analysis via Contrastive Learning and Extractive Argument Summarization". In: *Proceedings of the 8th Workshop on Argument Mining*. Association for Computational Linguistics, pp. 184–189.

D. Anastasiou (2022). "ENRICH4ALL: A First Luxembourgish BERT Model for a Multilingual Chatbot". In: *Proceedings of the 1st Annual Meeting of the ELRA/ISCA Special Interest Group on Under-Resourced Languages*. European Language Resources Association, pp. 207–212.

L. Anastasiou and A. De Liddo (2021). "Making Sense of Online Discussions: Can Automated Reports Help?" In: *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI EA '21. Association for Computing Machinery.

L. Anastasiou and A. De Liddo (2026). "From Shallow Discourse to Structured Deliberation: The BCause Approach". In: *Human-Computer Interaction – INTERACT 2025*. Ed. by C. Ardito, S. Diniz Junqueira Barbosa, T. Conte, A. Freire, I. Gasparini, P. Palanque, and R. Prates. Springer Nature Switzerland, pp. 94–99.

L. Anastasiou, A. De Moor, B. Brayshay, and A. De Liddo (2023). "A tale of struggles: an evaluation framework for transitioning from individually usable to community-useful online deliberation tools". In: *Proceedings of the 11th International Conference on Communities and Technologies*. C&T '23. Association for Computing Machinery, pp. 144–155.

A. A. Anderson, D. Brossard, D. A. Scheufele, M. A. Xenos, and P. Ladwig (2014). "The "nasty effect:" Online incivility and risk perceptions of emerging technologies". In: *Journal of computer-mediated communication* 19.3, pp. 373–387.

M. Arana-Catania, F.-A. V. Lier, R. Procter, N. Tkachenko, Y. He, A. Zubiaga, and M. Liakata (2021a). "Citizen Participation and Machine Learning for a Better Democracy". In: *Digit. Gov.: Res. Pract.* 2.3.

M. Arana-Catania, R. Procter, Y. He, and M. Liakata (2021b). "Evaluation of Abstractive Summarisation Models with Machine Translation in Deliberative Processes". In: *Proceedings of the Third Workshop on New Frontiers in Summarization*. Association for Computational Linguistics, pp. 57–64.

L. P. Argyle, C. A. Bail, E. C. Busby, J. R. Gubler, T. Howe, C. Rytting, T. Sorensen, and D. Wingate (2023). "Leveraging AI for democratic discourse: Chat interventions can improve online political conversations at scale". In: *Proceedings of the National Academy of Sciences* 120.41, e2311627120.

A. Askell, Y. Bai, A. Chen, D. Drain, D. Ganguli, T. Henighan, A. Jones, N. Joseph, B. Mann, N. DasSarma, N. Elhage, Z. Hatfield-Dodds, D. Hernandez, J. Kernion, K. Ndousse, C. Olsson, D. Amodei, T. B. Brown, J. Clark, S. McCandlish, C. Olah, and J. Kaplan (2021). "A General Language Assistant as a Laboratory for Alignment". In: *CoRR* abs/2112.00861.

M. Assady (2015). "Incremental Hierarchical Topic Modeling for Multi-party Conversation Analysis". PhD thesis.

M. El-Assady, R. Sevastjanova, D. Keim, and C. Collins (2018). "ThreadReconstructor: Modeling reply-chains to untangle conversational text through visual analytics". In: *Computer Graphics Forum*. Vol. 37. 3. Wiley Online Library, pp. 351–365.

P. Atanasova, J. G. Simonsen, C. Lioma, and I. Augenstein (2020). "Generating Fact Checking Explanations". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 7352–7364.

J. L. Ba, J. R. Kiros, and G. E. Hinton (2016). *Layer Normalization*. arXiv: 1607.06450 [stat.ML].

I. D. Babatunde, O. M. Nnanna, and M. Klein (2025). "Moderating Large Scale Online Deliberative Processes with Large Language Models (LLMs): Enhancing Collective Decision-Making." In: *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*. SAC '25. Association for Computing Machinery, pp. 996–1003.

A. Bächtiger, J. S. Dryzek, J. Mansbridge, and M. D. Warren (2018). *The Oxford Handbook of Deliberative Democracy*. Oxford University Press.

A. Bächtiger, M. Gerber, and E. Fournier-Tombs (2022). "Discourse Quality Index". In: *Research Methods in Deliberative Democracy*. Oxford University Press. eprint: https://academic.oup.com/book/0/chapter/378695331/chapter-pdf/53454043/oso-9780192848925-chapter-6.pdf.

A. Bächtiger, S. Niemeyer, M. Neblo, M. R. Steenbergen, and J. Steiner (2009a). "Disentangling Diversity in Deliberative Democracy: Competing Theories, Their Blind Spots and Complementarities". In: *Journal of Political Philosophy* 18.1, pp. 32–63.

A. Bächtiger, S. Shikano, S. Pedrini, and M. Ryser (2009b). "Measuring deliberation 2.0: standards, discourse types, and sequenzialization". In: *ECPR General Conference*. Potsdam, pp. 5–12.

D. Bahdanau, K. Cho, and Y. Bengio (2014). "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473*.

M. Bakker, M. Chadwick, H. Sheahan, M. Tessler, L. Campbell-Gillingham, J. Balaguer, N. McAleese, A. Glaese, J. Aslanides, M. Botvinick, and C. Summerfield (2022). "Fine-tuning language models to find agreement among humans with diverse preferences". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., pp. 38176–38189.

D. Balta, P. Kuhn, M. Sellami, D. Kulus, C. Lieven, and H. Krcmar (2019). "How to Streamline AI Application in Government? A Case Study on Citizen Participation in Germany". In: *Electronic Government: 18th IFIP WG 8.5 International Conference, EGOV 2019, San Benedetto Del Tronto, Italy, September 2–4, 2019, Proceedings*. Springer-Verlag, pp. 233–247.

R. Bao, Z. Zhang, and H. Zhao (2021). "Span Fine-tuning for Pre-trained Language Models". In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Ed. by M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih. Association for Computational Linguistics, pp. 1970–1979.

A.-C. Băroiu and Ș. Trăușan-Matu (2022). "Automatic Sarcasm Detection: Systematic Literature Review". In: *Information* 13.8.

M. Barrett and I. Brunton-Smith (2014). "Political and Civic Engagement and Participation: Towards an Integrative Perspective". In: *Journal of Civil Society* 10.1, pp. 5–28. eprint: https://doi.org/10.1080/17448689.2013.871911.

V. Barriere and A. Balahur (2023). "Multilingual Multi-Target Stance Recognition in Online Public Consultations". In: *Mathematics* 11.9.

V. Barriere, A. Balahur, and B. Ravenet (2022a). "Debating Europe: A Multilingual Multi-Target Stance Classification Dataset of Online Debates". In: *Proceedings of the LREC 2022 workshop on Natural Language Processing for Political Sciences*, pp. 16–21.

V. Barriere, G. G. Jacquet, and L. Hemamou (2022b). "CoFE: A New Dataset of Intra-Multilingual Multi-target Stance Classification from an Online European Participatory Democracy Platform". In: *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Association for Computational Linguistics, pp. 418–422.

N. Beauchamp (2020). "321Modeling and Measuring Deliberation Online". In: *The Oxford Handbook of Networked Communication*. Oxford University Press. eprint: https://academic.oup.com/book/0/chapter/290658653/chapter-ag-pdf/44521967/book_34286_section_290658653.ag.pdf.

M. Behrendt and S. Harmeling (2021). "ArgueBERT: How To Improve BERT Embeddings for Measuring the Similarity of Arguments". In: *Proceedings of the 17th Conference on Natural Language Processing (KONVENS 2021)*. KONVENS 2021 Organizers, pp. 28–36.

M. Behrendt, S. S. Wagner, and S. Harmeling (2025a). "MaxPoolBERT: Enhancing BERT Classification via Layer-and Token-Wise Aggregation". In: *arXiv preprint arXiv:2505.15696*.

M. Behrendt, S. S. Wagner, M. Warne, J. L. Peters, M. Ziegele, and S. Harmeling (2025b). "Supporting Online Discussions: Integrating AI Into the adhocracy+ Participation Platform To Enhance Deliberation". In: *Proceedings of the Fourth Workshop on Bridging Human-Computer Interaction and Natural Language Processing (HCI+NLP)*. Ed. by S. L. Blodgett, A. C. Curry, S. Dev, S. Li, M. Madaio, J. Wang, S. T. Wu, Z. Xiao, and D. Yang. Association for Computational Linguistics, pp. 7–16.

M. Behrendt, S. S. Wagner, C. Weinmann, M. Bormann, M. Warne, and S. Harmeling (2026). "Machine Learning for Enhancing Deliberation in Online Political Discussions and Participatory Processes: A Survey". In: *arXiv preprint arXiv:2506.02533*.

M. Behrendt, S. S. Wagner, M. Ziegele, L. Wilms, A. Stoll, D. Heinbach, and S. Harmeling (2024). "AQuA – Combining Experts' and Non-Experts' Views To Assess Deliberation Quality in Online Discussions Using LLMs". In: *Proceedings of the First Workshop on Language-driven Deliberation Technology (DELITE) @ LREC-COLING 2024*. Ed. by A.

Hautli-Janisz, G. Lapesa, L. Anastasiou, V. Gold, A. D. Liddo, and C. Reed. ELRA and ICCL, pp. 1–12.

E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell (2021). "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Association for Computing Machinery, pp. 610–623.

E. M. Bender, J. T. Morgan, M. Oxley, M. Zachry, B. Hutchinson, A. Marin, B. Zhang, and M. Ostendorf (2011). "Annotating social acts: authority claims and alignment moves in Wikipedia talk pages". In: *Proceedings of the Workshop on Languages in Social Media*. LSM '11. Association for Computational Linguistics, pp. 48–57.

M. Bhidya (2019). *UtkMl's Twitter Spam Detection Competition*. `https://kaggle.com/competitions/utkmls-twitter-spam-detection-competition`.

R. Bjarnason, D. Gambrell, and J. Lanthier-Welch (2024). "Using Artificial Intelligence to Accelerate Collective Intelligence: Policy Synth and Smarter Crowdsourcing". In: *arXiv preprint arXiv:2407.13960*.

L. W. Black (2008). "Listening to the city: Difference, identity, and storytelling in online deliberative groups". In: *Journal of Deliberative Democracy* 5.1.

J. Bohman and W. Rehg (1997). *Deliberative democracy: Essays on reason and politics*. MIT press.

F. Boltužić and J. Šnajder (2014). "Back up your Stance: Recognizing Arguments in Online Discussions". In: *Proceedings of the First Workshop on Argumentation Mining*. Association for Computational Linguistics, pp. 49–58.

F. Boltužić and J. Šnajder (2015). "Identifying Prominent Arguments in Online Debates Using Semantic Textual Similarity". In: *Proceedings of the 2nd Workshop on Argumentation Mining*. Ed. by C. Cardie. Association for Computational Linguistics, pp. 110–115.

S. Bonechi (2024). "Development of an Automated Moderator for Deliberative Events". In: *Electronics* 13.3.

E. Bossens, D. Geerts, E. Storms, M. Nuytemans, and J. Boesman (2022). "RHETORiC: An Audience Conversation Tool That Restores Civility in News Comment Sections". In: *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI EA '22. Association for Computing Machinery.

S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning (2015). "A large annotated corpus for learning natural language inference". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Ed. by L. Màrquez, C. Callison-Burch, and J. Su. Association for Computational Linguistics, pp. 632–642.

M. Brenneis, M. Behrendt, and S. Harmeling (2021). "How Will I Argue? A Dataset for Evaluating Recommender Systems for Argumentations". In: *Proceedings of the 22nd*

Annual Meeting of the Special Interest Group on Discourse and Dialogue. Association for Computational Linguistics, pp. 360–367.

M. Brenneis, M. Behrendt, S. Harmeling, and M. Mauve (2020). "How Much Do I Argue Like You? Towards a Metric on Weighted Argumentation Graphs". In: *Proceedings of the Third International Workshop on Systems and Algorithms for Formal Argumentation co-located with the 8th International Conference on Computational Models of Argument (COMMA 2020), September 8, 2020*. Ed. by S. A. Gaggl, M. Thimm, and M. Vallati. Vol. 2672. CEUR Workshop Proceedings. CEUR-WS.org, pp. 2–13.

M. Brenneis and M. Mauve (2020). "deliberate–Online Argumentation with Collaborative Filtering". In: *Computational Models of Argument: Proceedings of COMMA*. IOS Press, pp. 453–454.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 1877–1901.

M. X. D. Carpini, F. L. Cook, and L. R. Jacobs (2004). "PUBLIC DELIBERATION, DISCURSIVE PARTICIPATION, AND CITIZEN ENGAGEMENT: A Review of the Empirical Literature". In: *Annual Review of Political Science* 7.1, pp. 315–344. eprint: `https://doi.org/10.1146/annurev.polisci.7.121003.091630`.

D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia (2017). "SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation". In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Ed. by S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. Cer, and D. Jurgens. Association for Computational Linguistics, pp. 1–14.

L. Cernuzzi, M. Alcaraz, C. Parra, and J. Saldivar (2020). "A Sentiment Analysis Approach to Process Civic Contributions". In: *2020 XLVI Latin American Computing Conference (CLEI)*, pp. 455–461.

S. Chambers (2003). "Deliberative Democratic Theory". In: *Annual Review of Political Science* 6.1, pp. 307–326. eprint: `https://doi.org/10.1146/annurev.polisci.6.121901.085538`.

B. Chan, S. Schweter, and T. Möller (2020). "German's Next Language Model". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Ed. by D. Scott, N. Bel, and C. Zong. International Committee on Computational Linguistics, pp. 6788–6796.

H.-S. Chang, R.-Y. Sun, K. Ricci, and A. McCallum (2023). "Multi-CLS BERT: An Efficient Alternative to Traditional Ensembling". In: *Proceedings of the 61st Annual Meeting of*

the Association for Computational Linguistics (Volume 1: Long Papers). Ed. by A. Rogers, J. Boyd-Graber, and N. Okazaki. Association for Computational Linguistics, pp. 821–854.

J. P. Chang, C. Chiam, L. Fu, A. Wang, J. Zhang, and C. Danescu-Niculescu-Mizil (2020). "ConvoKit: A Toolkit for the Analysis of Conversations". In: *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Ed. by O. Pietquin, S. Muresan, V. Chen, C. Kennington, D. Vandyke, N. Dethlefs, K. Inoue, E. Ekstedt, and S. Ultes. Association for Computational Linguistics, pp. 57–60.

J. P. Chang and C. Danescu-Niculescu-Mizil (2019). "Trouble on the Horizon: Forecasting the Derailment of Online Conversations as they Develop". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by K. Inui, J. Jiang, V. Ng, and X. Wan. Association for Computational Linguistics, pp. 4743–4754.

C. Chemudugunta, P. Smyth, and M. Steyvers (2006). "Modeling General and Specific Aspects of Documents with a Probabilistic Topic Model". In: *Advances in Neural Information Processing Systems*. Ed. by B. Schölkopf, J. Platt, and T. Hoffman. Vol. 19. MIT Press.

C. Chen and K. Shu (2024a). "Can LLM-Generated Misinformation Be Detected?" In: *The Twelfth International Conference on Learning Representations*.

C. Chen and K. Shu (2024b). "Combating misinformation in the age of llms: Opportunities and challenges". In: *AI Magazine* 45.3, pp. 354–368.

G. M. Chen (2017). *Online incivility and public debate: Nasty talk*. Springer.

Q. Chen, W. Wang, Q. Zhang, C. Deng, M. Yukun, and S. Zheng (2023). "Improving BERT with Hybrid Pooling Network and Drop Mask". In: *arXiv preprint arXiv:2307.07258*.

K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by A. Moschitti, B. Pang, and W. Daelemans. Association for Computational Linguistics, pp. 1724–1734.

K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning (2020). "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: *International Conference on Learning Representations*.

L. Clever, J. Klapproth, and L. Frischlich (2022). "Automatisierte (Gegen-)Rede? Social Bots als digitales Sprachrohr ihrer Nutzer*innen". In: *Gegenrede digital: Neue und alte Herausforderungen interkultureller Bildungsarbeit in Zeiten der Digitalisierung*. Springer Fachmedien Wiesbaden, pp. 11–26.

K. Coe, K. Kenski, and S. A. Rains (2014). "Online and Uncivil? Patterns and Determinants of Incivility in Newspaper Website Comments". In: *Journal of Communication*

64.4, pp. 658–679. eprint: https://academic.oup.com/joc/article-pdf/64/4/658/22322347/jjnlcom0658.pdf.

J. Cohen (1989). *Deliberative Democracy and Democratic Legitimacy Good Polity*.

M. Conroy, J. T. Feezell, and M. Guerrero (2012). "Facebook and political engagement: A study of online political group membership and offline political engagement". In: *Computers in Human behavior* 28.5, pp. 1535–1546.

N. Curato, J. Sass, S. A. Ercan, and S. Niemeyer (2022). "Deliberative democracy in the age of serial crisis". In: *International Political Science Review* 43.1, pp. 55–66. eprint: https://doi.org/10.1177/0192512120941882.

I. Dagan, O. Glickman, and B. Magnini (2006). "The PASCAL Recognising Textual Entailment Challenge". In: *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*. Ed. by J. Quiñonero-Candela, I. Dagan, B. Magnini, and F. d'Alché-Buc. Springer Berlin Heidelberg, pp. 177–190.

T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré (2022). "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., pp. 16344–16359.

J. Davies, M. Arana-Catania, R. Procter, F.-A. van Lier, and Y. He (2021). "Evaluating the application of NLP tools in mainstream participatory budgeting processes in Scotland". In: *14th International Conference on Theory and Practice of Electronic Governance*, pp. 362–366.

J. Davies and R. Procter (2020). "Online Platforms of Public Participation: A Deliberative Democracy or a Delusion?" In: *Proceedings of the 13th International Conference on Theory and Practice of Electronic Governance*. ICEGOV '20. Association for Computing Machinery, pp. 746–753.

A. De Liddo and R. Strube (2021). "Understanding Failures and Potentials of Argumentation Tools for Public Deliberation". In: *C&T '21: Proceedings of the 10th International Conference on Communities & Technologies - Wicked Problems in the Age of Tech*. C&T '21. Association for Computing Machinery, pp. 75–88.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Association for Computational Linguistics, pp. 4171–4186.

N. Diakopoulos (2015). "Picking the NYT picks: Editorial criteria and automation in the curation of online news comments". In: *ISOJ Journal* 5.1, pp. 147–166.

T. M. Doan and J. A. Gulla (2022). "A survey on political viewpoints identification". In: *Online Social Networks and Media* 30, p. 100208.

J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. Smith (2020). "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping". In: *arXiv preprint arXiv:2002.06305*.

W. B. Dolan and C. Brockett (2005). "Automatically Constructing a Corpus of Sentential Paraphrases". In: *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

J. S. Dryzek (2002). *Deliberative democracy and beyond: Liberals, critics, contestations*. Oxford University Press, USA.

J. S. Dryzek (2005). "Deliberative democracy in divided societies: Alternatives to agonism and analgesia". In: *Political theory* 33.2, pp. 218–242.

L. Dumani, P. J. Neumann, and R. Schenkel (2020). "A Framework for Argument Retrieval". In: *Advances in Information Retrieval*. Ed. by J. M. Jose, E. Yilmaz, J. Magalhães, P. Castells, N. Ferro, M. J. Silva, and F. Martins. Springer International Publishing, pp. 431–445.

T. Durotoye, M. Goyanes, R. Berganza, and H. Gil de Zúñiga (2025). "Online and Social Media Political Participation: Political Discussion Network Ties and Differential Social Media Platform Effects Over Time". In: *Social Science Computer Review*, p. 08944393251332640.

I. Dylko, I. Dolgov, W. Hoffman, N. Eckhart, M. Molina, and O. Aaziz (2017). "The dark side of technology: An experimental investigation of the influence of customizability technology on online political selective exposure". In: *Computers in Human Behavior* 73, pp. 181–190.

S. Elguendouze, L. Anastasiou, E. Hain, E. Cabrio, A. De Liddo, and S. Villata (2025). "AM4DSP: Argumentation Mining in Structured Decentralized Discussion Platforms for Deliberative Democracy". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by I. Habernal, P. Schulam, and J. Tiedemann. Association for Computational Linguistics, pp. 796–805.

S. A. Ercan, H. Asenbaum, N. Curato, and R. F. Mendonça (2022). *Research methods in deliberative democracy*. Oxford University Press.

K. Esau, D. Fleuß, and S.-M. Nienhaus (2021). "Different arenas, different deliberative quality? Using a systemic framework to evaluate online deliberation on immigration policy in Germany". In: *Policy & Internet* 13.1, pp. 86–112.

K. Esau, D. Friess, and C. Eilders (2017). "Design Matters! An Empirical Analysis of Online Deliberation on Different News Platforms". In: *Policy & Internet* 9.3, pp. 321–342. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/poi3.154`.

A. Fabbri, F. Rahman, I. Rizvi, B. Wang, H. Li, Y. Mehdad, and D. Radev (2021). "ConvoSumm: Conversation Summarization Benchmark and Improved Abstractive Summarization with Argument Mining". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 6866–6880.

N. Falk, I. Jundi, E. M. Vecchi, and G. Lapesa (2021). "Predicting Moderation of Deliberative Arguments: Is Argument Quality the Key?" In: *Proceedings of the 8th Workshop on Argument Mining*. Association for Computational Linguistics, pp. 133–141.

N. Falk and G. Lapesa (2022a). "Reports of personal experiences and stories in argumentation: datasets and analysis". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 5530–5553.

N. Falk and G. Lapesa (2022b). "Scaling up Discourse Quality Annotation for Political Science". In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. European Language Resources Association, pp. 3301–3318.

N. Falk and G. Lapesa (2023a). "Bridging Argument Quality and Deliberative Quality Annotations with Adapters". In: *Findings of the Association for Computational Linguistics: EACL 2023*. Ed. by A. Vlachos and I. Augenstein. Association for Computational Linguistics, pp. 2469–2488.

