Reasoning for Fact Verification Using Language Models

by

Mohammadamin Kanaani

Bachelor of Computer Engineering, The University of Guilan, 2020

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s):	Ali A. Ghorbani, Ph.D., Computer Science
Examining Board:	Andrew McAllister, Ph.D., Computer Science, Chair
	Michael Fleming, Ph.D., Computer Science
	Hamed Asgari Moslehabadi, Ph.D., Mechanical Engineering

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

February, 2024

Abstract

In response to the proliferation of misinformation on social media platforms, this thesis introduces the Triple-R framework (Retriever, Ranker, Reasoner) to enhance fact-checking by leveraging the Web for evidence retrieval and generating understandable explanations for its decisions. Unlike existing methods, Triple-R incorporates external sources for evidence and provides explanations for datasets lacking them. By fine-tuning a causal language model, it produces natural language explanations and labels for evidence-claim pairs, aiming for greater transparency and interpretability in fact-checking systems. Evaluated on a popular dataset, Triple-R achieved a state-of-the-art accuracy of 42.72% on the LIAR benchmark, outperforming current automated fact verification methods. This underscores its effectiveness in integrating web sources and offering clear reasons, presenting a significant step forward in the fight against online misinformation.

Dedication

This thesis is dedicated to my beloved wife, whose unwavering support, love, and encouragement have been essential throughout this challenging journey. Her patience and understanding have served as my solid foundation, while her confidence in my abilities has driven my determination to achieve this milestone. I also express my gratitude to my family, for their inspirational role and strength. Their continuous encouragement and sacrifices have been instrumental in my progress. This thesis stands as a tribute to their constant support.

Acknowledgements

I owe a profound debt of gratitude to Professor Ali Ghorbani, my supervisor, for his unparalleled guidance, support, and mentorship throughout my thesis journey. Collaborating with the Canadian Institute for Cybersecurity (CIC) under his supervision has been both an honor and a transformative experience. I am equally thankful to my colleagues at CIC, whose expertise and dedication have significantly contributed to my research. Their collaborative spirit under Professor Ghorbani's leadership has not only propelled this thesis forward but also equipped me with invaluable lessons and skills for my professional future. This period stands as a pivotal chapter in my life, marking a turning point that has prepared me for upcoming challenges and opportunities.

Table of Contents

List of Tables

List of Figures

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
XAI	Explainable Artificial Intelligence
NLP	Natural Language Processing
LM	Language Model
SVM	Support Vector Machine
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term memory
BiLSTM	Bidirectional Long Short-Term Memory
BERT	Bidirectional Encoder Representations from Transformers
LM	Language Model
LLM	Large Language Model
ICL	In-Context Learning
CoT	Chain of Thought
GPT	Generative Pre-Trained Transformers
LoRA	Low-Rank Adaptation
QLoRA	Quantized Low-Rank Adaptation
NF4	4-bit Normal Float
PEFT	Parameter-Efficient Fine-Tuning
RLHF	Reinforcement Learning from Human Feedback
PPO	Proximal Policy Optimization

Table 1: List of Abbreviations mentioned in this thesis.

Nomenclatures

Symbol	Definition
$\sum$	Summation
$\prod$	Multiplication
$\cdot$	Inner product
f	Function
$\mathcal{L}$	Loss function
λ	A hyper-parameter controlling the weight of statistical model scores
$\mathbb{R}$	Real numbers

Table 2: List of Nomenclatures mentioned in this thesis.

Chapter 1

Introduction

1.1 Introduction

In recent years, social media platforms have witnessed an exceptional surge in their number of users. By October 2023, the global count of internet users had reached 5.3 billion, constituting 65.7% of the world's population. Among these, 4.95 billion individuals, equivalent to 61.4% of the global populace, were active users of social media platforms. [4]. This increasing trend of social media popularity is proving to be advantageous for both individuals and business owners.

In addition to the entertainment aspects of social media, these platforms, such as X and Facebook, have become a primary source of news for a large segment of the population. The majority of U.S. adults (86%) frequently or occasionally receive news via smartphones, computers, or tablets, with 56% doing so often—surpassing 49% in 2022 and 51% in 2021. However, news consumption from television remains relatively stable at 31% in 2022 and 32% in 2023 for those who often receive news. Radio and print publications are less popular sources compared to digital devices and television [3].

While these platforms offer various benefits, they are also susceptible to the spread

of misinformation, such as fake news and rumors. This susceptibility poses risks of significant political, social, and financial harm. Recent surveys indicate a mere 21% trust in social media content among Canadians [5]. A misleading news story can lead to plummeting stock prices, damage a business's reputation, or create unrealistic customer expectations. For instance, a false announcement from an imposter Eli Lilly pharmaceutical account claiming "insulin is free now" resulted in a 4.5% drop in the company's stock [2].

Moreover, recognizing false news is increasingly challenging due to the emergence of a dedicated industry. Individuals are hired to produce sensational stories and craft clickbait content, aiming to boost website traffic. Both misinformation (presenting misleading information as fact) and disinformation (intentionally spreading false information) can be camouflaged to appear legitimate, quickly reaching thousands of users within minutes.

Several fact-checking websites, such as Snopes[1], FactCheck.org[2], and Full Fact[3], strive to identify instances of misinformation. Figure 1.1 shows an example from the PolitiFact website. These platforms heavily depend on human involvement. However, the overwhelming volume of fake news, rumors, and misinformation circulated daily on social media challenges the capacity of fact-checking websites to promptly detect and interpret such content. As a result, there is a pressing need for an automated system that can accurately and efficiently assess the veracity of claims made on social media.

To automate fact verification, natural language processing techniques can be leveraged. The goal of fact verification is primarily to classify claims as either true or false. Numerous studies have attempted to address this problem by leveraging text classification methods such as machine learning and deep learning algorithms. Some

[1]https://www.snopes.com
[2]https://www.factcheck.org
[3]https://fullfact.org

Figure 1.1: Illustration of claims scrutinized by the PolitiFact website, where assessments rely on credible third-party sources, leading to the assignment of a corresponding label based on the evaluation.

studies extract textual features from content and apply machine learning algorithms on these features to classify them [6, 66]. However, some works use deep learning to train end-to-end models [37, 44].

Beyond textual information, instances arise where the text alone may lack non-factual claims; however, when coupled with other modalities like images, it can contribute to the formation of deceptive claims. Consequently, several studies explore the application of multi-modal models for effectively classifying pairs of text and images [89, 80].

Analyzing fake news based solely on its content is insufficient because fake news is inherently designed to deceive, posing a unique challenge. Unlike authentic news, the deceptive intent of fake news complicates straightforward content analysis. Even expert fact-checkers frequently rely on cross-referencing multiple sources to draw accurate conclusions about a claim. Therefore, incorporating additional contextual information into the fact-checking process becomes crucial, as it enhances the models' ability to discern the nuances and deceitful nature of fake news, ultimately improving overall accuracy.

Integrating context in fact verification systems is another challenge due to various sources. On social media, user profiling, source assessment, the relationship between

users, propagation of claims, etc., are mainly taken into account [76, 78, 57].

Another kind of information is textual data that supports or opposes the claim. A popular method to incorporate textual data is knowledge graphs. A knowledge graph is a graphical representation of a large amount of interconnected data, organized and linked together to represent concepts and relationships between entities. In this method, mainly the entities are extracted from the claim, and then their associated relationships are fetched from the knowledge graph [53, 34]. However, extracting the relationship between entities and creating the corresponding graph is challenging. In general, creating a knowledge graph consists of four steps, coreference resolution, named entity recognition, entity linking, and relationship extraction. Each step requires different settings and methods, which makes creating a knowledge graph an intricate process.

Contextual information can also be derived from evidence presented in natural language. This approach involves retrieving pieces of evidence associated with claims from a factual information database and subsequently processing both the claim and its corresponding evidence. However, a significant challenge lies in determining the source of knowledge and developing effective methods for evidence extraction. Some studies utilize information from Wikipedia as a source of facts, as evidenced by works like Thorne et al. [82] and Sathe et al. [70]. Other research efforts, such as the dEFEND work [75], focus on extracting relevant comments associated with claims. Additionally, search engines are explored as external knowledge sources, as exemplified in the work by Popat et al. [62].

Fact verification systems often lack explainability, even with a detection component. If such a system labels a human claim, users may be hesitant to trust the decision of the model without sufficient justification. Furthermore, the provision of explanations can aid non-expert users in comprehending domain-specific claims. Finally, they would provide researchers with insight into the logic behind the decision-making of

such systems, facilitating the development of more robust systems. Therefore, the incorporation of explainable models is an essential component of fact verification systems.

Some methods aim to gauge the significance of input features, typically words, by employing attention mechanisms that assign scores to each word [62] or via post-hoc methods [8, 81]. Additionally, certain approaches focus on identifying triples or Horn rules to delineate the reasoning path of the model [26, 7, 49]. However, interpreting the role of these important words or ruling triples in the model's decision remains the responsibility of the user, potentially limiting the overall clarity of the explanation. While retrieved evidence aids in detection, certain methodologies focus on scoring the evidence list and presenting the most relevant pieces as explanations for the model's decision [9, 75]. Furthermore, models can justify their decisions in natural language, enhancing comprehension for human users [85]. Nevertheless, training a model capable of generating explanations in the natural language format requires a dataset with annotated explanations, hindering the development of such systems.

1.2 Motivation and Objectives

This research seeks to overcome the limitations of prior studies by developing an interpretable fact verification system that considers claim content. Our framework, Triple-R, comprises three key components—Retriever, Ranker, and Reasoner—each playing a distinct role in the fact verification process. The Retriever sources external knowledge from the web, while the Ranker employs a hybrid method to score internet-gathered text, forming evidence related to the claim. Subsequently, we employ an auto-regressive language model based on the transformer architecture to generate both a reason grounded in the evidence and to assign a label to the claim. Additionally, in addressing the challenge of data scarcity in explainable fact verifi-

Claim: Over 10 years, (extending tax cuts for the wealthy) would add less to the deficit than Obama added with the stimulus in one year.
Evidence: 1. "Over 10 years, (extending tax cuts for the wealthy) (...) 2. "The president says we can't afford the tax cuts for (...) 3. Ultimately, we feel that Will had a point worth making (...) 4. Just as the upper-income tax cuts are being charged (...)
Reason: The statement is partially true but lacks important context. While it is true that the tax cuts for the wealthy would add less to the deficit over a 10-year period than the stimulus did in one year, it is misleading to suggest that the stimulus is vastly more expensive than the tax cuts. The stimulus package included various funding streams and tax breaks, and the Congressional Budget Office (CBO) reported that it would increase the deficit in each year until 2015 before reducing it in subsequent years. Therefore, it is important to consider the overall impact of the stimulus and the tax cuts over a longer period rather than comparing their individual yearly costs.
Label: barely-true

Table 1.1: Table showcasing an example claim analyzed through the Triple-R framework, utilizing web-based information retrieval to find pertinent evidence. The framework subsequently formulates a reasoned judgment and assigns a label indicating the veracity of the claim.

cation systems, we introduce a method leveraging the reasoning capabilities of large language models to create synthetic rationales for datasets lacking human-annotated reasons. This synthetic dataset is then used to fine-tune a smaller language model. Furthermore, we leverage novel techniques to mitigate the computational cost associated with training and inferencing large language models. Table 1.1 illustrates an example analyzed by the Triple-R framework.

1.3 Summary of Contributions

This section presents the main contributions of the research, highlighting the advancements in the field of explainable fact verification. The research has made significant strides in developing a framework that combines various powerful language models to present an explainable fact verification system. Additionally, the research

introduces techniques to create a synthetic dataset to train language models.
In summary, our contributions are as follows:

- We propose a novel causal language model designed specifically for logical reasoning in the domain of fact verification. This model provides natural language explanations detailing the rationale behind its assessment of a claim's veracity. By generating this reasoning process explicitly, our method enables greater transparency and interpretability compared to standard classification models.

- We introduce a teacher-student training approach that transfers knowledge from a large, powerful language model to a smaller, more efficient model. This allows us to leverage the reasoning abilities of cutting-edge models while maintaining feasible computational requirements for real-world usage.

- We develop an innovative hybrid ranker combining neural and statistical methods to retrieve related textual evidence from the web. The ranker serves a dual purpose - the extracted passages provide vital contextual information to inform the reasoning process and also act as additional justification to accompany the generated explanation.

1.4 Thesis Organization

In the upcoming chapters, we conduct a thorough literature review on fake news detection in Section 2. Section 3 outlines the architecture and components of our proposed models in detail. The implementation specifics of the proposed method, along with the experimental results and model evaluations, are detailed in Section 4. Lastly, the concluding section presents our final reflections and suggests potential avenues for future research.

Chapter 2

Literature Review

In this chapter, we delineate the research landscape of fact verification by classifying existing approaches into three broad categories: Content-based Detection, Context-enriched Methods, and Explainable Fact Verification. Additionally, we review some works concerning explainable AI that are relevant to the issue of fact verification.

2.1 Content-based Detection

In the realm of fact verification, content-based detection approaches aim to determine the truthfulness of a claim based solely on the features presented within its content. To this end, researchers have explored various techniques for extracting and analyzing textual features.

Ahmad et al. [6] introduced an ensemble learning strategy employing linguistic features to detect fake news across diverse domains. They utilized the LIWC2015 tool to extract 93 numerical features, encompassing aspects like emotion, grammar, and punctuation. These features were employed to train machine learning algorithms, including logistic regression, SVM, and ensemble techniques like bagging, boosting, and voting classifiers. Results from experiments on real-world datasets covering politics, sports, tech, and entertainment domains exhibited superior performance in

accuracy, precision, and recall for ensemble models compared to individual learner baselines. Notably, XGBoost-based boosting demonstrated the best outcomes by effectively amalgamating weak classification models, minimizing errors. The key innovation lies in incorporating model diversity through various ensemble methods and linguistic inputs, enabling more generalized fake news identification across domains. The integration of representation learning and classifier fusion yielded significant improvements over single-model approaches, overcoming limitations observed in prior works focused solely on specific political domains or features.

Pérez-Rosas and colleagues [61] introduced two fresh datasets for detecting fake news, acquired through crowdsourcing and sourced from celebrity news websites. They constructed classification models utilizing n-gram, punctuation, psycholinguistic, readability, and syntax features. The top-performing models achieved accuracies comparable to human evaluators, surpassing human judgment on the diverse news dataset while demonstrating similar performance on celebrity news. Linguistic analysis uncovered distinctions in social, positive, certainty, and time-focused words between fake and authentic news. The noteworthy contributions encompass the creation of new datasets spanning various domains, a comparative linguistic analysis revealing semantic differences in fake content, and the development of highly accurate classifiers relying on textual features rather than source heuristics. This underscores the effectiveness of computational linguistics in detecting fake news across diverse domains. Future endeavors could involve expanding the datasets, incorporating indicators of source reliability, and investigating the impact of larger training sizes.

The FNDNet technique [37] proposed a deep convolutional neural network architecture designed for detecting fake news. Unlike traditional approaches relying on manually crafted features, FNDNet is structured to autonomously learn distinctive features through multiple hidden layers. It employs parallel convolution layers with various kernel sizes applied to word embeddings from GloVe, facilitating feature ex-

traction at different levels. The subsequent application of max pooling, flattening, and dense layers generates predictions. Notably, experiments conducted on well-established fake news datasets reveal a state-of-the-art accuracy of 98.36%, showcasing a noteworthy advancement compared to existing machine learning and deep learning methods in fake news classification. The authors assert that FNDNet stands out as a promising deep learning model for precise fake news detection. Key aspects worthy of emphasis in a literature review encompass automated feature learning, the integration of multiple parallel convolutional layers, and its superior performance compared to prior methodologies.

Abdul Nasir et al. [58] introduced an innovative hybrid deep learning model, merging convolutional and recurrent neural networks (CNN-RNN) to classify fake news. The model employs a CNN layer to extract local text features initially, followed by an LSTM layer to capture long-term dependencies among these features. Through experiments on FA-KES (comprising 376 fake and 426 real articles on the Syrian war) and ISOT (with 23,481 fake and 21,417 real articles), the hybrid approach outperforms non-hybrid baseline methods such as logistic regression, random forests, and standalone CNN or RNN models across metrics like accuracy, precision, recall, and F1-score. Notably, the model achieves an outstanding accuracy of 99% on the ISOT dataset. The authors also undertake initial generalization experiments, training on ISOT and testing on FA-KES. They address overfitting limitations and discuss future enhancements to enhance model portability across datasets. In summary, this research underscores the potential of deep neural networks, particularly hybrid approaches, in automating the detection of fake news.

Kaliyar et al. [36] suggested a new deep learning method named FakeBERT designed for detecting fake news on social media. FakeBERT merges a pre-trained bidirectional transformer model, known as BERT [21], with multiple parallel 1D convolutional neural network blocks. This combination effectively captures semantic

nuances and long-distance dependencies in sentences. The results showcase FakeBERT's superior performance over previous benchmarks, achieving a 98.9% accuracy on a real-world dataset from the 2016 US Presidential Election. The authors emphasize that the bidirectional training approach of FakeBERT enables it to model pertinent information in fake news more effectively than earlier unidirectional models. Additionally, they suggest that FakeBERT's fusion of BERT and 1D-CNN adeptly handles ambiguity, a critical aspect in tasks like fake news detection within natural language understanding. In summary, Kaliyar et al. present FakeBERT as a precise deep learning architecture for discerning fake content and misinformation based on writing style and news content.

2.2 Context-Enriched Methods

Evaluating the credibility of factual information in textual content presents distinctive challenges that set it apart from other text classification tasks. Content-based detection systems face two primary issues. Firstly, false claims are crafted to closely mimic true claims, often sharing similar linguistic features. This makes it challenging to distinguish between them. Secondly, the dynamic nature of events and news poses a challenge; once a fact verification system is trained, it lacks information about future events, hindering its ability to identify claims beyond the training dataset. To enhance the efficacy of fact verification systems, several studies have incorporated contextual information, with a focus on two main types of external context: social media information and textual evidence.

2.2.1 Social Media Information

In the realm of social media, numerous factors, such as users' information, relationships between users, and patterns of news propagation, among others, can signifi-

cantly impact the veracity of presented facts. Consequently, researchers have investigated various forms of this auxiliary information to evaluate the facts disseminated through social media.

The TriFN approach, as presented in [77], introduces a framework for detecting fake news by leveraging the tri-relationship among publishers, news content, and social media engagements. The method contends that jointly modeling these three components enhances the classification of fake news by capturing complementary information. Specifically, it recognizes that publisher partisan bias can indicate potential manipulation, while considering user credibilities and stances in engagements also provides signals regarding veracity. The TriFN framework employs non-negative matrix factorization to learn latent feature representations of news content, integrates user credibility and homophily in the modeling process, and utilizes semi-supervised learning.

Through experiments conducted on two comprehensive fake news datasets, TriFN demonstrates superior performance compared to methods that solely focus on news content or social context. It achieves a notable 5-9% relative improvement in accuracy. These findings underscore the significance of concurrently modeling the relationships between publishers and news, as well as news and users, for an effective fake news detection system.

Shu et al. [79] explored the utilization of social media user profiles for identifying fake news. Their study involved identifying user groups based on their tendencies to share either fake or real news, examining both explicit profile features available in metadata and implicit features derived from content and behaviors. The analysis highlighted disparities in the distribution of real and fake news sharers, particularly in aspects such as location, profile images, and political bias features. These features demonstrated effectiveness in classifying fake news, consistently surpassing existing content-based methods, with implicit features, such as political bias, proving to be

the most distinguishing. The findings suggest that incorporating user profile signals, especially implicit ones, holds promise for improving fake news detection on social media.

Dou et al. [22] introduced a unique framework named UPFD (User Preference-aware Fake News Detection) aimed at enhancing fake news detection. In contrast to previous studies concentrating solely on news content, UPFD innovatively integrates user preferences and social context. It employs BERT to encode both news content and user preferences, incorporating historical posts to capture endorsement bias. Additionally, UPFD represents social context through a propagation graph, and a graph neural network combines endogenous user preferences with exogenous context for accurate news veracity classification. Experimental results on two Twitter datasets indicate that UPFD surpasses existing models, underscoring the significance of incorporating user preferences alongside news content in fake news detection. This multi-modal strategy opens a novel avenue for leveraging both user psychology and social engagement to combat misinformation.

2.2.2 Textual Evidence

The DEAP-FAKED system [54] presented an innovative approach to fake news detection using a knowledge graph-based framework. The prevalence of fake news on social media poses substantial challenges, influencing significant events such as elections and the COVID-19 pandemic. In this method, bidirectional LSTM models encode the textual content of news titles to form representations. Additionally, named entities within the titles are identified, linked to a knowledge graph (Wikidata), and represented using ComplEx embeddings. These news and entity representations are then combined and inputted into a classifier for fake news detection.

The system's effectiveness is evaluated on two datasets: one sourced from Kaggle, consisting of approximately 14,000 political/tech/business articles, and another from

the COVID-19 dataset (CoAID) with 632 preprocessed articles. To mitigate biases, such as news sources, and prevent spurious correlations, the evaluation removes them. The results showcase notable improvements, with F1 scores reaching 88% and 78% on the respective datasets. This signifies a 21% and 3% enhancement over baselines, emphasizing the significance of incorporating knowledge graph information to complement textual representations. The study suggests potential avenues for further exploration, such as experimenting with various combinations of knowledge graph and Wikipedia entity encoders.

Hu et al. [34] introduced an innovative end-to-end graph neural network named CompareNet for detecting fake news. The main concept involves evaluating news content against external information from Wikipedia using entities to determine its credibility. The paper constructs a directed heterogeneous document graph that includes sentences, topics, and entities for each news document. To enhance the news representation, a heterogeneous graph attention network is applied, capturing both topic-enriched news representation and contextual entity representations that encode the news content's semantics. These contextual entity representations are then compared to corresponding entity representations from both structured knowledge graph triplets and unstructured Wikipedia entity descriptions using a carefully crafted comparison function. The resulting entity comparison features are integrated with the news content representation for the final classification of fake news.

Experiments on two benchmark fake news datasets reveal that the proposed CompareNet model outperforms existing methods significantly. The comparisons to external knowledge and the inclusion of topic information are demonstrated to provide valuable signals. The directed heterogeneous document graph is designed to aggregate evidence from external knowledge effectively without distorting the news representation. This paper presents a fresh perspective on utilizing external knowledge for fake news detection through entity comparison.

2.3 Explainable Fact Verification

Szczepański et al. [81] introduced a novel approach to enhance the interpretability of BERT-based fake news detection models. Emphasizing the necessity for AI systems to be both comprehensible and trustworthy, the authors advocate for the adoption of model-agnostic explanation techniques such as LIME and Anchors. These methods can be seamlessly integrated into existing models, offering localized explanations without necessitating model retraining. The experimentation involved BERT and BiLSTM classifiers applied to fake news datasets. Evaluation encompassed various test scenarios, including accurately and inaccurately classified instances of both fake and real news. The findings indicate that these explanation methods effectively capture significant patterns influencing model decisions, revealing some shared highlighted words between LIME and Anchors. Nevertheless, Anchors occasionally struggled to provide explanations for fake news. The authors suggest that combining multiple explanation methods yields more comprehensive insights, asserting that this approach facilitates the application of interpretability to opaque models like BERT-based detectors.

Popat et al. [62] introduced DeClarE, a neural network system designed for evaluating the credibility of textual claims. DeClarE operates as an end-to-end model, autonomously assessing the credibility of natural language claims without relying on handcrafted features or lexicons. A notable feature is its capacity to integrate signals from external evidence articles on the web, offering additional context to the claim under scrutiny. The model employs a bidirectional LSTM for language modeling of the evidence articles and incorporates an attention mechanism to highlight the most relevant sections pertaining to the claim. DeClarE also innovatively considers multiple factors influencing credibility, such as the language within the articles, the credibility of sources, and the context of the claim. Through extensive experiments across four datasets, DeClarE demonstrated superior performance compared to ex-

isting state-of-the-art methods in credibility classification and regression. Notably, DeClarE is praised for its interpretability; the attention weights it generates allow for the identification of key excerpts from evidence articles contributing to the credibility assessment. This transparency enhances the model's explanatory power in justifying its predictions.

Chen et al. [15] introduced the Hierarchical Reasoning-based Heterogeneous Graph Neural Network (HHGN) method for fact verification. Traditional methods often rely on single-granularity sentence-level features to represent evidence, neglecting valuable features such as entity-level and context-level information, and lacking interpretability in reasoning. In response, HHGN utilizes a heterogeneous graph, representing entities, sentences, and context, to incorporate multi-granularity features. It employs a hierarchical reasoning process to enhance interpretability by propagating features from fine-grained to coarse-grained nodes. Initially, HHGN constructs a heterogeneous graph from retrieved evidence sentences and initializes node representations using BERT. Subsequently, it propagates features hierarchically from entities to sentences to context. The model extracts reasoning paths, predicts labels for each path based on claim relevance, and aggregates path results for the overall prediction. Experimental results on the FEVER dataset showcase HHGN's superiority over competitive baselines in both single and multiple evidence scenarios. Notably, in multiple evidence cases, HHGN achieves a 3.42% increase in accuracy and a 5.09% improvement in FEVER Score compared to the best baseline. Additional analysis highlights the interpretability of HHGN's evidence reasoning process.

FACE-KEG [85] is a novel deep learning framework for explainable fact checking. Given an input fact or claim, it constructs a relevant knowledge graph and retrieves contextual textual information from a knowledge base. The knowledge graph and textual context are encoded using a graph transformer network and bidirectional RNN respectively. This encoded information is then utilized to jointly train a classi-

fier to predict the veracity of the input claim, as well as an RNN decoder to generate a natural language explanation justifying the predicted veracity.

A key contribution of this work is providing concise, coherent textual explanations for fact checking decisions, instead of just predicting a veracity label. The graphical representation and additional context allow FACE-KEG to learn more informative explanations compared to logic rules or knowledge base patterns used in prior work. Evaluations on three large-scale datasets demonstrate that FACE-KEG outperforms competitive baselines in accurately detecting fact veracity. Human judgments also indicate that the generated explanations are of higher quality - more relevant, adequate, and coherent in justifying veracity predictions.

2.4 Explainable Artificial Intelligence

Explainable artificial intelligence (XAI) refers to methods and techniques that aim to make AI and machine learning models more interpretable and explainable. As AI systems are increasingly deployed in high-stakes domains like finance, healthcare, and defense, there is a need for the decisions and predictions made by these "black box" models to be understandable by humans. XAI encompasses approaches that help demystify how models work internally or explain the reasoning behind individual decisions.

The XAI can be categorized into three broad types - intrinsically interpretable models, model-agnostic methods, and example-based explanations. Intrinsically interpretable models like linear regression and decision trees are transparent in how they operate. Model-agnostic methods can explain the predictions of any machine learning model by various means like feature importance, partial dependence plots, and local surrogate models. Finally, example-based methods select influential, prototypical or counterfactual instances from the dataset to provide explanation analogies for

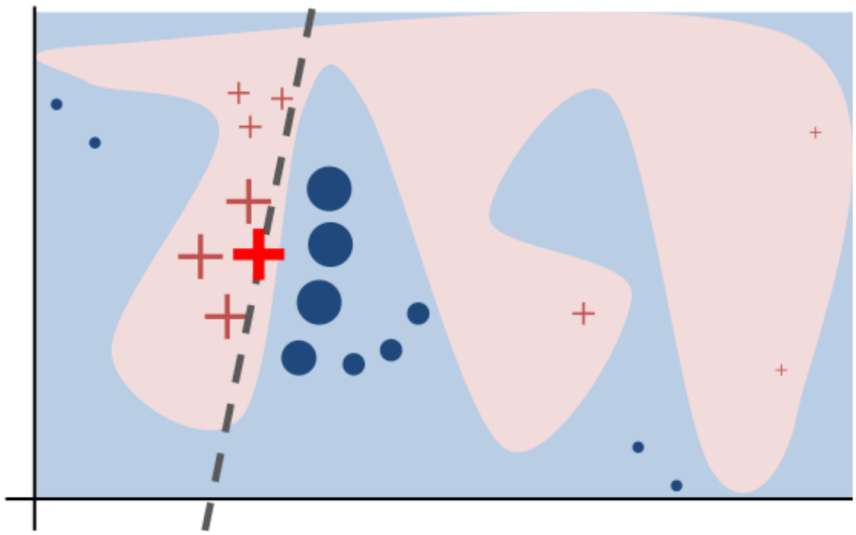


Figure 2.1: The figure illustrates an example elucidating the intuition behind LIME. The blue/pink background represents the complex decision function f of the black-box model (unknown to LIME). The bold red cross denotes the instance under explanation. LIME samples instances, obtains predictions using f, and assigns weights based on proximity to the explained instance (indicated by size). The dashed line represents the locally faithful learned explanation, capturing the model's behavior in the vicinity of the specific instance.

model behavior. Each category has its own advantages and limitations in enabling model transparency. Ongoing research is focused on developing XAI techniques that can provide faithful, customized and quantifiable explanations [35]. To enhance our comprehension of XAI, we delve into two widely used model-agnostic methods: Local Interpretable Model-agnostic Explanations (LIME) [67] and Deep Learning Important Feature (DeepLIFT) [74].

