UNIVERSITY OF CALIFORNIA,
IRVINE

Incorporating and Eliciting Knowledge in Neural Language Models

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Robert Lockwood Logan

Dissertation Committee:
Sameer Singh, Associate Professor, Chair
Padhraic Smyth, Chancellor's Professor
Stephan Mandt, Assistant Professor

2022

DEDICATION

In loving memory of the simpleton from Singleton.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to express my extreme gratitude to my advisor and mentor Sameer Singh. I am incredibly thankful to have been led through my doctoral studies by someone so infected with passion for their research, as well as patience and good humor. His foresight has steered my research interests towards numerous topics that have later become central to the field of NLP, and, under his guidance, I have developed from an unsure masters student into a confident and capable NLP researcher. I will always look back fondly on the late hours spent before deadlines making millimeter adjustments to tables and figures, and sneaking \vspace's into captions, to ensure that our papers looked as good as we thought they were.

I would also like to sincerely thank my co-advisor Padhraic Smyth, as well as Stephan Mandt for serving on my defense committee. I am particularly grateful for Padhraic's active role in recruiting me to UCI's Ph.D. program, and also for allowing me the chance to broaden my research horizons outside the field of NLP.

I have had the great fortune to have worked alongside a number of great colleagues and friends in my time at UCI: Caterina Belem, Alex Boyd, Anthony Chen, Dheeru Dua, Chis Galbraith, Casey Graff, Shivanshu Gupta, Tamanna Hossain-Kay, Disi Ji, Markelle Kelly, Gavin Kerrigan, Moshe Lichman, Rachel Longjohn, Yoshitomo Matsubara, Eric Nalisnick, Kolby Nottingham, Jihyun Park, Preston Putzel, Yasaman Razeghi, Edgar Robles, Dylan Slack, Preethi Seshadri, Sam Showalter, and Zhengli Zhao. I'd like to particularly thank Alex Boyd for acting as a statistical consultant for a number of the papers included in this dissertation; sorry if I get your statistician card revoked. I'd also like to thank Yasaman Razeghi for periodically writing nice things about me to strangers.

The work presented in this dissertation would not have been possible without the help of my fantastic co-authors: Ivana Balažević, Ming-Wei Chang, Matt Gardner, Vidur Joshi, Nelson F. Liu, Mark Neumann, Alexandre Passos, Matthew E. Peters, Fabio Petroni, Sebastian Riedel, Roy Schwartz, Taylor Shin, Noah A. Smith, and Eric Wallace. I would like to especially thank Matt Gardner for acting as a pseudo-co-co-advisor earlier on in my PhD, Nelson Liu for teaching me what a unit test was, and Eric Wallace for his meticulous copy editing abilities.

I was lucky to have had two internship opportunities at Google and one at Diffbot during the course of my PhD, and would like to thank my hosts—Dan Bikel, Ming-Wei Chang, Samuel Humeau, Andrew McCallum, Alexandre Passos, and Mike Tung—for giving me a preview of what to expect when I am thrust from this ivory tower into the jaws of industry.

Finally, on a more personal note, I would like to thank my family and friends. To the Sanders family, thank you for putting me up for the first three months of this journey; it was a trip getting to re-experience the later years of high school with the boys. To my second family, James, Brittany, and Mona, thank you for being there during all the highs and lows these past few years. To AJ, thank you for giving me a chance to relax and vent each weekend. To mom, thank you for always having a spot at home ready for me, and for all of the nice meals. Last but not least, to my Rachel, thank you for your undying support. I love you for always and forever.

The material presented in this dissertation was funded in part by: NSF awards #IIS-1817183 and #CNS-1730158, the DARPA MCS program under Contract No. N660011924033 with the United States Office Of Naval Research, the Irvine Initiative in AI, Law, and Society fellowship, the Allen Institute of Artificial Intelligence (AI2), Amazon, and Google. The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

VITA

Robert Lockwood Logan

EDUCATION

Doctor of Philosophy in Computer Science **2022**
University of California, Irvine

Bachelor of Arts in Mathematics and Economics **2013**
University of California, Santa Cruz

RESEARCH EXPERIENCE

Graduate Research Assistant **2017–2022**
University of California, Irvine *Irvine, California*

Research Scientist, Intern **2020, 2021**
Google *Remote*

Research Scientist, Intern **2018**
Diffbot *Mountain View, California*

TEACHING EXPERIENCE

Reader **2017–2020**
CS 161, 177, 272

ACADEMIC SERVICE

Reviewer **2017–2022**
EMNLP: 2018, 2019, 2020 (outstanding reviewer), 2021
NeurIPS: 2019 (top reviewer), 2020
AAAI: 2020, 2021, 2022
AKBC: 2020
COLING: 2020
ACL SRW: 2020
EACL SRW: 2021
CSKGs: 2021
ENLSP: 2021
AI Open: 2021
SoCal NLP: 2018, 2019
WeCNLP: 2020