N. Falk and G. Lapesa (2023b). "StoryARG: a corpus of narratives and personal experiences in argumentative texts". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by A. Rogers, J. Boyd-Graber, and N. Okazaki. Association for Computational Linguistics, pp. 2350–2372.

N. Falk, E. Vecchi, I. Jundi, and G. Lapesa (2024). "Moderation in the Wild: Investigating User-Driven Moderation in Online Discussions". In: *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Y. Graham and M. Purver. Association for Computational Linguistics, pp. 992–1013.

S. Faridani, E. Bitton, K. Ryokai, and K. Goldberg (2010). "Opinion Space: A Scalable Tool for Browsing Online Comments". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '10. Association for Computing Machinery, pp. 1175–1184.

M. Fazil, A. K. Sah, and M. Abulaish (2021). "DeepSBD: A Deep Neural Network Model With Attention Mechanism for SocialBot Detection". In: *IEEE Transactions on Information Forensics and Security* 16, pp. 4211–4223.

T. Fischer, F. Schneider, F. Petersen-Frey, A. S. M. Haque, I. Eiser, G. Koch, and C. Biemann (2024). "Extending the Discourse Analysis Tool Suite with Whiteboards for Visual Qualitative Analysis". In: *Proceedings of the 2024 Joint International Conference on*

Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024). Ed. by N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue. ELRA and ICCL, pp. 7017–7022.

J. S. Fishkin and P. Laslett (2008). *Debating deliberative democracy*. John Wiley & Sons.

E. Fournier-Tombs and G. Di Marzo Serugendo (2020). "DelibAnalysis: Understanding the quality of online political discourse with machine learning". In: *Journal of Information Science* 46.6, pp. 810–822. (Visited on 07/27/2022).

E. Fournier-Tombs and M. K. MacKenzie (2021). "Big data and democratic speech: Predicting deliberative quality using machine learning techniques". In: *Methodological Innovations* 14.2, p. 20597991211010416.

D. G. Freelon (2010). "Analyzing online political discussion using three models of democratic communication". In: *New media & society* 12.7, pp. 1172–1190.

R. Friedman, L. Dankin, Y. Hou, R. Aharonov, Y. Katz, and N. Slonim (2021). "Overview of the 2021 Key Point Analysis Shared Task". In: *Proceedings of the 8th Workshop on Argument Mining*. Association for Computational Linguistics, pp. 154–164.

D. Friess and C. Eilders (2015). "A systematic review of online deliberation research". In: *Policy & Internet* 7.3, pp. 319–339.

D. Friess, C. Weinmann, and M. Warné (2025). "AI and Deliberation: Normative Ideals in the Light of Current AI Research-A Review". In: *Journal of Deliberative Democracy* 21.1.

D. Friess, M. Ziegele, and D. Heinbach (2021). "Collective Civic Moderation for Deliberation? Exploring the Links between Citizens' Organized Engagement in Comment Sections and the Deliberative Quality of Online Discussions". In: *Political Communication* 38.5, pp. 624–646. eprint: `https://doi.org/10.1080/10584609.2020.1830322`.

O. Galal, A. H. Abdel-Gawad, and M. Farouk (2024). "Rethinking of BERT sentence embedding for text classification". In: *Neural Computing and Applications* 36.32, pp. 20245–20258.

D. Gambrell (2025). "Ai for Egovernance: Combining Artificial Intelligence and Collective Intelligence to Develop Evidence-Based Ai Policy". In: *2025 Eleventh International Conference on eDemocracy & eGovernment (ICEDEG)*. IEEE, pp. 317–324.

F. Gao, T. Zhang, and T. Franklin (2013). "Designing asynchronous online discussion environments: Recent progress and possible future directions". In: *British Journal of Educational Technology* 44.3, pp. 469–483.

J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin (2017). "Convolutional Sequence to Sequence Learning". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1243–1252.

J. Geiping and T. Goldstein (2023). "Cramming: Training a Language Model on a single GPU in one day." In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 11117–11143.

M. Gerber, A. Bächtiger, S. Shikano, S. Reber, and S. Rohr (2018). "Deliberative Abilities and Influence in a Transnational Deliberative Poll (EuroPolis)". In: *British Journal of Political Science* 48.4, pp. 1093–1118.

F. Gerlach and C. Eilders, eds. (2022). *#meinfernsehen 2021*. Nomos.

A. Giannakopoulos, M. Coriou, A. Hossmann, M. Baeriswyl, and C. Musat (2019). "Resilient Combination of Complementary CNN and RNN Features for Text Classification through Attention and Ensembling". In: *2019 6th Swiss Conference on Data Science (SDS)*, pp. 57–62.

E. Gilbert (2014). "Vader: A parsimonious rule-based model for sentiment analysis of social media text". In: *Proceedings of the international AAAI conference on web and social media*. Vol. 8. 1, pp. 216–225.

X. Glorot and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. PMLR, pp. 249–256.

V. Gold, M. El-Assady, A. Hautli-Janisz, T. Bögel, C. Rohrdantz, M. Butt, K. Holzinger, and D. Keim (2015). "Visual linguistic analysis of political discussions: Measuring deliberative quality". In: *Digital Scholarship in the Humanities* 32.1, pp. 141–158. eprint: `https://academic.oup.com/dsh/article-pdf/32/1/141/11046544/fqv033.pdf`.

S. Goyal, A. R. Choudhury, S. Raje, V. Chakaravarthy, Y. Sabharwal, and A. Verma (2020). "Power-bert: Accelerating bert inference via progressive word-vector elimination". In: *International Conference on Machine Learning*. PMLR, pp. 3690–3699.

T. Graham (2010). "The Use of Expressives in Online Political Talk: Impeding or Facilitating the Normative Goals of Deliberation?" In: *Electronic Participation*. Ed. by E. Tambouris, A. Macintosh, and O. Glassey. Springer Berlin Heidelberg, pp. 26–41.

H. Guo, J. Cao, Y. Zhang, J. Guo, and J. Li (2018). "Rumor Detection with Hierarchical Social Attention Network". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. CIKM '18. Association for Computing Machinery, pp. 943–951.

Z. Guo, M. Schlichtkrull, and A. Vlachos (2022). "A Survey on Automated Fact-Checking". In: *Transactions of the Association for Computational Linguistics* 10, pp. 178–206.

A. Gupta and M. Klein (2025). "CivicParse: A Benchmark and Pipeline for Structured Online Deliberation". In: *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*.

A. Gutmann and D. F. Thompson (2004). *Why deliberative democracy?* Princeton University Press.

J. Habermas (1991). *The structural transformation of the public sphere: An inquiry into a category of bourgeois society*. MIT press.

J. Habermas (1994). "THREE NORMATIVE MODELS OF DEMOCRACY". In: *Constellations* 1.1, pp. 1–10. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8675.1994.tb00001.x.

R. Hadfi and T. Ito (2022). "Augmented Democratic Deliberation: Can Conversational Agents Boost Deliberation in Social Media?" In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '22. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1794–1798.

L. Hagen, S. Neely, T. E. Keller, R. Scharf, and F. E. Vasquez (2022). "Rise of the Machines? Examining the Influence of Social Bots on a Political Discussion Network". In: *Social Science Computer Review* 40.2, pp. 264–287. eprint: https://doi.org/10.1177/0894439320908190.

A. P. Hamlin and P. Pettit (1991). *The Good Polity: Normative Analysis of the State*.

Y. Hao, L. Dong, F. Wei, and K. Xu (2020). "Investigating Learning Dynamics of BERT Fine-Tuning". In: *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*. Ed. by K.-F. Wong, K. Knight, and H. Wu. Association for Computational Linguistics, pp. 87–92.

W. Harly and A. S. Girsang (2022). "CNN-BERT for measuring agreement between argument in online discussion". In: *International Journal of Web Information Systems* ahead-of-print.

K. He, X. Zhang, S. Ren, and J. Sun (2016). *Deep Residual Learning for Image Recognition*.

T. Heide-Jørgensen, G. Eady, and A. Rasmussen (2025). "Understanding the success and failure of online political debate: Experimental evidence using large language models". In: *Science Advances* 11.30, eadv7864. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.adv7864.

D. Heinbach, L. Wilms, and M. Ziegele (2022). "Effects of empowerment moderation in online discussions: A field experiment with four news outlets". In: *72nd Annual Conference of the International Communication Association (ICA)*.

D. Helbing, S. Mahajan, R. H. Fricker, A. Musso, C. I. Hausladen, C. Carissimo, D. Carpentras, E. Stockinger, J. Argota Sanchez-Vaquerizo, J. C. Yang, M. C. Ballandies, M. Korecki, R. K. Dubey, and E. Pournaras (2023). "Democracy by Design: Perspectives

for Digitally Assisted, Participatory Upgrades of Society". In: *Journal of Computational Science* 71, p. 102061.

J. Hirschberg and C. D. Manning (2015). "Advances in natural language processing". In: *Science* 349.6245, pp. 261–266.

R. Hofstra, A. Michels, and A. Meijer (2023). "Online democratic participation during COVID-19: Assessing implications for inclusivity of citizen engagement". In: *Information Polity* 28.3, pp. 395–410.

M. Horta Ribeiro, J. Cheng, and R. West (2023). "Automated Content Moderation Increases Adherence to Community Guidelines". In: *Proceedings of the ACM Web Conference 2023*. WWW '23. Association for Computing Machinery, pp. 2666–2676.

N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly (2019). "Parameter-Efficient Transfer Learning for NLP". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 2790–2799.

T. Hu, C. Meinel, and H. Yang (2024). "A flexible BERT model enabling width- and depth-dynamic inference". In: *Computer Speech & Language* 87, p. 101646.

H. Hua, X. Li, D. Dou, C. Xu, and J. Luo (2021). "Noise Stability Regularization for Improving BERT Fine-tuning". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou. Association for Computational Linguistics, pp. 3229–3241.

L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu (2025). "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions". In: *ACM Trans. Inf. Syst.* 43.2.

J. Introne, R. Laubacher, G. Olson, and T. Malone (2011). "The Climate CoLab: Large scale model-based collaborative planning". In: *2011 International Conference on Collaboration Technologies and Systems (CTS)*, pp. 40–47.

T. Ito, R. Hadfi, and S. Suzuki (2022). "An agent that facilitates crowd discussion". In: *Group Decision and Negotiation* 31.3, pp. 621–647.

T. Ito, Y. Imi, T. Ito, and E. Hideshima (2014). "COLLAGREE: A faciliator-mediated large-scale consensus support system". In: *Collective Intelligence* 2014, pp. 10–12.

T. Ito, Y. Imi, M. Sato, T. Ito, and E. Hideshima (2015). "Incentive mechanism for managing large-scale internet-based discussions on collagree". In: *Collective Intelligence* 2015.

P. Izsak, M. Berchansky, and O. Levy (2021). "How to Train BERT with an Academic Budget". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language*

Processing. Ed. by M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih. Association for Computational Linguistics, pp. 10644–10652.

C. Janiesch, P. Zschech, and K. Heinrich (2021). "Machine learning and deep learning". In: *Electronic markets* 31.3, pp. 685–695.

M. Jasim, E. Hoque, A. Sarvghad, and N. Mahyar (2021). "CommunityPulse: Facilitating Community Input Analysis by Surfacing Hidden Insights, Reflections, and Priorities". In: *Proceedings of the 2021 ACM Designing Interactive Systems Conference*. DIS '21. Association for Computing Machinery, pp. 846–863.

Q. Jia, Y. Liu, S. Ren, and K. Q. Zhu (2023). "Taxonomy of Abstractive Dialogue Summarization: Scenarios, Approaches, and Future Directions". In: *ACM Comput. Surv.* 56.3.

A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al. (2023). "Mistral 7B". In: *arXiv preprint arXiv:2310.06825*.

Jigsaw (2025). *Perspective API*. URL: `https://perspectiveapi.com/` (visited on 10/17/2025).

M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy (2020). "SpanBERT: Improving Pre-training by Representing and Predicting Spans". In: *Transactions of the Association for Computational Linguistics* 8, pp. 64–77. eprint: `https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl\_a\_00300/1923170/tacl\_a\_00300.pdf`.

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov (2017). "Bag of Tricks for Efficient Text Classification". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Ed. by M. Lapata, P. Blunsom, and A. Koller. Association for Computational Linguistics, pp. 427–431.

G. Karadzhov, T. Stafford, and A. Vlachos (2021). "DeliData: A Dataset for Deliberation in Multi-party Problem Solving". In: *Proceedings of the ACM on Human-Computer Interaction* 7, pp. 1–25.

M. Katsaros, K. Yang, and L. Fratamico (2022). "Reconsidering tweets: Intervening during tweet creation decreases offensive content". In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 16, pp. 477–487.

O. Kelm, T. Neumann, M. Behrendt, M. Brenneis, K. Gerl, S. Marschall, F. Meißner, S. Harmeling, G. Vowe, and M. Ziegele (2023). "How algorithmically curated online environments influence users' political polarization: Results from two experiments with panel data". In: *Computers in Human Behavior Reports* 12, p. 100343.

S. Khan, M. Fazil, A. L. Imoize, B. I. Alabduallah, B. M. Albahlal, S. A. Alajlan, A. Almjally, and T. Siddiqui (2023). "Transformer Architecture-Based Transfer Learning for Politeness Prediction in Conversation". In: *Sustainability* 15.14.

S. Kim, J. Eun, J. Seering, and J. Lee (2021). "Moderator Chatbot for Deliberative Discussion: Effects of Discussion Structure and Discussant Facilitation". In: *Proc. ACM Hum.-Comput. Interact.* 5.CSCW1.

I. Kivlichan, J. Sorensen, J. Elliott, L. Vasserman, M. Görner, and P. Culliton (2020). *Jigsaw Multilingual Toxic Comment Classification*. `https://kaggle.com/competitions/jigsaw-multilingual-toxic-comment-classification`.

M. Klein (2011). *How to harvest collective wisdom on complex problems: An introduction to the mit deliberatorium*.

M. Klein (2015). "A Critical Review of Crowd-Scale Online Deliberation Technologies". In: *Available at SSRN 2652888*.

M. Klein (2025). "How Can AI Help Achieve Effective Deliberation at Scale?" In: *Available at SSRN 5509438*.

V. Kolhatkar, H. Wu, L. Cavasso, E. Francis, K. Shukla, and M. Taboada (2020). "The SFU opinion and comments corpus: A corpus for the analysis of online news comments". In: *Corpus Pragmatics* 4, pp. 155–190.

M. Kolla, S. Salunkhe, E. Chandrasekharan, and K. Saha (2024). "LLM-Mod: Can Large Language Models Assist Content Moderation?" In: *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. CHI EA '24. Association for Computing Machinery.

X. Kong, Z. Liu, C. Chen, S. Liu, Z. Xu, and Q. Tang (2025). "Exploratory study of an AI-supported discussion representational tool for online collaborative learning in a Chinese university". In: *The Internet and Higher Education* 64, p. 100973.

T. Kuo, A. Hernani, and J. Grossklags (2023). "The Unsung Heroes of Facebook Groups Moderation: A Case Study of Moderation Practices and Tools". In: *Proc. ACM Hum.-Comput. Interact.* 7.CSCW1.

M. J. Kushin and K. Kitchener (2009). "Getting political on social network sites: Exploring online political discourse on Facebook". In: *First Monday*.

N. Lago, M. Durieux, J.-A. Pouleur, C. Scoubeau, C. Elsen, and C. Schelings (2019). "Citizen Participation through Digital Platforms: the Challenging Question of Data Processing for Cities." In: *Proceedings of the Eighth International Conference on Smart Cities, Systems, Devices and Technologies*. DGTRE - Région wallonne. Direction générale des Technologies, de la Recherche et de l'Énergie.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut (2020). "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *International Conference on Learning Representations*.

H. Landemore (2024). "Can artificial intelligence bring deliberation to the masses". In: *Conversations in philosophy, law, and politics*, pp. 39–69.

J. Lawrence and C. Reed (2020). "Argument Mining: A Survey". In: *Computational Linguistics* 45.4, pp. 765–818. eprint: `https://direct.mit.edu/coli/article-pdf/45/4/765/1847520/coli\_a\_00364.pdf`.

C. Lee, K. Cho, and W. Kang (2020). "Mixout: Effective Regularization to Finetune Large-scale Pretrained Language Models". In: *International Conference on Learning Representations*.

E. Lee, F. Rustam, P. B. Washington, F. E. Barakaz, W. Aljedaani, and I. Ashraf (2022). "Racism Detection by Analyzing Differential Opinions Through Sentiment Analysis of Tweets Using Stacked Ensemble GCR-NN Model". In: *IEEE Access* 10, pp. 9717–9728.

J. Lehečka, J. Švec, P. Ircing, and L. Šmídl (2020). "Adjusting BERT's pooling layer for large-scale multi-label text classification". In: *International Conference on Text, Speech, and Dialogue*. Springer, pp. 214–221.

H. J. Levesque, E. Davis, and L. Morgenstern (2012). "The Winograd schema challenge". In: *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*. KR'12. AAAI Press, pp. 552–561.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer (2020). "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault. Association for Computational Linguistics, pp. 7871–7880.

M. Liebeck, K. Esau, and S. Conrad (2016). "What to Do with an Airport? Mining Arguments in the German Online Participation Project Tempelhofer Feld". In: *Proceedings of the Third Workshop on Argument Mining (ArgMining2016)*. Association for Computational Linguistics, pp. 144–153.

E. Lieu, J. Cole, and C. Watkins (2022). "Bring Something to the Potluck: A System for Inclusive and Reciprocal Online Discussion". In: *35th International BCS Human-Computer Interaction Conference 35*, pp. 1–6.

C. Lieven (2017). "DIPAS–Towards an integrated GIS-based system for civic participation". In: *Procedia Computer Science* 112, pp. 2473–2485.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov (2019). "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692*.

J. Maillard, V. Karpukhin, F. Petroni, W.-t. Yih, B. Oguz, V. Stoyanov, and G. Ghosh (2021). "Multi-Task Retrieval for Knowledge-Intensive Tasks". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 1098–1111.

J. Mansbridge, J. Bohman, S. Chambers, D. Estlund, A. Føllesdal, A. Fung, C. Lafont, B. Manin, and J. luis Martí (2010). "The Place of Self-Interest and the Role of Power in Deliberative Democracy". In: *Journal of Political Philosophy* 18.1, pp. 64–100.

L. McInnes, J. Healy, N. Saul, and L. Großberger (2018). "UMAP: Uniform Manifold Approximation and Projection". In: *Journal of Open Source Software* 3.29, p. 861.

N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan (2021). "A Survey on Bias and Fairness in Machine Learning". In: *ACM Comput. Surv.* 54.6.

J. Mendonca, A. Lavie, and I. Trancoso (2022). "QualityAdapt: an Automatic Dialogue Quality Estimation Framework". In: *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Ed. by O. Lemon, D. Hakkani-Tur, J. J. Li, A. Ashrafzadeh, D. H. Garcia, M. Alikhani, D. Vandyke, and O. Dušek. Association for Computational Linguistics, pp. 83–90.

R. Mihalcea and P. Tarau (2004). "TextRank: Bringing Order into Text". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pp. 404–411.

A. Mikhaylovskaya (2024). "Enhancing deliberation with digital democratic innovations". In: *Philosophy & Technology* 37.1, p. 3.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. Ed. by C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger. Vol. 26. Curran Associates, Inc.

S.-J. Min (2007). "Online vs. Face-to-Face Deliberation: Effects on Civic Engagement". In: *Journal of Computer-Mediated Communication* 12.4, pp. 1369–1387. eprint: `https://academic.oup.com/jcmc/article-pdf/12/4/1369/22316714/jjcmcom1369.pdf`.

S. Mishra, S. Suryavardan, A. Bhaskar, P. Chopra, A. N. Reganti, P. Patwa, A. Das, T. Chakraborty, A. P. Sheth, A. Ekbal, et al. (2022). "FACTIFY: A Multi-Modal Fact Verification Dataset." In: *DE-FACTIFY@ AAAI*.

A. Misra, B. Ecker, and M. Walker (2016). "Measuring the Similarity of Sentential Arguments in Dialogue". In: *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Ed. by R. Fernandez, W. Minker, G. Carenini, R. Higashinaka, R. Artstein, and A. Gainer. Association for Computational Linguistics, pp. 276–287.

M. Mosbach, M. Andriushchenko, and D. Klakow (2021). "On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines". In: *International Conference on Learning Representations*.

N. Mouter, J. I. Hernandez, and A. V. Itten (2021). "Public participation in crisis policy-making. How 30,000 Dutch citizens advised their government on relaxing COVID-19 lockdown measures". In: *PLoS One* 16.5, e0250614.

S. Muhammed T and S. K. Mathew (2022). "The disaster of misinformation: a review of research in social media". In: *International journal of data science and analytics* 13.4, pp. 271–285.

V. Nair and G. E. Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Omnipress, pp. 807–814.

R. Negrinho (2020). *Sane tikz*.

N. Ng, K. Yee, A. Baevski, M. Ott, M. Auli, and S. Edunov (2019). "Facebook FAIR's WMT19 News Translation Task Submission". In: *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*. Ed. by O. Bojar, R. Chatterjee, C. Federmann, M. Fishel, Y. Graham, B. Haddow, M. Huck, A. J. Yepes, P. Koehn, A. Martins, C. Monz, M. Negri, A. Névéol, M. Neves, M. Post, M. Turchi, and K. Verspoor. Association for Computational Linguistics, pp. 314–319.

S. Niemeyer, F. Veri, J. S. Dryzek, and A. Bächtier (2024). "How Deliberation Happens: Enabling Deliberative Reason". In: *American Political Science Review* 118.1, pp. 345–362.

Z. Nussbaum, J. X. Morris, A. Mulyar, and B. Duderstadt (2025). "Nomic Embed: Training a Reproducible Long Context Text Embedder". In: *Transactions on Machine Learning Research*.

F. Ouyang and L. Zhang (2024). "AI-driven learning analytics applications and tools in computer-supported collaborative learning: A systematic review". In: *Educational Research Review* 44, p. 100616.

Z. Papacharissi (2004). "Democracy online: civility, politeness, and the democratic potential of online political discussion groups". In: *New Media & Society* 6.2, pp. 259–283. eprint: https://doi.org/10.1177/1461444804041444.