LIME (Local Interpretable Model-Agnostic Explanations) is a technique to explain the predictions of any machine learning classifier or model in an interpretable manner. The key idea is to approximate the model locally around a specific prediction using an interpretable and simple model like a linear model or decision tree.

Formally, for a model f and an instance x being predicted, LIME learns an interpretable explanation model g from a dataset Z of perturbed samples around x. The domain of g is $\{0,1\}^{d'}$, i.e. g acts over absence/presence of the interpretable compo-

nents. The perturbed samples Z are generated by randomly turning interpretable components on and off. The samples are weighted by their proximity $\pi x(z)$ to x. The explanation $\xi(x)$ is obtained by solving an optimization problem that minimizes the locality-aware loss L between f and g on Z, regularized by the complexity $\Omega(g)$ of g. Figure 2.1 depicts an example of LIME method. This ensures both local fidelity as well as interpretability of the explanation:

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi x) + \Omega(g)$$

By explaining individual predictions, LIME aims to provide interpretability and assess trust in models. The paper also proposes SP-LIME, which selects a diverse, representative set of explanations to provide a global understanding of a model. Experiments on text and image models, with both simulated and human subjects, demonstrate the utility of LIME explanations for various trust-related tasks.

DeepLIFT is a method for explaining the output predictions of neural networks by assigning importance scores to the inputs. It propagates an importance signal backward through the network layers in a single pass, assigning contribution scores $C\Delta x_i \Delta t$ to the difference from reference Δx_i of each input neuron. The key idea is to explain the difference in output Δt relative to a reference output t_0. By using difference-from-reference instead of absolute values, DeepLIFT can assign importance even when gradients are zero, avoiding the limitations of gradient-based methods.

The DeepLIFT rules define how to compute contribution scores and multipliers $m\Delta x_i \Delta t$ layer-by-layer via backpropagation. The summation-to-delta property requires that the contribution scores sum up to the output difference Δt. A unique aspect of DeepLIFT is allowing separate treatment of positive and negative contributions with the RevealCancel rule, enabling it to identify dependencies missed by other methods in situations where positive and negative terms cancel out in nonlinear

neurons. Experiments show DeepLIFT assigns more meaningful input importance scores compared to gradients or gradient x input on tasks like MNIST digit classification and motif discovery in genomics.

2.5 Concluding Remarks

In the literature review chapter, we explore three main themes: Content-based Detection, Context-Enriched Methods, and Explainable Fact Verification. In Content-based Detection, Ahmad et al.'s ensemble strategy, Pérez-Rosas's new datasets, and FNDNet's deep learning model stand out for their effectiveness in fake news identification. Moving to Context-Enriched Methods, TriFN and Shu et al. emphasize the importance of social media information, while UPFD integrates user preferences and social context, outperforming existing models. DEAP-FAKED leverages a knowledge graph, and CompareNet uses external knowledge and topic information, both showcasing significant improvements.

Explainable Fact Verification explores approaches enhancing interpretability. Szczepański et al. propose model-agnostic explanation techniques, DeClarE demonstrates superior performance and interpretability, HHGN introduces a hierarchical reasoning-based model, and FACE-KEG not only predicts veracity but also provides coherent textual explanations, outperforming baselines.

Additionally, the literature delves into Explainable Artificial Intelligence (XAI) and two widely used model-agnostic methods: Local Interpretable Model-agnostic Explanations (LIME) and Deep Learning Important Feature (DeepLIFT). LIME provides interpretable explanations for black-box models, enhancing trust. DeepLIFT assigns meaningful input importance scores compared to other methods.

The Proposed Method chapter will build upon these insights, detailing the techniques and methods used to implement the fact verification approach.

Chapter 3

Proposed Approach for Fact Verification

3.1 Overview

In the present day, a majority of individuals prefer obtaining the latest news and information through social media channels like X[1]. However, the open nature of these platforms allows for the rapid dissemination of false information. Consequently, when confronted with a claim, one potential method to verify its authenticity is to conduct an online search. Search engines employ advanced algorithms, which often present the most relevant links in the initial results. By exploring the content of these early websites, one can find relevant information pertaining to the claim. Utilizing this collected evidence from the web, one can make an informed judgment regarding the legitimacy of the claim.

In the context of fact verification, the process of determining the validity of a claim is aimed to be replicated. This process typically involves conducting online searches to gather relevant information and evidence, which is then analyzed to assess the veracity of the claim. To emulate this approach, a three-stage fact verification model

[1]https://twitter.com/

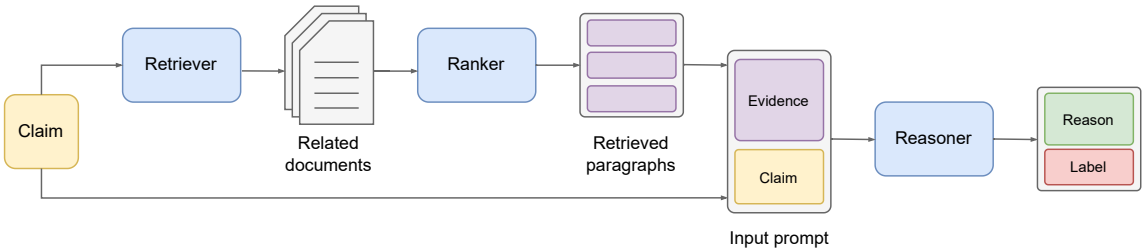


Figure 3.1: The three stages for fact verification, designed to emulate the step-by-step process a human would undertake when assessing the validity of a claim. These stages involve information gathering, evidence synthesis, and logical reasoning to arrive at a conclusion regarding the claim's veracity. The fact verification model aims to replicate the human-like approach for determining the credibility of the given claim.

has been developed, as depicted in Figure 3.1. This model seeks to mimic the human process of information gathering, evidence synthesis, and logical reasoning to arrive at a conclusion regarding the validity of a given claim.

This proposed framework to assess the veracity of a claim is named **Triple-R**, consisting of **R**etriever, **R**anker, and **R**easoner. The stages are discussed briefly below:

- **Retriever:** This stage involves sourcing information related to the claim from the web, which is used as an external knowledge base.
- **Ranker:** In this stage, a score is assigned to each paragraph of the retrieved documents based on their similarity to the claim. The top-scoring paragraphs are then selected as evidence.
- **Reasoner:** The final stage employs a generative language model to simultaneously label the claim and provide a rationale for its veracity based on the collected evidence.

3.2 Retriever

During the fine-tuning process, a pre-trained language model acquires a deep understanding of the specific domain it is being trained on. For instance, in sentiment

analysis, when provided with a sufficiently large dataset, the model discerns crucial features like semantics, syntax, and sentiment-related words that influence the sentiment of a given text. This acquired knowledge enables the model to effectively classify input texts based on their sentiments. Once trained, the model can maintain its classification capabilities over time, delivering consistent performance without significant degradation.

In the fact verification task, the scenario differs from other tasks as false information is intentionally crafted to mimic true information, making it challenging to distinguish between them. The crux of the matter lies in the facts presented in the claims, which is the most significant difference between correct and erroneous information. Given the constantly evolving nature of news, new events introduce novel facts that were not encountered during model training, leading to difficulties in detecting false claims that arise independently of past events. Even human fact-checkers, with their expertise, often face the challenge of assessing new events without external references. This makes fact verification a complex and dynamic task that requires specialized approaches.

To overcome the mentioned challenge, incorporating an external knowledge source into fact verification systems proves to be beneficial as it enables the model to access relevant information. In this study, we leverage the web as the external source to provide additional context for the model. The web stands as a prime resource due to its vastness and up-to-date content, making it a valuable repository of information. Moreover, search engines employ sophisticated algorithms to index websites and retrieve the most pertinent information related to input queries, further enhancing the model's access to relevant data. By tapping into this external knowledge, the fact verification system gains an edge in handling the dynamic nature of fact-checking.

To gather relevant information, a search engine is employed to search the exact claim verbatim, without any further preprocessing. Employing the claim verbatim

as a search query may be a straightforward method, but it serves as a suitable initial step, particularly considering that search engines often incorporate supplementary procedures to enhance user interactions. Then, the search engine tries to find the most relevant articles based on the input claim. Several search engine APIs are available for this purpose, including the Google Search API[2], Mojeek Web Search API [3], and Bing Web Search API [4]. In this study, the Bing Search API is utilized as the retriever. After performing a verbatim search of the claim, the URLs of the top k articles are selected. These URLs are then processed through a scraper to extract the content from these articles. Finally, the retrieved articles undergo preprocessing steps to eliminate unwanted elements, such as URLs and noisy punctuation.

3.3 Ranker

When the retriever finds the related URLs to the claim, it extracts all the text from the HTML content of the URLs, including content that is not part of the main article, such as footers, headers, menu names, ads, etc. These extraneous elements can introduce noise and negatively impact the system's performance. Furthermore, articles may span multiple pages and contain paragraphs that are irrelevant to the input claim, adding to the noise in the data. The model in the reasoner component also has a limited context length, and articles with many pages can exceed this limit. To filter out unwanted content, a ranking model is employed to assess the relevance of information within the articles. The articles are first segmented into paragraphs, and then both the paragraphs and the claim are encoded as numerical vectors to calculate the similarity scores between the paragraphs and the claim. Inspired by [52], a hybrid model is utilized for ranking, incorporating both a term-based model and a neural network model to score each paragraph. This hybrid approach enables

[2]https://developers.google.com/custom-search
[3]https://www.mojeek.com/services/search/web-search-api/
[4]https://www.microsoft.com/en-us/bing/apis/bing-web-search-api

accurate identification and extraction of the most pertinent paragraphs while mitigating the influence of irrelevant content. In the following sections, both models will be discussed in detail.

3.3.1 Neural Network Model

Transformer-based neural language models have become state-of-the-art for generating contextualized text representations. As described in Appendix A.6.1, pre-trained transformer encoders can encode both individual tokens and entire sequences into dense vector embeddings that capture semantic meaning. This advanced capability enables us to effectively map textual claims and evidence paragraphs into numerical representations suitable for comparison and analysis. In this study, we leverage the capabilities of a readily available pre-trained transformer encoder known as DistilBERT [69].

DistilBERT is a transformer encoder that employs a technique known as knowledge distillation during its pre-training phase. In this approach, the authors use the BERT [21] model as a teacher and the DistilBERT model as a student to enhance learning. The overall architecture of DistilBERT closely resembles BERT, with a specific focus on optimizing computational efficiency by reducing the number of layers by half. To achieve this, the authors initialize the student model with knowledge from the teacher model, employing every other layer. DistilBERT is trained on the same dataset as the original BERT model. The training process involves a loss function that combines the distillation loss $\mathcal{L}_{ce}$, the supervised training loss, which includes the masked language modeling loss $\mathcal{L}_{mlm}$, and a cosine embedding loss $\mathcal{L}_{cos}$ that aligns the hidden state vectors of the student and teacher models.

To compute the similarity between a claim and retrieved paragraphs, they are initially transformed into tokens. Let $C = \{c_1, \ldots, c_n\}$ represent a claim with n tokens, and $P = \{p_1, \ldots, p_m\}$ denote a paragraph with m tokens. The DistilBERT

encoder takes an input item x and generates a real-valued vector as its encoding. Just like BERT, DistilBERT includes special classification tokens [CLS] and [SEP] at the sequence's start and end, respectively. This means that P is encoded as $(\texttt{[CLS]}, p_1, \ldots, p_m, \texttt{[SEP]})$, with the final representation of the [CLS] token, $h_{\texttt{[CLS]}} \in \mathbb{R}^d$, as the paragraph encoding $\mathbf{p}$. Similarly, the claim C is encoded as $(\texttt{[CLS]}, c_1, \ldots, c_n, \texttt{[SEP]})$ and then processed by DistilBERT to obtain the claim encoding $\mathbf{c}$.

The similarity between the claim and paragraph is determined by calculating the inner product of their representations:

$$\text{sim}(\mathbf{c}, \mathbf{p}) = \mathbf{c} \cdot \mathbf{p} = \mathbf{c}^\top \mathbf{p} \tag{3.1}$$

In the study by [52], two fine-tuned BERT models were employed, one for encoding questions and another for encoding associated passages. In contrast, in this study, a pre-trained DistilBERT is utilized in a zero-shot manner to encode both a claim and its corresponding paragraphs without requiring any additional fine-tuning.

3.3.2 Statistical Model

While DistilBERT excels in contextual understanding, we enhance its performance by identifying and retrieving paragraphs that contain the words present in the claim as closely as possible. To achieve this, we incorporate the Okapi BM25 algorithm, a powerful zero-shot model that has demonstrated superior performance over supervised neural models in various applications [68].

The Okapi BM25 algorithm is a widely used information retrieval method that assesses the relevance of a document with respect to a given query. This algorithm takes into account the term frequency and inverse document frequency of the query terms, as well as the length of the document and the average length of documents

in the collection.

For a claim C and a paragraph P, BM25 is calculated as the following similarity score:

$$\text{BM25}(P, C) = \sum_{i=1}^{n} IDF(c_i) \cdot \frac{f(c_i, P) \cdot (k_1 + 1)}{f(c_i, P) + k_1 \cdot (1 - b + b \cdot \frac{|P|}{avgdl})} \tag{3.2}$$

where n is the number of claim words, $f(c_i, P)$ is the frequency of the i^{th} claim term in the paragraph, $|P|$ is the length of the paragraph, $avgdl$ is the average length of paragraphs in the collection, k_1 and b are hyperparameters, and $IDF(c_i)$ is the inverse document frequency of the i^{th} claim term, defined as:

$$IDF(c_i) = log\frac{N}{n(c_i)} \tag{3.3}$$

where N is the total number of paragraphs in the collection and $n(c_i)$ is the number of documents that contain the i^{th} claim term. The higher the Okapi BM25 score of a paragraph, the more relevant it is to the claim.

To arrive at the ultimate score, we combine these two scores. However, it's important to note that the neural network similarity score often has a significantly larger magnitude compared to the statistical score. Thus, a simple summation wouldn't accurately reflect the influence of the statistical score. To address this, both the neural network and statistical scores undergo L2 normalization. The L2 normalization of a vector is computed as follows:

$$\text{norm}(\mathbf{v}) = \frac{\mathbf{v}}{\|\mathbf{v}\|_2} \tag{3.4}$$

where

$$\|\mathbf{v}\|_2 = \sqrt{\sum_{i}^{n} v_i^2} \tag{3.5}$$

After the normalization of the similarity vectors, the final similarity score for a claim and a paragraph is obtained by summing their normalized neural network and

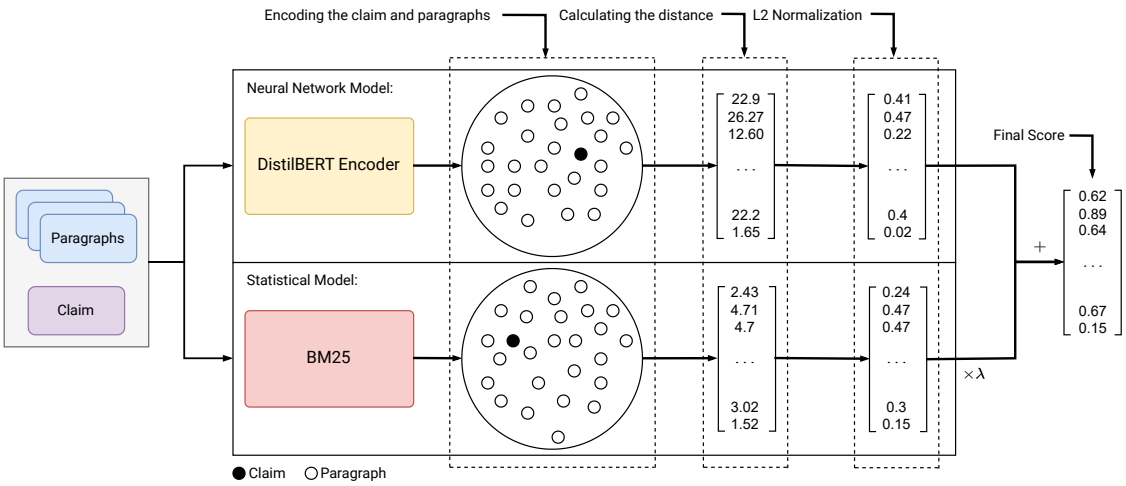


Figure 3.2: Illustration of the ranker procedure. Paragraphs and the claim are encoded into d and $|V|$ dimensions using DistilBERT and BM25 models, respectively. The claim's distance from each paragraph is computed, and both similarity vectors undergo L2 normalization. The final score is a combination of DistilBERT scores and a weighted BM25 score.

statistical similarity scores:

$$\text{sim}(c, p_i) = \lambda < c^{bm25}, p_i^{bm25} > + < c^{nn}, p_i^{nn} > \tag{3.6}$$

Here, λ represents an interpolation hyper-parameter that adjusts the relative weight between the BM25 and neural model contributions, and p_i denotes the i-th paragraph within the collection of retrieved paragraphs with respect to claim c. Once the paragraphs are scored, the top-k paragraphs exhibiting the highest similarity scores are chosen as the supporting evidence.

3.4 Reasoner

When a collection of evidence is available for a claim, humans can naturally determine the claim's accuracy based on that evidence. The innate ability for logical reasoning is a shared trait among humans, irrespective of the depth of their reasoning. This reasoning prowess is a trait that researchers aim to impart to machine

learning models. Recent advancements, like large language models (LLMs), have shown promise in this area, demonstrating emerging capabilities, including logical reasoning. This discussion explores how this ability is used to assess the truthfulness of a claim using evidence sourced from the web.

The last component in the Triple-R system is the reasoner, accomplished by developing a causal language model. As discussed in Appendix A.6.2, causal language models are skilled at generating text. We utilize this capability to create reasons and labels for given claims and evidence pairs. Consider a dataset $D = \{(c_1, y_1), \ldots, (c_N, y_N)\}$ with N samples, where each c_i is a claim, and y_i is its corresponding label. For every claim c_i, the retriever identifies a set of Internet articles denoted as $A_i = \{a_1^i, \ldots, a_m^i\}$, where m represents the number of top articles. The ranker then breaks down these articles into paragraphs and selects the top-k most relevant paragraphs for the claim. This forms the evidence set $e_i = \{e_i^1, \ldots, e_i^k\}$. Ultimately, when presented with a set of evidence and a claim, the role of the reasoner is to generate both the reason and the label.

$$\text{reasoner}(\{e_i^1, \ldots, e_i^k\}, c_i) = (r_i, \hat{y}_i) \tag{3.7}$$

To fine-tune a causal language model for generating reasons and labels based on input evidence-claim pairs, a suitable dataset with annotated reasons is essential for the fine-tuning process. However, many existing datasets lack these annotations, which are vital for establishing the connection between claims and their supporting evidence. Without this linkage, the model's learning process could be hindered. In the following subsection on reason generation, we explore how to create a synthetic dataset where each claim is provided with a reason derived from the evidence. The subsequent part covers the fine-tuning process, detailing how the causal language model is refined using this synthetic dataset to proficiently generate both reasons and labels for the input claims.

3.4.1 Reason Generation

To fine-tune a model for generating reasons from evidence, an annotated dataset is crucial. In this dataset, each claim must be associated with a natural language reason derived from the gathered evidence. This connection is vital because the model needs to understand how to draw logical reasons from the evidence. One way to construct such a dataset is to enlist human annotators to create reasons based on the evidence for each claim. However, this method is expensive, time-consuming, and demands expertise in reason formulation. Hence, an alternative approach is necessary to address this challenge effectively.

Recent years have witnessed remarkable progress in natural language processing, particularly in text generation tasks, driven by the increasing of language models' weight size and training data. These advanced models also exhibit emerging capabilities not observed in their predecessors. One such capability is in-context learning (ICL), introduced by GPT-3 [13]. ICL leverages a well-structured natural language prompt, incorporating task descriptions and a few task examples as demonstrations. This prompt configuration comprises a task description followed by task-specific examples from the dataset, meticulously sequenced using specific templates. The resultant prompt, enriched by adding the test instance to these demonstrations, serves as input to large language models (LLMs) to generate desired outputs. Through a thorough analysis of the task description and demonstrations, LLMs gain insight into the task's essence and the underlying patterns, enabling them to tackle novel tasks without explicit gradient adjustments. Further elaboration on the concept of in-context learning can be found in Appendix A.7.

The ICL method utilizes natural language prompts and requires no changes in the model's architecture, allowing for flexible template design and manipulation. Researchers introduced the Chain-of-Thought (CoT) [90] technique as an improved prompting method, enhancing large language models' (LLMs) performance in intri-

Claim: Trump Management was charged with discriminating against African-Americans and breaking federal law.
Evidence: 1. That year, the federal government filed a complaint against Trump, (...) 2. In October of that year, the federal government filed a complaint (...) 3. Clinton said that Trump "started his career back in 1973 (...) 4. Says Donald Trump "started his career back in 1973 (...)
Reason: Trump Management, which included Donald Trump and his father Fred Trump, was indeed charged with violating the Fair Housing Act and engaging in racial discrimination against African-Americans. The federal government filed a complaint in 1973, alleging that Trump Management refused to rent apartments to African-Americans based on their race, which is a violation of federal law. This charge and subsequent legal action demonstrate that Trump Management was accused of discriminatory practices and breaking federal law.
Label: true

Table 3.1: Table illustrated an example of reason generation. The reason and label are generated based on the pair of the claim and evidence.

cate reasoning tasks like arithmetic, commonsense, and symbolic reasoning. CoT involves a series of succinct sentences simulating human reasoning, enabling LLMs to break down complex problems into manageable intermediate steps. This approach proves beneficial for tasks exceeding standard few-shot prompting capabilities. Combining CoT with large-scale LLMs yields significant performance enhancements across diverse reasoning tasks. The CoT method is detailed in Appendix A.8.

Building upon the concepts of ICL and CoT, we utilized a large language model to construct a synthetic dataset. Let M represent the large language model, and $D = \{(e_1, c_1, y_1), \ldots, (e_N, c_N, y_N)\}$ denote the dataset. Following the CoT approach, we randomly select k examples from this dataset, where k is significantly smaller than N, and manually compose natural language reasons for the claims based on their retrieved evidence. This process yields a natural language prompt $P = \{I, d_1, \ldots, d_k\}$, where d_i is $f(e_i, c_i, r_i, y_i)$, a function mapping the i-th set of evidence, claim, reason, and label to a natural language prompt. Figure 3.3a illustrates the process for generating a reason and label for a sample. Here, r_i is the written reason for claim c_i

based on the retrieved evidence e_i, and I provides an instruction describing the fact verification task to the large language model. Similar to the few-shot prompting in the ICL method, we append each example in D to the prompt P. Given this input prompt, the large language model's output for the pair (e_j, c_j) from the test set can be formulated as follows:

$$M(I, \underbrace{d_1, \ldots, d_k}_{\text{demonstrations}}, f(\underbrace{e_j, c_j}_{\text{test}}, \underbrace{\_\_\_\_\_}_{\text{reason}}, \underbrace{\_\_\_\_\_}_{\text{label}})) \rightarrow (r_j, \hat{y}_j) \tag{3.8}$$

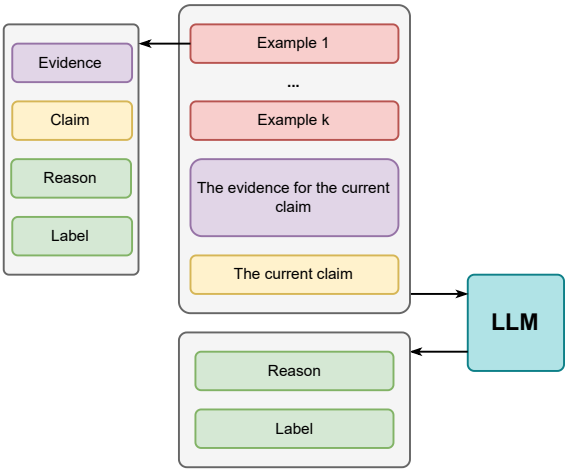
Following the provided instruction, the model comprehends the task and then employs the pattern of reasons outlined in the prompt P to generate both a reason r_j and a label $\hat{y}_j$. The demonstrations guide the model to reason based on the information provided in the evidence section. If the model accurately predicts the label, we naturally assume that it effectively utilizes the evidence to produce a logical reason leading to the accurate answer. Therefore, we focus solely on reasons that correspond to correct answers, $(\hat{y}_j = y_j)$.

The success of ICL greatly hinges on well-designed demonstrations within the prompt. Given the input evidence and claim, the model generates the reason and label, forming two key segments in each demonstration: the input part comprises the evidence and claim, while the output part encompasses the reason and label. This gives rise to two permutations for both input and output arrangements. While it's feasible to position the claim before the evidence in the input, studies have demonstrated that arranging the claim after the evidence leads to greater accuracy [46]. Additionally, it's possible to place the reason after the label. This approach makes the model first predict the label and then generate the reason, isolating whether the model actually depends on the produced reason. However, such a configuration tends to yield poorer performance compared to the alternative, as the produced reason significantly aids the model in making more informed decisions [90], Table 3.1 shows an example of

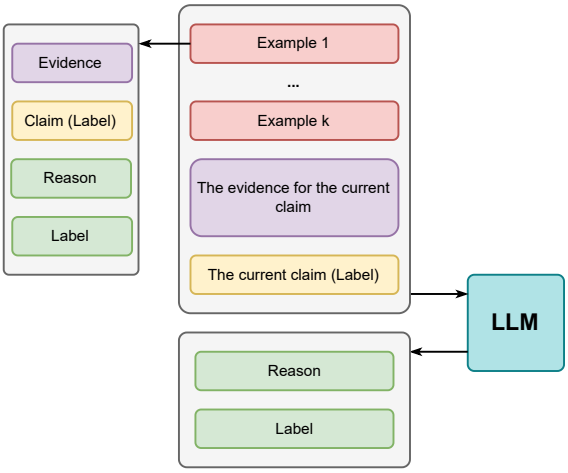
reason generation.

Generating reasons for claims using this approach has a limitation: progress stalls when the model struggles to create a reason for a claim. This is due to the lack of training signals from unsuccessful examples. Taking inspiration from the STaR method [93], we adopt a technique known as rationalization. In this process, we provide the model with the correct label as a hint and ask it to generate a reason in a manner similar to the earlier reason generation step. With the label at hand, the model can work backward, making it easier to formulate reasons that lead to the correct answer. For example, as shown in Figure 3.3b, we embed the hint in the prompt within parentheses to guide the reason generation. Rationalization is applied to claims where the model fails to accurately predict the label. When we include a reason generated through rationalization in the synthetic dataset, we omit the hint from its corresponding prompt, giving the impression that the model devised the reason autonomously. Figure 3.4 provides an overview of the entire process for generating reasons for each claim. Table 3.2 provides an example where the model initially failed to predict the correct label, but achieved success when rationalization was applied.

The STaR [93] method leverages the existing reasoning abilities of an LLM, using iterative bootstrapping to generate rationales. The model generates its own rationales through few-shot prompts, and when it struggles with a particular problem, a rationale is crafted by providing the model with the correct answer as a hint. After creating a rationale for each example, the model's proficiency is refined by fine-tuning on rationales that lead to accurate answers. This iterative process, where enhanced rationale generation refines training data, and improved training data, in turn, boosts rationale generation, is synergistic. This cycle is repeated until the model does not show any improvement. Notably, the original pre-trained model is trained on the newly collected dataset to prevent overfitting, avoiding continuous



(a) Without hint



(b) With hint

Figure 3.3: (a) The structure of the prompt which forces the model to generate a reason and a label according to the examples in the prompt. Each example contains evidence, claim, reason, and label. (b) For the prompt in the rationalization method, we add the actual label as a hint at the end of claims for current input and examples.

Without hint	**Claim:** President Obama's refusal to grant permits for offshore drilling is one reason gasoline prices are soaring.
	Reason: The claim that President Obama's refusal to grant permits for offshore drilling is one reason for soaring gasoline prices is not entirely accurate. While it is true that President Obama had implemented regulations and restrictions on offshore drilling, it is an oversimplification to attribute rising gasoline prices solely to this factor. Gasoline prices are influenced by a multitude of factors, including global oil prices, supply and demand dynamics, geopolitical events, and market speculation. It is important to consider the complex nature of the energy market and the various factors that contribute to fluctuations in gasoline prices.
	Predicted label: half-true
With hint	**Claim:** President Obama's refusal to grant permits for offshore drilling is one reason gasoline prices are soaring. (hint the label is pants-fire)
	Reason: The statement that President Obama refused to grant permits for offshore drilling as a reason for soaring gasoline prices is false. In reality, President Obama did grant permits for offshore drilling during his presidency. In fact, his administration expanded offshore drilling in certain areas, although it also implemented safety regulations following the Deepwater Horizon oil spill. Gasoline prices are influenced by a variety of factors, including global oil prices, supply and demand dynamics, geopolitical events, and market speculation. It is misleading to attribute rising gasoline prices solely to President Obama's actions regarding offshore drilling permits.
	Predicted label: pants-fire

Table 3.2: This table illustrates the impact of the rationalization method. Initially, the model's label prediction was incorrect; however, upon applying rationalization, it generated both the correct reason and label.