J. Park, S. Klingel, C. Cardie, M. Newhart, C. Farina, and J.-J. Vallbé (2012). "Facilitative Moderation for Online Participation in ERulemaking". In: *Proceedings of the 13th Annual International Conference on Digital Government Research*. dg.o '12. Association for Computing Machinery, pp. 173–182.

K. F. Pearson (1901). "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11, pp. 559–572. eprint: https://doi.org/10.1080/14786440109462720.

J. Pennington, R. Socher, and C. D. Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543.

J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych (2021). "AdapterFusion: Non-Destructive Task Composition for Transfer Learning". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Ed. by P. Merlo, J. Tiedemann, and R. Tsarfaty. Association for Computational Linguistics, pp. 487–503.

J. Pfeiffer, A. Rücklé, C. Poth, A. Kamath, I. Vulić, S. Ruder, K. Cho, and I. Gurevych (2020). "AdapterHub: A Framework for Adapting Transformers". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by Q. Liu and D. Schlangen. Association for Computational Linguistics, pp. 46–54.

A. Pokharana and U. Garg (2023). "A Review on diverse algorithms used in the context of Plagiarism Detection". In: *2023 International Conference on Advancement in Computation & Computer Technologies (InCACCT)*, pp. 1–6.

J. Portes, A. Trott, S. Havens, D. KING, A. Venigalla, M. Nadeem, N. Sardana, D. Khudia, and J. Frankle (2023). "MosaicBERT: A Bidirectional Encoder Optimized for Fast Pretraining". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., pp. 3106–3130.

C. Poth, H. Sterz, I. Paul, S. Purkayastha, L. Engländer, T. Imhof, I. Vulić, S. Ruder, I. Gurevych, and J. Pfeiffer (2023). *Adapters: A Unified Library for Parameter-Efficient and Modular Transfer Learning*. arXiv: `2311.11077 [cs.CL]`.

M. Potthast, J. Kiesel, K. Reinartz, J. Bevendorff, and B. Stein (2018). "A Stylometric Inquiry into Hyperpartisan and Fake News". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pp. 231–240.

J. Pougué-Biyong, V. Semenova, A. Matton, R. Han, A. Kim, R. Lambiotte, and D. Farmer (2021). "DEBAGREEMENT: A comment-reply dataset for (dis)agreement detection in online debates". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

D. A. Prabowo and G. Budi Herwanto (2019). "Duplicate Question Detection in Question Answer Website using Convolutional Neural Network". In: *2019 5th International Conference on Science and Technology (ICST)*. Vol. 1, pp. 1–6.

I. Quinto, L. Iandoli, A. de Liddo, et al. (2021). "Designing online collaboration for the individual and social good: A collective argumentation approach". In: *27th Annual Americas Conference on Information Systems, AMCIS 2021*. Association for Information Systems.

A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. (2018). *Improving language understanding by generative pre-training*.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang (2016). "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Ed. by J. Su, K. Duh, and X. Carreras. Association for Computational Linguistics, pp. 2383–2392.

S. Rao, A. K. Verma, and T. Bhatia (2021). "A review on social spam detection: Challenges, open issues, and future directions". In: *Expert Systems with Applications* 186, p. 115742.

S.-A. Rebuffi, H. Bilen, and A. Vedaldi (2017). "Learning multiple visual domains with residual adapters". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc.

N. Reimers and I. Gurevych (2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by K. Inui, J. Jiang, V. Ng, and X. Wan. Association for Computational Linguistics, pp. 3982–3992.

N. Reimers, B. Schiller, T. Beck, J. Daxenberger, C. Stab, and I. Gurevych (2019). "Classification and Clustering of Arguments with Contextualized Word Embeddings". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by A. Korhonen, D. Traum, and L. Màrquez. Association for Computational Linguistics, pp. 567–578.

P. Rescala, M. H. Ribeiro, T. Hu, and R. West (2024). "Can Language Models Recognize Convincing Arguments?" In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Ed. by Y. Al-Onaizan, M. Bansal, and Y.-N. Chen. Association for Computational Linguistics, pp. 8826–8837.

J. Risch, A. Stoll, L. Wilms, and M. Wiegand (2021). "Overview of the GermEval 2021 Shared Task on the Identification of Toxic, Engaging, and Fact-Claiming Comments". In: *Proceedings of the GermEval 2021 Shared Task on the Identification of Toxic, Engaging, and Fact-Claiming Comments*. Ed. by J. Risch, A. Stoll, L. Wilms, and M. Wiegand. Association for Computational Linguistics, pp. 1–12.

Y. M. Rocha, G. A. de Moura, G. A. Desidério, C. H. de Oliveira, F. D. Lourenço, and L. D. de Figueiredo Nicolete (2021). "The impact of fake news on social media and its influence on health during the COVID-19 pandemic: A systematic review". In: *Journal of Public Health*, pp. 1–10.

A. Rogers, O. Kovaleva, and A. Rumshisky (2020). "A Primer in BERTology: What We Know About How BERT Works". In: *Transactions of the Association for Computational Linguistics* 8. Ed. by M. Johnson, B. Roark, and A. Nenkova, pp. 842–866.

J. Romberg and S. Conrad (2021). "Citizen Involvement in Urban Planning - How Can Municipalities Be Supported in Evaluating Public Participation Processes for Mobility

Transitions?" In: *Proceedings of the 8th Workshop on Argument Mining*. Association for Computational Linguistics, pp. 89–99.

J. Romberg and T. Escher (2022). "Automated Topic Categorisation of Citizens' Contributions: Reducing Manual Labelling Efforts Through Active Learning". In: *Electronic Government*. Ed. by M. Janssen, C. Csáki, I. Lindgren, E. Loukis, U. Melin, G. Viale Pereira, M. P. Rodríguez Bolívar, and E. Tambouris. Springer International Publishing, pp. 369–385.

J. Romberg and T. Escher (2023). "Making Sense of Citizens' Input through Artificial Intelligence: A Review of Methods for Computational Text Analysis to Support the Evaluation of Contributions in Public Participation". In: *Digit. Gov.: Res. Pract.*

C. Ruess, C. P. Hoffmann, S. Boulianne, and K. Heger (2023). "Online political participation: the evolution of a concept". In: *Information, Communication & Society* 26.8, pp. 1495–1512. eprint: https://doi.org/10.1080/1369118X.2021.2013919.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf (2019). "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108*.

C. Sasaki, T. Oyama, C. Oshima, S. Kajihara, and K. Nakayama (2021). "Online Discussion Support System with Facilitation Function". In: *International Journal of Advanced Computer Science and Applications* 12.8.

C. Schluger, J. P. Chang, C. Danescu-Niculescu-Mizil, and K. Levy (2022). "Proactive Moderation of Online Discussions: Existing Practices and the Potential for Algorithmic Support". In: *Proc. ACM Hum.-Comput. Interact.* 6.CSCW2.

M. Schütz, J. Boeck, D. Liakhovets, D. Slijepcevic, A. Kirchknopf, M. Hecht, J. Bogensperger, S. Schlarb, A. Schindler, and M. Zeppelzauer (2021). "Automatic Sexism Detection with Multilingual Transformer Models". In: *CoRR* abs/2106.04908.

R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni (2020). "Green AI". In: *Commun. ACM* 63.12, pp. 54–63.

M. F. Scudder (2022). "Measuring Democratic Listening: A Listening Quality Index". In: *Political Research Quarterly* 75.1, pp. 175–187. eprint: https://doi.org/10.1177/1065912921989449.

S. Shalev-Shwartz and S. Ben-David (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

N. A. Sharma, A. S. Ali, and M. A. Kabir (2025). "A review of sentiment analysis: tasks, applications, and deep learning techniques". In: *International journal of data science and analytics* 19.3, pp. 351–388.

N. Shazeer (2020). "Glu variants improve transformer". In: *arXiv preprint arXiv:2002.05202*.

P. Sheth, T. Kumarage, R. Moraffah, A. Chadha, and H. Liu (2023). "PEACE: Cross-Platform Hate Speech Detection - A Causality-Guided Framework". In: *Machine Learning and Knowledge Discovery in Databases: Research Track: European Conference, ECML PKDD 2023, Turin, Italy, September 18–22, 2023, Proceedings, Part I*. Springer-Verlag, pp. 559–575.

B. Shin and M. Rask (2021). "Assessment of Online Deliberative Quality: New Indicators Using Network Analysis and Time-Series Analysis". In: *Sustainability* 13.3.

R. Shortall, A. Itten, M. van der Meer, P. Murukannaiah, and C. Jonker (2022). "Reason against the machine? Future directions for mass online deliberation". In: *Frontiers in Political Science* 4, p. 946589.

J. Simon, T. Bass, V. Boelman, and G. Mulgan (2017). *Digital democracy*.

A. Simonofski, E. Hertoghe, M. Steegmans, M. Snoeck, and Y. Wautelet (2021). "Engaging citizens in the smart city through participation platforms: A framework for public servants and developers". In: *Computers in Human Behavior* 124, p. 106901.

C. Small, M. Bjorkegren, T. Erkkilä, L. Shaw, and C. Megill (2021). "Polis: Scaling Deliberation by Mapping High Dimensional Opinion Spaces". In: *Recerca: Revista de Pensament i Anàlisi* 26.2.

C. T. Small, I. Vendrov, E. Durmus, H. Homaei, E. Barry, J. Cornebise, T. Suzman, D. Ganguli, and C. Megill (2023). "Opportunities and Risks of LLMs for Scalable Deliberation with Polis". In: *CoRR* abs/2306.11932.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts (2013). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Ed. by D. Yarowsky, T. Baldwin, A. Korhonen, K. Livescu, and S. Bethard. Association for Computational Linguistics, pp. 1631–1642.

L. South, M. Schwab, N. Beauchamp, L. Wang, J. Wihbey, and M. A. Borkin (2020). "DebateVis: Visualizing Political Debates for Non-Expert Users". In: *2020 IEEE Visualization Conference (VIS)*, pp. 241–245.

C. Stab, J. Daxenberger, C. Stahlhut, T. Miller, B. Schiller, C. Tauchmann, S. Eger, and I. Gurevych (2018a). "ArgumenText: Searching for Arguments in Heterogeneous Sources". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. Ed. by Y. Liu, T. Paek, and M. Patwardhan. Association for Computational Linguistics, pp. 21–25.

C. Stab and I. Gurevych (2017). "Parsing Argumentation Structures in Persuasive Essays". In: *Computational Linguistics* 43.3, pp. 619–659. eprint: `https://direct.mit.edu/coli/article-pdf/43/3/619/1808352/coli\_a\_00295.pdf`.

C. Stab, T. Miller, B. Schiller, P. Rai, and I. Gurevych (2018b). "Cross-topic Argument Mining from Heterogeneous Sources". In: *Proceedings of the 2018 Conference on Empiri-*

cal Methods in Natural Language Processing. Ed. by E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii. Association for Computational Linguistics, pp. 3664–3674.

F. Stahlberg (2020). "Neural machine translation: A review". In: *Journal of Artificial Intelligence Research* 69, pp. 343–418.

L. Stankevičius and M. Lukoševičius (2024). "Extracting Sentence Embeddings from Pretrained Transformer Models". In: *Applied Sciences* 14.19.

M. R. Steenbergen, A. Bächtiger, M. Spörndli, and J. Steiner (2003). "Measuring political deliberation: A discourse quality index". In: *Comparative European Politics* 1, pp. 21–48.

S. Strauß (2021). "Deep Automation Bias: How to Tackle a Wicked Problem of AI?" In: *Big Data and Cognitive Computing* 5.2, p. 18.

J. Stromer-Galley and A. Wichowski (2011). "Political Discussion Online". In: *The Handbook of Internet Studies*. John Wiley & Sons, Ltd. Chap. 8, pp. 168–187. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781444314861.ch8`.

J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu (2024). "RoFormer: Enhanced transformer with Rotary Position Embedding". In: *Neurocomput.* 568.C.

J. Suler (2004). "The online disinhibition effect". In: *Cyberpsychology & behavior* 7.3, pp. 321–326.

H. Sun, Q. Tu, J. Li, and R. Yan (2023). "ConvNTM: Conversational Neural Topic Model". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 11, pp. 13609–13617.

S. Suryavardan, S. Mishra, P. Patwa, M. Chakraborty, A. Rani, A. Reganti, A. Chadha, A. Das, A. Sheth, M. Chinnakotla, et al. (2022). "Factify 2: A Multimodal Fake News and Satire News Dataset". In: *Proceedings of De-Factify 2: 2nd Workshop on Multimodal Fact Checking and Hate Speech Detection, co-located with AAAI 2023*.

I. Sutskever, O. Vinyals, and Q. V. Le (2014). "Sequence to sequence learning with neural networks". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. MIT Press, pp. 3104–3112.

S. Suzuki, T. Ito, A. Moustafa, and R. Hadfi (2021). "A Node Classification Approach for Dynamically Extracting the Structures of Online Discussions". In: *Advances in Artificial Intelligence*. Ed. by K. Yada, D. Katagami, Y. Takama, T. Ito, A. Abe, E. Sato-Shimokawara, J. Mori, N. Matsumura, and H. Kashima. Springer International Publishing, pp. 1–12.

C. Tan, V. Niculae, C. Danescu-Niculescu-Mizil, and L. Lee (2016). "Winning Arguments: Interaction Dynamics and Persuasion Strategies in Good-faith Online Discussions". In: *Proceedings of the 25th International Conference on World Wide Web*. WWW '16. International World Wide Web Conferences Steering Committee, pp. 613–624.

Y. R. Tausczik and J. W. Pennebaker (2010). "The Psychological Meaning of Words: LIWC and Computerized Text Analysis Methods". In: *Journal of Language and Social Psychology* 29.1, pp. 24–54. eprint: https://doi.org/10.1177/0261927X09351676.

W. L. Taylor (1953). ""Cloze procedure": A new tool for measuring readability". In: *Journalism quarterly* 30.4, pp. 415–433.

N. Team, M. R. Costa-jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard, A. Sun, S. Wang, G. Wenzek, A. Youngblood, B. Akula, L. Barrault, G. M. Gonzalez, P. Hansanti, J. Hoffman, S. Jarrett, K. R. Sadagopan, D. Rowe, S. Spruit, C. Tran, P. Andrews, N. F. Ayan, S. Bhosale, S. Edunov, A. Fan, C. Gao, V. Goswami, F. Guzmán, P. Koehn, A. Mourachko, C. Ropers, S. Saleem, H. Schwenk, and J. Wang (2022). *No Language Left Behind: Scaling Human-Centered Machine Translation*. arXiv: 2207.04672 [cs.CL].

N. Thakur, N. Reimers, J. Daxenberger, and I. Gurevych (2021). "Augmented SBERT: Data Augmentation Method for Improving Bi-Encoders for Pairwise Sentence Scoring Tasks". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou. Association for Computational Linguistics, pp. 296–310.

S. S. Al-Thanyyan and A. M. Azmi (2021). "Automated Text Simplification: A Survey". In: *ACM Comput. Surv.* 54.2.

J. Torell (2006). "Political participation and three theories of democracy: A research inventory and agenda". In: *European Journal of Political Research* 45.5, pp. 787–810. eprint: https://ejpr.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1475-6765.2006.00636.x.

S. Toshniwal, H. Shi, B. Shi, L. Gao, K. Livescu, and K. Gimpel (2020). "A Cross-Task Analysis of Text Span Representations". In: *Proceedings of the 5th Workshop on Representation Learning for NLP*. Ed. by S. Gella, J. Welbl, M. Rei, F. Petroni, P. Lewis, E. Strubell, M. Seo, and H. Hajishirzi. Association for Computational Linguistics, pp. 166–176.

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom (2023). *Llama 2: Open Foundation and Fine-Tuned Chat Models*. arXiv: 2307.09288 [cs.CL].

L. L. Tsai, A. Pentland, A. Braley, N. Chen, J. R. Enríquez, and A. Reuel (2024). "Generative AI for Pro-Democracy Platforms". In: *An MIT Exploration of Generative AI*.

I. Turc, M. Chang, K. Lee, and K. Toutanova (2019). "Well-Read Students Learn Better: The Impact of Student Initialization on Knowledge Distillation". In: *CoRR* abs/1908.08962. arXiv: 1908.08962.

P. D. Turney (2002). "Thumbs up or Thumbs down? Semantic Orientation Applied to Unsupervised Classification of Reviews". In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL '02. Association for Computational Linguistics, pp. 417–424.

C. J. Uhlaner (2015). "Politics and Participation". In: *International Encyclopedia of the Social & Behavioral Sciences (Second Edition)*. Ed. by J. D. Wright. Second Edition. Elsevier, pp. 504–508.

C. Vaccari and A. Valeriani (2018). "Digital Political Talk and Political Participation: Comparing Established and Third Wave Democracies". In: *Sage Open* 8.2, p. 2158244018784986. eprint: https://doi.org/10.1177/2158244018784986.

S. Valenzuela, Y. Kim, and H. Gil de Zúñiga (2011). "Social Networks that Matter: Exploring the Role of Political Discussion for Online Political Participation". In: *International Journal of Public Opinion Research* 24.2, pp. 163–184. eprint: https://academic.oup.com/ijpor/article-pdf/24/2/163/2292246/edr037.pdf.

J. Vamvas and R. Sennrich (2020). "X-stance: A Multilingual Multi-Target Dataset for Stance Detection". In: *Proceedings of the 5th Swiss Text Analytics Conference (SwissText) & 16th Conference on Natural Language Processing (KONVENS)*. CEUR Workshop Proceedings. CEUR-WS.org.

M. Van der Meer, M. Reuver, U. Khurana, L. Krause, and S. Baez Santamaria (2022). "Will It Blend? Mixing Training Paradigms & Prompting for Argument Quality Prediction". In: *Proceedings of the 9th Workshop on Argument Mining*. International Conference on Computational Linguistics, pp. 95–103.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc.

E. M. Vecchi, N. Falk, C. Quensel, I. Jundi, and G. Lapesa (2025). "PerspectiveMod: A Perspectivist Resource for Deliberative Moderation". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Ed. by C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng. Association for Computational Linguistics, pp. 34163–34186.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio (2018). "Graph Attention Networks". In: *International Conference on Learning Representations*.

M. Völske, M. Potthast, S. Syed, and B. Stein (2017). "TL;DR: Mining Reddit to Learn Automatic Summarization". In: *Workshop on New Frontiers in Summarization at EMNLP*

2017. Ed. by G. Carenini, J. Cheung, F. Liu, and L. Wang. Association for Computational Linguistics, pp. 59–63.

H. Wachsmuth, G. Lapesa, E. Cabrio, A. Lauscher, J. Park, E. M. Vecchi, S. Villata, and T. Ziegenbein (2024). "Argument Quality Assessment in the Age of Instruction-Following Large Language Models". In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Ed. by N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue. ELRA and ICCL, pp. 1519–1538.

H. Wachsmuth, S. Syed, and B. Stein (2018). "Retrieval of the Best Counterargument without Prior Topic Knowledge". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by I. Gurevych and Y. Miyao. Association for Computational Linguistics, pp. 241–251.

S. S. Wagner, M. Behrendt, M. Ziegele, and S. Harmeling (2025). "The Power of LLM-Generated Synthetic Data for Stance Detection in Online Political Discussions". In: *The Thirteenth International Conference on Learning Representations*.

M. Walker, J. F. Tree, P. Anand, R. Abbott, and J. King (2012a). "A Corpus for Research on Deliberation and Debate". In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. European Language Resources Association (ELRA), pp. 812–817.

M. A. Walker, P. Anand, R. Abbott, J. E. F. Tree, C. Martell, and J. King (2012b). "That is your evidence?: Classifying stance in online political debate". In: *Decision Support Systems* 53.4, pp. 719–729.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman (2018). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Ed. by T. Linzen, G. Chrupała, and A. Alishahi. Association for Computational Linguistics, pp. 353–355.

W. Y. Wang (2017). ""Liar, Liar Pants on Fire": A New Benchmark Dataset for Fake News Detection". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pp. 422–426.

Y. Wang and N. Diakopoulos (2022). "Highlighting High-quality Content as a Moderation Strategy: The Role of New York Times Picks in Comment Quality and Engagement". In: *Trans. Soc. Comput.* 4.4.

B. Warner, A. Chaffin, B. Clavié, O. Weller, O. Hallström, S. Taghadouini, A. Gallagher, R. Biswas, F. Ladhak, T. Aarsen, G. T. Adams, J. Howard, and I. Poli (2025). "Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by W. Che, J.

Nabende, E. Shutova, and M. T. Pilehvar. Association for Computational Linguistics, pp. 2526–2547.

A. Warstadt, A. Singh, and S. R. Bowman (2019). "Neural Network Acceptability Judgments". In: *Transactions of the Association for Computational Linguistics* 7. Ed. by L. Lee, M. Johnson, B. Roark, and A. Nenkova, pp. 625–641.

A. Williams, N. Nangia, and S. Bowman (2018). "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Ed. by M. Walker, H. Ji, and A. Stent. Association for Computational Linguistics, pp. 1112–1122.

L. Wilms, A. Stoll, M. Ziegele, and K. Gerl (2023). "Bildungsbezogene Biases in crowd-annotierten Daten zur automatischen Klassifikation von konstruktiven und inzivilen Kommentaren (Educational biases in crowd-annotated data for the automatic classification of constructive and incivil comments)". In: *Annual Conference of the Political Communication Devision of the German Association of Communication Science (DGPuK)*.

C. Wohlin (2014). "Guidelines for snowballing in systematic literature studies and a replication in software engineering". In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE '14. Association for Computing Machinery.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. (2016). "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144*.

D. Wyss, S. Beste, and A. Bächtiger (2015). "A decline in the quality of debate? The evolution of cognitive complexity in Swiss parliamentary debates on immigration (1968–2014)". In: *Swiss Political Science Review* 21.4, pp. 636–653.

Y.-G. Xu, X.-P. Qiu, L.-G. Zhou, and X.-J. Huang (2023). "Improving BERT Fine-Tuning via Self-Ensemble and Self-Distillation". In: *J. Comput. Sci. Technol.* 38.4, pp. 853–866.

Y. Xu, X. Zhong, A. J. J. Yepes, and J. H. Lau (2020). "Forget Me Not: Reducing Catastrophic Forgetting for Domain Adaptation in Reading Comprehension". In: *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8.

X. Yan, J. Guo, Y. Lan, and X. Cheng (2013). "A Biterm Topic Model for Short Texts". In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW '13. Association for Computing Machinery, pp. 1445–1456.

H. Yang, J. Callan, and S. Shulman (2006). "Next Steps in Near-Duplicate Detection for ERulemaking". In: *Proceedings of the 2006 International Conference on Digital Government Research*. dg.o '06. Digital Government Society of North America, pp. 239–248.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le (2019). "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc.

S. Yeo, G. Lim, J. Gao, W. Zhang, and S. T. Perrault (2024). "Help Me Reflect: Leveraging Self-Reflection Interface Nudges to Enhance Deliberativeness on Online Deliberation Platforms". In: *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. CHI '24. Association for Computing Machinery.

Z. Yu, L. Otto, D. Assenmacher, and C. Wagner (2024). "A Systematic Review of the Effects of AI-Assisted Moderation on Individuals and Groups". In: *Human-Machine Communication* 9.1, p. 10.

T. Zhang, F. Wu, A. Katiyar, K. Q. Weinberger, and Y. Artzi (2021). "Revisiting Few-sample BERT Fine-tuning". In: *International Conference on Learning Representations*.

Y. Zhang, J. Baldridge, and L. He (2019). "PAWS: Paraphrase Adversaries from Word Scrambling". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Association for Computational Linguistics, pp. 1298–1308.

C. Zhou, C. Qiu, L. Liang, and D. E. Acuna (2025). "Paraphrase Identification With Deep Learning: A Review of Datasets and Methods". In: *IEEE Access* 13, pp. 65797–65822.

S. Zhu, S. Yang, M. A. Bakker, A. Pentland, and J. Pei (2025). "Can AI Truly Represent Your Voice in Deliberations? A Comprehensive Study of Large-Scale Opinion Aggregation with LLMs". In: *arXiv preprint arXiv:2510.05154*.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler (2015). "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books". In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 19–27.

M. Ziegele, O. Quiring, K. Esau, and D. Friess (2020). "Linking News Value Theory With Online Deliberation: How News Factors and Illustration Factors in News Articles Affect the Deliberative Quality of User Discussions in SNS' Comment Sections". In: *Communication Research* 47.6, pp. 860–890. eprint: `https://doi.org/10.1177/0093650218797884`.

H. G. de Zúñiga, M. Barnidge, and T. Diehl (2018). "Political persuasion on social media: A moderated moderation model of political discussion disagreement and civil reasoning". In: *The Information Society* 34.5, pp. 302–315. eprint: `https://doi.org/10.1080/01972243.2018.1497743`.

A Appendix

This appendix presents the implementation details of the two methods introduced in this work. The implementations are structured according to the respective methods and are documented in separate sections. The included code excerpts focus on the essential components required to understand the architecture and experimental procedures.

A.1 Code: ArgueBERT

This section provides the implementation details required to reproduce the pre-training procedure of ArgueBERT, introduced in Chapter 5. In particular, it includes the code used to construct the pre-training dataset as well as the scripts for running the modified pre-training objectives.

Since ArgueBERT is based on adapting BERT's pre-training process to better capture argument similarity, the generation of suitable training data and the design of the training pipeline are central components of the approach. The provided code documents these steps and enables full reproducibility of the experiments presented in this thesis.

The full implementation is available at: `https://github.com/mabehrendt/argueBERT`.

Create Pre-Training Data.

In the following, the code to create the pre-training data for each of the three pre-training tasks, namely graph edge validation, similarity prediction and argument order prediction is provided.

Listing 1: Create Pre-Training Data for Graph Edge Validation

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import collections
import random
import tokenization
import tensorflow as tf

flags = tf.flags

FLAGS = flags.FLAGS

flags.DEFINE_string("input_file", None,
                    "Input raw text file (or comma-separated list of files).")

flags.DEFINE_string(
    "output_file", None,
    "Output TF example file (or comma-separated list of files).")

```

```
flags.DEFINE_string("vocab_file", None,
                    "The vocabulary file that the BERT model was trained on.")

flags.DEFINE_bool(
    "do_lower_case", True,
    "Whether to lower case the input text. Should be True for uncased "
    "models and False for cased models.")

flags.DEFINE_bool(
    "do_whole_word_mask", False,
    "Whether to use whole word masking rather than per-WordPiece masking.")

flags.DEFINE_integer("max_seq_length", 128, "Maximum sequence length.")

flags.DEFINE_integer("max_predictions_per_seq", 20,
                     "Maximum number of masked LM predictions per sequence.")

flags.DEFINE_integer("random_seed", 12345, "Random seed for data generation.")

flags.DEFINE_integer(
    "dupe_factor", 10,
    "Number of times to duplicate the input data (with different masks).")

flags.DEFINE_float("masked_lm_prob", 0.15, "Masked LM probability.")

flags.DEFINE_float(
    "short_seq_prob", 0.1,
    "Probability of creating sequences which are shorter than the "
    "maximum length.")


class TrainingInstance(object):
  """A single training instance (sentence pair)."""

# random_next param by is_edge param
  def __init__(self, tokens, segment_ids, masked_lm_positions, masked_lm_labels,
               is_edge):
    self.tokens = tokens
    self.segment_ids = segment_ids
    self.is_edge = is_edge
    self.masked_lm_positions = masked_lm_positions
    self.masked_lm_labels = masked_lm_labels

  def __str__(self):
    s = ""
    s += "tokens: %s\n" % (" ".join(
        [tokenization.printable_text(x) for x in self.tokens]))
    s += "segment_ids: %s\n" % (" ".join([str(x) for x in self.segment_ids]))
    s += "is_edge: %s\n" % self.is_edge
    s += "masked_lm_positions: %s\n" % (" ".join(
        [str(x) for x in self.masked_lm_positions]))
    s += "masked_lm_labels: %s\n" % (" ".join(
        [tokenization.printable_text(x) for x in self.masked_lm_labels]))
    s += "\n"
    return s

  def __repr__(self):
    return self.__str__()


def write_instance_to_example_files(instances, tokenizer, max_seq_length,
                                    max_predictions_per_seq, output_files):
  """Create TF example files from `TrainingInstance`s."""
  writers = []
  for output_file in output_files:
    writers.append(tf.python_io.TFRecordWriter(output_file))

  writer_index = 0

  total_written = 0
  for (inst_index, instance) in enumerate(instances):
    input_ids = tokenizer.convert_tokens_to_ids(instance.tokens)
    input_mask = [1] * len(input_ids)
    segment_ids = list(instance.segment_ids)
    assert len(input_ids) <= max_seq_length

    while len(input_ids) < max_seq_length:
      input_ids.append(0)
      input_mask.append(0)
      segment_ids.append(0)

    assert len(input_ids) == max_seq_length
    assert len(input_mask) == max_seq_length
    assert len(segment_ids) == max_seq_length
```

```
105
106       masked_lm_positions = list(instance.masked_lm_positions)
107       masked_lm_ids = tokenizer.convert_tokens_to_ids(instance.masked_lm_labels)
108       masked_lm_weights = [1.0] * len(masked_lm_ids)
109
110       while len(masked_lm_positions) < max_predictions_per_seq:
111         masked_lm_positions.append(0)
112         masked_lm_ids.append(0)
113         masked_lm_weights.append(0.0)
114
115       edge_label = 1 if instance.is_edge else 0
116
117       features = collections.OrderedDict()
118       features["input_ids"] = create_int_feature(input_ids)
119       features["input_mask"] = create_int_feature(input_mask)
120       features["segment_ids"] = create_int_feature(segment_ids)
121       features["masked_lm_positions"] = create_int_feature(masked_lm_positions)
122       features["masked_lm_ids"] = create_int_feature(masked_lm_ids)
123       features["masked_lm_weights"] = create_float_feature(masked_lm_weights)
124       features["edge_labels"] = create_int_feature([edge_label])
125
126       tf_example = tf.train.Example(features=tf.train.Features(feature=features))
127
128       writers[writer_index].write(tf_example.SerializeToString())
129       writer_index = (writer_index + 1) % len(writers)
130
131       total_written += 1
132
133       if inst_index < 20:
134         tf.logging.info("*** Example ***")
135         tf.logging.info("tokens: %s" % " ".join(
136             [tokenization.printable_text(x) for x in instance.tokens]))
137
138         for feature_name in features.keys():
139           feature = features[feature_name]
140           values = []
141           if feature.int64_list.value:
142             values = feature.int64_list.value
143           elif feature.float_list.value:
144             values = feature.float_list.value
145           tf.logging.info(
146               "%s: %s" % (feature_name, " ".join([str(x) for x in values])))
147
148     for writer in writers:
149       writer.close()
150
151     tf.logging.info("Wrote %d total instances", total_written)
152
153
154   def create_int_feature(values):
155     feature = tf.train.Feature(int64_list=tf.train.Int64List(value=list(values)))
156     return feature
157
158
159   def create_float_feature(values):
160     feature = tf.train.Feature(float_list=tf.train.FloatList(value=list(values)))
161     return feature
162
163
164   def create_training_instances(input_files, tokenizer, max_seq_length,
165                                 dupe_factor, short_seq_prob, masked_lm_prob,
166                                 max_predictions_per_seq, rng):
167     """Create `TrainingInstance`s from raw text."""
168     all_documents = [[]]
169
170     # Input file format:
171     # (1) One sentence per line. These should ideally be actual sentences, not
172     # entire paragraphs or arbitrary spans of text. (Because we use the
173     # sentence boundaries for the "next sentence prediction" task).
174     # (2) Blank lines between documents. Document boundaries are needed so
175     # that the "next sentence prediction" task doesn't span between documents.
176     for input_file in input_files:
177       with tf.gfile.GFile(input_file, "r") as reader:
178         while True:
179           line = tokenization.convert_to_unicode(reader.readline())
180           if not line:
181             break
182           line = line.strip()
183
184           # Empty lines are used as document delimiters
185           if not line:
186             all_documents.append([])
187           tokens = tokenizer.tokenize(line)
188           if tokens:
```

```
            all_documents[-1].append(tokens)

    # Remove empty documents
    all_documents = [x for x in all_documents if x]
    rng.shuffle(all_documents)

    vocab_words = list(tokenizer.vocab.keys())
    instances = []
    for _ in range(dupe_factor):
      for document_index in range(len(all_documents)):
        instances.extend(
            create_instances_from_document(
                all_documents, document_index, max_seq_length, short_seq_prob,
                masked_lm_prob, max_predictions_per_seq, vocab_words, rng))

    rng.shuffle(instances)
    return instances


def create_instances_from_document(
    all_documents, document_index, max_seq_length, short_seq_prob,
    masked_lm_prob, max_predictions_per_seq, vocab_words, rng):
    """Creates `TrainingInstance`s for a single document."""
    document = all_documents[document_index]

    # Account for [CLS], [SEP], [SEP]
    max_num_tokens = max_seq_length - 3

    # For the task of predicting if an edge is existing, all sentence pairs are own documents.
    # The first sentence is always sentence A and in 50% of the cases sentence B is a sentence that is adjacent to sentence A in an argument
    # in 50% of the cases sentence B is a sentence from another random document (not adjacent to sentence A).
    # The same is applied for the other way round (Second sentence is sentence A.)
    instances = []
    sentences = []
    sentences.extend(document)
    i = 1
    half_max_tokens = int(max_num_tokens/2)
    while i < len(document):
      tokens_a = sentences[0]
      # Edge existing?
      tokens_b = sentences[i]
      if (len(tokens_a)+len(tokens_b)) > max_num_tokens:
          tokens_a = sentences[0][:half_max_tokens]
          tokens_b = sentences[i][:half_max_tokens]
          print(len(tokens_a)+len(tokens_b))
      is_edge = True
      i += 1
      if rng.random() < 0.5:
        i -= 1
        is_edge = False

          # This should rarely go for more than one iteration for large
          # corpora. However, just to be careful, we try to make sure that
          # the random document is not the same as the document
          # we're processing.
        for _ in range(10):
          random_document_index = rng.randint(0, len(all_documents) - 1)
          if random_document_index != document_index:
            break

        random_document = all_documents[random_document_index]
        tokens_b = random_document[rng.randint(0,len(random_document) - 1)]
        if (len(tokens_a)+len(tokens_b)) > max_num_tokens:
            tokens_a = sentences[0][:half_max_tokens]
            tokens_b = random_document[rng.randint(0,len(random_document) - 1)][:half_max_tokens]
            print(len(tokens_a)+len(tokens_b))

      assert len(tokens_a) >= 1
      assert len(tokens_b) >= 1

      tokens = []
      segment_ids = []
      tokens.append("[CLS]")
      segment_ids.append(0)
      for token in tokens_a:
        tokens.append(token)
        segment_ids.append(0)

      tokens.append("[SEP]")
      segment_ids.append(0)

      for token in tokens_b:
        tokens.append(token)
        segment_ids.append(1)
```

```
    tokens.append("[SEP]")
    segment_ids.append(1)

    (tokens, masked_lm_positions,
      masked_lm_labels) = create_masked_lm_predictions(
        tokens, masked_lm_prob, max_predictions_per_seq, vocab_words, rng)
    instance = TrainingInstance(
      tokens=tokens,
      segment_ids=segment_ids,
      is_edge=is_edge,
      masked_lm_positions=masked_lm_positions,
      masked_lm_labels=masked_lm_labels)
    instances.append(instance)
  return instances


MaskedLmInstance = collections.namedtuple("MaskedLmInstance",
                                          ["index", "label"])


def create_masked_lm_predictions(tokens, masked_lm_prob,
                                 max_predictions_per_seq, vocab_words, rng):
  """Creates the predictions for the masked LM objective."""

  cand_indexes = []
  for (i, token) in enumerate(tokens):
    if token == "[CLS]" or token == "[SEP]":
      continue
    # Whole Word Masking means that if we mask all of the wordpieces
    # corresponding to an original word. When a word has been split into
    # WordPieces, the first token does not have any marker and any subsequence
    # tokens are prefixed with ##. So whenever we see the ## token, we
    # append it to the previous set of word indexes.
    #
    # Note that Whole Word Masking does *not* change the training code
    # at all -- we still predict each WordPiece independently, softmaxed
    # over the entire vocabulary.
    if (FLAGS.do_whole_word_mask and len(cand_indexes) >= 1 and
        token.startswith("##")):
      cand_indexes[-1].append(i)
    else:
      cand_indexes.append([i])

  rng.shuffle(cand_indexes)

  output_tokens = list(tokens)

  num_to_predict = min(max_predictions_per_seq,
                       max(1, int(round(len(tokens) * masked_lm_prob))))

  masked_lms = []
  covered_indexes = set()
  for index_set in cand_indexes:
    if len(masked_lms) >= num_to_predict:
      break
    # If adding a whole-word mask would exceed the maximum number of
    # predictions, then just skip this candidate.
    if len(masked_lms) + len(index_set) > num_to_predict:
      continue
    is_any_index_covered = False
    for index in index_set:
      if index in covered_indexes:
        is_any_index_covered = True
        break
    if is_any_index_covered:
      continue
    for index in index_set:
      covered_indexes.add(index)

      masked_token = None
      # 80% of the time, replace with [MASK]
      if rng.random() < 0.8:
        masked_token = "[MASK]"
      else:
        # 10% of the time, keep original
        if rng.random() < 0.5:
          masked_token = tokens[index]
        # 10% of the time, replace with random word
        else:
          masked_token = vocab_words[rng.randint(0, len(vocab_words) - 1)]

      output_tokens[index] = masked_token

      masked_lms.append(MaskedLmInstance(index=index, label=tokens[index]))
```

```
  assert len(masked_lms) <= num_to_predict
  masked_lms = sorted(masked_lms, key=lambda x: x.index)

  masked_lm_positions = []
  masked_lm_labels = []
  for p in masked_lms:
    masked_lm_positions.append(p.index)
    masked_lm_labels.append(p.label)

  return (output_tokens, masked_lm_positions, masked_lm_labels)


def truncate_seq_pair(tokens_a, tokens_b, max_num_tokens, rng):
  """Truncates a pair of sequences to a maximum sequence length."""
  while True:
    total_length = len(tokens_a) + len(tokens_b)
    if total_length <= max_num_tokens:
      break

    trunc_tokens = tokens_a if len(tokens_a) > len(tokens_b) else tokens_b
    assert len(trunc_tokens) >= 1

    # We want to sometimes truncate from the front and sometimes from the
    # back to add more randomness and avoid biases.
    if rng.random() < 0.5:
      del trunc_tokens[0]
    else:
      trunc_tokens.pop()


def main(_):
  tf.logging.set_verbosity(tf.logging.INFO)

  tokenizer = tokenization.FullTokenizer(
      vocab_file=FLAGS.vocab_file, do_lower_case=FLAGS.do_lower_case)

  input_files = []
  for input_pattern in FLAGS.input_file.split(","):
    input_files.extend(tf.gfile.Glob(input_pattern))

  tf.logging.info("*** Reading from input files ***")
  for input_file in input_files:
    tf.logging.info("  %s", input_file)

  rng = random.Random(FLAGS.random_seed)
  instances = create_training_instances(
      input_files, tokenizer, FLAGS.max_seq_length, FLAGS.dupe_factor,
      FLAGS.short_seq_prob, FLAGS.masked_lm_prob, FLAGS.max_predictions_per_seq,
      rng)
  output_files = FLAGS.output_file.split(",")
  tf.logging.info("*** Writing to output files ***")
  for output_file in output_files:
    tf.logging.info("  %s", output_file)

  write_instance_to_example_files(instances, tokenizer, FLAGS.max_seq_length,
                                  FLAGS.max_predictions_per_seq, output_files)


if __name__ == "__main__":
  flags.mark_flag_as_required("input_file")
  flags.mark_flag_as_required("output_file")
  flags.mark_flag_as_required("vocab_file")
  tf.app.run()
```

Listing 2: Create Pre-Training Data for Similarity Prediction

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import collections
import random
import tokenization
import tensorflow as tf

flags = tf.flags

FLAGS = flags.FLAGS

flags.DEFINE_string("input_file", None,
                    "Input raw text file (or comma-separated list of files).")

flags.DEFINE_string(
```

```
    "output_file", None,
    "Output TF example file (or comma-separated list of files).")

flags.DEFINE_string("vocab_file", None,
                    "The vocabulary file that the BERT model was trained on.")

flags.DEFINE_bool(
    "do_lower_case", True,
    "Whether to lower case the input text. Should be True for uncased "
    "models and False for cased models.")

flags.DEFINE_bool(
    "do_whole_word_mask", False,
    "Whether to use whole word masking rather than per-WordPiece masking.")

flags.DEFINE_integer("max_seq_length", 128, "Maximum sequence length.")

flags.DEFINE_integer("max_predictions_per_seq", 20,
                     "Maximum number of masked LM predictions per sequence.")

flags.DEFINE_integer("random_seed", 12345, "Random seed for data generation.")

flags.DEFINE_integer(
    "dupe_factor", 10,
    "Number of times to duplicate the input data (with different masks).")

flags.DEFINE_float("masked_lm_prob", 0.15, "Masked LM probability.")

flags.DEFINE_float(
    "short_seq_prob", 0.1,
    "Probability of creating sequences which are shorter than the "
    "maximum length.")


class TrainingInstance(object):
  """A single training instance (sentence pair)."""

# random_next param by is_similar param
  def __init__(self, tokens, segment_ids, masked_lm_positions, masked_lm_labels,
               is_similar):
    self.tokens = tokens
    self.segment_ids = segment_ids
    self.is_similar = is_similar
    self.masked_lm_positions = masked_lm_positions
    self.masked_lm_labels = masked_lm_labels

  def __str__(self):
    s = ""
    s += "tokens: %s\n" % (" ".join(
        [tokenization.printable_text(x) for x in self.tokens]))
    s += "segment_ids: %s\n" % (" ".join([str(x) for x in self.segment_ids]))
    s += "is_similar: %s\n" % self.is_similar
    s += "masked_lm_positions: %s\n" % (" ".join(
        [str(x) for x in self.masked_lm_positions]))
    s += "masked_lm_labels: %s\n" % (" ".join(
        [tokenization.printable_text(x) for x in self.masked_lm_labels]))
    s += "\n"
    return s

  def __repr__(self):
    return self.__str__()


def write_instance_to_example_files(instances, tokenizer, max_seq_length,
                                    max_predictions_per_seq, output_files):
  """Create TF example files from `TrainingInstance`s."""
  writers = []
  for output_file in output_files:
    writers.append(tf.python_io.TFRecordWriter(output_file))

  writer_index = 0

  total_written = 0
  for (inst_index, instance) in enumerate(instances):
    input_ids = tokenizer.convert_tokens_to_ids(instance.tokens)
    input_mask = [1] * len(input_ids)
    segment_ids = list(instance.segment_ids)
    assert len(input_ids) <= max_seq_length

    while len(input_ids) < max_seq_length:
      input_ids.append(0)
      input_mask.append(0)
      segment_ids.append(0)