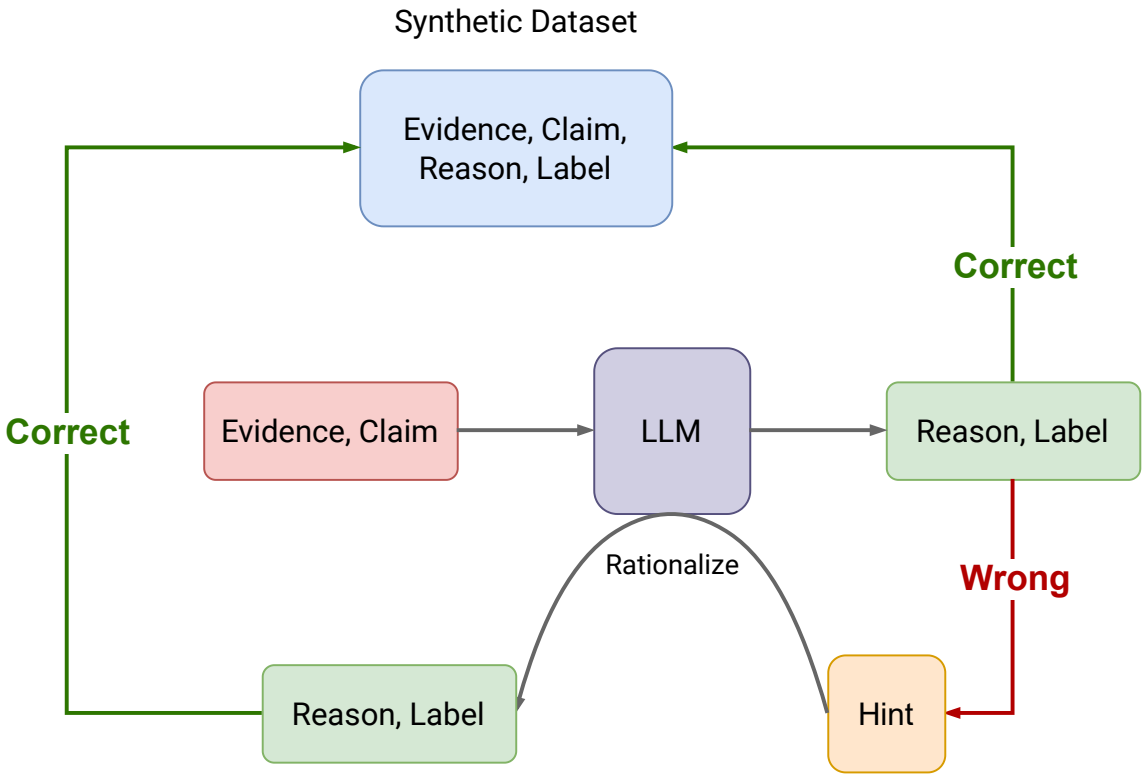


Figure 3.4: Rationalization process overview: The model predicts a reason and label based on the input prompt. If the label prediction is accurate, the resulting reason, alongside the evidence and claim, is included in the synthetic dataset. In case of an incorrect prediction, the model undergoes rationalization, involving the addition of a hint to the input prompt. If the model can then accurately predict the label with the hint, the reason, along with the evidence and claim (without the hint), is added to the dataset.

training of a single model. However, in this study, we deviate from the fine-tuning step. After generating the initial reasons, we compile them into a synthetic dataset.

3.4.2 Fine-tuning

In recent years, pre-trained large language models have proven to be effective in solving various natural language processing (NLP) tasks in zero-shot or few-shot settings. Although in zero-shot or few-shot settings, the need for the task-specific data is solved, it has been observed that fine-tuned models still outperform pre-trained models in many cases [13]. Moreover, the cost of maintaining large language models can be prohibitively high due to their massive number of parameters. Therefore, the development and utilization of smaller models are critical for practical applications of NLP.

Pre-trained language models amass substantial knowledge during pre-training through vast datasets. To utilize this knowledge for the fact verification task, a method called **fine-tuning** is employed. Fine-tuning entails constructing task-specific applications atop pre-trained models and adjusting either the entire set or a small subset of application-specific parameters. This procedure demands task-specific label data, which is obtained during the reason generation phase.

In this study, we perform fine-tuning on a pre-trained transformer decoder, utilizing the transformer architecture [84]. This decoder is designed to accept text input and produce text output, making it well-suited for tasks involving generation, such as contextual generation, summarization, and machine translation. The transformer decoder employs a self-attention mechanism to map input tokens $(x_1, \ldots, x_n)$ to corresponding output embeddings $(y_1, \ldots, y_n)$ of equal length. For a comprehensive understanding of the self-attention mechanism, refer to Appendix A.6.1.1.

Transformer decoders function as causal language models, generating words step by step based on previous choices. This involves assigning conditional probabilities to

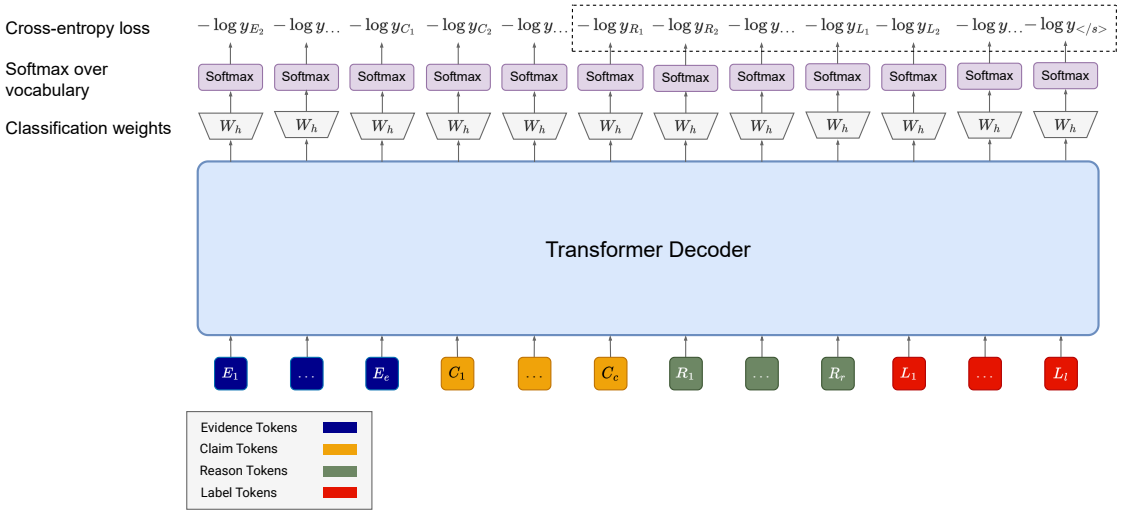


Figure 3.5: Illustration of the fine-tuning process using a transformer decoder as the causal language model. The model takes in the evidence, claim, reason, and label as inputs. For a given input token sequence $(x_1, \ldots, x_i)$, the model predicts the probability of the next token x_{i+1} using the softmax function. This probability is utilized to calculate the error through the cross-entropy loss function. The total loss for the input sequence is obtained by summing up the losses for the reason and label tokens.

potential next words, forming a probability distribution across the vocabulary. By using the chain rule, these models estimate the sequence probabilities by combining conditional probabilities for each word:

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i | w_0, \ldots, w_{i-1}) \tag{3.9}$$

After creating the new dataset containing the generated reason, this curated dataset is used to fine-tune a transformer decoder. In this dataset, every sample is the concatenation of evidence, a claim, a generated reason, and, in the end, a label, $X_i = f(e_i, c_i, r_i, y_i)$, where f is a function that maps its input into a natural language prompt. In contrast to the pre-training step, in which the loss is calculated for every token, in this work, the loss is only calculated for the target tokens. In other words, given the evidence, claim, reason, and label, the loss is only calculated based on the reason and label tokens. Therefore, the model calculates the probability of the

reason and label given the evidence and claim. The loss for a sample is calculated using the cross-entropy loss function, which is the negative log probability the model assigns to the next token in the training sequence:

$$\mathcal{L}_{CE}(\hat{x}, x) = -\sum_{i=r}^{N} \log(\hat{x}_i)$$

where x is the input sequence, and $\hat{x}$ is the predicted tokens. Also, r is the index where the reason tokens begin and $1 \leq r \leq N$, and N is the number of tokens in the input prompt. The model estimates this probability by using a softmax function over the set of possible outputs:

$$p(x_t|x_{t-1}, \ldots, x_1) = \text{softmax}(Wh_t + b) \tag{3.10}$$

where $h_t \in \mathbb{R}^d$ is the output vector of the neural network at time t, $W \in \mathbb{R}^{|V| \times d}$ is a parameter matrix that is learned during training and maps the output vector into a set of scores for tokens. Figure 3.5 illustrates the input and output of the reasoner model.

After training, our fine-tuned model is employed to generate reasons and labels for test samples. This process involves executing the retriever and ranker steps on the test samples to gather supporting evidence. By combining the retrieved evidence with the claim, we create a natural language input prompt $X_j = f(e_j, c_j)$, where the function f maps them into the appropriate format. Using this prompt, our fine-tuned model generates a reason and corresponding label:

$$(r_j, \hat{y}_j) = \text{reasoner}(e_j, c_j) \tag{3.11}$$

To extract the label, we incorporate a specific sentence in the reason generation step that encloses the label. After the model's output is generated, we search for this

sentence to extract the label, which is then used for evaluation purposes.

3.5 Concluding Remarks

In this chapter, we have introduced the Triple-R framework, which is designed to perform logical reasoning like humans and verify claims. The framework consists of three main stages, replicating the step-by-step process that humans usually follow when evaluating the accuracy of claims. This comprehensive overview provides a clear understanding of the proposed framework.

The first stage, the Retriever, focuses on gathering relevant contextual information from the web to provide knowledge regarding the claim. This is accomplished by searching the claim verbatim using a search engine API and retrieving the top articles returned. The inclusion of this external knowledge source is a vital aspect that provides the model with up-to-date information to better handle the dynamic nature of fact-checking.

The second stage, the Ranker, serves to filter the retrieved information, selecting only the most useful evidence. This is achieved through a hybrid ranking model that scores paragraphs based on their relevance to the claim using both a neural network and a statistical model. The neural network leverages contextualized embeddings from a transformer encoder to capture semantic similarity. Meanwhile, the statistical model identifies key overlapping terms using BM25. By combining these complementary approaches, the model can accurately retrieve the most pertinent passages.

The final stage, the Reasoner, is the core of the framework, performing the logical reasoning to assess the claim's veracity. This reasoning process is made possible through recent advances in generative causal language models. Specifically, first, a large language model is leveraged to generate natural language reasons for samples in the training dataset given a specific input prompt. Then, this curated dataset is

used to fine-tune a smaller language model so it can take new claim-evidence inputs to simultaneously produce a veracity label and reason.

A key benefit of Triple-R is its incorporation of external knowledge and replication of human logical reasoning, providing an advantage in tackling the dynamic nature of fact-checking. Moreover, the framework requires no human-written reason for training data, instead relying on a data generation process. Overall, Triple-R offers a novel approach to claim verification that closely emulates the human process.

Moving forward, the next chapter will discuss the implementation specifics and experimental setups used to evaluate Triple-R's performance on benchmark fact-checking datasets. The empirical results will demonstrate how the proposed framework compares against existing baselines and benchmarks on this complex reasoning task. The analyses will provide insights into the capabilities and limitations of the framework.

Chapter 4

Implementation and Results

4.1 Overview

In this chapter, we focus on implementing the proposed methods and evaluating their effectiveness. We begin by explaining the techniques used for implementation. Then, we discuss the obtained outcomes and evaluation methodologies. This chapter presents the practical realization and empirical assessment of our proposed methods.

4.2 Implementation

In this section, we detail the techniques and tools employed to implement each component of the framework, covering aspects such as models, data, software libraries, and computing infrastructure. We also address important implementation challenges and their corresponding solutions. As outlined in the proposed method chapter, the framework comprises three main components: retriever, ranker, and reasoner. Consequently, this section is divided into three sub-sections, each dedicated to explaining one of these components.

The framework is built using Python, a high-level programming language recognized for its simplicity, readability, and rich standard library. With its straightforward syn-

```python
def web_crawler(claim, top_urls=3):
  # Use Bing API to get top search result URLs for claim
  urls = get_urls(claim)

  # Download HTML content in parallel for top URLs
  articles = get_urls_content(urls[:top_urls])

  # Extract text from HTML tags
  # Also remove unwanted elements like URLs
  preprocessed_articles = preprocess_articles(articles)

  # Segment articles into paragraphs
  paragraphs = []
  length = 20
  for article in preprocessed_articles:

    # Split article into paragraphs by newline
    article_paragraphs = article.split('\n')

    # Filter paragraphs based on min length
    filtered_paragraphs = list(filter(lambda p: len(p.split()) > length and p != '', article_paragraphs))

    # Append valid paragraphs
    paragraphs.extend(filtered_paragraphs)

  return paragraphs
```

Figure 4.1: Python code implementing the Retriever component to extract relevant paragraphs from the web. The key steps include using the Bing API to obtain search result URLs, downloading and extracting text from the HTML content, splitting into paragraphs, and filtering based on length. Comments clarify how the code reflects the process of leveraging search engines and web scraping to gather textual evidence paragraphs given an input claim.

tax and wide array of modules, Python is well-suited for tasks spanning web development, data analysis, scientific computing, and artificial intelligence. Its adaptability and ability to work across platforms have led to its widespread adoption across a spectrum of applications, ranging from small scripts to intricate software systems.

4.2.1 Retriever

To retrieve relevant articles, we make use of the Bing Web Search API[1], performing a direct search with the claim text. We utilize the Request library to interact with

[1]https://www.microsoft.com/en-us/bing/apis/bing-web-search-api

this API, which offers a simple and Pythonic way of handling HTTP requests and web APIs. This library's design abstracts HTTP complexities, simplifying tasks like GET, POST, and PUT operations. By using the GET method, we send the claim as a parameter within the "params" variable for URL parameters. Additionally, we include the subscription key in the headers to accompany the API request. Once the API result is received, it's converted into JSON format for further processing.

Upon obtaining the search results, we select the top-3 URLs for further analysis and content retrieval. Empirically, we have observed that analyzing the top-3 URLs provides sufficient context to either support or refute the claim, with additional URLs adding redundant information. This involves crafting a function designed to fetch a URL's content using the Request library. The function incorporates specific constraints. A 10-second timeout is implemented to avoid links with sizable files. Additionally, the `Content-Type` value from the HTTP header is checked; if not `text/html`, the function returns an empty string. This step ensures the content retrieved is in HTML format rather than other file types like PDFs or PowerPoint presentations. Efficiency is enhanced by the parallel execution of this function. Since it's primarily an IO-bound process, we opt for multithreading over multiprocessing. The downloaded content from URLs arrives in HTML format. Initially, we extract text enclosed within HTML tags and further refine it for clarity. Specifically, we target the text within the `body` HTML tag. To do this, we utilize the Beautiful Soup library, a Python tool for parsing structured data. This library enables HTML interaction akin to developer tools on a web page. Once the content becomes a Beautiful Soup object, selection of the `body` tag is possible. Text extraction from the HTML tags inside `body` is executed using the html2text package[2], a Python script streamlining HTML-to-ASCII conversion. Despite html2text's proficiency in text extraction, it retains some undesired content like URLs, internal links, and

[2]https://pypi.org/project/html2text/

certain punctuation. To tackle this, we design custom Regex patterns to purify the text by eliminating these superfluous elements.

Once the textual content of URLs is obtained, it is segmented into paragraphs. This segmentation process involves identifying instances of \n, which denotes a new line in the content. After this initial splitting, paragraphs with empty content or containing fewer than 20 words are systematically filtered out. This two-step process ensures that the extracted paragraphs are meaningful and substantial in length.

4.2.2 Ranker

The ranker takes as input a list of paragraphs from the retriever. As outlined in the proposed method chapter, it aims to identify the paragraphs most relevant to the claim using a hybrid neural network and statistical model. The neural network computes semantic similarity between the claim and each paragraph. Meanwhile, the statistical model identifies key overlapping words. By combining these complementary techniques, the most pertinent paragraphs can be accurately extracted for use as evidence. The following sections detail the technical implementation of both models within the ranker.

The neural network model leverages DistilBERT [69], a condensed and optimized version of the BERT model [21]. DistilBERT retains 97% of BERT's language understanding skills while being 60% faster and 40% smaller in size, making it more efficient. This is achieved by reducing the number of encoder layers from 12 in BERT-base to just 6, without compromising too much accuracy. Furthermore, DistilBERT preserves the 512 tokens input length. The combination of strong performance, smaller footprint, and speed enhancements makes DistilBERT well-suited as an encoder for the ranker compared to the larger BERT model.

To implement DistilBERT, we use the SentenceTransformers package[3], a versatile

[3]https://www.sbert.net/

Python framework known for its ability to generate embeddings for text, sentences, and images. This framework supports over 100 languages and allows users to compute embeddings that can be compared using techniques like cosine similarity to find similar sentences. It provides a variety of pre-trained models fine-tuned for different tasks, and for this work, we opt for the `distilbert-base-cased` model. After initializing the model, it takes a list of string values as input and returns the corresponding encodings. To achieve this, the package tokenizes each item in the input list using the WordPiece [91] tokenizer, which automatically adds special tokens like `[CLS]` and `[SEP]` to the input. Utilizing pre-trained language models reduces the need for extensive preprocessing, eliminating the need for particular pre-processing steps. Once tokenized, the input is passed to the model, and the vector for the `[CLS]` token is used as the input embedding. Finally, all these embeddings for the list of texts are aggregated into a NumPy matrix[4].

To compute the similarity between the claim and the related paragraphs, we individually feed them into the model for encoding. This results in two matrices, $P \in \mathbb{R}^{N \times 768}$ and $C \in \mathbb{R}^{1 \times 768}$, where N represents the number of paragraphs, and 768 signifies the dimension of the DistilBERT's last hidden layer. The similarity score is determined by taking the inner dot product between these matrices, which we efficiently calculate using NumPy's built-in functions. Subsequently, we normalize the scores using the scikit-learn package, a Python-based, open-source machine learning library. Figure 4.2 illustrates the code for the process.

The retriever's other component is the statistical model, employing the BM25 algorithm [68]. Unlike the DistilBERT model, which is capable of handling raw text effectively, the BM25 requires multiple preprocessing steps to enhance its performance. These preprocessing steps are facilitated through the spaCy package[5], an open-source library renowned for advanced natural language processing. Specifically,

[4]https://numpy.org/
[5]https://spacy.io/

```python
def dense_ranker(paragraphs, claim):
  # Encode paragraphs and claim using DistilBERT
  embeddings = DistilBERT.encode(paragraphs)
  query_embedding = DistilBERT.encode([claim])

  # Calculate similarity dot product
  # embeddings shape: N x 768
  # query_embedding shape: 1 x 768
  # d_scores shape: N x 1
  d_scores = numpy.dot(embeddings, query_embedding.T)

  # L2 normalize scores
  d_norm = sklearn.preprocessing.normalize(d_scores, axis=0).ravel()

  # Return final normalized scores
  return d_norm
```

Figure 4.2: Python code snippet demonstrating the implementation of the neural ranker using DistilBERT embeddings. The key steps shown include encoding the claim and paragraphs into vector representations, computing similarity via dot product, normalizing the scores, and returning the final rankings.

we employ the `en_core_web_lg` pipeline, which integrates various text processing components of spaCy, including tagger, lemmatizer, parser, and entity recognizer. The preprocessing steps consist of the following:

- **Lowercasing**: To ensure case insensitivity, all words are converted to lowercase. This ensures that words like "Book" are represented consistently, regardless of case.

- **Special Character Removal**: The text is cleansed of special characters, including double quotes, punctuation, and possessive pronouns.

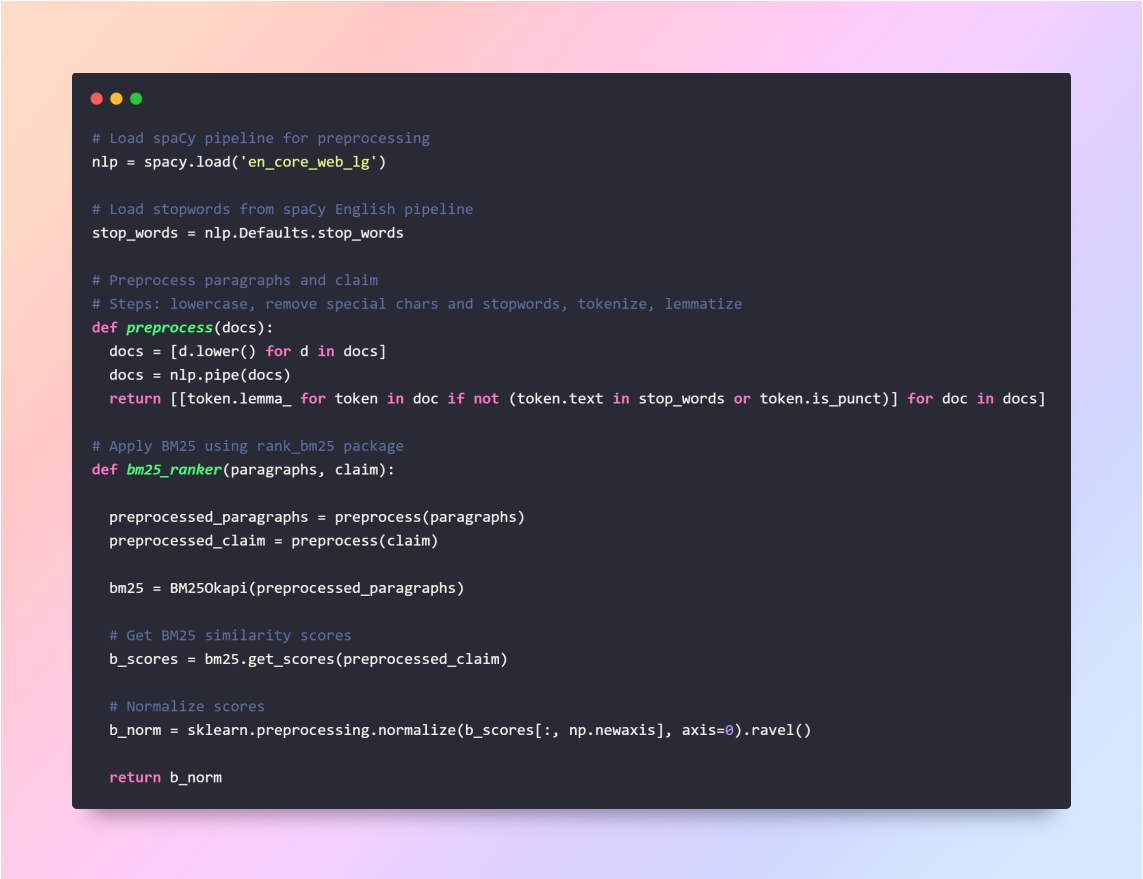
- **Stopword Removal**: Common and non-meaningful words, or "stopwords," such as "I" and "the," are removed. This step is vital as it allows the model to focus on informative words that convey the most significant information.

- **Tokenization**: Text tokenization breaks the text into smaller, meaningful units, facilitating detailed analysis in the context of other elements.

- **Lemmatization**: Lemmatization, a fundamental natural language processing technique, reduces words to their base or root form, known as a "lemma." Its primary purpose is to normalize words, ensuring different forms of the same word are treated as one. For instance, in English, words like "running," "ran," and "runs" are all lemmatized to the common lemma, "run." Even words like "better" and "best" are reduced to their shared lemma, "good." Importantly, lemmatization considers a word's part of speech, generating accurate lemmas depending on whether a word functions as a noun, verb, adjective, or another grammatical role.

After preprocessing both the paragraphs and the claim, we apply the BM25 algorithm to them. To execute this algorithm, we harness the rank_bm25 package[6], which

[6]https://github.com/dorianbrown/rank_bm25



```python
# Load spaCy pipeline for preprocessing
nlp = spacy.load('en_core_web_lg')

# Load stopwords from spaCy English pipeline
stop_words = nlp.Defaults.stop_words

# Preprocess paragraphs and claim
# Steps: lowercase, remove special chars and stopwords, tokenize, lemmatize
def preprocess(docs):
  docs = [d.lower() for d in docs]
  docs = nlp.pipe(docs)
  return [[token.lemma_ for token in doc if not (token.text in stop_words or token.is_punct)] for doc in docs]

# Apply BM25 using rank_bm25 package
def bm25_ranker(paragraphs, claim):

  preprocessed_paragraphs = preprocess(paragraphs)
  preprocessed_claim = preprocess(claim)

  bm25 = BM25Okapi(preprocessed_paragraphs)

  # Get BM25 similarity scores
  b_scores = bm25.get_scores(preprocessed_claim)

  # Normalize scores
  b_norm = sklearn.preprocessing.normalize(b_scores[:, np.newaxis], axis=0).ravel()

  return b_norm
```

Figure 4.3: Python code for the statistical ranker using BM25. The key steps include text preprocessing with spaCy, applying the BM25 algorithm from rank_bm25 package, calculating relevance scores for the claim, normalizing scores, and returning the final rankings.

offers various versions of BM25 algorithms. Specifically, we employ the BM25Okapi algorithm. Initially, we provide the preprocessed paragraphs to the algorithm to compute the associated weight for each word. This step is crucial for determining the importance of words within the paragraphs. Subsequently, the preprocessed claim is passed through the algorithm to ascertain its relevance score concerning the paragraphs. This score reflects how closely the claim aligns with the content of the paragraphs. After obtaining all the scores, we perform normalization using the scikit-learn normalization function.

To calculate the final score, we introduce a weighting parameter, denoted as λ. This is due to the recognition that the scores obtained from the neural network ranker carry more critical information. Consequently, we merge these two sets of scores. In our study, we have determined that setting λ to 0.9 achieves an optimal balance between the BM25 ranker and the neural network ranker. Following the ranking process of the paragraphs, we opt to select the top four paragraphs as the evidence. Our findings suggest that these four paragraphs typically encompass sufficient information to ascertain the veracity of a given claim.

4.2.3 Reasoner

4.2.3.1 Reason Generation

The reasoner is a language model tasked with generating both a reason and a label for a given set of evidence and a claim. However, a common challenge in existing datasets is the scarcity of annotated reasons. To tackle this issue and produce reasons for each claim based on retrieved evidence, we harness the reasoning prowess of a Large Language Model (LLM). Research has demonstrated that larger models exhibit enhanced reasoning abilities [90]. In our case, we utilize GPT-3 [13], an advanced language model built on the transformer decoder architecture pioneered by OpenAI. GPT-3 stands out due to its sheer scale, boasting an impressive 175

billion parameters distributed across 175 layers in its neural network. This extensive architecture empowers GPT-3 to excel in a wide array of natural language processing tasks, spanning text generation, translation, and summarization, among others.

We leverage the GPT-3.5-Turbo model, which is accessible via the OpenAI API, implemented within the OpenAI Python package[7]. A standout feature of the GPT-3.5-Turbo model is its multi-turn capability, a significant upgrade from the GPT-3 model, which handles only single-turn text prompts. This advancement empowers users to provide a series of messages as input, enabling preset scenarios and previous responses to serve as contextual information for generating more contextually relevant responses.

To utilize this model, we make use of the chat completions API. The GPT-3.5-Turbo model processes a list of messages as input and furnishes a model-generated message as output. While the chat format is primarily designed for facilitating multi-turn conversations, it remains equally effective for single-turn tasks that don't involve any prior conversation.

The primary input is the "message" parameter, which should be an array of message objects. Each message object includes a "role" (either "system," "user," or "assistant") and "content." Conversations can vary in length, from a single message to an extended exchange. Typically, a conversation is structured with an initial system message, followed by alternating user and assistant messages. The system message serves to define the model's behavior. In our case, we set the system message to describe the task's purpose: generating a reason based on the evidence and assigning a label to the claim.

User messages are used to make requests or provide comments for the assistant to respond to, while assistant messages can serve as a repository of previous assistant responses or provide examples of desired behavior. In our approach, we convert these

[7]https://pypi.org/project/openai/

```
# Create empty message list
messages = []

# Add system message
messages.append({"role": "system", "content": system_message})

# Add demonstration examples as conversation
for user_i, assistant_i in demonstrations:
  messages.append({"role": "user", "content": user_i})
  messages.append({"role": "assistant", "content": assistant_i})

# Create empty responses list
responses = []

# Iterate through training dataset
for evidence_i, statement_i in training_dataset:

  # Format evidence and statement as user message
  prompt = "Evidence:"
  prompt += evidence_i
  prompt += "Statement:"
  prompt += statement_i

  # Add formatted message to history
  messages.append({"role": "user", "content": prompt})

  # Call API with conversation history
  response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=messages,
    temperature=0.3,
    max_tokens=300,
  )

  # Add response to list
  responses.append(response)
```

Figure 4.4: This figure depicts the step-by-step procedure of structuring messages and utilizing the OpenAI API to generate responses. The process begins with the creation of an empty message list, followed by the addition of a system message that sets the model's behavior. Demonstrations are then included as part of the conversation. Next, a user message is constructed from the evidence and statement in the training dataset. This user message is added to the message history. Subsequently, the OpenAI API is called with the conversation history to obtain model responses. These responses are collected and utilized in the fine-tuning process.

demonstrations into a structured conversation history. To do this, we randomly select one example from each label present in the dataset as a demonstration. Then, we manually craft a reason for the claim based on the available evidence. In each turn of the conversation history, we designate the evidence and statement in the user role using `Evidence:` and `Statement:` tags, while the reason and label are placed in the assistant role, marked by `Reason:`and `Label:` tags. This structured format guides the assistant in understanding the context and desired responses.

For each pair of evidence and statement in the training dataset, we format them using the "Evidence:" and "Statement:" tags and include them as user messages in our conversation. This message history is then passed to the model through the API. The model, guided by the provided demonstrations and the defined objectives in the system message, generates a reason and label for the input.

To extract the label, we employ regex to locate the `Label:` tag and compare it with the actual label. In cases where the model fails to correctly predict the label, we employ a rationalization method. Here, we structure the messages in the same manner as before but include the actual label as a hint at the end of each statement in both the demonstrations and the current statement. This provides the model with additional guidance regarding the veracity of the statement. Figure 4.4 illustrates the process of creating reasons for the training dataset without the rationalization method.

It's worth noting that in addition to the message parameter, we utilize specific settings by sending a temperature of 0.3 and specifying a maximum token length of 300 to the API. The temperature parameter influences the randomness of the generated text, with lower values making the output more deterministic and higher values introducing more variability. Also, the maximum token length parameter serves as a constraint, limiting the length of the generated text to a predefined character or token limit, ensuring that the output remains within desired boundaries.

```
<s>[INST] <<SYS>>
{{ system_prompt }}
<</SYS>>

{{ user_msg_1 }} [/INST] {{ model_answer_1 }} </s><s>[INST] {{ user_msg_2 }} [/INST]
```

Figure 4.5: The template for the chat version of Llama 2 models.

4.2.3.2 Fine-Tuning

Once the dataset is prepared, we proceed with fine-tuning a smaller language model. This model strikes a balance between computational efficiency and performance, surpassing the resource-intensive GPT-3.5-Turbo model. Specifically, we've chosen the Llama-2-7b-chat model from the Llama 2 family, as detailed in [83]. The Llama 2 family offers three variants with varying parameter counts: 7 billion, 13 billion, and 70 billion, all trained on a substantial corpus of 2 trillion publicly available tokens. The Llama 2-Chat, a specialized version of Llama 2, is fine-tuned for dialogue applications. This fine-tuning process involves iterative refinements through Reinforcement Learning with Human Feedback (RLHF), as explained in [59]. RLHF employs techniques such as rejection sampling and Proximal Policy Optimization (PPO), as outlined in [72], to bring the model's behavior in closer alignment with human preferences and instructions.

To manage dialogues effectively, the creators of Llama 2 developed a prompt template tailored for chat variations. Much like the OpenAI chat API, this template delineates three distinct roles: system, user, and assistant. Illustrated in Figure 4.5, the guidance nestled between the special «SYS» tokens furnishes contextual cues to the model, clarifying the anticipated response. This methodology proves effective due to the consistent format applied during training, encompassing an array of system prompts designed for diverse tasks.

As the conversation unfolds, every exchange between the user and the model is added to the existing prompt, enclosed within [INST] markers. For this study, we employ

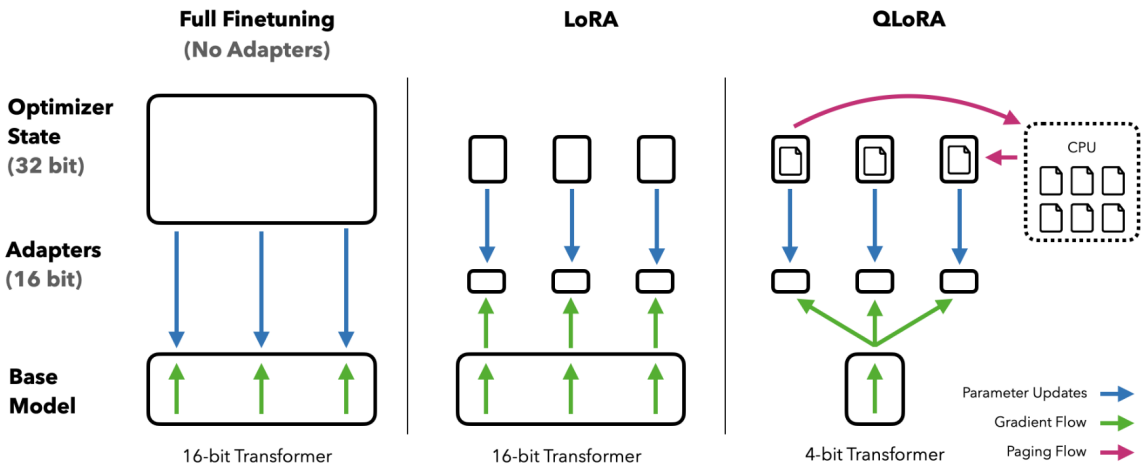


Figure 4.6: Comparison of fine-tuning methods and their memory demands. QLoRA enhances LoRA by introducing 4-bit precision quantization for the transformer model and implementing paged optimizers to effectively manage memory fluctuations. [20]

the same system message for the reason generation section, as the task remains consistent across both sections. Here, we define the evidence and claim as the user message, while the model's response encompasses the reason and label. To make the data ready for the model, we use `DataCollatorForCompletionOnlyLM` data collator from the trl[8] package. This data collator makes sure that the loss is only calculated for the model answer, not the whole input.

While LLMs exhibit remarkable capabilities, their fine-tuning process can be costly. Specifically, fine-tuning involves allocating 2 bytes for weight, 2 bytes for weight gradient, and 8 bytes for optimizer state per parameter, totaling 12 bytes per parameter. For instance, considering the Llama-2 (7b) model, this amounts to 96 gigabytes of VRAM (7b $\times$ 12 bytes). However, our computational resources are limited to a V100 GPU with 16GB of VRAM, which barely accommodates the model's extensive weight parameters. To overcome this limitation, we've implemented a parameter-efficient fine-tuning technique known as QLoRA, as discussed in [20].

QLoRA represents an extension of LoRA [33] and introduces a range of innovative strategies to optimize memory usage while maintaining top-notch performance.

[8]https://huggingface.co/docs/trl/index

These innovations include:

- **NF4 Quantization:** NF4 quantization capitalizes on the natural weight distribution of pre-trained neural networks, typically characterized by a zero-centered normal distribution with specific standard deviations. By standardizing all weights to a fixed distribution within the NF4 range ($[-1, 1]$), NF4 quantization efficiently quantifies weights without needing resource-intensive quantile estimation algorithms.

- **Double Quantization:** To mitigate the memory overhead associated with quantization constants, Double Quantization proves instrumental. This approach substantially reduces the memory footprint without compromising performance by quantizing the quantization constants themselves. It involves utilizing 8-bit Floats with a block size 256 for the second quantization step, resulting in an economical storage requirement of approximately 0.37 bits per parameter.

- **Paged Optimizers:** Paged Optimizers enable the evocation of optimizer states to the CPU RAM when GPU memory capacity is exhausted, subsequently paging them back into GPU VRAM when required for gradient backward passes to update weights. This technique serves as an optimization strategy, particularly beneficial for longer sequences where GPU VRAM proves insufficient to retain all optimizer states.

These concepts come together to refine the LoRA approach. Low-Rank Adaptation (LoRA) fine-tuning, described in [33], is a technique that minimizes memory requirements by using a smaller set of adaptable parameters known as adapters while keeping the model parameters frozen. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, its update is restricted through a low-rank decomposition expressed as $W = W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$, with $r << min(d, k)$. During training, W_0

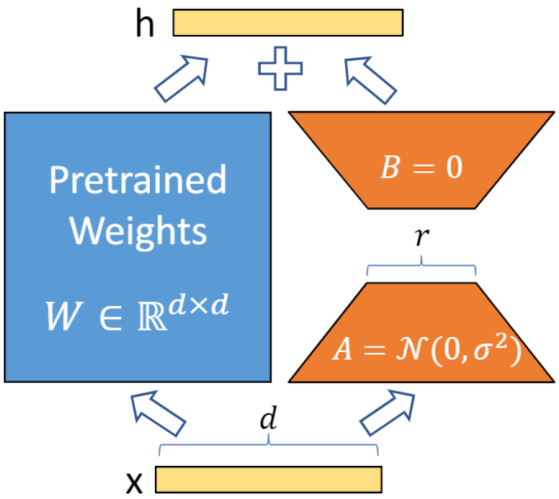


Figure 4.7: Visual representation of the LoRA process. This method involves the freezing of pre-trained weights, with a focus on training the low-rank matrices. [33]

remains static, not receiving any gradient updates, while A and B hold the trainable parameters. It's important to note that both W_0 and $\Delta W = BA$ are applied to the same input, x, and their resulting output vectors are summed coordinate-wise. In the context of $h = Wx$, the adjusted forward pass can be expressed as:

$$h = W_0 x + \Delta W = W_0 x + BAx$$

Figure 4.7 illustrates this process.

We perform fine-tuning on the Llama-2 (7b) model using a V100 GPU equipped with 16GB of VRAM. However, the GPU's VRAM is barely sufficient to accommodate the model's extensive weights (7b $\times$ 2 bytes = 14 GB in FP16). Factoring in additional requirements for optimizer states, gradients, and forward activations, we face limitations. To address this, we implement a parameter-efficient fine-tuning technique known as QLoRA [20]. QLoRA utilizes 4-bit quantization to backpropagate gradients through a frozen pre-trained language model into Low-Rank Adapters, significantly reducing VRAM usage. We set the QLoRA rank to 8 and the scaling parameter to 16. The fine-tuning process involves 5 epochs, utilizing the AdamW optimizer [51] with a learning rate of $1e-4$. A warm-up linear schedule [28] with

Dataset Statistics	
Training set size	10,269
Validation set size	1,284
Testing set size	1,283
Avg. statement length (tokens)	17.9
Top-3 Speaker Affiliations	
Democrats	4,150
Republicans	5,687
None (e.g., FB posts)	2,185

Table 4.1: The LIAR dataset statistics

100 warm-up steps is incorporated into the learning rate schedule, with a batch size of 4 for optimization.

During the fine-tuning process, a tag denoted as "Label: " is added to the end of each sample, representing the label for the claim. After the models are fine-tuned, a regular expression (regex) search is conducted to identify this sentence at the end of the generated text. If the correct format is found, the extracted label is considered the predicted label for the claim.

The main package used for loading and training the model in this work is transformers from Hugging Face[9]. Transformers package provides APIs and tools to easily download and train state-of-the-art pretrained models. To load the Llama-2-7b-chat model in 4-bit, we use the bitsandbytes package[10]. The bitsandbytes is a lightweight wrapper around CUDA custom functions, in particular 8-bit optimizers, matrix multiplication (LLM.int8()), and quantization functions. Finally, we use the PEFT package[11] to train the LoRA adaptors and combine them with the model weights after training.

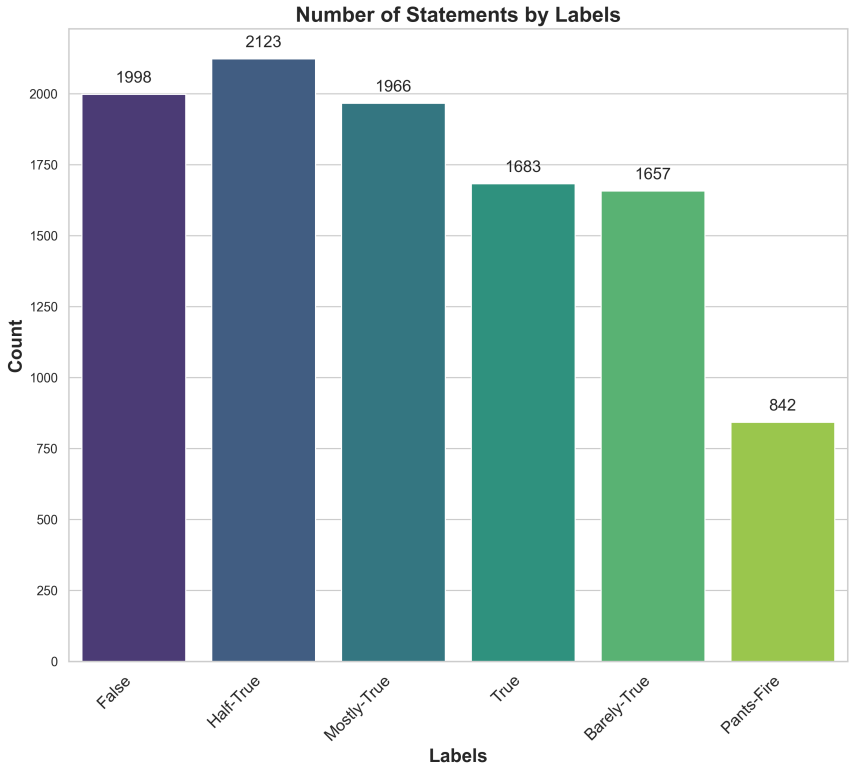


Figure 4.8: Distribution of statements in the LIAR training set.

4.3 Dataset

In this study, we employ the LIAR dataset [86] to assess our proposed Triple-R framework. The LIAR dataset consists of 12,836 brief statements manually evaluated for truthfulness by PolitiFact[12]. Table 4.1 presents the statistical overview of the LIAR dataset. Notably, PolitiFact covers a diverse array of political subjects, offering meticulous judgments with nuanced labels. These labels gauge the truthfulness level of each statement, spanning from "true" to the most extreme "pants-on-fire" falsehood, as illustrated in Figure 4.8.

As the claims in the dataset are sourced from the PolitiFact[13] website, we restrict the search engine to retrieve documents only from the PolitiFact domain. We then

[9]https://huggingface.co/docs/transformers/index
[10]https://github.com/TimDettmers/bitsandbytes
[11]https://huggingface.co/docs/peft/index
[12]https://www.politifact.com/
[13]https://www.politifact.com/

select the top three results and score the paragraphs using the ranker component. To balance the neural network and statistical score, we set the $\lambda = 0.9$, and then select the top four paragraphs.

4.4 Results and Discussion

In our experiment, our framework performs two tasks: identifying relevant evidence for each claim and generating a reason for the claim based on the identified evidence. We have compared the performance of our model with other models that classify claims with and without evidence.

In our ablation study, we contrast the Triple-R framework with two alternative configurations. Given that our proposed framework involves both reason generation and contextual evidence utilization, one configuration involves a model solely classifying the statements, while the other entails a model classifying pairs of statements and evidence. As detailed in A.6.1.9, transformer encoders, particularly the BERT model, prove apt for classification tasks. Hence, we implement the bert-base-uncased model, encompassing 110 million parameters.

For each input, a designated token `[CLS]` is inserted at the sequence's start. The output of the last layer for this token serves as the sequence representation. In both configurations, we utilize this vector as the input representation. In the context of evidence usage, we employ the `[SEP]` special token to demarcate the claim and evidence. Following the recommendations in the BERT paper [21], we fine-tune our model using a set of specified learning rates ($5e-5$, $4e-5$, $3e-5$, and $2e-5$), with the learning rate of $3e-5$ delivering the best results. Both models undergo training for 2 epochs with Adam optimizer [39] with a batch size of 16. Additionally, we assess our framework against in-context learning with larger language models to gauge the efficacy of fine-tuning. For the few-shot classification, we prompted GPT-3.5-Turbo

Claim: The U.S. Constitution owes its notion of democracy to the Iroquois Tribes.
Evidence: 1. We zeroed in on the meme's underlying premise, that (...) 2. The meme features the following claim: "The U.S. (...) 3. Even if there was some Iroquois influence, it (...) 4. Similarities in government structure. The Iroquois (...)
Reason: The statement is not entirely accurate. While there were indeed similarities in government structure between the Iroquois Confederacy and the United States government, it is an oversimplification to claim that the U.S. Constitution solely owes its notion of democracy to the Iroquois Tribes. The influence of ancient Greek and European thought, as well as other sources, played a significant role in shaping the Constitution. The Iroquois Confederacy was one of several influences that contributed to the development of democratic principles in the United States, but it is not the sole or primary source.
Label: barely-true
Predicted label: half-true

Claim: Barack Obama's health care plan "will cost taxpayers $1,700 more to cover each new person."
Evidence: 1. Obama's plan "will cost taxpayers $1,700 (...) 2. For evidence that Obama's plan will cost (...) 3. Sen. Hillary Clinton has a new health care (...) 4. Gruber studied broad models for providing (...)
Reason: The statement accurately reflects the claim made in the Clinton campaign's ad and supported by Paul Krugman of the New York Times. Multiple sources, including Gruber, have found that Obama's plan, compared to Clinton's plan, results in a $1,700 per year difference in per-person costs. This difference in cost is attributed to the mandate included in Clinton's plan, which requires everyone to have health insurance.
Label: barely-true
Predicted label: true

Table 4.2: The tables display two instances of the model's incorrect label predictions.

with the same prompt used for the reason generation part.

To evaluate the performance of the models, we use four metrics:

1. **Accuracy:** Accuracy is a measure of the overall correctness of the model's predictions. It calculates the ratio of correctly predicted instances to the total number of instances:

$$Accuracy = \frac{TP + TN}{TN + TP + FN + FP}$$

2. **Recall (Sensitivity or True Positive Rate):** Recall assesses the model's ability to capture all the relevant instances of a class. It calculates the ratio of true positive predictions to the total actual positives:

$$Recall = \frac{TP}{TP + FN}$$

3. **Precision (Positive Predictive Value):** Precision gauges the accuracy of the positive predictions made by the model. It calculates the ratio of true positive predictions to the total predicted positives:

$$Precision = \frac{TP}{TP + FP}$$

4. **F1:** The F1 score is the harmonic mean of precision and recall, providing a balanced assessment of a model's performance. It is particularly useful when there is an imbalance between classes:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Where:

Models	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
BERT/Without Evidence	29.3	29.83	27.7	27.92
GPT-3.5-Turbo	37.1	36.16	34.35	30.3
BERT/With Evidence	40.29	43.19	39.74	40.83
Llama 2 (7b)	42.72 ± 0.19	50.1 ± 2.7	41.1 ± 0.52	39.16± 0.34

(a) Ablation study result

	Accuracy (%)
Wang et al. [86]	27.4
Koloski et al. [41]	26.75
Long et al. [50]	41.5
Triple-R	42.72

(b) Previous works vs. Triple-R framework

Table 4.3: Table (a) presents a comparative analysis of model performance in an ablation study. Table (b) provides a summary of accuracy scores achieved in previous studies, contrasting them with the proposed method.

- *TP* is True Positives (correctly predicted positive instances).
- *TN* is True Negatives (correctly predicted negative instances).
- *FP* is False Positives (incorrectly predicted as positive instances).
- *FN* is False Negatives (incorrectly predicted as negative instances).

Collectively, these metrics offer a comprehensive evaluation of the model's classification performance. For a more robust comparison between models, we conduct five separate evaluations of the Triple-R framework. The results of these evaluations are presented in Table 4.3a.

The results of fact verification for the Llama 2 (7b) model's performance on a subset of the LIAR dataset are summarized in Table 4.3. The evaluations show that GPT-3.5-Turbo was only able to generate a reason for around 36% of the dataset during reason generation, leaving the rest without correctly predicting the label. However, when combined with the rationalization method, the model managed to generate a reason for nearly 97.8% of the dataset.

As illustrated in Table 4.3a, fine-tuning the BERT model exclusively on claims reveals a substantial performance gap compared to other configurations. This observation underscores the need for more sophisticated techniques beyond simple text classification for effective fact verification. Notably, when contextual data is integrated, the same model with identical settings achieves a significantly higher accuracy. Additionally, the Llama 2 (7b) model outperforms the GPT-3.5-Turbo model by nearly 15%, even though the latter generates the training data for the former. Even the BERT model, fine-tuned on pairs of evidence and claims, outpaces the GPT-3.5-Turbo model while boasting approximately 1000 times fewer parameters. This leads us to the conclusion that fine-tuned smaller models can outperform few-shot classification using pre-trained larger language models. Additionally, we also contrasted our results with those from previous studies, which can be found in Table 4.3b. In Table 4.2, two examples are provided where the model failed to predict the correct label.

Figure 4.9 depicts the confusion matrix, which is the average of five experiments. The confusion matrix is a visual representation of the model's performance, providing a detailed breakdown of its predictions across different classes. This matrix is particularly useful in multiclass classification scenarios. It consists of four main components: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). Each row of the matrix corresponds to the actual class, while each column represents the predicted class. The diagonal elements (top-left to bottom-right) indicate correct predictions, while off-diagonal elements highlight instances where the model has misclassified.

The confusion matrix provided indicates the model's best performance on the false and true classes, as evidenced by the high counts of 179 and 146 on the matrix diagonal. The model tends to misclassify pants-fire and barely-true instances as false. Moreover, there's a notable confusion between true and mostly-true classes, as indi-

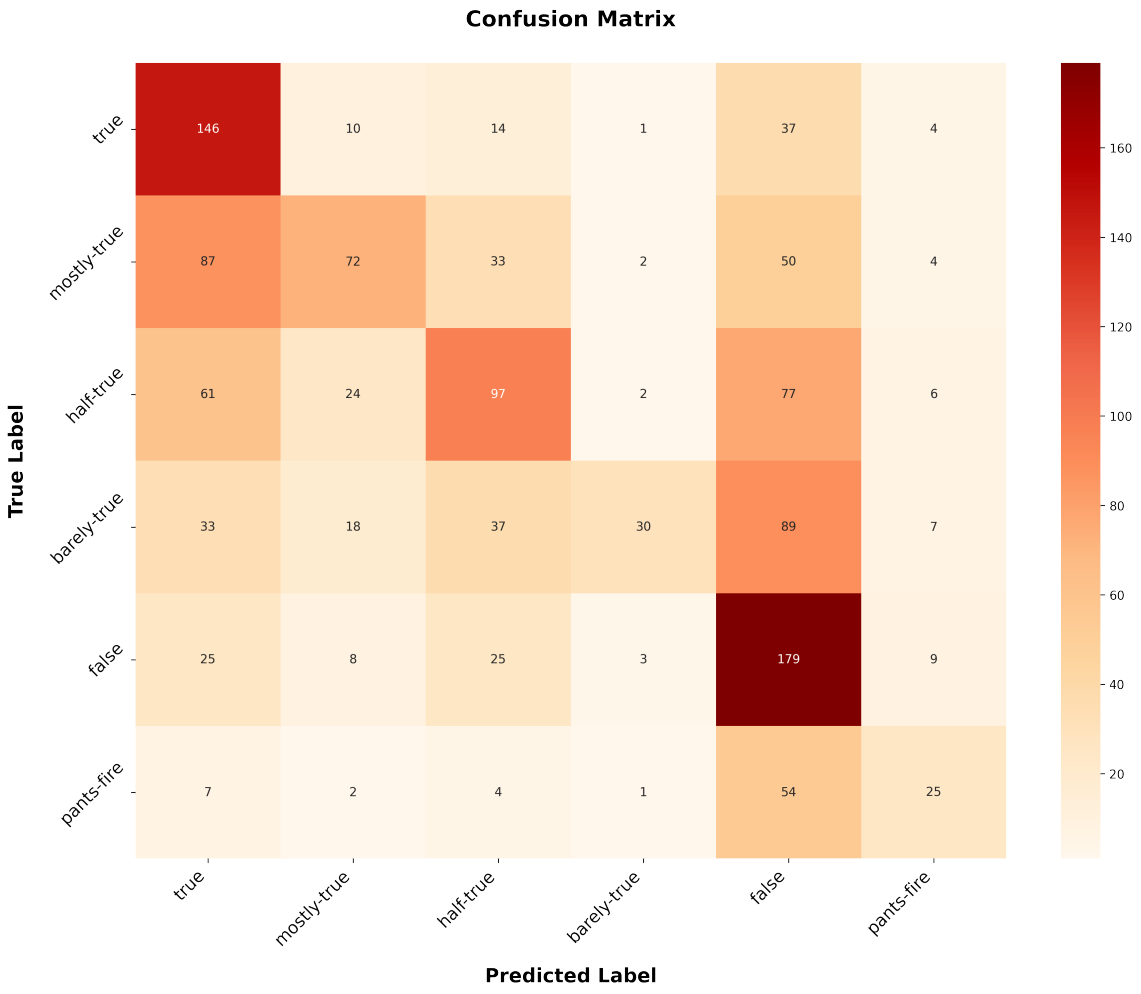


Figure 4.9: Confusion matrix illustrating the performance evaluation of the proposed model. Rows represent actual classes, while columns depict predicted classes.

cated by the substantial numbers off-diagonal, showing that the model may struggle to differentiate between these similar classes. The confusion is more pronounced among adjacent categories, such as pants-fire and false.

To improve the model's accuracy, two strategies could be employed. The first approach involves consolidating the labels into two broader categories: true (encompassing true, mostly-true, and half-true) and false (including barely-true, false, and pants-fire). This would simplify the classification task for the model. The second recommendation is to acquire more data for the classes that are underrepresented, namely barely-true, mostly-true, and pants-fire, to provide the model with more examples to learn from and thereby enhance its ability to distinguish between these nuanced categories.

Within the test set comprising 1283 samples, it's noteworthy that 19 of these samples failed to yield any search engine results. Nevertheless, during the evaluation of its performance on these samples, Llama 2 consistently achieved an average accuracy of approximately 34.7%. This achievement is particularly notable since the model is originally trained to make inferences and classifications based on pairs of evidence and claims. This suggests that the model exhibits the capability to classify claims independently too.

Furthermore, in the course of five evaluations, the model predominantly made label predictions as if they were next-sentence predictions. Interestingly, there was a singular instance where the model generated a label for a claim that did not fall under the predefined label categories.

4.5 Concluding Remarks

In this chapter, we have presented the implementation details and experimental results for our proposed Triple-R fact verification framework. We began by explaining

the techniques and tools used to realize each component, including the retriever, ranker, and reasoner modules. Careful considerations were made regarding efficiency and scalability during implementation.

Our framework employs a combination of neural networks and statistical models for evidence retrieval, leveraging the complementary strengths of semantic similarity and lexical matching. For reason generation, we harness the reasoning capabilities of large language models. To balance performance and computational constraints, the framework culminates in a fine-tuned smaller model.

Through an ablation study contrasting statement-only classification, statement-evidence classification, and reason generation models, we have demonstrated the value of our holistic approach. By retrieving contextual evidence and formulating logical reasons, our method achieves a state-of-the-art accuracy of 42.72% on the LIAR benchmark dataset.

In the next chapter, we conclude this thesis by summarizing our contributions and suggesting potential avenues for future work to advance automated fact verification systems.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

In the digital age, the surge of false information on social media has made fact verification an increasingly vital task. While fact-checking agencies strive to counteract misleading claims, the sheer volume of online content necessitates automated solutions. This thesis introduces the Triple-R framework, an automated system designed to address this challenge. Triple-R not only evaluates the veracity of claims but also provides reasons for its judgments, making the decision-making process transparent and trustworthy for users. By combining factual assessment with explanatory capabilities, Triple-R represents a significant step forward in the field of explainable fact verification.

We introduced the Triple-R framework, which comprises three pivotal components: Retriever, Ranker, and Reasoner. Our approach seeks to mimic the human process of claim assessment by gathering evidence from the internet and conducting reasoned analysis. The Retriever harnesses the web as its external knowledge source, employing a search engine to locate pertinent articles related to a claim. Subsequently, it extracts and segments the content into paragraphs. However, some of these para-

graphs may contain irrelevant information, potentially introducing noise. To address this, the Ranker component utilizes a hybrid method to score these paragraphs, leveraging two distinct approaches. The first employs a neural network model based on pre-trained DistilBERT, which gauges the contextual similarity between the claim and retrieved paragraphs. The second method utilizes Okapi BM25, a statistical approach that computes scores based on word overlap between the claim and paragraphs. Together, the Ranker and Retriever components furnish the system with contextual data, enabling it to remain updated on unobserved events and support its decision-making process.