```

```
    assert len(input_ids) == max_seq_length
    assert len(input_mask) == max_seq_length
    assert len(segment_ids) == max_seq_length

    masked_lm_positions = list(instance.masked_lm_positions)
    masked_lm_ids = tokenizer.convert_tokens_to_ids(instance.masked_lm_labels)
    masked_lm_weights = [1.0] * len(masked_lm_ids)

    while len(masked_lm_positions) < max_predictions_per_seq:
      masked_lm_positions.append(0)
      masked_lm_ids.append(0)
      masked_lm_weights.append(0.0)

    similar_label = 1 if instance.is_similar else 0

    features = collections.OrderedDict()
    features["input_ids"] = create_int_feature(input_ids)
    features["input_mask"] = create_int_feature(input_mask)
    features["segment_ids"] = create_int_feature(segment_ids)
    features["masked_lm_positions"] = create_int_feature(masked_lm_positions)
    features["masked_lm_ids"] = create_int_feature(masked_lm_ids)
    features["masked_lm_weights"] = create_float_feature(masked_lm_weights)
    features["similar_labels"] = create_int_feature([similar_label])

    tf_example = tf.train.Example(features=tf.train.Features(feature=features))

    writers[writer_index].write(tf_example.SerializeToString())
    writer_index = (writer_index + 1) % len(writers)

    total_written += 1

    if inst_index < 20:
      tf.logging.info("*** Example ***")
      tf.logging.info("tokens: %s" % " ".join(
          [tokenization.printable_text(x) for x in instance.tokens]))

      for feature_name in features.keys():
        feature = features[feature_name]
        values = []
        if feature.int64_list.value:
          values = feature.int64_list.value
        elif feature.float_list.value:
          values = feature.float_list.value
        tf.logging.info(
            "%s: %s" % (feature_name, " ".join([str(x) for x in values])))

  for writer in writers:
    writer.close()

  tf.logging.info("Wrote %d total instances", total_written)


def create_int_feature(values):
  feature = tf.train.Feature(int64_list=tf.train.Int64List(value=list(values)))
  return feature


def create_float_feature(values):
  feature = tf.train.Feature(float_list=tf.train.FloatList(value=list(values)))
  return feature


def create_training_instances(input_files, tokenizer, max_seq_length,
                              dupe_factor, short_seq_prob, masked_lm_prob,
                              max_predictions_per_seq, rng):
  """Create `TrainingInstance`s from raw text."""
  all_documents = [[]]

  # Input file format:
  # (1) One sentence per line. These should ideally be actual sentences, not
  # entire paragraphs or arbitrary spans of text. (Because we use the
  # sentence boundaries for the "next sentence prediction" task).
  # (2) Blank lines between documents. Document boundaries are needed so
  # that the "next sentence prediction" task doesn't span between documents.
  for input_file in input_files:
    with tf.gfile.GFile(input_file, "r") as reader:
      while True:
        line = tokenization.convert_to_unicode(reader.readline())
        if not line:
          break
        line = line.strip()

        # Empty lines are used as document delimiters
        if not line:
```

```
            all_documents.append([])
          tokens = tokenizer.tokenize(line)
          if tokens:
            all_documents[-1].append(tokens)

    # Remove empty documents
    all_documents = [x for x in all_documents if x]
    rng.shuffle(all_documents)

    vocab_words = list(tokenizer.vocab.keys())
    instances = []
    for _ in range(dupe_factor):
      for document_index in range(len(all_documents)):
        instances.extend(
            create_instances_from_document(
                all_documents, document_index, max_seq_length, short_seq_prob,
                masked_lm_prob, max_predictions_per_seq, vocab_words, rng))

    rng.shuffle(instances)
    return instances


def create_instances_from_document(
    all_documents, document_index, max_seq_length, short_seq_prob,
    masked_lm_prob, max_predictions_per_seq, vocab_words, rng):
    """Creates `TrainingInstance`s for a single document."""
    document = all_documents[document_index]

    # Account for [CLS], [SEP], [SEP]
    max_num_tokens = max_seq_length - 3

    # For the task of predicting similarity, all sentence pairs are own documents.
    # The first sentence is always sentence A and in 50% of the cases sentence B is a paraphrased version of sentence A
    # in 50% of the cases sentence B is a sentence from another random document (no paraphrase of sentence A).
    # The same is applied for the other way round (Second sentence is sentence A.)
    instances = []
    sentences = []
    sentences.extend(document)

    for i in range(0,len(document)-1,2):
      tokens_a = sentences[i][:int(max_num_tokens/2)]

      # Similarity
      tokens_b = sentences[i+1][:int(max_num_tokens/2)]
      is_similar = True
      if rng.random() < 0.5:
        is_similar = False

          # This should rarely go for more than one iteration for large
          # corpora. However, just to be careful, we try to make sure that
          # the random document is not the same as the document
          # we're processing.
        for _ in range(10):
          random_line_index = rng.randint(0, len(document) - 1)
          if random_line_index != i and random_line_index != i+1:
            break

        random_line = document[random_line_index]
        tokens_b = random_line

      assert len(tokens_a) >= 1
      assert len(tokens_b) >= 1

      tokens = []
      segment_ids = []
      tokens.append("[CLS]")
      segment_ids.append(0)
      for token in tokens_a:
        tokens.append(token)
        segment_ids.append(0)

      tokens.append("[SEP]")
      segment_ids.append(0)

      for token in tokens_b:
        tokens.append(token)
        segment_ids.append(1)
      tokens.append("[SEP]")
      segment_ids.append(1)

      (tokens, masked_lm_positions,
        masked_lm_labels) = create_masked_lm_predictions(
          tokens, masked_lm_prob, max_predictions_per_seq, vocab_words, rng)
      instance = TrainingInstance(
```

```
      tokens=tokens,
      segment_ids=segment_ids,
      is_similar=is_similar,
      masked_lm_positions=masked_lm_positions,
      masked_lm_labels=masked_lm_labels)
    instances.append(instance)
  return instances


MaskedLmInstance = collections.namedtuple("MaskedLmInstance",
                                          ["index", "label"])


def create_masked_lm_predictions(tokens, masked_lm_prob,
                                 max_predictions_per_seq, vocab_words, rng):
  """Creates the predictions for the masked LM objective."""

  cand_indexes = []
  for (i, token) in enumerate(tokens):
    if token == "[CLS]" or token == "[SEP]":
      continue
    # Whole Word Masking means that if we mask all of the wordpieces
    # corresponding to an original word. When a word has been split into
    # WordPieces, the first token does not have any marker and any subsequence
    # tokens are prefixed with ##. So whenever we see the ## token, we
    # append it to the previous set of word indexes.
    #
    # Note that Whole Word Masking does *not* change the training code
    # at all -- we still predict each WordPiece independently, softmaxed
    # over the entire vocabulary.
    if (FLAGS.do_whole_word_mask and len(cand_indexes) >= 1 and
        token.startswith("##")):
      cand_indexes[-1].append(i)
    else:
      cand_indexes.append([i])

  rng.shuffle(cand_indexes)

  output_tokens = list(tokens)

  num_to_predict = min(max_predictions_per_seq,
                       max(1, int(round(len(tokens) * masked_lm_prob))))

  masked_lms = []
  covered_indexes = set()
  for index_set in cand_indexes:
    if len(masked_lms) >= num_to_predict:
      break
    # If adding a whole-word mask would exceed the maximum number of
    # predictions, then just skip this candidate.
    if len(masked_lms) + len(index_set) > num_to_predict:
      continue
    is_any_index_covered = False
    for index in index_set:
      if index in covered_indexes:
        is_any_index_covered = True
        break
    if is_any_index_covered:
      continue
    for index in index_set:
      covered_indexes.add(index)

      masked_token = None
      # 80% of the time, replace with [MASK]
      if rng.random() < 0.8:
        masked_token = "[MASK]"
      else:
        # 10% of the time, keep original
        if rng.random() < 0.5:
          masked_token = tokens[index]
        # 10% of the time, replace with random word
        else:
          masked_token = vocab_words[rng.randint(0, len(vocab_words) - 1)]

      output_tokens[index] = masked_token

      masked_lms.append(MaskedLmInstance(index=index, label=tokens[index]))
  assert len(masked_lms) <= num_to_predict
  masked_lms = sorted(masked_lms, key=lambda x: x.index)

  masked_lm_positions = []
  masked_lm_labels = []
  for p in masked_lms:
    masked_lm_positions.append(p.index)
```

```
    masked_lm_labels.append(p.label)

  return (output_tokens, masked_lm_positions, masked_lm_labels)


def truncate_seq_pair(tokens_a, tokens_b, max_num_tokens, rng):
  """Truncates a pair of sequences to a maximum sequence length."""
  while True:
    total_length = len(tokens_a) + len(tokens_b)
    if total_length <= max_num_tokens:
      break

    trunc_tokens = tokens_a if len(tokens_a) > len(tokens_b) else tokens_b
    assert len(trunc_tokens) >= 1

    # We want to sometimes truncate from the front and sometimes from the
    # back to add more randomness and avoid biases.
    if rng.random() < 0.5:
      del trunc_tokens[0]
    else:
      trunc_tokens.pop()


def main(_):
  tf.logging.set_verbosity(tf.logging.INFO)

  tokenizer = tokenization.FullTokenizer(
      vocab_file=FLAGS.vocab_file, do_lower_case=FLAGS.do_lower_case)

  input_files = []
  for input_pattern in FLAGS.input_file.split(","):
    input_files.extend(tf.gfile.Glob(input_pattern))

  tf.logging.info("*** Reading from input files ***")
  for input_file in input_files:
    tf.logging.info("  %s", input_file)

  rng = random.Random(FLAGS.random_seed)
  instances = create_training_instances(
      input_files, tokenizer, FLAGS.max_seq_length, FLAGS.dupe_factor,
      FLAGS.short_seq_prob, FLAGS.masked_lm_prob, FLAGS.max_predictions_per_seq,
      rng)
  output_files = FLAGS.output_file.split(",")
  tf.logging.info("*** Writing to output files ***")
  for output_file in output_files:
    tf.logging.info("  %s", output_file)

  write_instance_to_example_files(instances, tokenizer, FLAGS.max_seq_length,
                                  FLAGS.max_predictions_per_seq, output_files)


if __name__ == "__main__":
  flags.mark_flag_as_required("input_file")
  flags.mark_flag_as_required("output_file")
  flags.mark_flag_as_required("vocab_file")
  tf.app.run()
```

Listing 3: Create Pre-Training Data for Argument Order Prediction

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import collections
import random
import tokenization
import tensorflow as tf

flags = tf.flags

FLAGS = flags.FLAGS

flags.DEFINE_string("input_file", None,
                    "Input raw text file (or comma-separated list of files).")

flags.DEFINE_string(
    "output_file", None,
    "Output TF example file (or comma-separated list of files).")

flags.DEFINE_string("vocab_file", None,
                    "The vocabulary file that the BERT model was trained on.")

flags.DEFINE_bool(
```

```
    "do_lower_case", True,
    "Whether to lower case the input text. Should be True for uncased "
    "models and False for cased models.")

flags.DEFINE_bool(
    "do_whole_word_mask", False,
    "Whether to use whole word masking rather than per-WordPiece masking.")

flags.DEFINE_integer("max_seq_length", 128, "Maximum sequence length.")

flags.DEFINE_integer("max_predictions_per_seq", 20,
                     "Maximum number of masked LM predictions per sequence.")

flags.DEFINE_integer("random_seed", 12345, "Random seed for data generation.")

flags.DEFINE_integer(
    "dupe_factor", 10,
    "Number of times to duplicate the input data (with different masks).")

flags.DEFINE_float("masked_lm_prob", 0.15, "Masked LM probability.")

flags.DEFINE_float(
    "short_seq_prob", 0.1,
    "Probability of creating sequences which are shorter than the "
    "maximum length.")


class TrainingInstance(object):
  """A single training instance (sentence pair)."""

  def __init__(self, tokens, segment_ids, masked_lm_positions, masked_lm_labels,
               is_sorted):
    self.tokens = tokens
    self.segment_ids = segment_ids
    self.is_sorted = is_sorted
    self.masked_lm_positions = masked_lm_positions
    self.masked_lm_labels = masked_lm_labels

  def __str__(self):
    s = ""
    s += "tokens: %s\n" % (" ".join(
        [tokenization.printable_text(x) for x in self.tokens]))
    s += "segment_ids: %s\n" % (" ".join([str(x) for x in self.segment_ids]))
    s += "is_sorted: %s\n" % self.is_sorted
    s += "masked_lm_positions: %s\n" % (" ".join(
        [str(x) for x in self.masked_lm_positions]))
    s += "masked_lm_labels: %s\n" % (" ".join(
        [tokenization.printable_text(x) for x in self.masked_lm_labels]))
    s += "\n"
    return s

  def __repr__(self):
    return self.__str__()


def write_instance_to_example_files(instances, tokenizer, max_seq_length,
                                    max_predictions_per_seq, output_files):
  """Create TF example files from `TrainingInstance`s."""
  writers = []
  for output_file in output_files:
    writers.append(tf.python_io.TFRecordWriter(output_file))

  writer_index = 0

  total_written = 0
  for (inst_index, instance) in enumerate(instances):
    input_ids = tokenizer.convert_tokens_to_ids(instance.tokens)
    input_mask = [1] * len(input_ids)
    segment_ids = list(instance.segment_ids)
    assert len(input_ids) <= max_seq_length*2

    while len(input_ids) < max_seq_length*2:
      input_ids.append(0)
      input_mask.append(0)
      segment_ids.append(0)

    assert len(input_ids) == max_seq_length*2
    assert len(input_mask) == max_seq_length*2
    assert len(segment_ids) == max_seq_length*2

    masked_lm_positions = list(instance.masked_lm_positions)
    masked_lm_ids = tokenizer.convert_tokens_to_ids(instance.masked_lm_labels)
    masked_lm_weights = [1.0] * len(masked_lm_ids)

```

```
    while len(masked_lm_positions) < max_predictions_per_seq:
      masked_lm_positions.append(0)
      masked_lm_ids.append(0)
      masked_lm_weights.append(0.0)

    is_sorted_label = 1 if instance.is_sorted else 0

    features = collections.OrderedDict()
    features["input_ids"] = create_int_feature(input_ids)
    features["input_mask"] = create_int_feature(input_mask)
    features["segment_ids"] = create_int_feature(segment_ids)
    features["masked_lm_positions"] = create_int_feature(masked_lm_positions)
    features["masked_lm_ids"] = create_int_feature(masked_lm_ids)
    features["masked_lm_weights"] = create_float_feature(masked_lm_weights)
    features["is_sorted_labels"] = create_int_feature([is_sorted_label])

    tf_example = tf.train.Example(features=tf.train.Features(feature=features))

    writers[writer_index].write(tf_example.SerializeToString())
    writer_index = (writer_index + 1) % len(writers)

    total_written += 1

    if inst_index < 20:
      tf.logging.info("*** Example ***")
      tf.logging.info("tokens: %s" % " ".join(
          [tokenization.printable_text(x) for x in instance.tokens]))

      for feature_name in features.keys():
        feature = features[feature_name]
        values = []
        if feature.int64_list.value:
          values = feature.int64_list.value
        elif feature.float_list.value:
          values = feature.float_list.value
        tf.logging.info(
            "%s: %s" % (feature_name, " ".join([str(x) for x in values])))

  for writer in writers:
    writer.close()

  tf.logging.info("Wrote %d total instances", total_written)


def create_int_feature(values):
  feature = tf.train.Feature(int64_list=tf.train.Int64List(value=list(values)))
  return feature


def create_float_feature(values):
  feature = tf.train.Feature(float_list=tf.train.FloatList(value=list(values)))
  return feature


def create_training_instances(input_files, tokenizer, max_seq_length,
                              dupe_factor, short_seq_prob, masked_lm_prob,
                              max_predictions_per_seq, rng):
  """Create `TrainingInstance`s from raw text."""
  all_documents = [[]]

  # Input file format:
  # (1) One post per line. These could be
  # entire paragraphs or arbitrary spans of text. (Because we use the
  # sentence boundaries for the "next sentence prediction" task).
  # (2) Blank lines between documents. Document boundaries are needed so
  # that the "next sentence prediction" task doesn't span between documents.
  for input_file in input_files:
    with tf.gfile.GFile(input_file, "r") as reader:
      while True:
        line = tokenization.convert_to_unicode(reader.readline())
        if not line:
          break
        line = line.strip()

        # Empty lines are used as document delimiters
        if not line:
          all_documents.append([])
        tokens = tokenizer.tokenize(line)
        if tokens:
          all_documents[-1].append(tokens)

  # Remove empty documents
  all_documents = [x for x in all_documents if x]
  rng.shuffle(all_documents)
```

```
193
194     vocab_words = list(tokenizer.vocab.keys())
195     instances = []
196     for _ in range(dupe_factor):
197       for document_index in range(len(all_documents)):
198         instances.extend(
199             create_instances_from_document(
200                 all_documents, document_index, max_seq_length, short_seq_prob,
201                 masked_lm_prob, max_predictions_per_seq, vocab_words, rng))
202
203     rng.shuffle(instances)
204     return instances
205
206
207   def create_instances_from_document(
208       all_documents, document_index, max_seq_length, short_seq_prob,
209       masked_lm_prob, max_predictions_per_seq, vocab_words, rng):
210     """Creates `TrainingInstance`s for a single document."""
211     document = all_documents[document_index]
212     # Account for [CLS], [SEP], [SEP]
213     max_num_tokens = max_seq_length - 3
214
215     # We *usually* want to fill up the entire sequence since we are padding
216     # to `max_seq_length` anyways, so short sequences are generally wasted
217     # computation. However, we *sometimes*
218     # (i.e., short_seq_prob == 0.1 == 10% of the time) want to use shorter
219     # sequences to minimize the mismatch between pre-training and fine-tuning.
220     # The `target_seq_length` is just a rough target however, whereas
221     # `max_seq_length` is a hard limit.
222     target_seq_length = max_num_tokens
223     #if rng.random() < short_seq_prob:
224     #  target_seq_length = rng.randint(2, max_num_tokens)
225
226     # We DON'T just concatenate all of the tokens from a document into a long
227     # sequence and choose an arbitrary split point because this would make the
228     # next sentence prediction task too easy. Instead, we split the input into
229     # segments "A" and "B" based on the actual "sentences" provided by the user
230     # input.
231
232     instances = []
233     for i in range(0,len(document)-1,2):
234       tokens_a = document[i][:max_num_tokens]
235       tokens_b = document[i+1][:max_num_tokens]
236       is_sorted = True
237       if rng.random() < 0.5:
238         is_sorted = False
239         tokens_a,tokens_b = tokens_b, tokens_a
240
241       assert len(tokens_a) >= 1
242       assert len(tokens_b) >= 1
243
244       tokens = []
245       segment_ids = []
246       tokens.append("[CLS]")
247       segment_ids.append(0)
248       for token in tokens_a:
249         tokens.append(token)
250         segment_ids.append(0)
251
252       tokens.append("[SEP]")
253       segment_ids.append(0)
254
255       for token in tokens_b:
256         tokens.append(token)
257         segment_ids.append(1)
258       tokens.append("[SEP]")
259       segment_ids.append(1)
260
261       (tokens, masked_lm_positions,
262        masked_lm_labels) = create_masked_lm_predictions(
263            tokens, masked_lm_prob, max_predictions_per_seq, vocab_words, rng)
264       instance = TrainingInstance(
265           tokens=tokens,
266           segment_ids=segment_ids,
267           is_sorted=is_sorted,
268           masked_lm_positions=masked_lm_positions,
269           masked_lm_labels=masked_lm_labels)
270       instances.append(instance)
271
272     return instances
273
274
275   MaskedLmInstance = collections.namedtuple("MaskedLmInstance",
276                                             ["index", "label"])
```

```
277
278
279 def create_masked_lm_predictions(tokens, masked_lm_prob,
280                                   max_predictions_per_seq, vocab_words, rng):
281   """Creates the predictions for the masked LM objective."""
282
283   cand_indexes = []
284   for (i, token) in enumerate(tokens):
285     if token == "[CLS]" or token == "[SEP]":
286       continue
287     # Whole Word Masking means that if we mask all of the wordpieces
288     # corresponding to an original word. When a word has been split into
289     # WordPieces, the first token does not have any marker and any subsequence
290     # tokens are prefixed with ##. So whenever we see the ## token, we
291     # append it to the previous set of word indexes.
292     #
293     # Note that Whole Word Masking does *not* change the training code
294     # at all -- we still predict each WordPiece independently, softmaxed
295     # over the entire vocabulary.
296     if (FLAGS.do_whole_word_mask and len(cand_indexes) >= 1 and
297         token.startswith("##")):
298       cand_indexes[-1].append(i)
299     else:
300       cand_indexes.append([i])
301
302   rng.shuffle(cand_indexes)
303
304   output_tokens = list(tokens)
305
306   num_to_predict = min(max_predictions_per_seq,
307                        max(1, int(round(len(tokens) * masked_lm_prob))))
308
309   masked_lms = []
310   covered_indexes = set()
311   for index_set in cand_indexes:
312     if len(masked_lms) >= num_to_predict:
313       break
314     # If adding a whole-word mask would exceed the maximum number of
315     # predictions, then just skip this candidate.
316     if len(masked_lms) + len(index_set) > num_to_predict:
317       continue
318     is_any_index_covered = False
319     for index in index_set:
320       if index in covered_indexes:
321         is_any_index_covered = True
322         break
323     if is_any_index_covered:
324       continue
325     for index in index_set:
326       covered_indexes.add(index)
327
328       masked_token = None
329       # 80% of the time, replace with [MASK]
330       if rng.random() < 0.8:
331         masked_token = "[MASK]"
332       else:
333         # 10% of the time, keep original
334         if rng.random() < 0.5:
335           masked_token = tokens[index]
336         # 10% of the time, replace with random word
337         else:
338           masked_token = vocab_words[rng.randint(0, len(vocab_words) - 1)]
339
340       output_tokens[index] = masked_token
341
342       masked_lms.append(MaskedLmInstance(index=index, label=tokens[index]))
343   assert len(masked_lms) <= num_to_predict
344   masked_lms = sorted(masked_lms, key=lambda x: x.index)
345
346   masked_lm_positions = []
347   masked_lm_labels = []
348   for p in masked_lms:
349     masked_lm_positions.append(p.index)
350     masked_lm_labels.append(p.label)
351
352   return (output_tokens, masked_lm_positions, masked_lm_labels)
353
354
355 def truncate_seq_pair(tokens_a, tokens_b, max_num_tokens, rng):
356   """Truncates a pair of sequences to a maximum sequence length."""
357   while True:
358     total_length = len(tokens_a) + len(tokens_b)
359     if total_length <= max_num_tokens:
360       break
```