Lastly, the Reasoner is anchored in an auto-regressive model based on the transformer architecture, known as Llama2-7b-chat. We fine-tune this model using evidence and the claim as inputs, generating a comprehensible rationale for the claim's veracity based on the evidence while also assigning a truthfulness label. This generated rationale not only aids human comprehension of the system's decision-making logic but also assists the model in reaching more accurate label assignments for claims.

Training the Reasoner to generate rationales necessitates a dataset comprising a rationale for each claim, derived from the Retrieved dataset. Yet, the challenge arises due to varying settings for the Ranker and Reasoner, potentially yielding distinct sets of evidence. Furthermore, the scarcity of annotated datasets poses a substantial obstacle to the advancement of explainable fact verification systems. To surmount this challenge, we introduce a novel method that capitalizes on the reasoning prowess of an LLM, specifically the GPT-3.5-Turbo model. This method enables the generation of rationales for individual claims based on their retrieved evidence.

To evaluate the Triple-R framework's performance, we carry out an extensive ablation study using the publicly accessible LIAR dataset, benchmarking our results against previous state-of-the-art methodologies. Our proposed method exhibits note-

worthy performance, achieving an impressive F1-score of 39.16% and an accuracy rate of 42.72%. The Triple-R framework offers several distinct advantages when compared to existing approaches, notably its ability to retrieve evidence from the web and generate human-readable rationales.

5.2 Future Works

In the future, our approach to explainable fact verification holds the potential for expansion on various fronts. Firstly, while our current method conducts verbatim searches for claims on the internet, exploring the extraction of claims in the format of search queries could yield more specific and relevant document retrieval. Secondly, in enhancing the Ranker component, we recognize the potential for improved quality by employing models fine-tuned specifically for semantic search tasks, thus enhancing the retrieval process. Thirdly, to fortify trust and accuracy, we need to refine our approach to reasoning. Currently, we associate correct label prediction with accurate reasoning. However, due to the inherent randomness in auto-regressive models, multiple valid reasons may arise from the same evidence and claim. Thus, it's crucial to develop techniques for assessing and selecting the most appropriate response that synthesizes information from the evidence section, thereby enhancing the model's trustworthiness. Fourthly, refining the Reasoner can benefit from advanced techniques like reinforcement learning from human feedback (RLHF), aligning the model with human preferences, and promoting the generation of more factual and robust rationales. Fifthly, our evidence-gathering approach considers all websites found by the search engine. In the future, strategies to prioritize reliable sources in evidence collection should be developed. Sixthly, we presently consider websites containing HTML content, but it's essential to expand our capability to extract evidence from other information sources, such as PDF files. Lastly, recognizing that misleading

information exists beyond textual content, there's a need for the development of multi-modal models that can incorporate various modalities, including images.

Overall, our proposed Triple-R framework presents a valuable mechanism for combating the spread of misinformation in online social media. It also lays a promising foundation for future research in the field of explainable fact verification.

References

[1] Common crawl. Accessed: 2023-06-20. URL: https://commoncrawl.org/.

[2] Eli Lilly Dives After Fake Twitter Account Promises Free Insulin; Takes Novo Nordisk, Sanofi With It. 2022. URL: https://www.investors.com/news/technology/lly-stock-dives-taking-novo-sanofi-with-it-after-fake-twitter-account-promises-free-insulin/.

[3] News Platform Fact Sheet. 2023. URL: https://www.pewresearch.org/journalism/fact-sheet/news-platform-fact-sheet/.

[4] Number of internet and social media users worldwide as of October 2023. 2023. URL: https://www.statista.com/statistics/617136/digital-population-worldwide/#:~:text=Worldwide%20digital%20population%202023&text=As%20of%20October%202023%2C%20there,population%2C%20were%20social%20media%20users.

[5] On The Same Page. 2023. URL: https://nmc-mic.ca/wp-content/uploads/2023/04/N247_2023-Fact-Sheet-1_FINAL-04.21.2023.pdf.

[6] Iftikhar Ahmad, Muhammad Yousaf, Suhail Yousaf, and Muhammad Ovais Ahmad. Fake news detection using machine learning ensemble methods. *Complexity*, 2020:1–11, 2020.

[7] Naser Ahmadi, Joohyung Lee, Paolo Papotti, and Mohammed Saeed. Explainable fact checking with probabilistic answer set programming. *arXiv preprint arXiv:1906.09198*, 2019.

[8] Mumtahina Ahmed, Mohammad Shahadat Hossain, Raihan Ul Islam, and Karl Andersson. Explainable text classification model for covid-19 fake news detection. *Journal of Internet Services and Information Security (JISIS)*, 12(2):51–69, 2022.

[9] Pepa Atanasova, Jakob Grue Simonsen, Christina Lioma, and Isabelle Augenstein. Generating fact checking explanations. *arXiv preprint arXiv:2004.05773*, 2020.

[10] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[11] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[12] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[14] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B Brown, Dawn Song, Ul-

far Erlingsson, et al. Extracting training data from large language models. In *USENIX Security Symposium*, volume 6, 2021.

[15] Chonghao Chen, Fei Cai, Xuejun Hu, Wanyu Chen, and Honghui Chen. Hhgn: A hierarchical reasoning-based heterogeneous graph neural network for fact verification. *Information Processing & Management*, 58(5):102659, 2021.

[16] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[17] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[18] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[19] Mark Davis and Martin Dürst. Unicode normalization forms, 2001.

[20] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[22] Yingtong Dou, Kai Shu, Congying Xia, Philip S Yu, and Lichao Sun. User preference-aware fake news detection. In *Proceedings of the 44th International*

ACM SIGIR Conference on Research and Development in Information Retrieval, pages 2051–2055, 2021.

[23] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR, 2022.

[24] Hao Fu, Yao; Peng and Tushar Khot. How does gpt obtain its ability? tracing emergent abilities of language models to their sources. *Yao Fu's Notion*, Dec 2022. URL: `https://yaofu.notion.site/How-does-GPT-Obtain-its-Ability-Tracing-Emergent-Abilities-of-Language-Models-to-their-Sources-b9a57ac0fcf74f30a1ab9e3e36fa1dc1`.

[25] Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*, 2022.

[26] Mohamed H Gad-Elrab, Daria Stepanova, Jacopo Urbani, and Gerhard Weikum. Exfakt: A framework for explaining facts over knowledge graphs and text. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 87–95, 2019.

[27] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[28] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large

minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[30] Danny Hernandez, Tom Brown, Tom Conerly, Nova DasSarma, Dawn Drain, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Tom Henighan, Tristan Hume, et al. Scaling laws and interpretability of learning from repeated data. *arXiv preprint arXiv:2205.10487*, 2022.

[31] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[32] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[33] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[34] Linmei Hu, Tianchi Yang, Luhao Zhang, Wanjun Zhong, Duyu Tang, Chuan Shi, Nan Duan, and Ming Zhou. Compare to the knowledge: Graph neural fake news detection with external knowledge. 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)*, pages 754–763, 2021.

[35] Sheikh Rabiul Islam, William Eberle, Sheikh Khaled Ghafoor, and Mohiuddin Ahmed. Explainable artificial intelligence approaches: A survey. *arXiv preprint arXiv:2101.09429*, 2021.

[36] Rohit Kumar Kaliyar, Anurag Goswami, and Pratik Narang. Fakebert: Fake news detection in social media with a bert-based deep learning approach. *Multimedia tools and applications*, 80(8):11765–11788, 2021.

[37] Rohit Kumar Kaliyar, Anurag Goswami, Pratik Narang, and Soumendu Sinha. Fndnet–a deep convolutional neural network for fake news detection. *Cognitive Systems Research*, 61:32–44, 2020.

[38] Nikhil Kandpal, Eric Wallace, and Colin Raffel. Deduplicating training data mitigates privacy risks in language models. In *International Conference on Machine Learning*, pages 10697–10707. PMLR, 2022.

[39] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[40] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[41] Boshko Koloski, Timen Stepišnik Perdih, Marko Robnik-Šikonja, Senja Pollak, and Blaž Škrlj. Knowledge graph informed fake news classification via heterogeneous representation ensembles. *Neurocomputing*, 496:208–226, 2022.

[42] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5, 2023.

[43] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[44] Sachin Kumar, Rohan Asthana, Shashwat Upadhyay, Nidhi Upreti, and Mohammad Akbar. Fake news detection using deep learning models: A novel approach. *Transactions on Emerging Telecommunications Technologies*, 31(2):e3767, 2020.

[45] Hugo Laurençon, Lucile Saulnier, Thomas Wang, Christopher Akiki, Albert Villanova del Moral, Teven Le Scao, Leandro Von Werra, Chenghao Mou, Eduardo González Ponferrada, Huu Nguyen, et al. The bigscience roots corpus: A 1.6 tb composite multilingual dataset. *Advances in Neural Information Processing Systems*, 35:31809–31826, 2022.

[46] Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. Internet-augmented language models through few-shot prompting for open-domain question answering. *arXiv preprint arXiv:2203.05115*, 2022.

[47] Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. On the advance of making language models better reasoners. *arXiv preprint arXiv:2206.02336*, 2022.

[48] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[49] Jinshuo Liu, Chenyang Wang, Chenxi Li, Ningxi Li, Juan Deng, and Jeff Z Pan. Dtn: Deep triple network for topic specific fake news detection. *Journal of Web Semantics*, 70:100646, 2021.

[50] Yunfei Long, Qin Lu, Rong Xiang, Minglei Li, and Chu-Ren Huang. Fake news detection through multi-perspective speaker profiles. In *Proceedings of the eighth*

international joint conference on natural language processing (volume 2: Short papers), pages 252–256, 2017.

[51] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[52] Ji Ma, Ivan Korotkov, Yinfei Yang, Keith Hall, and Ryan McDonald. Zero-shot neural passage retrieval via domain-targeted synthetic question generation. *arXiv preprint arXiv:2004.14503*, 2020.

[53] Mohit Mayank, Shakshi Sharma, and Rajesh Sharma. Deap-faked: Knowledge graph based approach for fake news detection. In *2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 47–51. IEEE, 2022.

[54] Mohit Mayank, Shakshi Sharma, and Rajesh Sharma. Deap-faked: Knowledge graph based approach for fake news detection. In *2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 47–51. IEEE, 2022.

[55] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[56] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[57] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*, 2019.

[58] Jamal Abdul Nasir, Osama Subhani Khan, and Iraklis Varlamis. Fake news detection: A hybrid cnn-rnn based deep learning approach. *International Journal of Information Management Data Insights*, 1(1):100007, 2021.

[59] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[60] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[61] Verónica Pérez-Rosas, Bennett Kleinberg, Alexandra Lefevre, and Rada Mihalcea. Automatic detection of fake news. *arXiv preprint arXiv:1708.07104*, 2017.

[62] Kashyap Popat, Subhabrata Mukherjee, Andrew Yates, and Gerhard Weikum. Declare: Debunking fake news and false claims using evidence-aware deep learning. *arXiv preprint arXiv:1809.06416*, 2018.

[63] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[64] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[65] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[66] Julio C. S. Reis, André Correia, Fabrício Murai, Adriano Veloso, and Fabrício Benevenuto. Supervised learning for fake news detection. *IEEE Intelligent Systems*, 34(2):76–81, 2019. doi:10.1109/MIS.2019.2899143.

[67] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[68] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[69] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[70] Aalok Sathe, Salar Ather, Tuan Manh Le, Nathan Perry, and Joonsuk Park. Automated fact-checking of claims from wikipedia. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 6874–6882, 2020.

[71] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

[72] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[73] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[74] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International conference on machine learning*, pages 3145–3153. PMLR, 2017.

[75] Kai Shu, Limeng Cui, Suhang Wang, Dongwon Lee, and Huan Liu. defend: Explainable fake news detection. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 395–405, 2019.

[76] Kai Shu, Suhang Wang, and Huan Liu. Exploiting tri-relationship for fake news detection. *arXiv preprint arXiv:1712.07709*, 8, 2017.

[77] Kai Shu, Suhang Wang, and Huan Liu. Exploiting tri-relationship for fake news detection. *arXiv preprint arXiv:1712.07709*, 8, 2017.

[78] Kai Shu, Suhang Wang, and Huan Liu. Understanding user profiles on social media for fake news detection. In *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 430–435, 2018. doi:10.1109/MIPR.2018.00092.

[79] Kai Shu, Xinyi Zhou, Suhang Wang, Reza Zafarani, and Huan Liu. The role of user profiles for fake news detection. In *Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining*, pages 436–439, 2019.

[80] Shivangi Singhal, Rajiv Ratn Shah, Tanmoy Chakraborty, Ponnurangam Kumaraguru, and Shin'ichi Satoh. Spotfake: A multi-modal framework for fake news detection. In *2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM)*, pages 39–47, 2019. doi:10.1109/BigMM.2019.00-44.

[81] Mateusz Szczepański, Marek Pawlicki, Rafał Kozik, and Michał Choraś. New explainability method for bert-based model in fake news detection. *Scientific reports*, 11(1):23705, 2021.

[82] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*, 2018.

[83] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[84] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[85] Nikhita Vedula and Srinivasan Parthasarathy. Face-keg: Fact checking explained using knowledge graphs. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 526–534, 2021.

[86] William Yang Wang. " liar, liar pants on fire": A new benchmark dataset for fake news detection. *arXiv preprint arXiv:1705.00648*, 2017.

[87] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[88] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Rationale-augmented ensembles in language models. *arXiv preprint arXiv:2207.00747*, 2022.

[89] Yaqing Wang, Fenglong Ma, Zhiwei Jin, Ye Yuan, Guangxu Xun, Kishlay Jha, Lu Su, and Jing Gao. Eann: Event adversarial neural networks for multi-modal fake news detection. In *Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining*, pages 849–857, 2018.

[90] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[91] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[92] Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomas Kocisky, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, et al. Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*, 2019.

[93] Eric Zelikman, Jesse Mu, Noah D Goodman, and Yuhuai Tony Wu. Star: Self-taught reasoner bootstrapping reasoning with reasoning. 2022.

[94] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[95] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

[96] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Appendix A

Reasoning with Language Models

Language models have changed the way information is extracted from unstructured text. Recently, pre-trained language models have become handy tools and have been applied in a task-agnostic and flexible way for many downstream tasks.

Language models have a wide range of applications, including text classification, question-answering, and even creative writing. They have been used to improve the accuracy of language-based tasks such as sentiment analysis and text summarization and have also been used to generate new text, from writing news articles to composing poetry.

In this appendix, we will explore the inner workings of language models, their applications, and their variations. We will also investigate the recent breakthroughs and the latest research in this field.

Overall, language models are a rapidly advancing field with a wide range of potential applications, making them an exciting and important area of study.

A.1 What are Language Models

Basically, a language model predicts the next word for a sequence of words. A simple form of language models is widely used on phones' keyboards to suggest the

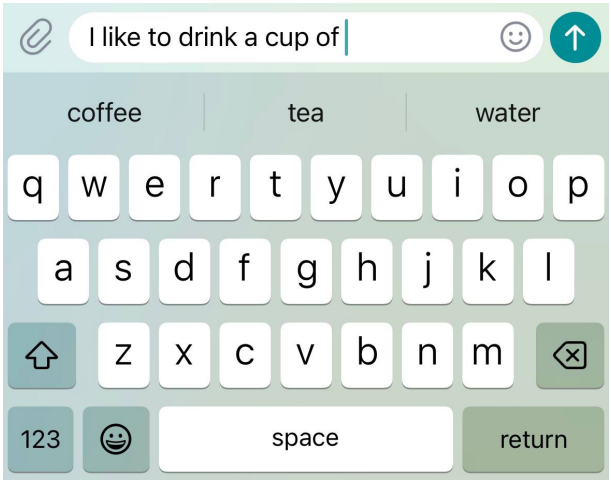


Figure A.1: A simple application of language models is to suggest the next word on phones' keyboards.

next word, Figure A.1. In other words, a language model produces a probability distribution over a sequence of words. Then, we choose a suitable word for the input words based on this, Figure A.2. This process is similar to the classification task. Here, instead of a few classes, the language model classifier has thousands of classes, and every class represents a token.

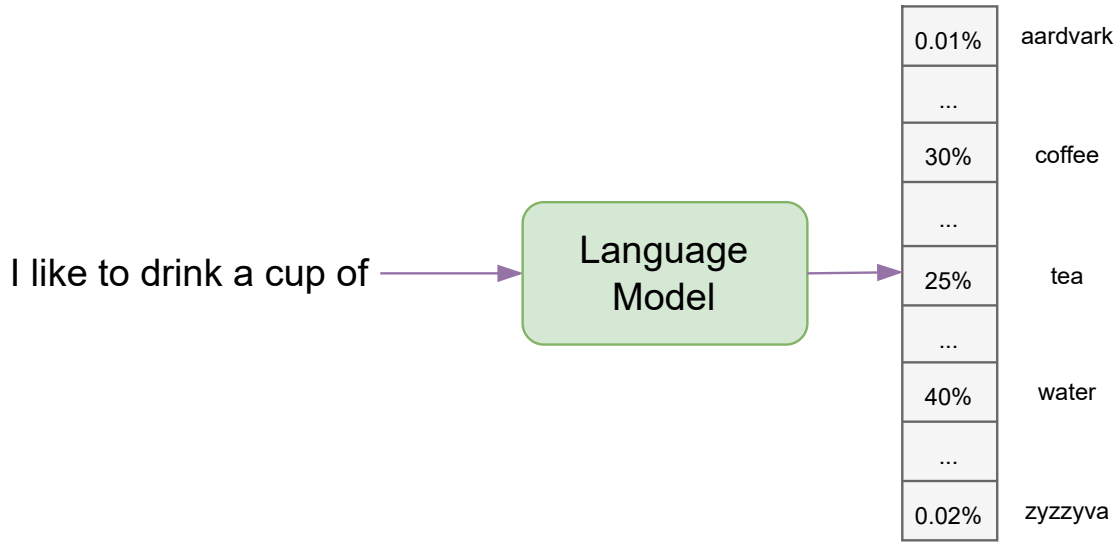


Figure A.2: Given a sequence of words, a language model generates a probability distribution for the next word.

A.2 The Stages of Language Models

The study of language models (LM) has garnered significant interest in academic literature, encompassing four distinct stages of advancement:

A.2.1 Statistical Language Models

During the 1990s, statistical learning methods gave rise to the development of statistical language models (SLMs). These models are built on the concept of the Markov assumption, which allows for predicting the likelihood of future word units without extensive consideration of past contexts. The underlying principle involves constructing a word prediction model using probabilistic techniques.

The n-gram model is widely used as a statistical language model (SLM). In this model, rather than calculating the probability of a word based on its complete history, we can estimate the history by considering only the most recent few words.

As an illustration, the bigram model approximates the probability of a word given all the previous words $P(w_n|w_{1:n-1})$ by considering solely the conditional probability of the preceding word $P(w_n|w_{n-1})$. For example, instead of calculating the probability

$$P(\text{coffee}|\text{I like to drink a cup of}) \tag{A.1}$$

we make an approximation using the probability

$$P(\text{coffee}|\text{of}) \tag{A.2}$$

This approximation reduces the dependency on the complete context and focuses on the conditional probability based on the last observed word.

By employing a bigram model to anticipate the conditional probability of the sub-

sequent word, we make the subsequent approximation:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1}) \tag{A.3}$$

This approximation involves estimating the likelihood of the current word based solely on the previous word, neglecting the influence of all other preceding words. For example, to compute a particular bigram probability of a word w_n given a previous word w_{n-1}, we compute the count of the bigram $C(w_{n-1}w_n)$ and normalize by the unigram count for word w_{n-1}:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{A.4}$$

N-gram language models have found extensive applications in improving performance in tasks related to information retrieval (IR) and natural language processing (NLP), such as speech recognition, machine translation, and text classification. However, they encounter the issue of sparsity, where certain N-grams may not be present in the training corpus, resulting in zero probabilities. This challenge can be addressed by employing smoothing techniques like Laplace smoothing and Good-Turing smoothing. Another limitation of the n-gram language model is its assumption that a word's probability is solely dependent on its preceding N-1 words, which may not hold true in real-world scenarios. Furthermore, the model fails to capture the semantic nuances of words and struggles with the inherent ambiguity of language.

A.2.2 Neural Language Models

Neural language models (NLM) aim to capture the likelihood of word sequences by leveraging neural networks, particularly recurrent neural networks (RNN), among other architectures.

An influential contribution can be observed in the research by Bengio et al. [12],

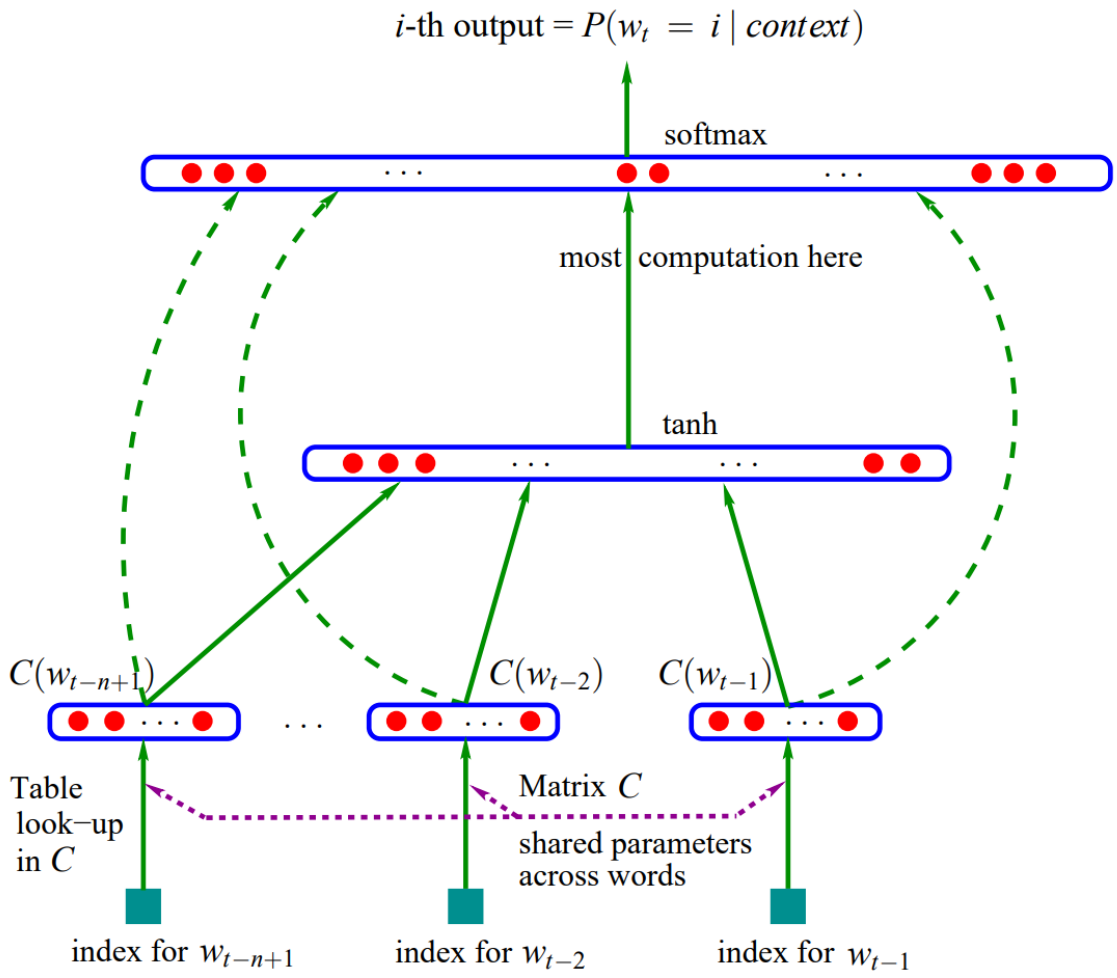


Figure A.3: Illustration of the neural architecture $f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))$, where g represents the neural network and $C(i)$ denotes the feature vector associated with the i-th word [12].

where they introduced the notion of *distributed representation*. This approach, akin to the n-gram model, estimates the probability of the subsequent word based on the n preceding words. However, what sets it apart is the utilization of a neural network, i.e., $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t|w_{1:t-1})$.

The function $f(w_t, \ldots, w_{t-n+1})$ is decomposed in two parts:

1. In the given context, the function C maps each element i from the vocabulary set V to a real vector $C(i) \in \mathbb{R}^m$. This mapping signifies the presence of *distributed feature vectors* associated with individual words in the vocabulary. To implement this concept, the function C is represented by a matrix of free parameters with dimensions $|V| \times m$.

2. The function g takes an input sequence of feature vectors for words in context, denoted as $(C(w_{t-n+1}), \ldots, C(w_{t-1}))$, and generates a conditional probability distribution over words in V for the next word w_t. The result produced by g is a vector, with each element estimating the probability $\hat{P}(w_t = i|w_{1:t-1})$ as depicted in Figure A.3.

$$f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1})) \tag{A.5}$$

Later, the matrix C came to be known as the word embeddings matrix. Rather than utilizing one-hot vectors, which are vectors with all elements being zero except for the index corresponding to the particular word being represented (which is one), word embedding techniques assign words to dense and continuous vectors within a high-dimensional space. It is a form of feature representation that aims to capture the semantic and syntactic relationships between words based on their contextual usage.

Word embeddings are typically learned from large corpora using unsupervised learning algorithms like Word2Vec [56] and GloVe [60]. These algorithms analyze the

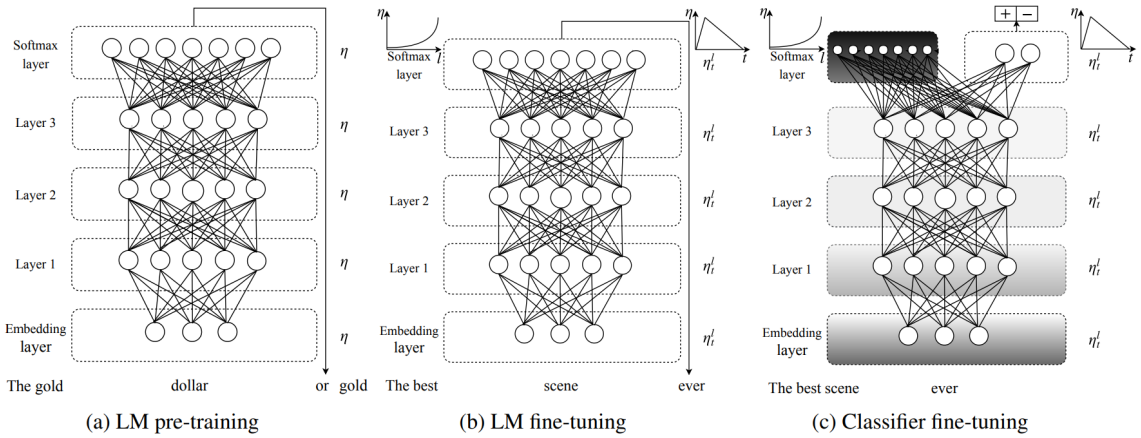


Figure A.4: Illustrating the three stages of the ULMFiT method, which involves training a language model (LM) on a general-domain corpus, fine-tuning the pre-trained LM on the target task data, and performing fine-tuning of the classifier on the target task data [32].

co-occurrence patterns of words in a given dataset and create dense vector representations where similar words are closer to each other in the vector space. The resulting word embeddings can be used as input features for various NLP tasks, such as text classification, sentiment analysis, machine translation, and information retrieval, to name a few.

Word embeddings have revolutionized NLP by enabling machines to understand and process human language more efficiently, bridging the gap between symbolic representations and numerical computations. They have significantly contributed to advancements in tasks like language understanding, document clustering, and semantic search.

A.2.3 Pre-trained Language Models

In its initial stages, an approach called universal language model fine-tuning (ULMFiT) was introduced [32]. ULMFiT serves as a transfer learning technique applicable to a wide range of tasks within natural language processing (NLP). It suggests techniques that enable the fine-tuning of a language model for specific tasks.

ULMFiT, which builds upon the AWD-LSTM architecture [55], follows a three-stage

process:

1. The language model (LM) undergoes training on a general-domain corpus to capture the language's overarching characteristics across different layers.

2. To address the distribution dissimilarity between the general-domain data and the target task data, the pre-trained LM is subjected to fine-tuning with discriminative fine-tuning and slanted triangular learning rates (STLR). This enables the model to acquire task-specific features.

3. The classifier is fine-tuned on the target task, employing gradual unfreezing and STLR techniques. This approach preserves low-level representations while adapting high-level representations.

All three stages are visually represented in Figure A.4.

The introduction of the Transformer architecture has revolutionized the field of Natural Language Processing (NLP). The Transformer, proposed in the seminal work "Attention is All You Need" [84], introduced a groundbreaking approach to sequence modeling and language understanding. It departed from traditional approaches, e.g., recurrent neural network (RNN) and convolutional neural network (CNN) models, and introduced a self-attention mechanism that enables the model to capture dependencies between words in an input sequence more effectively. The Transformer architecture has demonstrated remarkable performance in various NLP tasks, including machine translation, language generation, and text classification. Its ability to model long-range dependencies and capture contextual relationships in a parallelizable manner has led to significant advancements in the NLP field.

Drawing inspiration from the Transformers architecture, the concept of BERT [21] was introduced. BERT involves pre-training bidirectional language models with specifically designed tasks on extensive unlabeled corpora. These pre-trained word embeddings, which exhibit contextual awareness, serve as highly effective semantic

features with broad applicability across NLP tasks. Consequently, the introduction of BERT has significantly elevated the performance standards in the field. The success of BERT has sparked a wealth of subsequent research, establishing the paradigm of *pre-training* and *fine-tuning* learning. This paradigm has led to the development of numerous pre-trained language models (PLMs) that either adopt alternative architectures, such as GPT-2 [63] and T5 [65], or employ enhanced pre-training strategies. Within this framework, PLMs are commonly leveraged and adapted to diverse downstream tasks.

A.2.4 Large Language Models

Through the process of scaling pre-trained language models (PLMs), whether by enlarging the model size or expanding the training data size, notable advancements have been observed in downstream tasks. Extensive research has delved into the performance of PLMs by training progressively larger models, exemplified by the 175B-parameter GPT-3 [13] and the 540B-parameter PaLM [17]. While scaling primarily revolves around augmenting the model size while retaining a similar architecture and pre-training tasks, these larger PLMs exhibit distinct behaviors compared to their smaller counterparts, such as the 330M-parameter BERT and 1.5B-parameter GPT-2. These complex tasks elicit varied responses from the different-sized PLMs. For instance, GPT-3 showcases remarkable performance in few-shot tasks through in-context learning, whereas GPT-2 struggles in comparison. Consequently, the research community has coined the term *"large language models (LLM)"* to refer to these magnified PLMs.

In the subsequent sections, a comprehensive exploration of the various architectures of Transformers models and the emerging capabilities of LLMs will be presented.

A.3 Language Model Training

Language models, in particular those based on neural networks, have a key advantage over most machine learning models, which is that they do not require labeled data, making them well suited for many natural language processing tasks.

Language models utilize self-supervised learning to grasp patterns within text. Self-supervised learning capitalizes on the inherent structure of data to extract supervisory signals, enabling the prediction of concealed aspects based on observable segments. In the realm of NLP, a common practice involves masking a portion of a sentence and predicting the hidden words using the remaining context. Similarly, in video analysis, future or past frames can be predicted based on the present frames. By leveraging the intrinsic structure of the data, self-supervised learning can harness diverse supervisory signals spanning multiple modalities like video and audio, leveraging extensive datasets without explicit reliance on labeled data.

Additionally, obtaining labeled data can be time-consuming and expensive. A self-supervised model can learn from the massive amount of unstructured and unlabeled data, which is a more practical solution. Also, developing a model for a specific task requires designing a task-specific architecture and curating so many hyper-parameters; however, language models do not need dramatic changes in architecture for downstream tasks.

The two sections that follow, namely Data Sources and Data Preprocessing, draw heavily upon the research conducted by Zhao et al. in their paper titled "A Survey of Large Language Models" [96].

A.4 Data Sources

High-quality, extensive datasets are essential for the pre-training of language models, particularly large language models, as their model capacities heavily depend on the

pre-training corpus and its preprocessing. The acquisition of a capable language model necessitates the aggregation of significant quantities of natural language data from diverse sources. Typically, existing language models utilize a blend of diverse publicly available textual datasets as their pre-training corpus.

The origin of the pre-training corpus can be broadly classified into two categories: general data and specialized data. General data, such as webpages, books, and conversational text, is commonly utilized by most language models. This type of data is preferred due to its vast and diverse nature, making it valuable for improving language modeling and generalization abilities. Additionally, language models have demonstrated impressive generalization capabilities, prompting some studies to extend the pre-training corpus to include more specialized datasets. These specialized datasets can include multilingual data, scientific data, and code, which provide language models with specific capabilities for solving particular tasks. In the following sections, we will delve into a detailed description of these two types of pre-training data sources [96].

A.4.1 General Text Data

Most language models commonly utilize general-purpose pre-training data, including webpages, books, and conversational text, as these sources offer a wide range of textual information covering diverse topics.

- **Webpages**: The Internet has played a significant role in generating a wide range of data, enriching the linguistic knowledge available to language models, and boosting their ability to generalize. For instance, projects like CommonCrawl [1] have crawled massive amounts of web data, which is readily accessible for analysis by anyone. However, web data encompasses a mix of high-quality content like Wikipedia and lower-quality text such as spam emails. Hence, it becomes crucial to filter and preprocess webpages to enhance the overall data

quality.

- **Conversation text**: Conversational data plays a crucial role in training language models to engage in conversations and potentially improve their performance on various question-answering tasks. To incorporate conversational data, researchers can utilize publicly available conversation subsets, such as those from platforms like Reddit[1], or gather conversation data from online social media sources. Since online conversational data often involves multiple participants engaging in discussions, it is beneficial to organize the conversations into a tree structure. This structure links each utterance to the one it responds to, facilitating effective processing and analysis. By dividing the multi-party conversation tree into sub-conversations, valuable conversational data can be collected and included in the pre-training corpus.

- **Books**: Compared to other sources, books are beneficial to pre-training language models in different ways. They are an important source of formal long texts, which are potentially beneficial for language models to learn linguistic knowledge, model long-term dependency, and generate narrative and coherent texts. To obtain open-source book data, existing studies usually adopt Books3 and Bookcorpus2 datasets, which are available in the Pile dataset [27].

A.4.2 Specialized Text Data

Specialized datasets play a crucial role in enhancing the targeted capabilities of language models for specific downstream tasks. In the following sections, we present three distinct categories of specialized data.

- **Multilingual text**: Besides the text in the target language, integrating a multilingual corpus can enhance the multilingual abilities of language under-

[1]https://www.reddit.com/

standing and generation. For example, BLOOM [71], and PaLM [17] have curated multilingual data covering 46 and 122 languages, respectively, within their pre-training corpora. These models demonstrate impressive performance in multilingual tasks, such as translation, multilingual summarization, and multilingual question answering, and achieve comparable or superior performance to the state-of-the-art models that are fine-tuned on the corpus in the target language(s).

- **Scientific text**: The rapid growth of scientific publications attests to humanity's continuous exploration of science. To enrich the understanding of scientific knowledge within language models, incorporating a scientific corpus for pre-training proves valuable. By exposing language models to extensive scientific text during pre-training, they demonstrate remarkable performance in scientific and reasoning tasks. Constructing the scientific corpus primarily involves aggregating arXiv papers, scientific textbooks, math web pages, and other pertinent scientific resources. Given the intricate nature of scientific data, encompassing mathematical symbols and protein sequences, specialized tokenization and preprocessing techniques are often necessary. These techniques ensure that the diverse data formats are transformed into a standardized representation suitable for processing within language models.

- **Code**: The research community has extensively investigated program synthesis, particularly leveraging pre-trained language models (PLMs) trained on code. However, effectively generating high-quality and precise programs remains a challenge. Recent investigations [16, 10] have revealed that training language models on a large corpus of code can significantly enhance the quality of synthesized programs. These generated programs demonstrate successful execution on expert-designed unit-test cases [16] and proficiency in solving competitive programming questions [48]. Typically, two primary sources of code

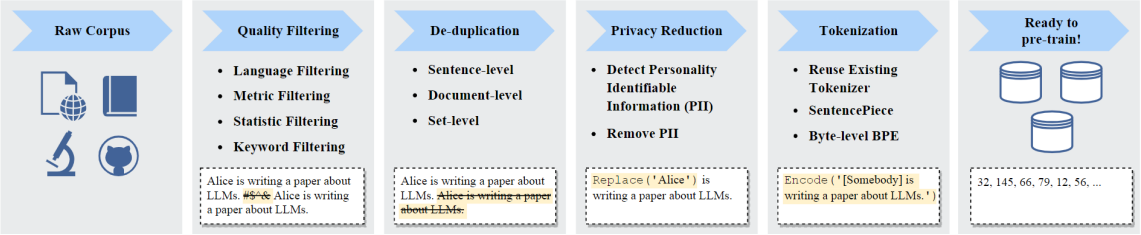


Figure A.5: A visualization depicting a standard data preprocessing pipeline employed for pre-training language models [96].

corpora are employed for pre-training language models. The first source involves programming question-answering communities, such as Stack Exchange, while the second source encompasses public software repositories like GitHub, where code data, including comments and docstrings, are harnessed. Unlike natural language text, code exhibits the format of a programming language, necessitating consideration of long-range dependencies and accurate execution logic. An insightful study [24] also speculates that code-based training could be instrumental in fostering complex reasoning abilities, such as chain-of-thought ability [90].

A.5 Data Preprocessing

The effectiveness and performance of language models heavily depend on the quality of the pre-training data. Once a substantial amount of text data has been gathered, it becomes essential to preprocess the data in order to construct a high-quality pre-training corpus. This preprocessing step involves the removal of noisy, redundant, irrelevant, and potentially harmful data. In this section, we outline the specific strategies employed for data preprocessing, which contribute to enhancing the overall quality of the collected data. To provide a visual representation of the preprocessing process for the language model's pre-training corpus, Figure A.5 illustrates a typical pipeline.

A.5.1 Quality Filtering

To eliminate low-quality content from the collected corpus, current research commonly employs two strategies: (1) classifier-based and (2) heuristic-based approaches. The classifier-based approach involves training a selection classifier using high-quality texts as positive instances and candidate data as negative instances. This classifier is then utilized to identify and filter out low-quality data based on a quality score assigned to each example. Notably, these methods [13, 17] often employ well-curated data like Wikipedia pages for training. However, some studies [64, 23] highlight that relying solely on classifier-based approaches can inadvertently remove valuable texts in dialectal, colloquial, and sociolectal languages, leading to bias in the pre-training corpus and reducing its overall diversity. In contrast, the heuristic-based approach, exemplified by BLOOM [71], applies a set of carefully crafted rules to identify and exclude low-quality texts, specifically characterized by the following:

- **Language-based filtering**: If a language model is primarily intended for specific language tasks, the text in other languages can be selectively filtered.
- **Metric-based filtering**: Evaluation metrics, such as perplexity, can be utilized to identify and eliminate unnatural sentences in the generated texts.
- **Statistic-based filtering**: Evaluation metrics, such as perplexity, can be utilized to identify and eliminate unnatural sentences in the generated texts.
- **Keyword-based filtering**: By employing a specific keyword set, it becomes possible to identify and eliminate noisy or irrelevant elements in the text, such as HTML tags, hyperlinks, boilerplates, and offensive words.

A.5.2 De-duplication

Previous research [30] has discovered that duplicate data within a corpus can diminish the diversity of language models, potentially leading to instability during training and affecting model performance. Hence, it becomes necessary to eliminate duplicates from the pre-training corpus. De-duplication can be carried out at different levels, including sentence level, document level, and dataset level. Initially, it is important to remove low-quality sentences that exhibit repeated words and phrases, as they can introduce repetitive patterns in language modeling [31]. At the document level, existing studies mostly rely on the comparison of surface features (e.g., word and n-gram overlap) to identify and discard duplicate documents with similar content [83, 64, 71]. Additionally, to prevent contamination between the training and evaluation sets [17], it is crucial to identify and remove any potential duplicate texts from the training set. It has been demonstrated that employing all three levels of de-duplication proves beneficial in enhancing the training process of language models [17], making their joint utilization recommended in practical scenarios.

A.5.3 Privacy Reduction

The acquisition of pre-training text data predominantly relies on web sources, encompassing content generated by users that may involve sensitive or personal information, thereby amplifying the potential risk of privacy breaches [14]. Consequently, it becomes essential to eliminate personally identifiable information (PII) from the pre-training corpus. One effective approach is the direct utilization of rule-based methods, such as keyword spotting, to identify and remove PII instances like names, addresses, and phone numbers [45]. Moreover, researchers have also discovered that the susceptibility of language models to privacy attacks can be attributed to the presence of duplicate PII data within the pre-training corpus [38]. Thus, de-duplication can partially mitigate privacy risks as well.

A.5.4 Tokenization

Tokenization is a vital step in data preprocessing as it involves breaking down the raw text into individual tokens, which are subsequently used as inputs for language models. While it may be convenient to utilize existing tokenizers (e.g., OPT [94] and GPT-3 [13] employ the GPT-2 tokenizer [63]), employing a tokenizer specifically designed for the pre-training corpus can yield significant benefits [42], particularly for corpora that encompass diverse domains, languages, and formats. Consequently, several recent language models have trained custom tokenizers, specifically tailored for their pre-training corpus, employing SentencePiece [43]. To ensure lossless information post-tokenization, the byte-level Byte Pair Encoding (BPE) algorithm [73] is employed [17, 64]. However, it is worth noting that certain normalization techniques within BPE, such as NFKC [19], may adversely affect tokenization performance [64, 71].

A.6 Architecture

The Transformers architecture [84] has emerged as the go-to model for language and sequence modeling tasks, surpassing the traditional reliance on RNN and CNN models. Originally presented with an encoder-decoder framework, the Transformers architecture has since evolved to accommodate various applications, with the encoder and decoder components often employed independently. This section delves into the diverse variations of the Transformers architecture, exploring its wide-ranging applications and the distinct pretraining objectives pursued in different contexts.

A.6.1 Bidirectional Encoder Models

In this section, the introduction of the bidirectional transformer encoder, which is trained using masked language modeling, is presented. With this method, the entire

text, encompassing both the left and right context, can be observed by the model. The BERT model [21] is then introduced as the extensively utilized version of the masked language modeling architecture.

In addition, two significant concepts commonly associated with these language models are examined. The first concept is **fine-tuning**, which involves further training the pre-trained models by incorporating an additional neural network classifier. This classifier utilizes the output of the model's top layer to perform various downstream tasks like named entity recognition or question answering. The underlying idea is that the pre-training phase facilitates the learning of a language model that captures nuanced word meanings, thus enabling easier adaptation or fine-tuning for specific language understanding tasks. The pretrain-finetune approach is an example of **transfer learning** in machine learning, where knowledge acquired from one task or domain is leveraged and applied to solve novel tasks.

The second concept is contextual embedding, which involves representing words based on their context. Unlike traditional word embedding techniques like word2vec [56] or GloVe [60], which assign a fixed embedding vector to each word in the vocabulary, contextual embeddings, such as those learned by masked language models like BERT, generate unique vectors for each occurrence of a word in different contexts.

The transformer architecture introduces novel mechanisms, such as self-attention and positional encodings, to effectively capture temporal information and understand the relationships between words across long distances. The encoder is constructed with a stack of N identical layers, each consisting of two sub-layers. The first sub-layer incorporates a multi-head self-attention mechanism, while the second sub-layer utilizes a simple position-wise, fully connected feed-forward network. Both sub-layers are accompanied by a residual connection, and the output of each sub-layer is normalized using a layer norm function. The subsequent subsections provide a comprehensive discussion of each component.

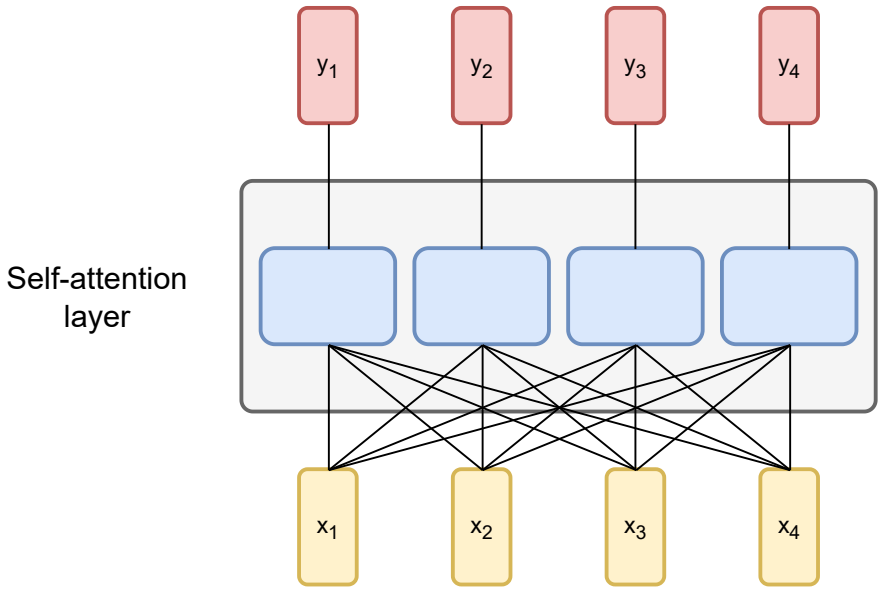


Figure A.6: The flow of information in a bidirectional transformer encoder, where each element is influenced by both preceding and succeeding context.

A.6.1.1 Self-Attention Networks

The self-attention layer is a fundamental breakthrough in transformer models. It enables the network to extract and incorporate information from extensive contexts, eliminating the reliance on intermediary recurrent connections, as seen in RNNs. By applying self-attention, input sequences $(x_1, \ldots, x_n)$ are transformed into output sequences of equal length $(y_1, \ldots, y_n)$, with each output vector enriched by incorporating information from both the preceding and succeeding context. Refer to Figure A.6 for an illustration of the encoding attention flow.

In order to compute the output, transformers introduced three weight matrices: W^Q, W^K, and W^V. These weights are trained during the training process and will be utilized to map each input vector x_i into a representation corresponding to its role as a *query*, *key*, and *value*. This mapping is achieved through the following equations:

$$q_i = x_i W^Q; \quad k_i = x_i W^K; \quad v_i = x_i W^V \tag{A.6}$$

Both the input x and output y of transformers, as well as the intermediate vectors

at different layers, have the same dimensionality of $N \times d_{model}$, where N is the number of tokens and d_{model} is the dimension of the sub-layers and layers outputs. It is important to note that the dimension of the q_i, k_i, and v_i vectors is smaller than the embedding dimension. Therefore, the transformer matrices are defined as $W^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W^K \in \mathbb{R}^{d_{model} \times d_k}$, and $W^V \in \mathbb{R}^{d_{model} \times d_v}$.

Based on the projected vectors, the score between the current focus of attention, x_i, and an element in the input, x_j, is computed by taking the dot product between the corresponding query vector, q_i, and the key vector of the element, k_j. Hence, the score for these two elements can be expressed as:

$$\text{score}(x_i, x_j) = q_i \cdot k_j \tag{A.7}$$

To prevent numerical issues and loss of gradients during training, it is necessary to scale the dot product in a suitable manner. A commonly used approach is the scaled dot-product, where the dot-product result is divided by the square root of the dimensionality of the query and key vectors, d_k. This ensures that the values stay within a reasonable range. Hence, the updated version of Equation A.7 becomes:

$$\text{score}(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}} \tag{A.8}$$

After computing the scores between the token x_i and other tokens, these scores are transformed into probabilities using a softmax function. The self-attention output for the input x_i is then obtained by taking a weighted sum over the corresponding value vectors v. This process is depicted in Figure A.7 when calculating the first output z_1 in a sequence. The self-attention output for x_i is represented as:

$$z_i = \sum_{j=1}^{N} \alpha_{ij} v_j \tag{A.9}$$

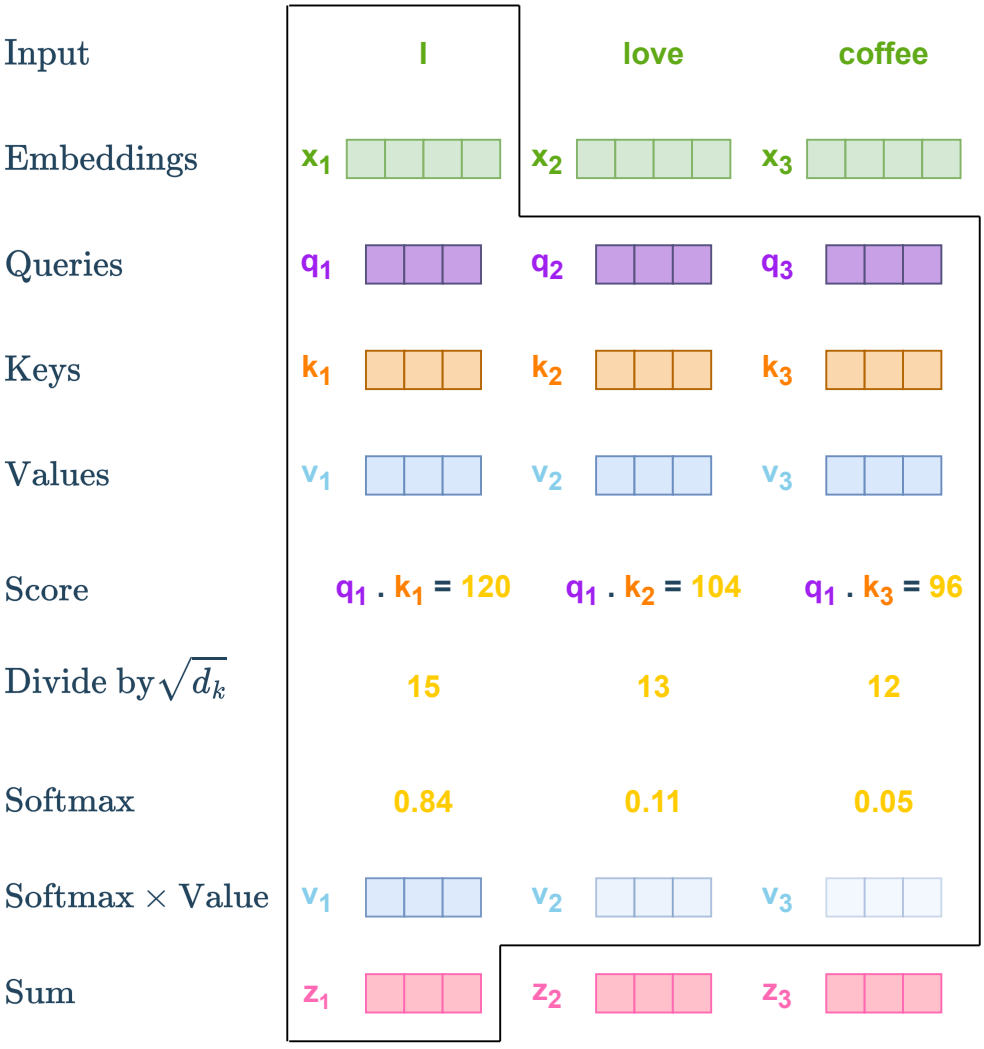


Figure A.7: Calculating the value of z_1 using self-attention

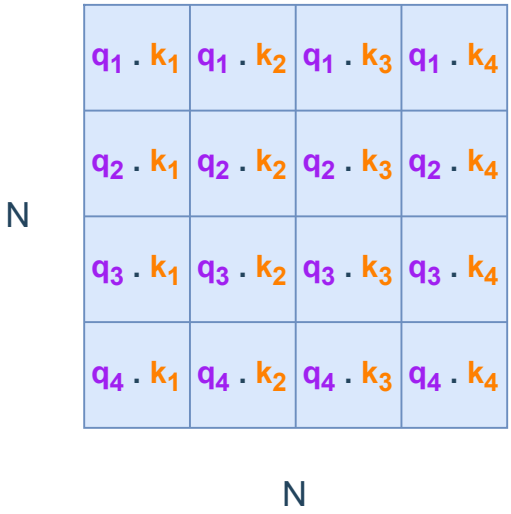


Figure A.8: The $N \times N$ matrix represents the dot product values $q_i \cdot k_j$ between query vectors q_i and key vectors k_j.

To enable parallel computation and leverage efficient matrix multiplication routines, the attention calculation can be optimized. This involves packing the input embeddings of N tokens from the input sequence into a single matrix $x \in \mathbb{R}^{N \times d_{model}}$, where each row represents the embedding of a token. Then, the key and query matrices, with dimensions $d_{model} \times d_k$, and the value matrix, with dimensions $d_{model} \times d_v$, are multiplied with the input matrix x. This multiplication yields matrices $Q \in \mathbb{R}^{N \times d_k}$, $K \in \mathbb{R}^{N \times d_k}$, and $V \in \mathbb{R}^{N \times d_v}$, which contain all the key, query, and value vectors, respectively. The matrix operations can be represented as:

$$Q = xW^Q; \quad K = xW^K; \quad V = xW^V \tag{A.10}$$

Using these matrices, the query-key comparisons can be efficiently computed by performing a single matrix multiplication between Q and K^T, resulting in an $N \times N$ matrix as shown in Fig. A.8. Subsequently, the attention scores matrix is scaled and passed through the softmax function. The resulting probability matrix is then multiplied by V, producing a matrix of shape $N \times d_v$. The entire self-attention

process for a sequence of N tokens can be summarized as follows:

$$\text{SelfAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{A.11}$$

A.6.1.2 Multi-Head Attention

In a text, words can exhibit multiple simultaneous relationships with one another. For example, within a sentence, verbs and their arguments may possess distinct syntactic, semantic, and discourse connections. Attempting to capture all these diverse parallel relations within a single transformer block would pose challenges. To address this, transformers employ linear projections to transform the queries, keys, and values h times, each with separate learned linear projections to dimensions d_k, d_k, and d_v, respectively. By utilizing these distinct parameter sets, each attention head can specialize in learning different aspects of the relationships present among inputs at the same level of abstraction. This allows for a more comprehensive understanding of the varied connections within the text.

To implement the **multi-head attention** mechanism, a self-attention layer contains multiple heads, denoted by i, each having its own set of key, query, and value matrices: W_i^Q, W_i^K, and W_i^V. These matrices allow each head to independently transform the input into separate key, value, and query embeddings while preserving the rest of the self-attention computation. The dimensions for each head are $d_k = d_v = d_{model}/h$, where h represents the number of heads. As a result, the output of the multi-head attention layer has a shape of $N \times hd_k$. To integrate these vectors for further processing, they are concatenated and then reduced back to the original input dimension d_{model} using another linear projection, $W^O \in \mathbb{R}^{hd_v \times d_{model}}$, producing a total output of $N \times d_{model}$ for the input. Figure A.9 illustrates this approach with 4 self-attention heads. The multi-head layer replaces the single self-attention layer

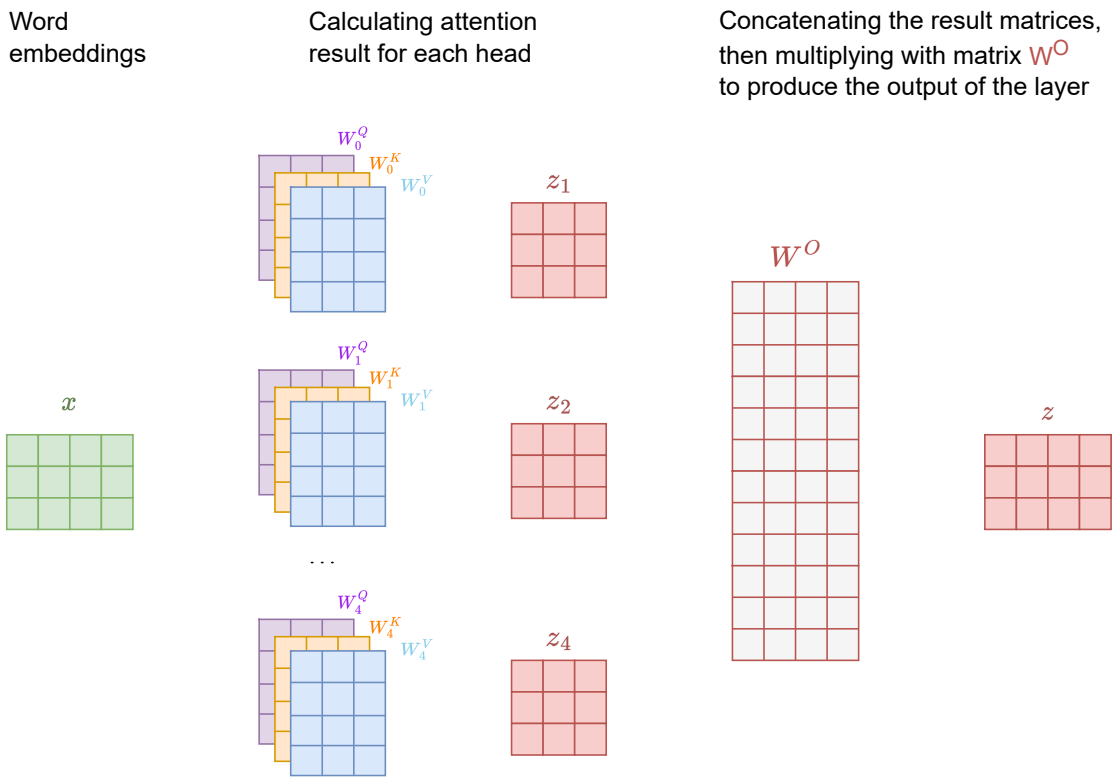


Figure A.9: The multi-head self-attention layers in the figure are equipped with the unique query, key, and value matrices. The outputs from each head are concatenated and then projected to d_{model}, resulting in an output that matches the input size.

as follows:

$$\text{MultiHeadAttention}(x) = (head_1 \oplus head_2 \cdots \oplus head_h)W^O \tag{A.12}$$

$$head_i = \text{SelfAttention}(Q_i, K_i, V_i) \tag{A.13}$$

$$Q_i = xW_i^Q; \quad K_i = xW_i^K; \quad V_i = xW_i^V \tag{A.14}$$

A.6.1.3 Transformer Block

The multi-head attention layer serves as the central component of a transformer block, which encompasses various other elements. Alongside the multi-head attention layer, a transformer block incorporates additional feed-forward layers, residual connections, and normalizing layers. These components are designed to maintain

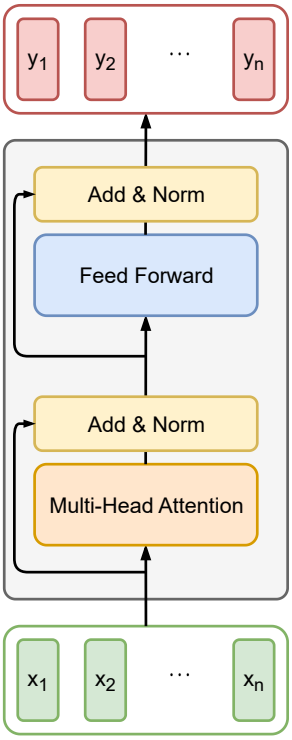


Figure A.10: A transformer block illustrating all the sub-layers

consistent input and output dimensions, enabling the stacking of multiple transformer blocks.