```
361
362       trunc_tokens = tokens_a if len(tokens_a) > len(tokens_b) else tokens_b
363       assert len(trunc_tokens) >= 1
364
365       # We want to sometimes truncate from the front and sometimes from the
366       # back to add more randomness and avoid biases.
367       if rng.random() < 0.5:
368         del trunc_tokens[0]
369       else:
370         trunc_tokens.pop()
371
372
373   def main(_):
374     tf.logging.set_verbosity(tf.logging.INFO)
375
376     tokenizer = tokenization.FullTokenizer(
377         vocab_file=FLAGS.vocab_file, do_lower_case=FLAGS.do_lower_case)
378
379     input_files = []
380     for input_pattern in FLAGS.input_file.split(","):
381       input_files.extend(tf.gfile.Glob(input_pattern))
382
383     tf.logging.info("*** Reading from input files ***")
384     for input_file in input_files:
385       tf.logging.info("  %s", input_file)
386
387     rng = random.Random(FLAGS.random_seed)
388     instances = create_training_instances(
389         input_files, tokenizer, FLAGS.max_seq_length, FLAGS.dupe_factor,
390         FLAGS.short_seq_prob, FLAGS.masked_lm_prob, FLAGS.max_predictions_per_seq,
391         rng)
392
393     output_files = FLAGS.output_file.split(",")
394     tf.logging.info("*** Writing to output files ***")
395     for output_file in output_files:
396       tf.logging.info("  %s", output_file)
397
398     write_instance_to_example_files(instances, tokenizer, FLAGS.max_seq_length,
399                                     FLAGS.max_predictions_per_seq, output_files)
400
401
402   if __name__ == "__main__":
403     flags.mark_flag_as_required("input_file")
404     flags.mark_flag_as_required("output_file")
405     flags.mark_flag_as_required("vocab_file")
406     tf.app.run()
```

Run Pre-Training.

In the following, the code to run pre-training for each of the three tasks, namely graph edge validation, similarity prediction and argument order prediction is provided.

Listing 4: Run Pre-Training for Graph Edge Validation

```
1   """Run masked LM/sorted masked_lm pre-training for BERT."""
2
3   from __future__ import absolute_import
4   from __future__ import division
5   from __future__ import print_function
6
7   import sys
8
9   sys.path.append('/content/drive/My Drive/Colab Notebooks')
10
11  import os
12  import modeling
13  import optimization
14  import tensorflow.compat.v1 as tf
15
16  def model_fn_builder(bert_config, init_checkpoint, learning_rate,
17                       num_train_steps, num_warmup_steps, use_tpu,
18                       use_one_hot_embeddings):
19    """Returns `model_fn` closure for TPUEstimator."""
20
21    def model_fn(features, labels, mode, params):  # pylint: disable=unused-argument
22      """The `model_fn` for TPUEstimator."""
23
```

```
    tf.logging.info("*** Features ***")
    for name in sorted(features.keys()):
      tf.logging.info("  name = %s, shape = %s" % (name, features[name].shape))

    input_ids = features["input_ids"]
    input_mask = features["input_mask"]
    segment_ids = features["segment_ids"]
    masked_lm_positions = features["masked_lm_positions"]
    masked_lm_ids = features["masked_lm_ids"]
    masked_lm_weights = features["masked_lm_weights"]
    is_edge_labels = features["is_edge_labels"]

    is_training = (mode == tf.estimator.ModeKeys.TRAIN)

    model = modeling.BertModel(
        config=bert_config,
        is_training=is_training,
        input_ids=input_ids,
        input_mask=input_mask,
        token_type_ids=segment_ids,
        use_one_hot_embeddings=use_one_hot_embeddings)

    (masked_lm_loss,
     masked_lm_example_loss, masked_lm_log_probs) = get_masked_lm_output(
         bert_config, model.get_sequence_output(), model.get_embedding_table(),
         masked_lm_positions, masked_lm_ids, masked_lm_weights)

    (is_edge_loss, is_edge_example_loss,
     is_edge_log_probs) = get_is_edge_output(
         bert_config, model.get_pooled_output(), is_edge_labels)

    total_loss = masked_lm_loss + is_edge_loss

    tvars = tf.trainable_variables()

    initialized_variable_names = {}
    scaffold_fn = None
    if init_checkpoint:
      (assignment_map, initialized_variable_names
      ) = modeling.get_assignment_map_from_checkpoint(tvars, init_checkpoint)
      if use_tpu:

        def tpu_scaffold():
          tf.train.init_from_checkpoint(init_checkpoint, assignment_map)
          return tf.train.Scaffold()

        scaffold_fn = tpu_scaffold
      else:
        tf.train.init_from_checkpoint(init_checkpoint, assignment_map)

    tf.logging.info("**** Trainable Variables ****")
    for var in tvars:
      init_string = ""
      if var.name in initialized_variable_names:
        init_string = ", *INIT_FROM_CKPT*"
      tf.logging.info("  name = %s, shape = %s%s", var.name, var.shape,
                      init_string)

    output_spec = None
    if mode == tf.estimator.ModeKeys.TRAIN:
      train_op = optimization.create_optimizer(
          total_loss, learning_rate, num_train_steps, num_warmup_steps, use_tpu)

      output_spec = tf.estimator.tpu.TPUEstimatorSpec(
          mode=mode,
          loss=total_loss,
          train_op=train_op,
          scaffold_fn=scaffold_fn)
    elif mode == tf.estimator.ModeKeys.EVAL:

      def metric_fn(masked_lm_example_loss, masked_lm_log_probs, masked_lm_ids,
                    masked_lm_weights, is_edge_example_loss,
                    is_edge_log_probs, is_edge_labels):
        """Computes the loss and accuracy of the model."""
        masked_lm_log_probs = tf.reshape(masked_lm_log_probs,
                                         [-1, masked_lm_log_probs.shape[-1]])
        masked_lm_predictions = tf.argmax(
            masked_lm_log_probs, axis=-1, output_type=tf.int32)
        masked_lm_example_loss = tf.reshape(masked_lm_example_loss, [-1])
        masked_lm_ids = tf.reshape(masked_lm_ids, [-1])
        masked_lm_weights = tf.reshape(masked_lm_weights, [-1])
        masked_lm_accuracy = tf.metrics.accuracy(
            labels=masked_lm_ids,
            predictions=masked_lm_predictions,
```

```
            weights=masked_lm_weights)
        masked_lm_mean_loss = tf.metrics.mean(
            values=masked_lm_example_loss, weights=masked_lm_weights)

        is_edge_log_probs = tf.reshape(
            is_edge_log_probs, [-1, is_edge_log_probs.shape[-1]])
        is_edge_predictions = tf.argmax(
            is_edge_log_probs, axis=-1, output_type=tf.int32)
        is_edge_labels = tf.reshape(is_edge_labels, [-1])
        is_edge_accuracy = tf.metrics.accuracy(
            labels=is_edge_labels, predictions=is_edge_predictions)
        is_edge_mean_loss = tf.metrics.mean(
            values=is_edge_example_loss)

        return {
            "masked_lm_accuracy": masked_lm_accuracy,
            "masked_lm_loss": masked_lm_mean_loss,
            "is_edge_accuracy": is_edge_accuracy,
            "is_edge_loss": is_edge_mean_loss,
        }

      eval_metrics = (metric_fn, [
          masked_lm_example_loss, masked_lm_log_probs, masked_lm_ids,
          masked_lm_weights, is_edge_example_loss,
          is_edge_log_probs, is_edge_labels
      ])
      output_spec = tf.estimator.tpu.TPUEstimatorSpec(
          mode=mode,
          loss=total_loss,
          eval_metrics=eval_metrics,
          scaffold_fn=scaffold_fn)
    else:
      raise ValueError("Only TRAIN and EVAL modes are supported: %s" % (mode))

    return output_spec

  return model_fn


def get_masked_lm_output(bert_config, input_tensor, output_weights, positions,
                         label_ids, label_weights):
  """Get loss and log probs for the masked LM."""
  input_tensor = gather_indexes(input_tensor, positions)

  with tf.variable_scope("cls/predictions"):
    # We apply one more non-linear transformation before the output layer.
    # This matrix is not used after pre-training.
    with tf.variable_scope("transform"):
      input_tensor = tf.layers.dense(
          input_tensor,
          units=bert_config.hidden_size,
          activation=modeling.get_activation(bert_config.hidden_act),
          kernel_initializer=modeling.create_initializer(
              bert_config.initializer_range))
      input_tensor = modeling.layer_norm(input_tensor)

    # The output weights are the same as the input embeddings, but there is
    # an output-only bias for each token.
    output_bias = tf.get_variable(
        "output_bias",
        shape=[bert_config.vocab_size],
        initializer=tf.zeros_initializer())
    logits = tf.matmul(input_tensor, output_weights, transpose_b=True)
    logits = tf.nn.bias_add(logits, output_bias)
    log_probs = tf.nn.log_softmax(logits, axis=-1)

    label_ids = tf.reshape(label_ids, [-1])
    label_weights = tf.reshape(label_weights, [-1])

    one_hot_labels = tf.one_hot(
        label_ids, depth=bert_config.vocab_size, dtype=tf.float32)

    # The `positions` tensor might be zero-padded (if the sequence is too
    # short to have the maximum number of predictions). The `label_weights`
    # tensor has a value of 1.0 for every real prediction and 0.0 for the
    # padding predictions.
    per_example_loss = -tf.reduce_sum(log_probs * one_hot_labels, axis=[-1])
    numerator = tf.reduce_sum(label_weights * per_example_loss)
    denominator = tf.reduce_sum(label_weights) + 1e-5
    loss = numerator / denominator

  return (loss, per_example_loss, log_probs)
```

```
def get_is_edge_output(bert_config, input_tensor, labels):
  """Get loss and log probs for the is_edge prediction."""

  # Simple binary classification. Note that 1 is "sorted" and 0 is
  # "not sorted". This weight matrix is not used after pre-training.
  with tf.variable_scope("cls/seq_relationship"):
    output_weights = tf.get_variable(
        "output_weights",
        shape=[2, bert_config.hidden_size],
        initializer=modeling.create_initializer(bert_config.initializer_range))
    output_bias = tf.get_variable(
        "output_bias", shape=[2], initializer=tf.zeros_initializer())

    logits = tf.matmul(input_tensor, output_weights, transpose_b=True)
    logits = tf.nn.bias_add(logits, output_bias)
    log_probs = tf.nn.log_softmax(logits, axis=-1)
    labels = tf.reshape(labels, [-1])
    one_hot_labels = tf.one_hot(labels, depth=2, dtype=tf.float32)
    per_example_loss = -tf.reduce_sum(one_hot_labels * log_probs, axis=-1)
    loss = tf.reduce_mean(per_example_loss)
    return (loss, per_example_loss, log_probs)


def gather_indexes(sequence_tensor, positions):
  """Gathers the vectors at the specific positions over a minibatch."""
  sequence_shape = modeling.get_shape_list(sequence_tensor, expected_rank=3)
  batch_size = sequence_shape[0]
  seq_length = sequence_shape[1]
  width = sequence_shape[2]

  flat_offsets = tf.reshape(
      tf.range(0, batch_size, dtype=tf.int32) * seq_length, [-1, 1])
  flat_positions = tf.reshape(positions + flat_offsets, [-1])
  flat_sequence_tensor = tf.reshape(sequence_tensor,
                                    [batch_size * seq_length, width])
  output_tensor = tf.gather(flat_sequence_tensor, flat_positions)
  return output_tensor


def input_fn_builder(input_files,
                     max_seq_length,
                     max_predictions_per_seq,
                     is_training,
                     num_cpu_threads=4):
  """Creates an `input_fn` closure to be passed to TPUEstimator."""

  def input_fn(params):
    """The actual input function."""
    batch_size = params["batch_size"]

    name_to_features = {
        "input_ids":
            tf.FixedLenFeature([max_seq_length], tf.int64),
        "input_mask":
            tf.FixedLenFeature([max_seq_length], tf.int64),
        "segment_ids":
            tf.FixedLenFeature([max_seq_length], tf.int64),
        "masked_lm_positions":
            tf.FixedLenFeature([max_predictions_per_seq], tf.int64),
        "masked_lm_ids":
            tf.FixedLenFeature([max_predictions_per_seq], tf.int64),
        "masked_lm_weights":
            tf.FixedLenFeature([max_predictions_per_seq], tf.float32),
        "is_edge_labels":
            tf.FixedLenFeature([1], tf.int64),
    }

    # For training, we want a lot of parallel reading and shuffling.
    # For eval, we want no shuffling and parallel reading doesn't matter.
    if is_training:
      d = tf.data.Dataset.from_tensor_slices(tf.constant(input_files))
      d = d.repeat()
      d = d.shuffle(buffer_size=len(input_files))

      # `cycle_length` is the number of parallel files that get read.
      cycle_length = min(num_cpu_threads, len(input_files))

      # `sloppy` mode means that the interleaving is not exact. This adds
      # even more randomness to the training pipeline.
      d = d.apply(
         # tf.contrib.data.parallel_interleave(
         #     tf.data.TFRecordDataset,
         #     sloppy=is_training,
         #     cycle_length=cycle_length))
```

```
          tf.data.experimental.parallel_interleave(
              tf.data.TFRecordDataset,
              sloppy=is_training,
              cycle_length=cycle_length))
      d = d.shuffle(buffer_size=100)
    else:
      d = tf.data.TFRecordDataset(input_files)
      # Since we evaluate for a fixed number of steps we don't want to encounter
      # out-of-range exceptions.
      d = d.repeat()

    # We must `drop_remainder` on training because the TPU requires fixed
    # size dimensions. For eval, we assume we are evaluating on the CPU or GPU
    # and we *don't* want to drop the remainder, otherwise we wont cover
    # every sample.
    d = d.apply(
        tf.data.experimental.map_and_batch(
            lambda record: _decode_record(record, name_to_features),
            batch_size=batch_size,
            num_parallel_batches=num_cpu_threads,
            drop_remainder=True))
    return d

  return input_fn


def _decode_record(record, name_to_features):
  """Decodes a record to a TensorFlow example."""
  example = tf.parse_single_example(record, name_to_features)

  # tf.Example only supports tf.int64, but the TPU only supports tf.int32.
  # So cast all int64 to int32.
  for name in list(example.keys()):
    t = example[name]
    if t.dtype == tf.int64:
      t = tf.to_int32(t)
    example[name] = t

  return example


def main(_):

  bert_config_file = "/content/drive/My Drive/uncased_L-8_H-512_A-8/config.json"
  input_file = "/content/drive/My Drive/Training Data Order/createdebate.tfrecord"
  output_dir = "/content/drive/My Drive/output/checkpoints2"
  max_seq_length = 128
  max_predictions_per_seq = 5
  do_train = True
  do_eval = True
  train_batch_size = 32
  eval_batch_size = 8
  learning_rate = 2e-5
  iterations_per_loop = 1000
  num_tpu_cores = 8
  max_eval_steps = 100
  use_tpu = False
  save_checkpoints_steps = 20000
  num_train_steps = 20000
  num_warmup_steps = 100
  #tpu_name = 'grpc://' + os.environ['COLAB_TPU_ADDR']
  tpu_name = None
  tpu_zone = None
  gcp_project = None
  master = None
  init_checkpoint = "/content/drive/My Drive/output/checkpoints"

  tf.logging.set_verbosity(tf.logging.INFO)

  if not do_train and not do_eval:
    raise ValueError("At least one of `do_train` or `do_eval` must be True.")

  bert_config = modeling.BertConfig.from_json_file(bert_config_file)

  tf.gfile.MakeDirs(output_dir)

  input_files = []
  for input_pattern in input_file.split(","):
    input_files.extend(tf.gfile.Glob(input_pattern))

  tf.logging.info("*** Input Files ***")
  for input_file in input_files:
    tf.logging.info("  %s" % input_file)

```

```
  tpu_cluster_resolver = None
  if use_tpu and tpu_name:
    #tpu_model = tf.contrib.tpu.keras_to_tpu_model(
    #    model,
    #    strategy=tf.contrib.tpu.TPUDistributionStrategy(
    #        tf.contrib.cluster_resolver.TPUClusterResolver(
    #            tpu_name)
    #    )
    #)
    tpu_cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver(
        tpu_name, zone=tpu_zone, project=gcp_project)

  #my_tpu_estimator = tf.contrib.tpu.TPUEstimator(
  #    model_fn=my_model_fn,
  #    config=tf.contrib.tpu.RunConfig()
  #    use_tpu=False
  #)
  is_per_host = tf.estimator.tpu.InputPipelineConfig.PER_HOST_V2
  run_config = tf.estimator.tpu.RunConfig(
      cluster=tpu_cluster_resolver,
      master=master,
      model_dir=output_dir,
      save_checkpoints_steps=save_checkpoints_steps,
      tpu_config=tf.estimator.tpu.TPUConfig(
          iterations_per_loop=iterations_per_loop,
          num_shards=num_tpu_cores,
          per_host_input_for_training=is_per_host))

  model_fn = model_fn_builder(
      bert_config=bert_config,
      init_checkpoint=init_checkpoint,
      learning_rate=learning_rate,
      num_train_steps=num_train_steps,
      num_warmup_steps=num_warmup_steps,
      use_tpu=use_tpu,
      use_one_hot_embeddings=use_tpu)

  # If TPU is not available, this will fall back to normal Estimator on CPU
  # or GPU.
  estimator = tf.estimator.tpu.TPUEstimator(
      use_tpu=use_tpu,
      model_fn=model_fn,
      config=run_config,
      train_batch_size=train_batch_size,
      eval_batch_size=eval_batch_size)

  if do_train:
    tf.logging.info("***** Running training *****")
    tf.logging.info("  Batch size = %d", train_batch_size)
    train_input_fn = input_fn_builder(
        input_files=input_files,
        max_seq_length=max_seq_length,
        max_predictions_per_seq=max_predictions_per_seq,
        is_training=True)
    estimator.train(input_fn=train_input_fn, max_steps=num_train_steps)

  if do_eval:
    tf.logging.info("***** Running evaluation *****")
    tf.logging.info("  Batch size = %d", eval_batch_size)

    eval_input_fn = input_fn_builder(
        input_files=input_files,
        max_seq_length=max_seq_length,
        max_predictions_per_seq=max_predictions_per_seq,
        is_training=False)

    result = estimator.evaluate(
        input_fn=eval_input_fn, steps=max_eval_steps)

    output_eval_file = os.path.join(output_dir, "eval_results.txt")
    with tf.gfile.GFile(output_eval_file, "w") as writer:
      tf.logging.info("***** Eval results *****")
      for key in sorted(result.keys()):
        tf.logging.info("  %s = %s", key, str(result[key]))
        writer.write("%s = %s\n" % (key, str(result[key])))


if __name__ == "__main__":
#  flags.mark_flag_as_required("input_file")
#  flags.mark_flag_as_required("bert_config_file")
#  flags.mark_flag_as_required("output_dir")
  tf.app.run()
```

Listing 5: Run Pre-Training for Similarity Prediction

```
"""Run masked LM/similar masked_lm pre-training for BERT."""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import modeling
import optimization
import tensorflow as tf

flags = tf.flags

FLAGS = flags.FLAGS

## Required parameters
flags.DEFINE_string(
    "bert_config_file", None,
    "The config json file corresponding to the pre-trained BERT model. "
    "This specifies the model architecture.")

flags.DEFINE_string(
    "input_file", None,
    "Input TF example files (can be a glob or comma separated).")

flags.DEFINE_string(
    "output_dir", None,
    "The output directory where the model checkpoints will be written.")