In Figure A.10, a standard transformer block is illustrated. It comprises a multi-head attention layer followed by a fully connected feed-forward layer, both of which are accompanied by residual connections and layer normalization. This layer operates on each position individually. It involves two linear transformations with a ReLU activation function in between:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{A.15}$$

Although the linear transformations remain consistent across different positions, they employ distinct parameters from one layer to another.

Residual connections are a valuable component in the transformer architecture. They allow information to bypass intermediate layers, creating direct connections from

lower to higher layers. This mechanism enhances learning by granting higher-level layers access to important information from lower levels [29]. In Figure A.10, you can see that residual connections are employed in both the attention and feed-forward sublayers. The summed vectors are then normalized using layer normalization to ensure stability and effective training [11]. The function computed within a transformer block can be expressed as:

$$z = \text{LayerNorm}(x + \text{MultiHeadAttention}(x)) \tag{A.16}$$

$$y = \text{LayerNorm}(z + \text{FFN}(z)) \tag{A.17}$$

Layer normalization, a technique used in deep neural networks, aims to improve training performance by keeping the values of a hidden layer within an optimal range for gradient-based training. It is a variation of the standard score (z-score) from statistics applied to a single hidden layer. To perform layer normalization, the mean (μ) and standard deviation (σ) of the vector to be normalized are calculated. For a hidden layer with dimensionality d_h, these values are computed as follows:

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i \tag{A.18}$$

$$\sigma = \sqrt{\frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2} \tag{A.19}$$

After obtaining the mean (μ) and standard deviation (σ), the vector components are normalized by subtracting the mean from each element and dividing it by the standard deviation. This normalization process results in a new vector, $\hat{x}$, with a mean of zero and a standard deviation of one. Mathematically, it is expressed as:

$$\hat{x} = \frac{(x - \mu)}{\sigma} \tag{A.20}$$

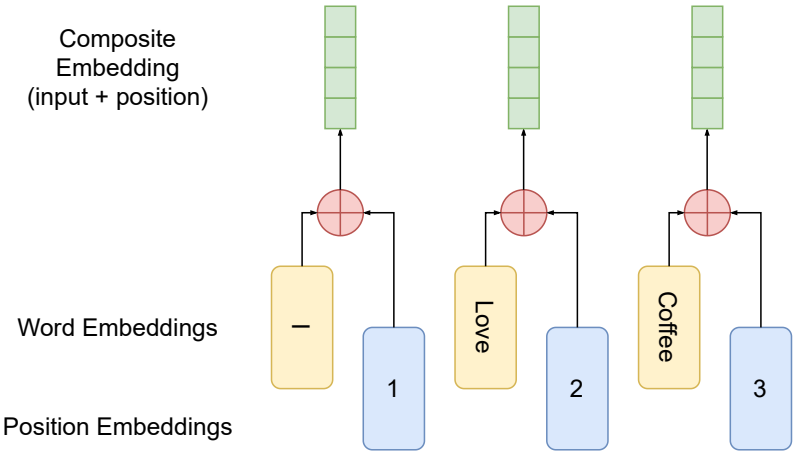


Figure A.11: Introducing positional embeddings by incorporating absolute position representations into the input word embeddings, resulting in a new embedding of equal dimensionality.

In the standard implementation of layer normalization, two learnable parameters, γ and β, are introduced to control the gain and offset of the normalized vector $\hat{x}$. These parameters allow for fine-tuning the normalization process. The layer normalization function can then be expressed as:

$$\text{LayerNorm} = \gamma\hat{x} + \beta \tag{A.21}$$

A.6.1.4 Positional Embeddings

Unlike recurrent neural networks (RNNs), which inherently capture the order of input sequences, transformers do not possess any inherent notion of input order. The transformer model, as presented so far, lacks the ability to understand the relative or absolute positions of tokens in the input. This means that even if the order of the input sequence is shuffled during the attention computation (as depicted in Figure A.7), the resulting output would remain unchanged.

One straightforward approach is to enhance the input embeddings by incorporating positional embeddings specific to each position in the input sequence. A common method involves initializing embeddings for each possible input position up to a de-

fined maximum length. For example, if there is an embedding for the word "coffee," there would also be an embedding for position 3. Similar to word embeddings, these positional embeddings are learned during training alongside other parameters. To generate an embedding that captures positional information, each word is combined with its corresponding positional embedding using summation. This resultant embedding then serves as the input for subsequent processing steps. The concept is illustrated in Figure A.11.

One potential issue with the simple absolute position embedding approach is the unequal distribution of training examples across input positions. This can result in poorer training for embeddings corresponding to positions at the outer length limits, leading to potential challenges in generalization during testing. Another approach to address positional information is to use a static function that maps integer inputs to real-valued vectors, capturing the inherent relationships among positions. The original transformer work employed a combination of sine and cosine functions with different frequencies for this purpose. Improving position representations is an ongoing research focus in the field.

A.6.1.5 Pre-training Bidirectional Encoders

Language models typically predict the next word in a given sequence. However, in the case of bidirectional encoders, this task becomes straightforward as the context provides direct access to the answer, and the attention layer allows previous words to gather information about the next word. As a result, a new training approach is required. Fortunately, the traditional learning objective offers a solution for pre-training bidirectional encoders. Instead of focusing on predicting the next word, the model is trained to perform a fill-in-the-blank task known as the **cloze task**.

During the learning task, the model is presented with an input sequence that contains one or more missing elements. The objective is to predict these missing elements.

Specifically, during training, the model is deprived of certain elements from the input sequence and tasked with generating a probability distribution over the vocabulary for each missing item. The learning process is driven by utilizing cross-entropy loss, which measures the discrepancy between the model's predictions and the ground truth.

A.6.1.6 Masking Words

The training approach employed for bidirectional encoders is known as **Masked Language Modeling** (MLM) [21]. Similar to other language model training methods, MLM utilizes unannotated text from a large corpus. During the training process, the model is presented with sentences from the training corpus, where a random selection of tokens from each sequence is chosen for the learning task. These selected tokens are then handled in one of the following three ways:

- They are replaced with a special vocabulary token, denoted as `[MASK]`.
- They are substituted with another token from the vocabulary, randomly sampled based on token unigram probabilities.
- They are left unchanged.

In BERT, a training sequence typically involves sampling 15% of the input tokens for the learning task. Within this subset, approximately 80% of the tokens are replaced with the special token `[MASK]`, 10% of the tokens are replaced with randomly selected tokens from the vocabulary, and the remaining 10% of the tokens are left unchanged. The goal of MLM training is to predict the original inputs for the masked tokens using a bidirectional encoder, as discussed in earlier sections. The training process is driven by the cross-entropy loss computed from these predictions, which updates all the parameters in the model. It is important to note that while all input tokens are involved in the self-attention process, only the sampled tokens are used for learning.

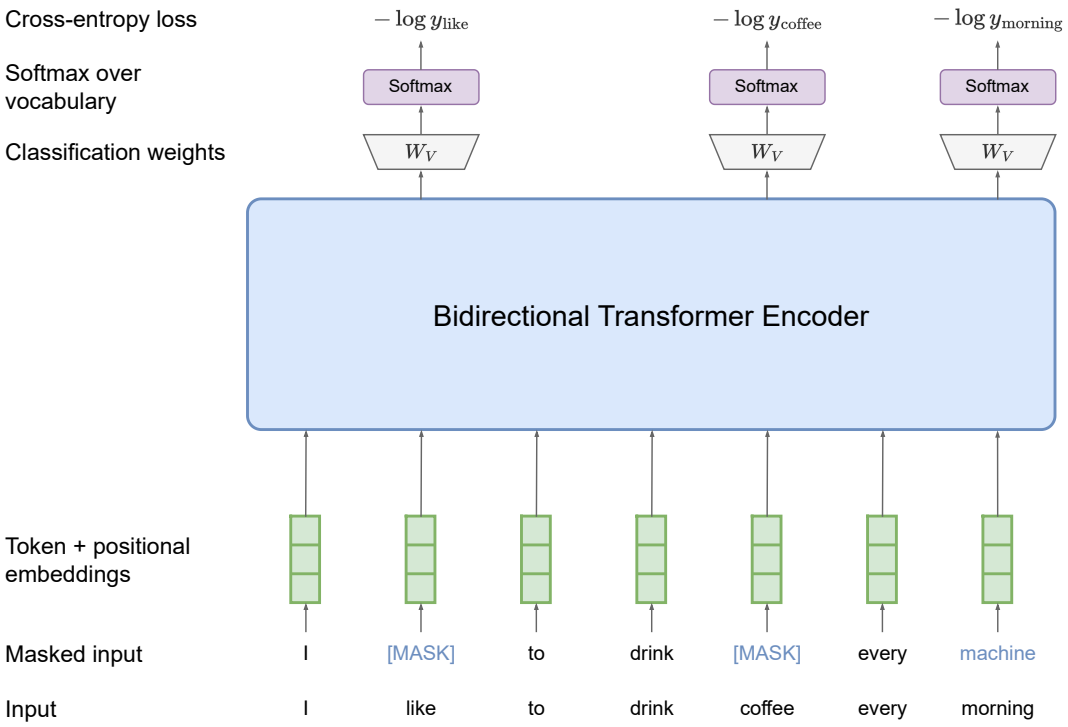


Figure A.12: The figure illustrates the sampling of three input tokens, where two of them are masked, and the third one is substituted with an unrelated word. The assigned probabilities of these tokens by the model are utilized to compute the loss.

To begin with, the original input sequence undergoes tokenization using a subword model. The sampled items, which are crucial for the learning process, are selected from the set of tokenized inputs. Word embeddings for all tokens in the input are extracted from the word embedding matrix. These word embeddings are then combined with positional embeddings to create the input for the transformer model. Figure A.12 illustrates this approach using a sample. In this example, the tokens "like," "coffee," and "morning" are sampled from the training sequence. The first two tokens are masked, while the third token is randomly substituted with a different token. These sampled tokens are then processed through a stack of bidirectional transformer blocks. The resulting output vectors from the last layer for these three tokens are multiplied by a classification weight matrix $W_V \in \mathbb{R}^{|V| \times d_h}$, where $|V|$ is the number of tokens in the vocabulary and d_h is the dimension of the vectors. Finally, a softmax function is applied to generate the required predictions over the

vocabulary.

$$y_i = \text{softmax}(W_V h_i) \tag{A.22}$$

After obtaining the predicted probability distribution for each masked token, the cross-entropy loss is calculated to measure the negative log probability, as depicted in Figure A.12. The gradients, which serve as the basis for weight updates, are derived from the average loss over the sampled learning items from either a single training sequence or a batch of sequences.

A.6.1.7 Next Sentence Prediction

While MLM primarily focuses on predicting missing words within surrounding contexts to create effective word-level representations, many NLP applications require understanding the relationships between pairs of sentences, which is not directly captured by MLM. These tasks encompass paraphrase detection, entailment, and discourse coherence.

To capture the relationship between sentences, BERT introduced a learning objective called **Next Sentence Prediction** (NSP) that can be easily derived from any monolingual corpus. In this task, pairs of sentences are presented to the model, and its goal is to predict whether each pair consists of a consecutive pair of sentences from the training corpus or a pair of unrelated sentences. For each pre-training example, the sentences `A` and `B` follow a specific pattern. In 50% of the cases, `B` is indeed the next sentence that follows `A` (labeled as `IsNext`). In the remaining 50% of the cases, `B` is a random sentence from the corpus (labeled as `NotNext`).

To perform the Next Sentence Prediction (NSP) task, BERT introduces two special tokens into the input representation. After tokenizing the input using the subword model, the special classification token `[CLS]` is added to the beginning of the sentence pair. It's worth noting that in both the MLM and NSP tasks, the first token of every sequence is always the `[CLS]` token. The final hidden state corresponding

to this token, denoted as $C \in \mathbb{R}^{d_h}$, serves as the aggregated representation of the entire sequence for classification purposes. To differentiate between the sentences, two modifications are made. First, a special token `[SEP]` is inserted between the sentences and after the final token of the second sentence. Second, alongside the positional embeddings, a learned embedding is appended to each token indicating whether it belongs to sentence `A` or sentence `B`.

During training, the output vector obtained from the final layer corresponding to the `[CLS]` token is utilized for the next sentence prediction task. Similar to the Masked Language Modeling (MLM) task, a set of learnable classification weights $W_{NSP} \in \mathbb{R}^{2 \times d_h}$ is employed to generate a binary prediction based on the raw `[CLS]` vector:

$$y = \text{softmax}(W_{NSP} h_{\texttt{[CLS]}}) \tag{A.23}$$

The NSP loss is calculated using cross-entropy for each sentence pair fed to the model. Figure A.13 provides an overview of the NSP training configuration. In BERT, the NSP loss is combined with the MLM training objective to form the overall loss.

A.6.1.8 Contextual Embeddings

Using a pre-trained language model and a new input sentence, the model can generate contextual embeddings for each token in the input. These embeddings capture the contextual meaning of the words and can be employed as contextual representations in downstream tasks that rely on word semantics.

Contextual embeddings are vector representations that capture the meaning of a token within the context of the entire sentence. As a result, each token can have a distinct vector representation depending on the surrounding context. For instance, in a sequence of input tokens $x_1, \ldots, x_n$, the output vector y_i corresponds to the representation of token x_i in the context of the sentence $x_1, \ldots, x_n$, obtained from the final layer of the model.

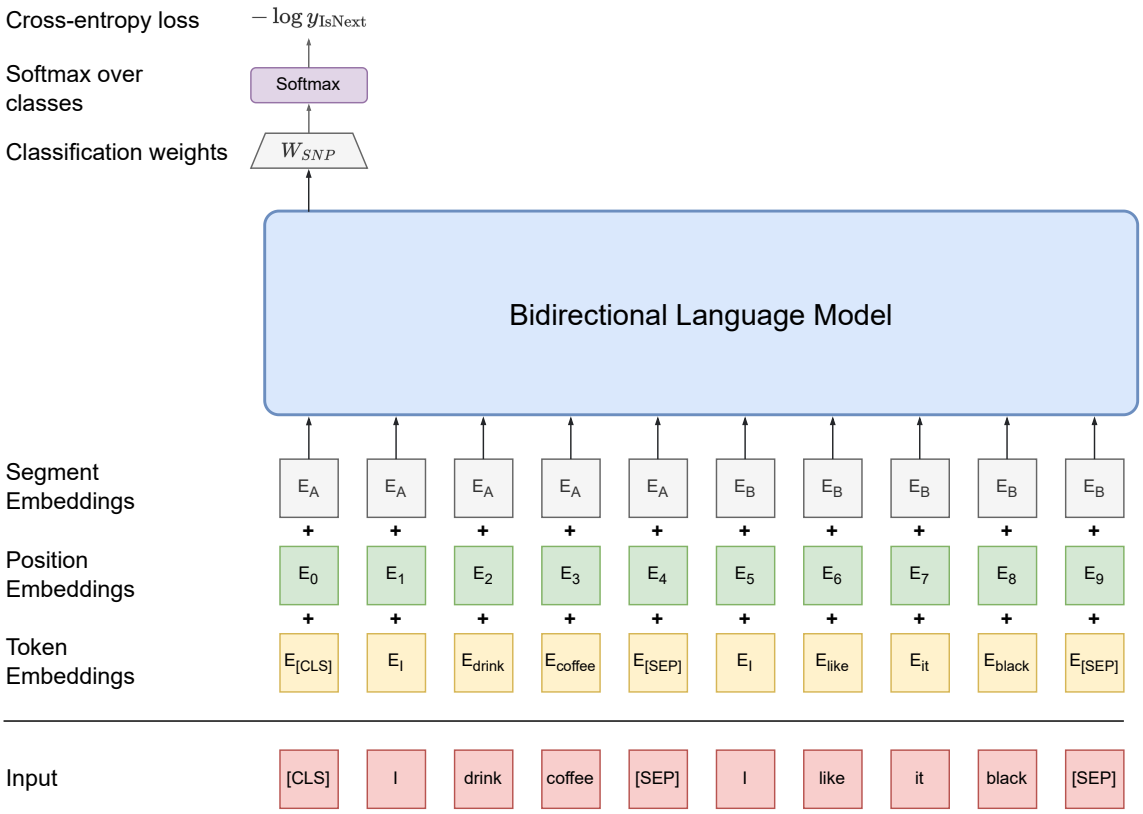


Figure A.13: Illustration of the Next Sentence Prediction (NSP) task. The [CLS] token serves as the representation of the input text. Alongside positional embeddings, token embeddings are incorporated to distinguish between the two sentences.

In addition to capturing contextual word embeddings, BERT is capable of generating a fixed representation vector for a sequence of tokens. The output vector from the final layer, corresponding to the `[CLS]` token, serves as a contextualized representation of the entire input sequence. During pre-training, the model learns to encode the complete representation of the input sequence into this token. This vector proves valuable in tasks that necessitate a consistent input representation regardless of the number of input tokens, such as semantic search. Semantic search aims to determine the similarity between two texts of varying lengths. For instance, consider two sentences: $x_1, \ldots, x_n$ and $x'_1, \ldots, x'_m$. If these sentences share a similar context, the distance between the vectors corresponding to the `[CLS]` token will be small; otherwise, the distance will be high.

A.6.1.9 Fine-Tuning the Bidirectional Encoder

Pre-trained language models have acquired extensive knowledge during pre-training from a large volume of data. This knowledge enables them to generalize well, making them valuable for various downstream tasks. To apply this knowledge to specific applications, a process called **fine-tuning** is used. Fine-tuning involves building task-specific applications on top of pre-trained models and adjusting a small set of application-specific parameters. This process requires labeled data specific to the target domain for further training. Typically, the pre-trained language model parameters are either frozen or minimally adjusted during this additional training phase.

Text classification is a commonly used application in NLP. In this task, an input sequence is transformed into a compact representation, and a model processes this vector to assign a label. In traditional RNN models, the last hidden state associated with the final element represents the entire sequence and is often referred to as the **sentence embedding**. In BERT, the special token `[CLS]` is used as the sentence

embedding. This token is added to the vocabulary and placed at the beginning of all input sequences. The output vector from the final layer of the model corresponding to the `[CLS]` input serves as the representation of the entire sequence. This vector is then fed into a **classifier head**, which can be a logistic regression or a neural network classifier, to make predictions.

An example of a text classification task is fact verification, which aims to determine the truthfulness of a claim. A simple fine-tuning approach for this task involves training a set of weights, denoted as W_C, that maps the output vector of the `[CLS]` token, represented as $h_{\texttt{[CLS]}}$, to a probability distribution over the classes. Assuming a binary classification scenario (true or false) and a hidden layer size of d_h for the pre-trained language model, the weight matrix W_C has dimensions of $\mathbb{R}^{2\times d_h}$. To classify unseen documents, the input text is passed through the pre-trained language model to obtain $h_{\texttt{[CLS]}}$, which is then multiplied by W_C. A softmax function is applied to the result to obtain the predicted probabilities for the classes:

$$y = \text{softmax}(W_C h_{\texttt{[CLS]}}) \tag{A.24}$$

To fine-tune the weights in W_C, supervised training data is required, which consists of labeled input sequences paired with their corresponding classes. Let $X = \{x_1, \ldots, x_n\}$ denote the input examples, and $Y = \{y_1, \ldots, y_n\}$ represent their associated labels. During training, the process is similar to training other neural networks. The cross-entropy loss is computed between the softmax output and the correct answer, serving as the driving force for updating the weights in W_C.

It is worth noting that the loss function can be employed not only to train the weights of the classifier but also to update the parameters of the pre-trained language model. In practice, satisfactory classification performance can often be achieved with minimal adjustments to the language model parameters.

A.6.2 Causal Transformer Models

In the previous section, the process of pretraining bidirectional transformer language models was discussed, showcasing their usefulness as a tool for a wide range of NLP tasks through the utilization of MLM and NSP objectives. This chapter introduces the second type of transformer model, known as the causal or left-to-right model, which allows for the generation of contextualized texts. Furthermore, the generation of coherent and meaningful sequences of words is explored, highlighting the potential applications of these models in natural language processing.

A.6.2.1 Decoder Architecture

This sub-section introduces the transformer decoder, which serves as the foundation for models like GPT [13]. While the previous section delved into the bidirectional transformer encoder, it is important to note that these encoder models are not inherently designed for text generation. Their primary focus lies in learning deep contextualized representations through methods such as MLM. As a result, they excel in tasks like sentiment analysis, question-answering, and named entity recognition. However, when it comes to generation tasks such as contextual generation, summarization, and machine translation, their effectiveness is limited. In contrast, decoder models are specifically trained using a unidirectional (causal) language modeling objective. This training approach enables them to generate text sequentially from left to right by predicting the next word based on the preceding context.

Figure A.6 illustrates how bidirectional encoders enable the self-attention mechanism to consider the entire input. In contrast, the focus of the decoder transformer is on predicting the next word. Like the encoder, the decoder employs the self-attention mechanism to map the input embeddings $(x_1, \ldots, x_n)$ to corresponding output embeddings $(y_1, \ldots, y_n)$ of the same length. However, unlike the bidirectional encoder, the decoder self-attention operates in a left-to-right manner, where each word only

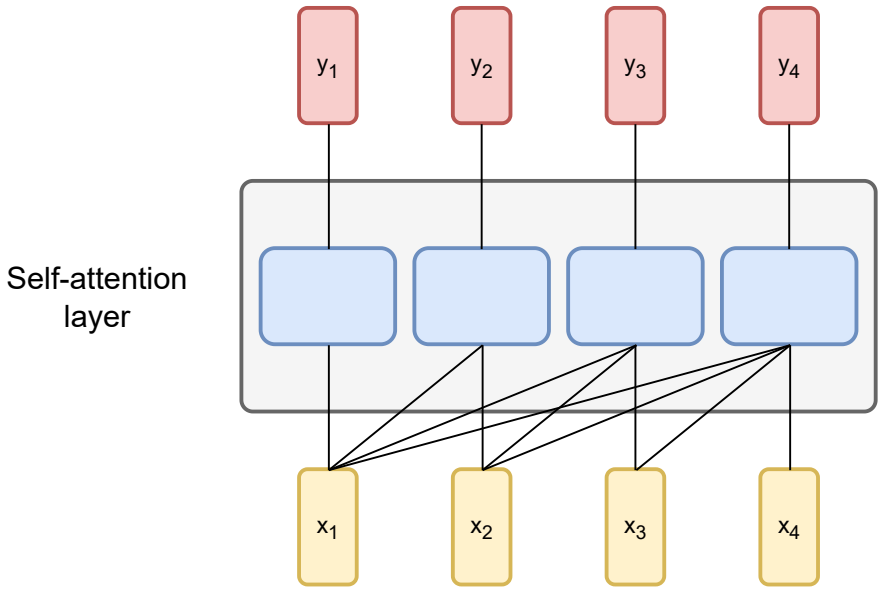


Figure A.14: The figure illustrates the information flow in the self-attention mechanism of a causal model. When processing each element of the sequence, the model attends to all the preceding inputs, including the current one. This enables the model to incorporate information from the left context while making predictions, ensuring that each element is influenced by its relevant context within the sequence.

has access to information from preceding words. The flow of information in a decoder self-attention layer is depicted in Figure A.14.

The prediction of the next word is achieved through the utilization of the self-attention mechanism, which is also employed in encoder models. This process involves generating key, query, and value embeddings for each token of the input vector x using learned weight matrices W^Q, W^K, and W^V. These weight matrices enable the projection of each input vector x_i into its respective role as a key, query, or value:

$$q_i = x_i W^Q, \quad k_i = x_i W^K, \quad v_i = x_i W^V \tag{A.25}$$

To obtain the output vector y_i corresponding to each input element x_i, a weighted sum of the input value vectors v is computed, considering all the preceding elements up to index i:

$$y_i = \sum_{j \leq i} \alpha_{ij} v_j \tag{A.26}$$

The weights α are determined through a softmax operation applied to the comparison scores calculated using dot products between each element of the input sequence treated as a query and every other element as a key. This process ensures that the weights reflect the relevance or importance of each element in relation to others. The computation of the weights and comparison scores can be defined as follows:

$$\alpha_{ij} = \frac{\exp(\text{score}(x_i, x_j))}{\sum_{k=1}^{n} \exp(\text{score}(x_i, x_k))} \tag{A.27}$$

$$\text{score}(x_i, x_j) = q_i \cdot k_j \tag{A.28}$$

By employing the dot product, the score captures the similarity or correlation between the query q_i and key k_j embeddings. The softmax function then normalizes the scores to obtain the weight α_{ij}, representing the relative importance of the j-th element in relation to the i-th element.

To enable efficient parallel processing of the entire sequence, the input embeddings x_i are packed into a matrix $x \in \mathbb{R}^{N \times d_h}$. This matrix representation allows for the simultaneous computation of key, query, and value vectors through matrix multiplication with weight matrices W^Q, W^K, and W^V, respectively. The resulting matrices $Q \in \mathbb{R}^{N \times d_k}$, $K \in \mathbb{R}^{N \times d_k}$, and $V \in \mathbb{R}^{N \times d_k}$ encapsulate all the key, query, and value information for the entire sequence, optimizing computational efficiency. This matrix operation can be expressed as:

$$Q = xW^Q; \quad K = xW^K; \quad V = xW^V \tag{A.29}$$

Given these matrices, the necessary query-key comparisons are computed in a single operation by multiplying Q and K^T. While the computations for the transformer encoder and decoder are similar, there is a distinction when it comes to the comparison matrix QK^T. In the context of language modeling, it is crucial to prevent the model from having access to future words when predicting the next word. To

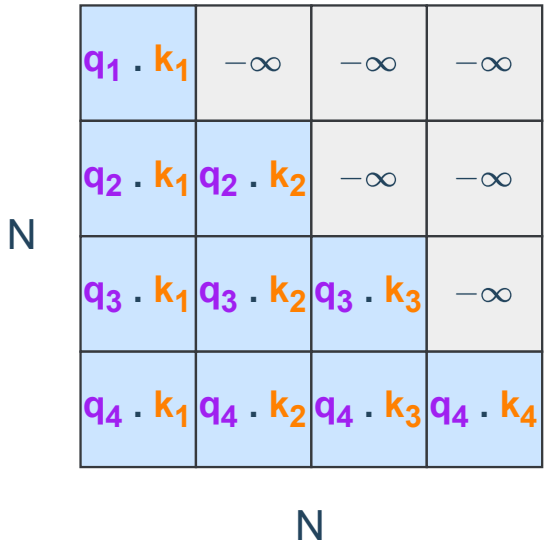


Figure A.15: The masked-out matrix QK^T is an $N \times N$ matrix that represents the dot product values $q_i \cdot k_j$ between each query-key pair. To ensure proper attention computation, the upper-triangle portion of the comparison matrix is set to $-\infty$, resulting in zeros after applying the softmax function.

achieve this, the upper-triangular portion of the matrix is modified by setting its elements to infinity. This effectively masks out the knowledge of words that follow in the sequence, ensuring that the model's predictions focus solely on the preceding context. Figure A.15 illustrates the modified QK^T matrix with the masked-out upper-triangular portion.