## Other parameters
flags.DEFINE_string(
    "init_checkpoint", None,
    "Initial checkpoint (usually from a pre-trained BERT model).")

flags.DEFINE_integer(
    "max_seq_length", 128,
    "The maximum total input sequence length after WordPiece tokenization. "
    "Sequences longer than this will be truncated, and sequences shorter "
    "than this will be padded. Must match data generation.")

flags.DEFINE_integer(
    "max_predictions_per_seq", 20,
    "Maximum number of masked LM predictions per sequence. "
    "Must match data generation.")

flags.DEFINE_bool("do_train", False, "Whether to run training.")

flags.DEFINE_bool("do_eval", False, "Whether to run eval on the dev set.")

flags.DEFINE_integer("train_batch_size", 32, "Total batch size for training.")

flags.DEFINE_integer("eval_batch_size", 8, "Total batch size for eval.")

flags.DEFINE_float("learning_rate", 5e-5, "The initial learning rate for Adam.")

flags.DEFINE_integer("num_train_steps", 100000, "Number of training steps.")

flags.DEFINE_integer("num_warmup_steps", 10000, "Number of warmup steps.")

flags.DEFINE_integer("save_checkpoints_steps", 1000,
                     "How often to save the model checkpoint.")

flags.DEFINE_integer("iterations_per_loop", 1000,
                     "How many steps to make in each estimator call.")

flags.DEFINE_integer("max_eval_steps", 100, "Maximum number of eval steps.")

flags.DEFINE_bool("use_tpu", False, "Whether to use TPU or GPU/CPU.")

tf.flags.DEFINE_string(
    "tpu_name", None,
    "The Cloud TPU to use for training. This should be either the name "
    "used when creating the Cloud TPU, or a grpc://ip.address.of.tpu:8470 "
    "url.")

tf.flags.DEFINE_string(
    "tpu_zone", None,
    "[Optional] GCE zone where the Cloud TPU is located in. If not "
    "specified, we will attempt to automatically detect the GCE project from "
    "metadata.")

tf.flags.DEFINE_string(
```

```
    "gcp_project", None,
    "[Optional] Project name for the Cloud TPU-enabled project. If not "
    "specified, we will attempt to automatically detect the GCE project from "
    "metadata.")

tf.flags.DEFINE_string("master", None, "[Optional] TensorFlow master URL.")

flags.DEFINE_integer(
    "num_tpu_cores", 8,
    "Only used if `use_tpu` is True. Total number of TPU cores to use.")


def model_fn_builder(bert_config, init_checkpoint, learning_rate,
                     num_train_steps, num_warmup_steps, use_tpu,
                     use_one_hot_embeddings):
  """Returns `model_fn` closure for TPUEstimator."""

  def model_fn(features, labels, mode, params):  # pylint: disable=unused-argument
    """The `model_fn` for TPUEstimator."""

    tf.logging.info("*** Features ***")
    for name in sorted(features.keys()):
      tf.logging.info("  name = %s, shape = %s" % (name, features[name].shape))

    input_ids = features["input_ids"]
    input_mask = features["input_mask"]
    segment_ids = features["segment_ids"]
    masked_lm_positions = features["masked_lm_positions"]
    masked_lm_ids = features["masked_lm_ids"]
    masked_lm_weights = features["masked_lm_weights"]
    similar_labels = features["similar_labels"]

    is_training = (mode == tf.estimator.ModeKeys.TRAIN)

    model = modeling.BertModel(
        config=bert_config,
        is_training=is_training,
        input_ids=input_ids,
        input_mask=input_mask,
        token_type_ids=segment_ids,
        use_one_hot_embeddings=use_one_hot_embeddings)

    (masked_lm_loss,
     masked_lm_example_loss, masked_lm_log_probs) = get_masked_lm_output(
         bert_config, model.get_sequence_output(), model.get_embedding_table(),
         masked_lm_positions, masked_lm_ids, masked_lm_weights)

    (similar_loss, similar_example_loss,
     similar_log_probs) = get_similar_output(
         bert_config, model.get_pooled_output(), similar_labels)

    total_loss = masked_lm_loss + similar_loss

    tvars = tf.trainable_variables()

    initialized_variable_names = {}
    scaffold_fn = None
    if init_checkpoint:
      (assignment_map, initialized_variable_names
      ) = modeling.get_assignment_map_from_checkpoint(tvars, init_checkpoint)
      if use_tpu:

        def tpu_scaffold():
          tf.train.init_from_checkpoint(init_checkpoint, assignment_map)
          return tf.train.Scaffold()

        scaffold_fn = tpu_scaffold
      else:
        tf.train.init_from_checkpoint(init_checkpoint, assignment_map)

    tf.logging.info("**** Trainable Variables ****")
    for var in tvars:
      init_string = ""
      if var.name in initialized_variable_names:
        init_string = ", *INIT_FROM_CKPT*"
      tf.logging.info("  name = %s, shape = %s%s", var.name, var.shape,
                      init_string)

    output_spec = None
    if mode == tf.estimator.ModeKeys.TRAIN:
      train_op = optimization.create_optimizer(
          total_loss, learning_rate, num_train_steps, num_warmup_steps, use_tpu)

      output_spec = tf.estimator.tpu.TPUEstimatorSpec(
```

```
          mode=mode,
          loss=total_loss,
          train_op=train_op,
          scaffold_fn=scaffold_fn)
    elif mode == tf.estimator.ModeKeys.EVAL:

      def metric_fn(masked_lm_example_loss, masked_lm_log_probs, masked_lm_ids,
                    masked_lm_weights, similar_example_loss,
                    similar_log_probs, similar_labels):
        """Computes the loss and accuracy of the model."""
        masked_lm_log_probs = tf.reshape(masked_lm_log_probs,
                                         [-1, masked_lm_log_probs.shape[-1]])
        masked_lm_predictions = tf.argmax(
            masked_lm_log_probs, axis=-1, output_type=tf.int32)
        masked_lm_example_loss = tf.reshape(masked_lm_example_loss, [-1])
        masked_lm_ids = tf.reshape(masked_lm_ids, [-1])
        masked_lm_weights = tf.reshape(masked_lm_weights, [-1])
        masked_lm_accuracy = tf.metrics.accuracy(
            labels=masked_lm_ids,
            predictions=masked_lm_predictions,
            weights=masked_lm_weights)
        masked_lm_mean_loss = tf.metrics.mean(
            values=masked_lm_example_loss, weights=masked_lm_weights)

        similar_log_probs = tf.reshape(
            similar_log_probs, [-1, similar_log_probs.shape[-1]])
        similar_predictions = tf.argmax(
            similar_log_probs, axis=-1, output_type=tf.int32)
        similar_labels = tf.reshape(similar_labels, [-1])
        similar_accuracy = tf.metrics.accuracy(
            labels=similar_labels, predictions=similar_predictions)
        similar_mean_loss = tf.metrics.mean(
            values=similar_example_loss)

        return {
            "masked_lm_accuracy": masked_lm_accuracy,
            "masked_lm_loss": masked_lm_mean_loss,
            "similar_accuracy": similar_accuracy,
            "similar_loss": similar_mean_loss,
        }

      eval_metrics = (metric_fn, [
          masked_lm_example_loss, masked_lm_log_probs, masked_lm_ids,
          masked_lm_weights, similar_example_loss,
          similar_log_probs, similar_labels
      ])
      output_spec = tf.estimator.tpu.TPUEstimatorSpec(
          mode=mode,
          loss=total_loss,
          eval_metrics=eval_metrics,
          scaffold_fn=scaffold_fn)
    else:
      raise ValueError("Only TRAIN and EVAL modes are supported: %s" % (mode))

    return output_spec

  return model_fn


def get_masked_lm_output(bert_config, input_tensor, output_weights, positions,
                         label_ids, label_weights):
  """Get loss and log probs for the masked LM."""
  input_tensor = gather_indexes(input_tensor, positions)

  with tf.variable_scope("cls/predictions"):
    # We apply one more non-linear transformation before the output layer.
    # This matrix is not used after pre-training.
    with tf.variable_scope("transform"):
      input_tensor = tf.layers.dense(
          input_tensor,
          units=bert_config.hidden_size,
          activation=modeling.get_activation(bert_config.hidden_act),
          kernel_initializer=modeling.create_initializer(
              bert_config.initializer_range))
      input_tensor = modeling.layer_norm(input_tensor)

    # The output weights are the same as the input embeddings, but there is
    # an output-only bias for each token.
    output_bias = tf.get_variable(
        "output_bias",
        shape=[bert_config.vocab_size],
        initializer=tf.zeros_initializer())
    logits = tf.matmul(input_tensor, output_weights, transpose_b=True)
    logits = tf.nn.bias_add(logits, output_bias)
```

```
    log_probs = tf.nn.log_softmax(logits, axis=-1)

    label_ids = tf.reshape(label_ids, [-1])
    label_weights = tf.reshape(label_weights, [-1])

    one_hot_labels = tf.one_hot(
        label_ids, depth=bert_config.vocab_size, dtype=tf.float32)

    # The `positions` tensor might be zero-padded (if the sequence is too
    # short to have the maximum number of predictions). The `label_weights`
    # tensor has a value of 1.0 for every real prediction and 0.0 for the
    # padding predictions.
    per_example_loss = -tf.reduce_sum(log_probs * one_hot_labels, axis=[-1])
    numerator = tf.reduce_sum(label_weights * per_example_loss)
    denominator = tf.reduce_sum(label_weights) + 1e-5
    loss = numerator / denominator

  return (loss, per_example_loss, log_probs)


def get_similar_output(bert_config, input_tensor, labels):
  """Get loss and log probs for the similarity prediction."""

  # Simple binary classification. Note that 1 is "similar" and 0 is
  # "random sentence". This weight matrix is not used after pre-training.
  with tf.variable_scope("cls/seq_relationship"):
    output_weights = tf.get_variable(
        "output_weights",
        shape=[2, bert_config.hidden_size],
        initializer=modeling.create_initializer(bert_config.initializer_range))
    output_bias = tf.get_variable(
        "output_bias", shape=[2], initializer=tf.zeros_initializer())

    logits = tf.matmul(input_tensor, output_weights, transpose_b=True)
    logits = tf.nn.bias_add(logits, output_bias)
    log_probs = tf.nn.log_softmax(logits, axis=-1)
    labels = tf.reshape(labels, [-1])
    one_hot_labels = tf.one_hot(labels, depth=2, dtype=tf.float32)
    per_example_loss = -tf.reduce_sum(one_hot_labels * log_probs, axis=-1)
    loss = tf.reduce_mean(per_example_loss)
    return (loss, per_example_loss, log_probs)


def gather_indexes(sequence_tensor, positions):
  """Gathers the vectors at the specific positions over a minibatch."""
  sequence_shape = modeling.get_shape_list(sequence_tensor, expected_rank=3)
  batch_size = sequence_shape[0]
  seq_length = sequence_shape[1]
  width = sequence_shape[2]

  flat_offsets = tf.reshape(
      tf.range(0, batch_size, dtype=tf.int32) * seq_length, [-1, 1])
  flat_positions = tf.reshape(positions + flat_offsets, [-1])
  flat_sequence_tensor = tf.reshape(sequence_tensor,
                                    [batch_size * seq_length, width])
  output_tensor = tf.gather(flat_sequence_tensor, flat_positions)
  return output_tensor


def input_fn_builder(input_files,
                     max_seq_length,
                     max_predictions_per_seq,
                     is_training,
                     num_cpu_threads=4):
  """Creates an `input_fn` closure to be passed to TPUEstimator."""

  def input_fn(params):
    """The actual input function."""
    batch_size = params["batch_size"]

    name_to_features = {
        "input_ids":
            tf.FixedLenFeature([max_seq_length], tf.int64),
        "input_mask":
            tf.FixedLenFeature([max_seq_length], tf.int64),
        "segment_ids":
            tf.FixedLenFeature([max_seq_length], tf.int64),
        "masked_lm_positions":
            tf.FixedLenFeature([max_predictions_per_seq], tf.int64),
        "masked_lm_ids":
            tf.FixedLenFeature([max_predictions_per_seq], tf.int64),
        "masked_lm_weights":
            tf.FixedLenFeature([max_predictions_per_seq], tf.float32),
        "similar_labels":
```

```
            tf.FixedLenFeature([1], tf.int64),
    }

    # For training, we want a lot of parallel reading and shuffling.
    # For eval, we want no shuffling and parallel reading doesn't matter.
    if is_training:
      d = tf.data.Dataset.from_tensor_slices(tf.constant(input_files))
      d = d.repeat()
      d = d.shuffle(buffer_size=len(input_files))

      # `cycle_length` is the number of parallel files that get read.
      cycle_length = min(num_cpu_threads, len(input_files))

      # `sloppy` mode means that the interleaving is not exact. This adds
      # even more randomness to the training pipeline.
      d = d.apply(
          # tf.contrib.data.parallel_interleave(
          #     tf.data.TFRecordDataset,
          #     sloppy=is_training,
          #     cycle_length=cycle_length))
          tf.data.experimental.parallel_interleave(
              tf.data.TFRecordDataset,
              sloppy=is_training,
              cycle_length=cycle_length))
      d = d.shuffle(buffer_size=100)
    else:
      d = tf.data.TFRecordDataset(input_files)
      # Since we evaluate for a fixed number of steps we don't want to encounter
      # out-of-range exceptions.
      d = d.repeat()

    # We must `drop_remainder` on training because the TPU requires fixed
    # size dimensions. For eval, we assume we are evaluating on the CPU or GPU
    # and we *don't* want to drop the remainder, otherwise we wont cover
    # every sample.
    d = d.apply(
        tf.data.experimental.map_and_batch(
            lambda record: _decode_record(record, name_to_features),
            batch_size=batch_size,
            num_parallel_batches=num_cpu_threads,
            drop_remainder=True))
    return d

  return input_fn


def _decode_record(record, name_to_features):
  """Decodes a record to a TensorFlow example."""
  example = tf.parse_single_example(record, name_to_features)

  # tf.Example only supports tf.int64, but the TPU only supports tf.int32.
  # So cast all int64 to int32.
  for name in list(example.keys()):
    t = example[name]
    if t.dtype == tf.int64:
      t = tf.to_int32(t)
    example[name] = t

  return example


def main(_):
  tf.logging.set_verbosity(tf.logging.INFO)

  if not FLAGS.do_train and not FLAGS.do_eval:
    raise ValueError("At least one of `do_train` or `do_eval` must be True.")

  bert_config = modeling.BertConfig.from_json_file(FLAGS.bert_config_file)

  tf.gfile.MakeDirs(FLAGS.output_dir)

  input_files = []
  for input_pattern in FLAGS.input_file.split(","):
    input_files.extend(tf.gfile.Glob(input_pattern))

  tf.logging.info("*** Input Files ***")
  for input_file in input_files:
    tf.logging.info("  %s" % input_file)

  tpu_cluster_resolver = None
  if FLAGS.use_tpu and FLAGS.tpu_name:
    tpu_cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver(
        FLAGS.tpu_name, zone=FLAGS.tpu_zone, project=FLAGS.gcp_project)

```

```
  is_per_host = tf.estimator.tpu.InputPipelineConfig.PER_HOST_V2
  run_config = tf.estimator.tpu.RunConfig(
      cluster=tpu_cluster_resolver,
      master=FLAGS.master,
      model_dir=FLAGS.output_dir,
      save_checkpoints_steps=FLAGS.save_checkpoints_steps,
      tpu_config=tf.estimator.tpu.TPUConfig(
          iterations_per_loop=FLAGS.iterations_per_loop,
          num_shards=FLAGS.num_tpu_cores,
          per_host_input_for_training=is_per_host))

  model_fn = model_fn_builder(
      bert_config=bert_config,
      init_checkpoint=FLAGS.init_checkpoint,
      learning_rate=FLAGS.learning_rate,
      num_train_steps=FLAGS.num_train_steps,
      num_warmup_steps=FLAGS.num_warmup_steps,
      use_tpu=FLAGS.use_tpu,
      use_one_hot_embeddings=FLAGS.use_tpu)

  # If TPU is not available, this will fall back to normal Estimator on CPU
  # or GPU.
  estimator = tf.estimator.tpu.TPUEstimator(
      use_tpu=FLAGS.use_tpu,
      model_fn=model_fn,
      config=run_config,
      train_batch_size=FLAGS.train_batch_size,
      eval_batch_size=FLAGS.eval_batch_size)

  if FLAGS.do_train:
    tf.logging.info("***** Running training *****")
    tf.logging.info("  Batch size = %d", FLAGS.train_batch_size)
    train_input_fn = input_fn_builder(
        input_files=input_files,
        max_seq_length=FLAGS.max_seq_length,
        max_predictions_per_seq=FLAGS.max_predictions_per_seq,
        is_training=True)
    estimator.train(input_fn=train_input_fn, max_steps=FLAGS.num_train_steps)

  if FLAGS.do_eval:
    tf.logging.info("***** Running evaluation *****")
    tf.logging.info("  Batch size = %d", FLAGS.eval_batch_size)

    eval_input_fn = input_fn_builder(
        input_files=input_files,
        max_seq_length=FLAGS.max_seq_length,
        max_predictions_per_seq=FLAGS.max_predictions_per_seq,
        is_training=False)

    result = estimator.evaluate(
        input_fn=eval_input_fn, steps=FLAGS.max_eval_steps)

    output_eval_file = os.path.join(FLAGS.output_dir, "eval_results.txt")
    with tf.gfile.GFile(output_eval_file, "w") as writer:
      tf.logging.info("***** Eval results *****")
      for key in sorted(result.keys()):
        tf.logging.info("  %s = %s", key, str(result[key]))
        writer.write("%s = %s\n" % (key, str(result[key])))


if __name__ == "__main__":
  flags.mark_flag_as_required("input_file")
  flags.mark_flag_as_required("bert_config_file")
  flags.mark_flag_as_required("output_dir")
  tf.app.run()
```

Listing 6: Run Pre-Training for Argument Order Prediction

```
"""Run masked LM/sorted masked_lm pre-training for BERT."""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import sys

sys.path.append('/content/drive/My Drive/Colab Notebooks')

import os
import modeling
import optimization
import tensorflow.compat.v1 as tf

```

```
def model_fn_builder(bert_config, init_checkpoint, learning_rate,
                     num_train_steps, num_warmup_steps, use_tpu,
                     use_one_hot_embeddings):
  """Returns `model_fn` closure for TPUEstimator."""

  def model_fn(features, labels, mode, params):  # pylint: disable=unused-argument
    """The `model_fn` for TPUEstimator."""

    tf.logging.info("*** Features ***")
    for name in sorted(features.keys()):
      tf.logging.info("  name = %s, shape = %s" % (name, features[name].shape))

    input_ids = features["input_ids"]
    input_mask = features["input_mask"]
    segment_ids = features["segment_ids"]
    masked_lm_positions = features["masked_lm_positions"]
    masked_lm_ids = features["masked_lm_ids"]
    masked_lm_weights = features["masked_lm_weights"]
    is_sorted_labels = features["is_sorted_labels"]

    is_training = (mode == tf.estimator.ModeKeys.TRAIN)

    model = modeling.BertModel(
        config=bert_config,
        is_training=is_training,
        input_ids=input_ids,
        input_mask=input_mask,
        token_type_ids=segment_ids,
        use_one_hot_embeddings=use_one_hot_embeddings)

    (masked_lm_loss,
     masked_lm_example_loss, masked_lm_log_probs) = get_masked_lm_output(
         bert_config, model.get_sequence_output(), model.get_embedding_table(),
         masked_lm_positions, masked_lm_ids, masked_lm_weights)

    (is_sorted_loss, is_sorted_example_loss,
     is_sorted_log_probs) = get_is_sorted_output(
         bert_config, model.get_pooled_output(), is_sorted_labels)

    total_loss = masked_lm_loss + is_sorted_loss

    tvars = tf.trainable_variables()

    initialized_variable_names = {}
    scaffold_fn = None
    if init_checkpoint:
      (assignment_map, initialized_variable_names
      ) = modeling.get_assignment_map_from_checkpoint(tvars, init_checkpoint)
      if use_tpu:

        def tpu_scaffold():
          tf.train.init_from_checkpoint(init_checkpoint, assignment_map)
          return tf.train.Scaffold()

        scaffold_fn = tpu_scaffold
      else:
        tf.train.init_from_checkpoint(init_checkpoint, assignment_map)

    tf.logging.info("**** Trainable Variables ****")
    for var in tvars:
      init_string = ""
      if var.name in initialized_variable_names:
        init_string = ", *INIT_FROM_CKPT*"
      tf.logging.info("  name = %s, shape = %s%s", var.name, var.shape,
                      init_string)

    output_spec = None
    if mode == tf.estimator.ModeKeys.TRAIN:
      train_op = optimization.create_optimizer(
          total_loss, learning_rate, num_train_steps, num_warmup_steps, use_tpu)

      output_spec = tf.estimator.tpu.TPUEstimatorSpec(
          mode=mode,
          loss=total_loss,
          train_op=train_op,
          scaffold_fn=scaffold_fn)
    elif mode == tf.estimator.ModeKeys.EVAL:

      def metric_fn(masked_lm_example_loss, masked_lm_log_probs, masked_lm_ids,
                    masked_lm_weights, is_sorted_example_loss,
                    is_sorted_log_probs, is_sorted_labels):
        """Computes the loss and accuracy of the model."""
        masked_lm_log_probs = tf.reshape(masked_lm_log_probs,
                                         [-1, masked_lm_log_probs.shape[-1]])
```

```
        masked_lm_predictions = tf.argmax(
            masked_lm_log_probs, axis=-1, output_type=tf.int32)
        masked_lm_example_loss = tf.reshape(masked_lm_example_loss, [-1])
        masked_lm_ids = tf.reshape(masked_lm_ids, [-1])
        masked_lm_weights = tf.reshape(masked_lm_weights, [-1])
        masked_lm_accuracy = tf.metrics.accuracy(
            labels=masked_lm_ids,
            predictions=masked_lm_predictions,
            weights=masked_lm_weights)
        masked_lm_mean_loss = tf.metrics.mean(
            values=masked_lm_example_loss, weights=masked_lm_weights)

        is_sorted_log_probs = tf.reshape(
            is_sorted_log_probs, [-1, is_sorted_log_probs.shape[-1]])
        is_sorted_predictions = tf.argmax(
            is_sorted_log_probs, axis=-1, output_type=tf.int32)
        is_sorted_labels = tf.reshape(is_sorted_labels, [-1])
        is_sorted_accuracy = tf.metrics.accuracy(
            labels=is_sorted_labels, predictions=is_sorted_predictions)
        is_sorted_mean_loss = tf.metrics.mean(
            values=is_sorted_example_loss)

        return {
            "masked_lm_accuracy": masked_lm_accuracy,
            "masked_lm_loss": masked_lm_mean_loss,
            "is_sorted_accuracy": is_sorted_accuracy,
            "is_sorted_loss": is_sorted_mean_loss,
        }

      eval_metrics = (metric_fn, [
          masked_lm_example_loss, masked_lm_log_probs, masked_lm_ids,
          masked_lm_weights, is_sorted_example_loss,
          is_sorted_log_probs, is_sorted_labels
      ])
      output_spec = tf.estimator.tpu.TPUEstimatorSpec(
          mode=mode,
          loss=total_loss,
          eval_metrics=eval_metrics,
          scaffold_fn=scaffold_fn)
    else:
      raise ValueError("Only TRAIN and EVAL modes are supported: %s" % (mode))

    return output_spec

  return model_fn


def get_masked_lm_output(bert_config, input_tensor, output_weights, positions,
                         label_ids, label_weights):
  """Get loss and log probs for the masked LM."""
  input_tensor = gather_indexes(input_tensor, positions)

  with tf.variable_scope("cls/predictions"):
    # We apply one more non-linear transformation before the output layer.
    # This matrix is not used after pre-training.
    with tf.variable_scope("transform"):
      input_tensor = tf.layers.dense(
          input_tensor,
          units=bert_config.hidden_size,
          activation=modeling.get_activation(bert_config.hidden_act),
          kernel_initializer=modeling.create_initializer(
              bert_config.initializer_range))
      input_tensor = modeling.layer_norm(input_tensor)

    # The output weights are the same as the input embeddings, but there is
    # an output-only bias for each token.
    output_bias = tf.get_variable(
        "output_bias",
        shape=[bert_config.vocab_size],
        initializer=tf.zeros_initializer())
    logits = tf.matmul(input_tensor, output_weights, transpose_b=True)
    logits = tf.nn.bias_add(logits, output_bias)
    log_probs = tf.nn.log_softmax(logits, axis=-1)

    label_ids = tf.reshape(label_ids, [-1])
    label_weights = tf.reshape(label_weights, [-1])

    one_hot_labels = tf.one_hot(
        label_ids, depth=bert_config.vocab_size, dtype=tf.float32)

    # The `positions` tensor might be zero-padded (if the sequence is too
    # short to have the maximum number of predictions). The `label_weights`
    # tensor has a value of 1.0 for every real prediction and 0.0 for the
    # padding predictions.
```

```
    per_example_loss = -tf.reduce_sum(log_probs * one_hot_labels, axis=[-1])
    numerator = tf.reduce_sum(label_weights * per_example_loss)
    denominator = tf.reduce_sum(label_weights) + 1e-5
    loss = numerator / denominator

  return (loss, per_example_loss, log_probs)


def get_is_sorted_output(bert_config, input_tensor, labels):
  """Get loss and log probs for the is_sorted prediction."""