Finally, the masked scores are scaled, passed through a softmax function, and then multiplied by V, producing a matrix of shape $N \times d_k$ where each row represents a contextualized output embedding for each preceding token along with the token itself in the input.

$$\text{SelfAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{A.30}$$

In addition to this simple modification, the remaining components of the transformer architecture remain consistent for the decoder models. The input sequences undergo subword tokenization and are then combined with positional embeddings. These

embeddings, along with the tokenized inputs, are processed through a sequence of standard transformer blocks. These blocks consist of self-attention and feed-forward layers, enhanced by the inclusion of residual connections and layer normalization. The overall structure of the decoder is illustrated in Figure A.10.

A.6.2.2 Transformers for Language Modeling

Language models predict the next word in a sequence based on the context that precedes it. They accomplish this by assigning conditional probabilities to each potential next word, resulting in a probability distribution across the entire vocabulary. By applying the chain rule, these models can further estimate the probabilities of entire sequences by combining the conditional probabilities of each word in the sequence:

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{<i}) \tag{A.31}$$

One effective way to deploy a language model is through self-supervised learning using a transformer decoder. In self-supervised learning, the training data does not provide explicit labels but relies on the inherent structure of the input. The model is trained to predict the next token in a sequence y_t using the cross-entropy loss as the training objective. The cross-entropy loss measures the discrepancy between the predicted token $\hat{y}_t$ and the actual token y_t. By optimizing this loss function, the language model learns to capture patterns and dependencies in the training corpus, enabling it to generate coherent and contextually appropriate sequences of text.

$$\mathcal{L}_{CE}(\hat{y}_t, y_t) = -\log \hat{y}_t[w_{t+1}] \tag{A.32}$$

The training approach employed is known as teacher forcing. It involves using the correct target token from the training data as the next input at each decoding time step (x_{t+1}) instead of relying solely on the decoder's output ($\hat{y}_t$). By incorporat-

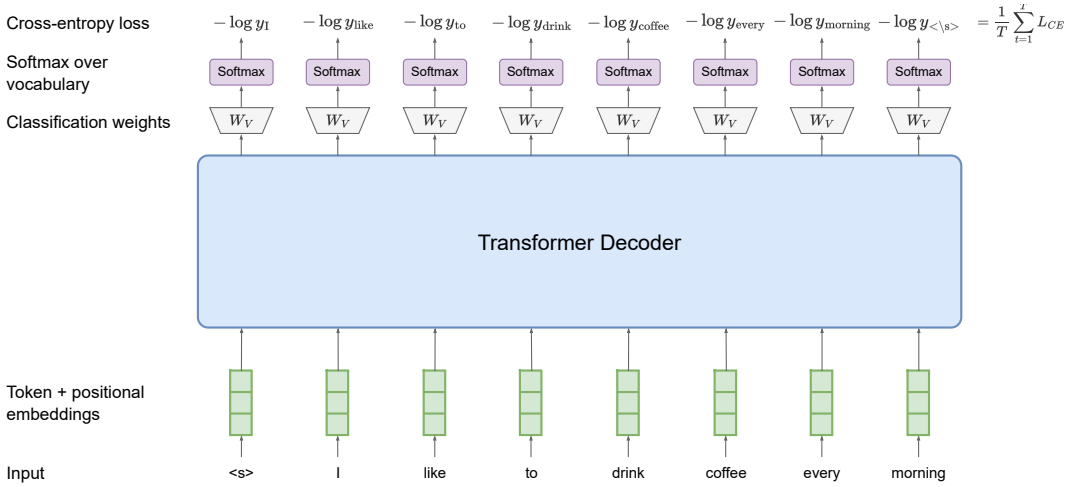


Figure A.16: Training a transformer decoder as a language model. The special tokens $< s >$ and $< \backslash s >$ represent the start and end of the input sequence, respectively.

ing this gold-standard information during training, the model receives more precise guidance, allowing it to learn the associations between input and output sequences effectively. This technique helps improve the quality of generated outputs and enhances the decoder's overall performance.

Figure A.16 provides an overview of the training process. To begin, the sequence is modified by adding special tokens at the start and end, representing the span of the sequence. The input to the model is then formed as $(< s >, x_1, \ldots, x_{n-1})$, while the model output is a probability distribution over the entire vocabulary. During training, the probabilities assigned to the $(x_2, \ldots, x_n, < \backslash s >)$ tokens are employed to compute the cross-entropy for the entire input sequence. It is important to note that the decoder's self-attention operates in a unidirectional manner, meaning the probability for each token t is determined based on the preceding tokens.

Once trained, the language model has the capability to generate new text in an autoregressive fashion. This approach, known as **autoregressive generation** or **causal LM generation**, involves incrementally generating words by sampling the next word based on the previous choices. Technically, an autoregressive model predicts the value at time t by considering a linear function of the previous values at

times $t-1$, $t-2$, and so on. Although language models are not strictly linear due to their multiple layers of non-linearities, this generation technique is colloquially referred to as autoregressive generation because each word generated at time step t is conditioned on the words selected by the network in the previous steps.

The generation process of the language model follows the following steps:

- Initiate the generation with the special token $< s >$ as the first input and sample a word from the resulting softmax distribution.
- Utilize the word embeddings of the sampled word as the input for the next time step and repeat the process of sampling the next word in a similar manner.
- Continue generating words until either the end of sentence marker $< \backslash s >$ is sampled, or a predetermined maximum length is reached.

This architecture forms the basis of advanced approaches used in applications like machine translation, summarization, and question-answering. The key to its effectiveness lies in providing the generation component with an appropriate context. Rather than starting with a basic token such as $< s >$, the model is given a more relevant and task-specific context. For translation, this context is the source language sentence, while for summarization, it is the text to be summarized. This context enhances the model's ability to generate accurate and contextually appropriate outputs.

In the process of generating text from a trained language model, each step in decoding involves selecting the output y_t by applying a softmax function over the available options in the vocabulary and choosing the token with the highest probability. Mathematically, this can be expressed as:

$$\hat{y}t = \text{argmax}_{w \in V} P(w|y_1, \ldots, y_{t-1}) \tag{A.33}$$

The approach of choosing the most probable token at each step, known as greedy decoding, focuses on making locally optimal decisions without considering their long-term consequences. It is a simple and efficient strategy that prioritizes immediate optimization without extensive exploration of alternative options.

Text generation is a prominent application of neural language models that has greatly influenced the field of NLP. This, along with image and code generation, has given rise to a new domain in AI known as generative AI. These advancements in language models have opened up possibilities for creating realistic and creative text through the power of neural networks.

A.7 In-Context Learning

In recent years, fine-tuning pre-trained transformer language models has revolutionized NLP applications, eliminating the necessity for task-specific architectures. This paradigm has significantly advanced various challenging NLP tasks, including text classification, summarization, machine translation, and question answering. Nevertheless, a significant drawback of this approach is its reliance on task-specific datasets and fine-tuning, even though the underlying architecture is agnostic to the task. Achieving robust generalization for specific domains often demands extensive datasets containing thousands or even hundreds of thousands of task-specific examples. Overcoming this limitation would be highly desirable for multiple reasons [13].

First, from a practical standpoint, acquiring labeled datasets for every new task can be costly and time-consuming, which limits the applicability of language models. With a wide range of potential language tasks, it becomes challenging to gather large supervised training data, particularly when having to repeat the process for each new task.

Second, despite the high flexibility of language models in their representations, the training data often exhibits a limited distribution and lacks diversity. This issue can result in the model relying on coincidental patterns in the data rather than truly grasping the underlying semantics. Consequently, the model may struggle to generalize effectively, leading to less reliable performance on real-world tasks. Evidence has shown that fine-tuning under this paradigm can further exacerbate the problem of poor generalization since the model becomes overly specific to the training distribution and fails to extend its understanding beyond it [92].

Third, humans do not require a large supervised dataset to learn new tasks. A simple and concise instruction in natural language, such as "Please determine if this sentence has a happy or sad tone," or even just a handful of examples, is often sufficient for a person to perform a new task. This innate adaptability allows humans to effortlessly transition between various tasks and skills, enabling them to seamlessly switch from one activity to another.

Researchers have observed significant advancements and emergent capabilities when scaling up pre-trained language models in terms of data size and model parameters. Among these capabilities is in-context learning (ICL), which was introduced by GPT-3 [13] and has the potential to address the aforementioned challenges. ICL leverages a formatted natural language prompt that includes a task description and/or a few task examples as demonstrations.

The input prompt consists of a task description followed by task-specific examples from the dataset, arranged in a carefully designed order using specific templates. The test instance is then appended to these demonstrations to form a comprehensive natural language prompt, serving as input to the large language models (LLMs) for generating the desired output. By analyzing the task description and demonstrations, LLMs can understand the task and recognize the patterns required to perform new tasks without explicit gradient updates. Figure A.17 depicts the illustration of

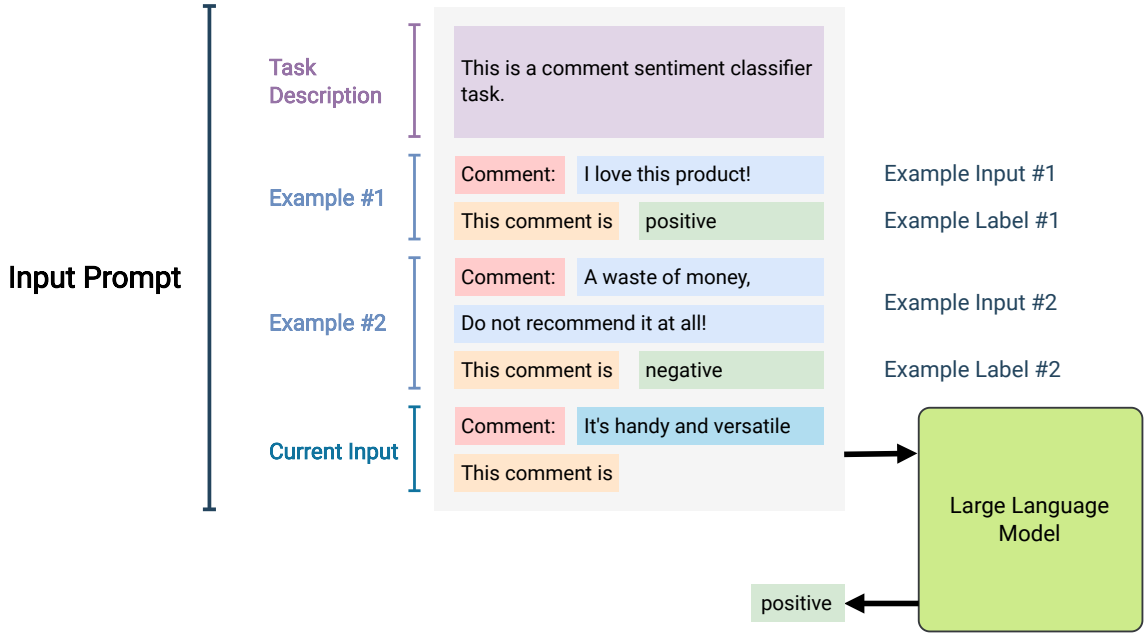


Figure A.17: Illustration of in-context learning: The input prompt, in natural language form, provides a description, multiple examples, and the test input to a large language model. The model comprehends the task and utilizes the information in the input prompt.

ICL.

In the context of ICL, let $D_k = \{f(x_1, y_1), \ldots, f(x_k, y_k)\}$ represent a set of demonstrations comprising k examples, where $f(x_i, y_i)$ is a function that maps the i-th task example to a natural language prompt. Given the task description I, the demonstration set D_k, and a test input query x_{k+1}, the LLMs' output prediction $\hat{y}_{k+1}$ can be formulated as follows:

$$\text{LLM}(I, \underbrace{f(x_1, y_1), \ldots, f(x_k, y_k)}_{\text{demonstrations}}, f(\underbrace{x_{k+1}}_{\text{test}}, \underbrace{\_\_\_\_\_}_{\text{answer}})) \rightarrow \hat{y}_{k+1} \tag{A.34}$$

Here, the actual answer y_{k+1} is left blank to be predicted by the LLM. As the performance of ICL is heavily influenced by the demonstrations, it is crucial to thoughtfully design them in the prompts. The construction process in Equation A.34 revolves around three key aspects: selecting appropriate examples for demonstrations, formatting each example using the function $f(\cdot)$ to fit the prompt, and arranging the

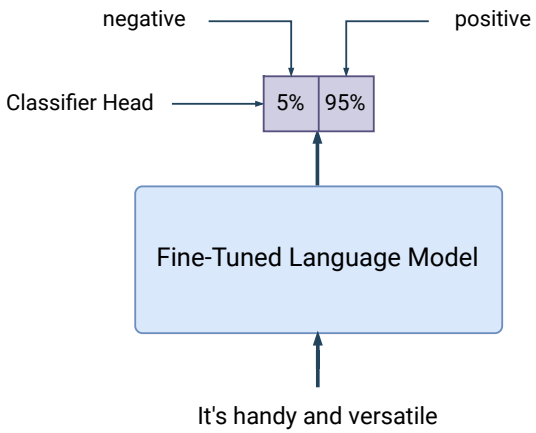


Figure A.18: In a binary classification scenario, the fine-tuned model's output is directed to a dedicated classifier that transforms it into a probability distribution.

demonstrations in a coherent format.

In fine-tuning, the language model head, responsible for mapping the output of a pre-trained language model into a probability distribution over the vocabulary, is substituted with a classifier head. This classifier maps the model's output to a probability distribution over the classes specific to the domain task. During training, using a domain-specific dataset, both the model and the classifier weights are updated through the gradient descent approach. After fine-tuning, when given a test sample, the model generates a vector representing a probability distribution over the classes, and the index with the highest probability corresponds to the predicted text label. Figure A.18 illustrates a binary classification model with a probability distribution assigned to the domain task.

On the other hand, In-Context Learning (ICL) maintains the language model head without additional fine-tuning. The pre-trained model addresses a new task as a text generation task, producing tokens until it reaches a special token indicating the end of the sequence. Instead of a probability distribution over the classes, the model output is a probability distribution over tokens, essentially generating the label in natural language form. An advantage of this approach is that a single model can effectively tackle multiple problems, eliminating the need to fine-tune several

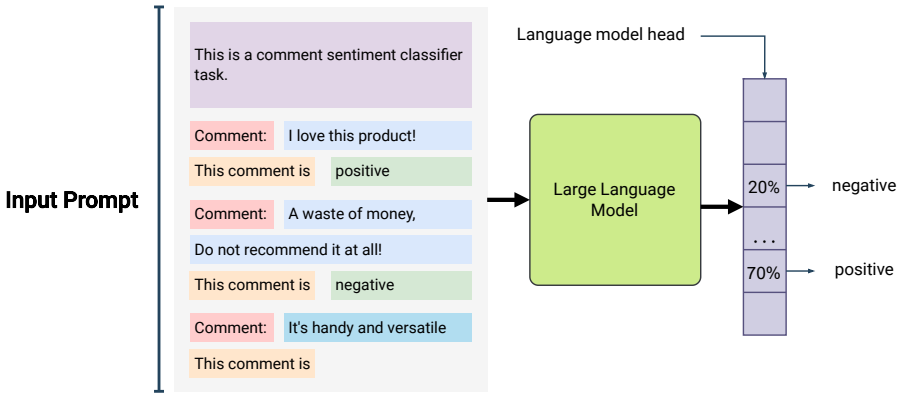


Figure A.19: Task-solving using ICL. The language model head is preserved, allowing the model to generate a new token in natural language form to solve the new task.

models for different tasks. Figure A.19 demonstrates the ICL process, where the large language model generates the label in the natural language format.

There are three distinct settings based on the number of demonstrations present in the input prompt [13]:

- **Few-Shot**: In this setting, the model receives a few task demonstrations during inference, without any weight updates applied. Figure A.20 illustrates that for a typical dataset, an example comprises a context and a desired completion, like an English sentence as the context and its French translation as the desired completion. For few-shot learning, K examples of context and completion are provided, followed by a final example of context, where the model is expected to provide the completion. Typically, K is set between 10 to 100, limited by how many examples can fit within the model's context window. The main advantages of few-shot learning include a significant reduction in the need for task-specific data and a lesser potential to overfit to a narrow distribution from a large but narrow fine-tuning dataset. However, results from this method have generally been less favorable compared to state-of-the-art fine-tuned models. Additionally, a small amount of task-specific data is still required.

- **One-Shot**: The one-shot setting is similar to few-shot, with the exception that

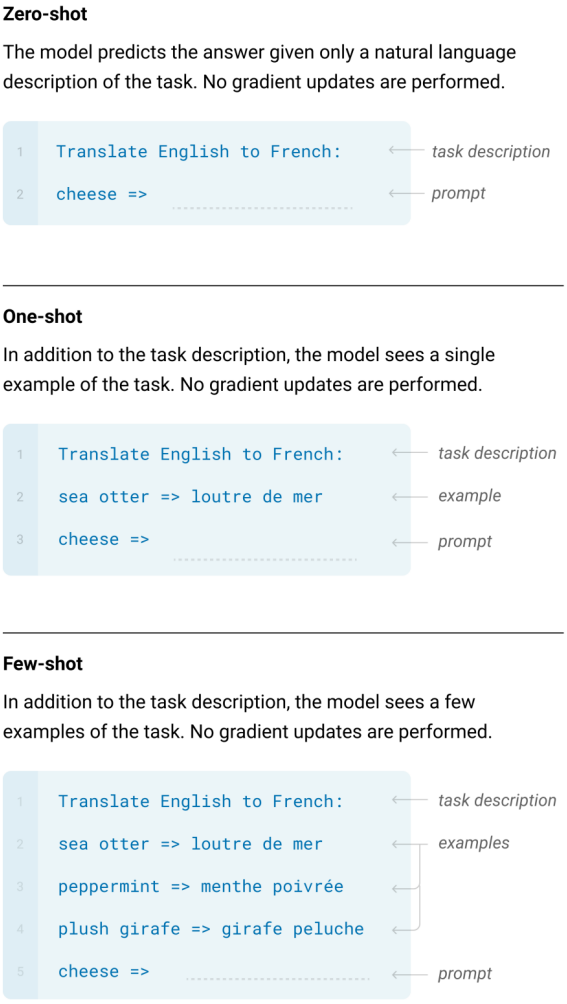


Figure A.20: Zero-shot, one-shot, and few-shot settings: This illustration showcases three distinct settings for conducting ICL with a language model to solve a task. In the zero-shot, one-shot, and few-shot scenarios, the model is required to perform the task using only forward passes during test time [13].

only one demonstration is allowed, along with a natural description of the task, as shown in Figure A.20. This distinction is made because it closely mirrors how some tasks are conveyed to humans. For instance, when humans are asked to generate a dataset on human worker services, it is common to provide one example of the task. The presence of a single demonstration allows for a better understanding of the task's content or format, making it more comprehensible than a zero-shot scenario.

- **Zero-Shot**: In this setting, the model is given only a natural language instruction describing the task, without any demonstrations allowed. Figure A.20 depicts this scenario, and it offers maximum convenience and potential for robustness, while also avoiding spurious correlations unless they are widely present across the large pre-training corpus. However, it is also the most challenging setting as it lacks any direct task-specific guidance. Humans may also struggle with understanding the task format without prior examples. Nevertheless, in some cases, zero-shot is the closest representation of how humans perform tasks. For instance, in the translation example shown in Figure A.20, a human may know what to do based solely on the text instruction.

A.8 Chain of Thought

Since the ICL method uses natural language prompts and does not need any changes in the model's architecture and weights, designing these templates brings flexibility, and one can manipulate the elements of the input template. Hence, researchers improvised a new method called **Chain-of-Thought** (CoT) [90], which is an improved prompting technique to boost the performance of LLMs on complex reasoning tasks, such as arithmetic reasoning, commonsense reasoning, and symbolic reasoning. Basically, CoT is a series of short sentences that mimic the reasoning process a person

might have when responding to a question.

LLMs have shown impressive performance on NLP tasks like text classification, even in few-shot and zero-shot settings through ICL. One way to interpret these findings is that LLMs excel at *system-1* tasks, which are tasks that humans handle quickly and intuitively. However, they encounter challenges with *system-2* tasks involving slow and deliberate thinking, such as logical reasoning or complex mathematical tasks. Even when LLMs are scaled up with hundreds of billions of parameters, they still struggle with tasks falling under this category. In other words, for many of these tasks, increasing the model size does not lead to significant performance improvements.

The CoT (Chain of Thought) is a method aimed at enhancing language models' reasoning abilities. Inspired by how humans approach multi-step reasoning problems, the idea is to enable language models to generate a coherent sequence of thoughts leading to the answer. By doing so, the language models can break down complex problems into intermediate steps, solving them individually instead of attempting to tackle the entire multi-hop problem in a single pass. This chain of thought approach proves beneficial for reasoning tasks that go beyond the capabilities of standard few-shot prompting. When combined with language models at scale, CoT prompting achieves remarkable improvements in performance across various reasoning tasks.

Imagine how one approaches a complex reasoning task, like a multi-step math word problem. Usually, the problem is broken down into intermediate steps, and each step is solved before arriving at the final answer. For instance, consider the problem: *"After Sarah buys 4 apples, she has 12... then after she gives 2 apples to her brother, she will have 10... so the answer is 10."* Figure A.21 illustrates an example of a model generating a chain of thought to solve a math word problem correctly, which it would have failed to do with ICL.

Chain-of-thought prompting offers several appealing features for enhancing reasoning

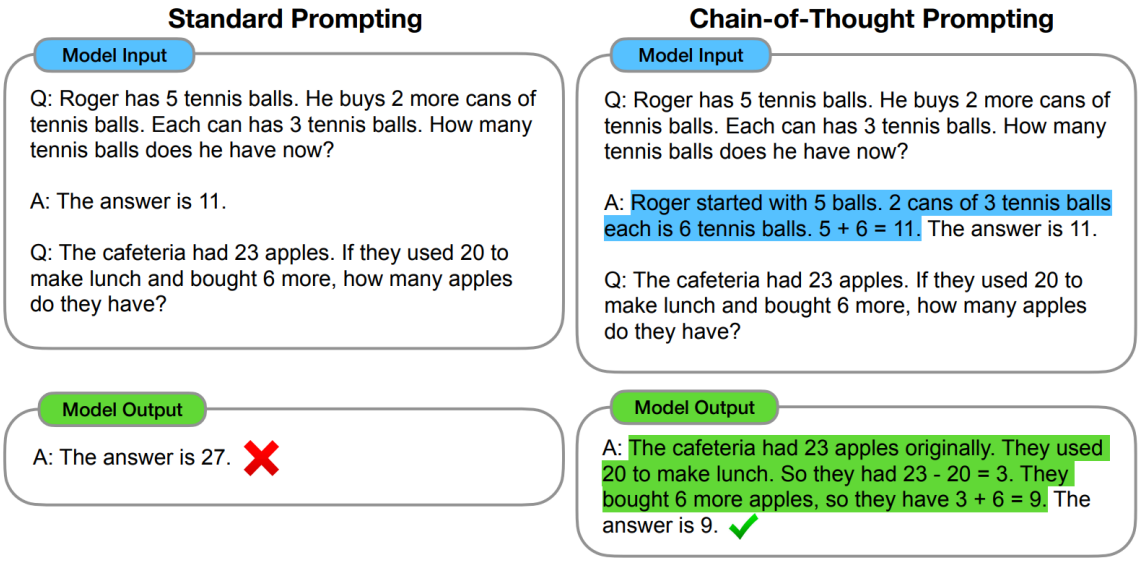


Figure A.21: The figure illustrates a comparison between chain-of-thought and in-context learning methods. While in-context learning may face challenges with complex tasks, chain-of-thought prompting empowers large language models to handle intricate arithmetic, commonsense, and symbolic reasoning tasks. The figure highlights the chain-of-thought reasoning process [90].

in language models:

1. Firstly, it enables models to break down multi-step problems into intermediate stages, allowing for additional computation allocation to tasks requiring more reasoning steps.

2. Secondly, a chain of thought provides transparency into the model's behavior, offering insights into how it arrived at a specific answer and providing opportunities to identify and debug reasoning errors (although fully understanding all computations supporting an answer remains challenging).

3. Thirdly, chain-of-thought reasoning can be applied to a wide range of tasks, including math word problems, commonsense reasoning, and symbolic manipulation, and holds the potential for addressing any task solvable through language by humans.

4. Lastly, eliciting chain-of-thought reasoning in large off-the-shelf language mod-

els is straightforward; it can be achieved by incorporating examples of chain-of-thought sequences into the few-shot prompts during training.

The CoT can be effectively utilized in two primary settings: few-shot and zero-shot, as introduced below [96].

A.8.1 Few-shot CoT

Few-shot CoT is a special form of few-show in ICL, in which each exemplar in few-shot prompting is augmented with a chain-of-thought for an associated answer, as illustrated in Figure A.21. Similar to Equation A.34, the demonstration set is $D_k = \{f(x_1, CoT_1, y_1), \ldots, f(x_k, CoT_k, y_k)\}$ where CoT_i is the reasoning steps for the set of (x_i, y_i). To implement this strategy, there are two key issues, i.e., how to design appropriate CoT prompts and how to utilize the generated CoTs for driving the final answer.

- **CoT prompt design:** Effective design of CoT prompts is crucial to elicit the complex reasoning ability of large language models (LLMs). Various approaches have been explored to enhance their performance. Using diverse CoTs, i.e., multiple reasoning paths for each problem, has been shown to be an effective strategy [47]. Additionally, prompts with more complex reasoning steps can better elicit the reasoning ability of LLMs, resulting in higher accuracy in generating correct answers [25]. However, these approaches rely on annotated CoT datasets, limiting their practical use. To overcome this limitation, Auto-CoT [95] proposes the use of zero-shot CoT, allowing LLMs to generate CoT reasoning paths without manual efforts. Auto-CoT further enhances performance by clustering questions in the training set and selecting those closest to the centroid of each cluster to represent the dataset effectively. While few-shot CoT can be seen as a special case of in-context learning (ICL), the ordering

of demonstrations appears to have a relatively small impact compared to the standard ICL prompt [90]. Furthermore, allocating the CoT before the answer in the demonstrations is crucial. Such prompts enable the model to access relevant knowledge acquired during pre-training more effectively. Conversely, an alternative configuration involves providing the CoT prompt only after the answer, allowing us to isolate whether the model truly depends on the produced CoT to arrive at the final answer. Interestingly, this variant performs similarly to the ICL setting without CoT, indicating that the sequential reasoning embedded in the CoT serves purposes beyond mere knowledge activation [90].

- **Enhanced CoT strategies:** CoT prompting not only enriches contextual information but also offers more options for inferring answers to questions. Existing studies have primarily focused on generating multiple reasoning paths and aiming to find a consensus among the derived answers [87, 88]. For example, self-consistency [87] has been proposed as a decoding strategy for CoT generation, where several reasoning paths are generated, and the final answer is determined through an ensemble of all the answers, such as selecting the most consistent answer through voting. This approach significantly boosts the performance of CoT reasoning and even improves certain tasks where CoT prompting typically underperforms compared to standard prompting, like closed-book question answering and natural language inference. Another study [88] expands on the self-consistency strategy to create a more general ensemble framework, extending it to ensemble not only on the answers but also on the prompts, emphasizing the importance of diverse reasoning paths for improved performance in CoT reasoning. These methods can be seamlessly integrated into CoT prompting without requiring additional training. In contrast, other studies involve training a scoring model to assess the reliability of generated reasoning paths [47] or continually training LLMs on the reasoning paths they

generate themselves to enhance performance [93].

A.8.2 Zero-Shot CoT

In contrast to few-shot CoT, zero-shot CoT does not rely on human-annotated task demonstrations in the prompts. Instead, it directly generates reasoning steps and utilizes these generated CoTs to arrive at the answers. The concept of zero-shot CoT was initially introduced in [40], where the LLM is prompted with instructions such as "Let's think step by step" to generate reasoning steps, followed by "Therefore, the answer is" to obtain the final answer. This strategy substantially enhances the model's performance when the model scale exceeds a certain threshold, although it is less effective with small-scale models, indicating notable emergent capabilities. To extend the CoT ability to more tasks, Flan-T5 and Flan-PaLM [18] incorporate instruction tuning on CoT annotations, leading to improved zero-shot performance on previously unseen tasks.

Vita

Candidate's full name: Mohammadamin Kanaani

University attended:

- **Master of Science in Computer Science**, University of New Brunswick, 2024
- **Bachelor of Science in Software Engineering**, University of Guilan, 2020

Publications:

- **Multimodal fake news analysis based on image–text similarity**, 2023 IEEE Transactions on Computational Social Systems

Conference Presentations:

- **Triple-R: Automatic Reasoning for Fact Verification Using Language Models**, 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, Italy - Torino, May 2024 (Submitted)
- **Few-Shot Learning for Intent Classification Using Pre-Trained BERT Model**, The 7th International Conference on Web Research, May 2021