  # Simple binary classification. Note that 1 is "sorted" and 0 is
  # "not sorted". This weight matrix is not used after pre-training.
  with tf.variable_scope("cls/seq_relationship"):
    output_weights = tf.get_variable(
        "output_weights",
        shape=[2, bert_config.hidden_size],
        initializer=modeling.create_initializer(bert_config.initializer_range))
    output_bias = tf.get_variable(
        "output_bias", shape=[2], initializer=tf.zeros_initializer())

    logits = tf.matmul(input_tensor, output_weights, transpose_b=True)
    logits = tf.nn.bias_add(logits, output_bias)
    log_probs = tf.nn.log_softmax(logits, axis=-1)
    labels = tf.reshape(labels, [-1])
    one_hot_labels = tf.one_hot(labels, depth=2, dtype=tf.float32)
    per_example_loss = -tf.reduce_sum(one_hot_labels * log_probs, axis=-1)
    loss = tf.reduce_mean(per_example_loss)
    return (loss, per_example_loss, log_probs)


def gather_indexes(sequence_tensor, positions):
  """Gathers the vectors at the specific positions over a minibatch."""
  sequence_shape = modeling.get_shape_list(sequence_tensor, expected_rank=3)
  batch_size = sequence_shape[0]
  seq_length = sequence_shape[1]
  width = sequence_shape[2]

  flat_offsets = tf.reshape(
      tf.range(0, batch_size, dtype=tf.int32) * seq_length, [-1, 1])
  flat_positions = tf.reshape(positions + flat_offsets, [-1])
  flat_sequence_tensor = tf.reshape(sequence_tensor,
                                    [batch_size * seq_length, width])
  output_tensor = tf.gather(flat_sequence_tensor, flat_positions)
  return output_tensor


def input_fn_builder(input_files,
                     max_seq_length,
                     max_predictions_per_seq,
                     is_training,
                     num_cpu_threads=4):
  """Creates an `input_fn` closure to be passed to TPUEstimator."""

  def input_fn(params):
    """The actual input function."""
    batch_size = params["batch_size"]

    name_to_features = {
        "input_ids":
            tf.FixedLenFeature([max_seq_length], tf.int64),
        "input_mask":
            tf.FixedLenFeature([max_seq_length], tf.int64),
        "segment_ids":
            tf.FixedLenFeature([max_seq_length], tf.int64),
        "masked_lm_positions":
            tf.FixedLenFeature([max_predictions_per_seq], tf.int64),
        "masked_lm_ids":
            tf.FixedLenFeature([max_predictions_per_seq], tf.int64),
        "masked_lm_weights":
            tf.FixedLenFeature([max_predictions_per_seq], tf.float32),
        "is_sorted_labels":
            tf.FixedLenFeature([1], tf.int64),
    }

    # For training, we want a lot of parallel reading and shuffling.
    # For eval, we want no shuffling and parallel reading doesn't matter.
    if is_training:
      d = tf.data.Dataset.from_tensor_slices(tf.constant(input_files))
      d = d.repeat()
      d = d.shuffle(buffer_size=len(input_files))

      # `cycle_length` is the number of parallel files that get read.
      cycle_length = min(num_cpu_threads, len(input_files))
```

```
268
269         # `sloppy` mode means that the interleaving is not exact. This adds
270         # even more randomness to the training pipeline.
271         d = d.apply(
272             # tf.contrib.data.parallel_interleave(
273             #     tf.data.TFRecordDataset,
274             #     sloppy=is_training,
275             #     cycle_length=cycle_length))
276             tf.data.experimental.parallel_interleave(
277                 tf.data.TFRecordDataset,
278                 sloppy=is_training,
279                 cycle_length=cycle_length))
280         d = d.shuffle(buffer_size=100)
281       else:
282         d = tf.data.TFRecordDataset(input_files)
283         # Since we evaluate for a fixed number of steps we don't want to encounter
284         # out-of-range exceptions.
285         d = d.repeat()
286
287       # We must `drop_remainder` on training because the TPU requires fixed
288       # size dimensions. For eval, we assume we are evaluating on the CPU or GPU
289       # and we *don't* want to drop the remainder, otherwise we wont cover
290       # every sample.
291       d = d.apply(
292           tf.data.experimental.map_and_batch(
293               lambda record: _decode_record(record, name_to_features),
294               batch_size=batch_size,
295               num_parallel_batches=num_cpu_threads,
296               drop_remainder=True))
297       return d
298
299     return input_fn
300
301
302   def _decode_record(record, name_to_features):
303     """Decodes a record to a TensorFlow example."""
304     example = tf.parse_single_example(record, name_to_features)
305
306     # tf.Example only supports tf.int64, but the TPU only supports tf.int32.
307     # So cast all int64 to int32.
308     for name in list(example.keys()):
309       t = example[name]
310       if t.dtype == tf.int64:
311         t = tf.to_int32(t)
312       example[name] = t
313
314     return example
315
316
317   def main(_):
318
319     bert_config_file = "/content/drive/My Drive/uncased_L-8_H-512_A-8/config.json"
320     input_file = "/content/drive/My Drive/Training Data Order/createdebate.tfrecord"
321     output_dir = "/content/drive/My Drive/output/checkpoints2"
322     max_seq_length = 128
323     max_predictions_per_seq = 5
324     do_train = True
325     do_eval = True
326     train_batch_size = 32
327     eval_batch_size = 8
328     learning_rate = 2e-5
329     iterations_per_loop = 1000
330     num_tpu_cores = 8
331     max_eval_steps = 100
332     use_tpu = False
333     save_checkpoints_steps = 20000
334     num_train_steps = 20000
335     num_warmup_steps = 100
336     #tpu_name = 'grpc://' + os.environ['COLAB_TPU_ADDR']
337     tpu_name = None
338     tpu_zone = None
339     gcp_project = None
340     master = None
341     init_checkpoint = "/content/drive/My Drive/output/checkpoints"
342
343     tf.logging.set_verbosity(tf.logging.INFO)
344
345     if not do_train and not do_eval:
346       raise ValueError("At least one of `do_train` or `do_eval` must be True.")
347
348     bert_config = modeling.BertConfig.from_json_file(bert_config_file)
349
350     tf.gfile.MakeDirs(output_dir)
351
```

```
352     input_files = []
353     for input_pattern in input_file.split(","):
354       input_files.extend(tf.gfile.Glob(input_pattern))
355
356     tf.logging.info("*** Input Files ***")
357     for input_file in input_files:
358       tf.logging.info("  %s" % input_file)
359
360     tpu_cluster_resolver = None
361     if use_tpu and tpu_name:
362       #tpu_model = tf.contrib.tpu.keras_to_tpu_model(
363       #    model,
364       #    strategy=tf.contrib.tpu.TPUDistributionStrategy(
365       #        tf.contrib.cluster_resolver.TPUClusterResolver(
366       #            tpu_name)
367       #    )
368       #)
369       tpu_cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver(
370           tpu_name, zone=tpu_zone, project=gcp_project)
371
372     #my_tpu_estimator = tf.contrib.tpu.TPUEstimator(
373     #    model_fn=my_model_fn,
374     #    config=tf.contrib.tpu.RunConfig()
375     #    use_tpu=False
376     #)
377     is_per_host = tf.estimator.tpu.InputPipelineConfig.PER_HOST_V2
378     run_config = tf.estimator.tpu.RunConfig(
379         cluster=tpu_cluster_resolver,
380         master=master,
381         model_dir=output_dir,
382         save_checkpoints_steps=save_checkpoints_steps,
383         tpu_config=tf.estimator.tpu.TPUConfig(
384             iterations_per_loop=iterations_per_loop,
385             num_shards=num_tpu_cores,
386             per_host_input_for_training=is_per_host))
387
388     model_fn = model_fn_builder(
389         bert_config=bert_config,
390         init_checkpoint=init_checkpoint,
391         learning_rate=learning_rate,
392         num_train_steps=num_train_steps,
393         num_warmup_steps=num_warmup_steps,
394         use_tpu=use_tpu,
395         use_one_hot_embeddings=use_tpu)
396
397     # If TPU is not available, this will fall back to normal Estimator on CPU
398     # or GPU.
399     estimator = tf.estimator.tpu.TPUEstimator(
400         use_tpu=use_tpu,
401         model_fn=model_fn,
402         config=run_config,
403         train_batch_size=train_batch_size,
404         eval_batch_size=eval_batch_size)
405
406     if do_train:
407       tf.logging.info("***** Running training *****")
408       tf.logging.info("  Batch size = %d", train_batch_size)
409       train_input_fn = input_fn_builder(
410           input_files=input_files,
411           max_seq_length=max_seq_length,
412           max_predictions_per_seq=max_predictions_per_seq,
413           is_training=True)
414       estimator.train(input_fn=train_input_fn, max_steps=num_train_steps)
415
416     if do_eval:
417       tf.logging.info("***** Running evaluation *****")
418       tf.logging.info("  Batch size = %d", eval_batch_size)
419
420       eval_input_fn = input_fn_builder(
421           input_files=input_files,
422           max_seq_length=max_seq_length,
423           max_predictions_per_seq=max_predictions_per_seq,
424           is_training=False)
425
426       result = estimator.evaluate(
427           input_fn=eval_input_fn, steps=max_eval_steps)
428
429       output_eval_file = os.path.join(output_dir, "eval_results.txt")
430       with tf.gfile.GFile(output_eval_file, "w") as writer:
431         tf.logging.info("***** Eval results *****")
432         for key in sorted(result.keys()):
433           tf.logging.info("  %s = %s", key, str(result[key]))
434           writer.write("%s = %s\n" % (key, str(result[key])))
435
```

```
436
437 if __name__ == "__main__":
438 #  flags.mark_flag_as_required("input_file")
439 #  flags.mark_flag_as_required("bert_config_file")
440 #  flags.mark_flag_as_required("output_dir")
441   tf.app.run()
```

A.2 Code: MaxPoolBERT

This section presents the implementation details of the MaxPoolBERT model described in Chapter 6. It includes the code for the modified architecture, which extends the standard BERT model by incorporating layer-wise and token-wise max pooling to improve classification performance.

As the main contribution of MaxPoolBERT lies in its architectural modification during fine-tuning, the provided implementation focuses on the integration of the pooling mechanism into the model pipeline. The code enables the reproduction of the experimental results and serves as a reference for applying the proposed approach to other classification tasks.

The implementation builds upon existing BERT frameworks, taken from Huggingface[30] and has been adapted to highlight the components relevant to the proposed method. The full implementation is available at: `https://github.com/mabehrendt/MaxPoolBERT`.

Model.

Listing 7: MaxPoolBERT Model Definition

```
1  from typing import List, Optional, Tuple, Union
2  import numpy as np
3
4  import torch
5  from torch import nn
6  import torch.nn.functional as F
7  from torch import Tensor
8  from torch.nn import BCEWithLogitsLoss, CrossEntropyLoss, MSELoss
9  from transformers import BertPreTrainedModel, BertModel
10 from transformers.modeling_outputs import SequenceClassifierOutput, BaseModelOutputWithPastAndCrossAttentions
11 from typing import Callable
12 import inspect
13
14 class MaxPoolBERTForSequenceClassification(BertPreTrainedModel):
15     """
16     MaxPoolBERT for sequence classification model.
17     ...
18
19     """
20     def __init__(self, config, pretrained_model_name_or_path, seed, num_layers_to_pool=3, pooling_strategy='max'):
21         """
22         Constructs all the necessary attributes for the MaxPoolBERTForSequenceClassification object.
23             Parameters
24             ----------
25             config : BertConfig
26                 config of the base model
27             pretrained_model_name_or_path : BertPreTrainedModel or str
28                 pretrained bert model as base for maxpoolbert or path to a pretrained model
29             seed : int
30                 random seed to train the model
```

[30] `https://huggingface.co/`

```
            num_layers_to_pool : int
                how many layers to consider for the max pooling operations. default = 3.
            pooling_strategy: str
                either max or mean
        """
        super().__init__(config)
        # set attributes
        self.num_labels = config.num_labels
        self.num_layers_to_pool = num_layers_to_pool
        self.pooling_strategy = pooling_strategy
        self.config = config

        # init base model
        self.bert = BertModel.from_pretrained(pretrained_model_name_or_path)
        classifier_dropout = (
            config.classifier_dropout if config.classifier_dropout is not None else config.hidden_dropout_prob
        )
        self.dropout = nn.Dropout(classifier_dropout)
        torch.manual_seed(seed)
        # define additional MHA layer
        self.cls_attention = nn.MultiheadAttention(embed_dim=self.config.hidden_size,num_heads=4,batch_first=True)
        # define classifier
        self.dense = nn.Linear(config.hidden_size, config.hidden_size)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)

        print("Num Labels: ", self.num_labels)
        # Initialize weights and apply final processing
        self.post_init()


    def init_att_weights(self, module):
        """Initialize the weights"""
        if isinstance(module, nn.MultiheadAttention):
            nn.init.kaiming_uniform_(module.in_proj_weight)
            nn.init.constant_(module.in_proj_bias, 0)
            nn.init.kaiming_uniform_(module.out_proj.weight)
            nn.init.constant_(module.out_proj.bias, 0)

    def forward(
        self,
        input_ids: Optional[torch.Tensor] = None,
        attention_mask: Optional[torch.Tensor] = None,
        token_type_ids: Optional[torch.Tensor] = None,
        position_ids: Optional[torch.Tensor] = None,
        head_mask: Optional[torch.Tensor] = None,
        inputs_embeds: Optional[torch.Tensor] = None,
        labels: Optional[torch.Tensor] = None,
        output_attentions: Optional[bool] = None,
        output_hidden_states: Optional[bool] = True,
        return_dict: Optional[bool] = None,
    ) -> Union[Tuple[torch.Tensor], SequenceClassifierOutput]:
        r"""
        labels (`torch.LongTensor` of shape `(batch_size,)`, *optional*):
        Labels for computing the sequence classification/regression loss. Indices should be in `[0, ...,
        config.num_labels - 1]`. If `config.num_labels == 1` a regression loss is computed (Mean-Square loss), If
        `config.num_labels > 1` a classification loss is computed (Cross-Entropy).
        """
        return_dict = return_dict if return_dict is not None else self.config.use_return_dict

        maxpoolbert_inputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )
        # get cls tokens as outputs
        cls_token = maxpoolbert_inputs['last_hidden_state'][:,0]
        hidden_states = torch.stack(maxpoolbert_inputs['hidden_states'][-self.num_layers_to_pool:], dim=1)
        if self.pooling_strategy == 'max':
            max_pool_hidden_states, _ = torch.max(hidden_states, dim=1)
        elif self.pooling_strategy == 'mean':
            max_pool_hidden_states = torch.mean(hidden_states, dim=1)
        else:
            raise ValueError("Unsupported pooling strategy")
        cls_att, _ = self.cls_attention(cls_token.unsqueeze(1), max_pool_hidden_states, max_pool_hidden_states)
        cls_token = cls_att.squeeze(1)
        pooled_output = self.dense(cls_token)
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
```

```
        loss = None
        if labels is not None:
            if self.config.problem_type is None:
                if self.num_labels == 1:
                    self.config.problem_type = "regression"
                elif self.num_labels > 1 and (labels.dtype == torch.long or labels.dtype == torch.int):
                    self.config.problem_type = "single_label_classification"
                else:
                    self.config.problem_type = "multi_label_classification"

            if self.config.problem_type == "regression":
                loss_fct = MSELoss()
                if self.num_labels == 1:
                    loss = loss_fct(logits.squeeze(), labels.squeeze())
                else:
                    loss = loss_fct(logits, labels)
            elif self.config.problem_type == "single_label_classification":
                loss_fct = CrossEntropyLoss()
                loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
            elif self.config.problem_type == "multi_label_classification":
                loss_fct = BCEWithLogitsLoss()
                loss = loss_fct(logits, labels)
        if not return_dict:
            output = (logits,) + output[2:]
            return ((loss,) + output) if loss is not None else output
        return SequenceClassifierOutput(
            loss=loss,
            logits=logits,
        )
```

A.3 Code: AQuA

This section includes the code to calculate AQuA scores. The code to train adapter models is based on the work of Falk and Lapesa (2023a) and can be found at `https://github.com/Blubberli/ArgQualityAdapters`. The trained models for AQuA are available at: `https://github.com/mabehrendt/AQuA`.

Inference

Listing 8: AQuA Inference

```
\UseRawInputEncoding
from transformers import AutoTokenizer, AutoConfig, AutoAdapterModel, AdapterTrainer
from data import InferenceDataset
from torch.utils.data import DataLoader
from tqdm import tqdm
import numpy as np
import torch.nn.functional as F
import torch
import argparse
import os
from utils import get_dynamic_parallel

# check for GPUs or CPU
if torch.cuda.is_available():
    device = torch.device("cuda")
    print('GPU in use:')
else:
    print('using the CPU')
    device = torch.device("cpu")

def predict(dataloader, model, dataset, output_path, task2identifier):
    """
    Predict AQuA scores for a dataset.

    Arguments:
    dataloader:         dataloader for inference
    model:              model used for inference
    dataset:            dataset for inference
    output_path:        path to the output csv file
    task2identifier:    dictionary including adapters and their path
    """
```

```
    output_dic = {}
    for k, v in task2identifier.items():
        output_dic[k] = []
    for id, batch in enumerate(tqdm(dataloader)):
        for k, v in batch.items():
            #if isinstance(v, torch.Tensor):
            batch[k] = v.to(device)
        # output length = num adapters
        with torch.no_grad():
            # output length = num adapters
            outputs = model(input_ids=batch["input_ids"], attention_mask=batch["attention_mask"])
        for i in range(len(outputs)):
            task = list(task2identifier.keys())[i]
            # gives predictions for one batch for one adapter
            predictions = outputs[i].logits
            prediction = torch.argmax(predictions, axis=1).detach().cpu().numpy()
            output_dic[task].extend(prediction)
    # add suffix to adapter predictions to not overwrite other annotations in the data
    for task, preds in output_dic.items():
        dataset[task+"_ad"] = torch.tensor(preds).detach().cpu().numpy()
    adapters = task2identifier.keys()
    score = dataset[list([s + "_ad" for s in adapters])].dot(weights)
    # normalize the score
    dataset["score"] = normalize_scores(score)
    dataset.to_csv(output_path, sep="\t", index=False)

def normalize_scores(scores, bound=5):
    """
    Returns the normalized scores in an interval between 0 and bound.

    Arguments:
    scores: scores to normalize
    bound:  upper interval bound
    """
    return ((scores-minval)/minmaxdif)*bound

weights = [0.20908452, 0.18285757, -0.11069402, 0.29000763, 0.39535126,
        0.14655912, -0.07331445, -0.03768367, 0.07019062, -0.02847408,
        0.21126469, -0.02674237, 0.01482095, 0.00732909, -0.01900971,
        -0.04995486, -0.05884586, -0.15170863, 0.02934227, 0.10628146]

maxval = 4.989267539999999
minval = -1.66928295
minmaxdif = maxval-minval

task2identifier = {"relevance": "trained adapters/relevance",
                   "fact": "trained adapters/fact",
                   "opinion": "trained adapters/opinion",
                   "justification": "trained adapters/justification",
                   "solproposal": "trained adapters/solproposal",
                   "addknowledge": "trained adapters/addknowledge",
                   "question": "trained adapters/question",

                   "refusers": "trained adapters/refusers",
                   "refmedium": "trained adapters/refmedium",
                   "refcontents": "trained adapters/refcontents",
                   "refpersonal": "trained adapters/refpersonal",
                   "refformat": "trained adapters/refformat",

                   "address": "trained adapters/address",
                   "respect": "trained adapters/respect",
                   "screaming": "trained adapters/screaming",
                   "vulgar": "trained adapters/vulgar",
                   "insult": "trained adapters/insult",
                   "sarcasm": "trained adapters/sarcasm",
                   "discrimination": "trained adapters/discrimination",

                   "storytelling": "trained adapters/storytelling"}


task2weight = {"relevance": 0.20908452,
                   "fact":0.18285757,
                   "opinion": -0.11069402,
                   "justification": 0.29000763,
                   "solproposal":0.39535126,
                   "addknowledge": 0.14655912,
                   "question":-0.07331445,

                   "refusers":-0.03768367,
                   "refmedium":0.07019062,
                   "refcontents":-0.02847408,
                   "refpersonal":0.21126469,
                   "refformat": -0.02674237,

```

```
                    "address": 0.01482095,
                    "respect": 0.00732909,
                    "screaming": -0.01900971,
                    "vulgar": -0.04995486,
                    "insult": -0.05884586,
                    "sarcasm":-0.15170863,
                    "discrimination":0.02934227,

                    "storytelling": 0.10628146}

if __name__ == '__main__':
    # read in arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('inference_data', type=str,
                        help='path to the test data')
    parser.add_argument('text_col', type=str, help="column name of text column")
    parser.add_argument('batch_size', type=int)
    parser.add_argument("output_path", type=str, help="path to output file")
    args = parser.parse_args()

    tokenizer = AutoTokenizer.from_pretrained("bert-base-multilingual-cased")
    model = AutoAdapterModel.from_pretrained("bert-base-multilingual-cased")
    adapter_counter = 0

    for k, v in task2identifier.items():
        print("loading adapter %s as adapter %d" % (k, adapter_counter))
        model.load_adapter(v, load_as="adapter%d" % adapter_counter, with_head=True,
                           set_active=True, source="hf")
        adapter_counter += 1
    print("loaded %d adapters" % adapter_counter)
    adapter_setup = get_dynamic_parallel(adapter_number=adapter_counter)
    model.active_adapters = adapter_setup
    test = InferenceDataset(path_to_dataset=args.inference_data, tokenizer=tokenizer, text_col=args.text_col)
    dataloader = DataLoader(test, batch_size=args.batch_size)
    if torch.cuda.is_available():
        model = model.cuda()
    predict(dataloader=dataloader, model=model, dataset=test.dataset,
                output_path=args.output_path, task2identifier=task2identifier)
```