REFEREED JOURNAL PUBLICATIONS

Detecting Conversation Topics in Primary Care Office Visits from Transcripts of Patient-Provider Interactions **2019**
Journal of the American Medical Informatics Association

REFEREED CONFERENCE PUBLICATIONS

FRUIT: Faithfully Reflecting Updated Information in Text **2022**
NAACL

Cutting Down on Prompts and Parameters: Simple Few-Shot Learning with Language Models **2022**
ACL Findings

Benchmarking Scalable Methods for Streaming Cross Document Entity Coreference **2021**
ACL

Active Bayesian Assessment for Black-Box Classifiers **2021**
AAAI

AutoPrompt: Eliciting Knowledge from Language Models Using Automatically Generated Prompts **2020**
EMNLP

On Importance Sampling-Based Evaluation of Latent Language Models **2020**
ACL

Knowledge Enhanced Contextual Word Representations **2019**
EMNLP

Barack's Wife Hillary: Using Knowledge Graphs for Fact-Aware Language Modeling **2019**
ACL

PoMo: Generating Entity-Specific Post-Modifiers in Context **2019**
NAACL

REFEREED WORKSHOP PUBLICATIONS

Cutting Down on Prompts and Parameters: Simple Few-Shot Learning with Language Models **2021**
Efficient Natural Language and Speech Processing @ NeurIPS (Best Poster)

Deriving Behavioral Tests from Common Sense Knowledge Graphs **2021**
Common Sense Knowledge Graphs @ AAAI

COVIDLies: Detecting COVID-19 Misinformation on Social Media **2020**
NLP-COVID19 Workshop @ EMNLP (Best Paper)

Bayesian Evaluation of Black-Box Classifiers **2019**
Uncertainty and Robustness in Deep Learning @ ICML

Multimodal Attribute Extraction **2017**
Automated Knowledge Base Construction @ NeurIPS

UNREFFEREED PREPRINTS

Impact of Pretraining Term Frequencies on Few-Shot Reasoning **2022**
arXiv

INVITED TALKS

Zero- and Few-Shot NLP with Pretrained Language Models **2022**
ACL 2022

Fill in the Blanks: Prompt-Based Solutions for NLP **2021**
UCI CML Seminar

Hyperparameters: The Bane of (Current) Deep Learning **2018**
CHASE-CI Workshop

Attribute-Value Extraction from Multimodal Data **2018**
SoCal NLP Symposium

ABSTRACT OF THE DISSERTATION

Incorporating and Eliciting Knowledge in Neural Language Models

By

Robert Lockwood Logan

Doctor of Philosophy in Computer Science

University of California, Irvine, 2022

Sameer Singh, Associate Professor, Chair

Neural language models have drastically changed the landscape of natural language processing (NLP). Originally used for language generation (e.g., in summarization and dialogue systems) and scoring (e.g., in automatic speech recognition and statistical machine translation), these models now widely serve as the universal starting point for transfer learning on almost all NLP tasks. This development is largely due to matters of scale—because language models require only raw text for supervision, they can be trained using the massive amounts of text readily available on the web. Accordingly, when this process is carried out on enough data, it exposes models to knowledge that subsequently improves their performance when they are transferred to downstream tasks. As neural language models become increasingly prevalent, it is crucial to be able to characterize and effectively leverage this knowledge, as well as provide recourse when it is incorrect.

In this dissertation, we address this need by exploring two research directions: 1) using the technique of prompting as a means for eliciting knowledge from language models, and 2) approaches for integrating the knowledge stored in language models and external knowledge bases.

In the first direction, we will begin by demonstrating how prompts can be used to reformulate NLP tasks as fill-in-the-blanks and complete-the-sentence problems that can naturally be solved using language models. We will then use prompts to diagnose which kinds of factual and task-specific knowledge are learned by language models during pretraining. Our analysis reveals that, while

language models struggle to memorize facts, they possess surprisingly powerful capabilities to perform certain tasks. Next, we will introduce AUTOPROMPT, a technique to automate prompt construction, and show that by using automatically constructed prompts, language models can achieve near state-of-the-art performance on some tasks without requiring task-specific finetuning. Based on this insight, we conclude our exploration of this topic by looking into how prompting can be best combined with finetuning to apply language models in few-shot learning settings.

In the second direction, we will explore how language models and knowledge bases can be integrated in order to improve their coverage of facts. We will first introduce two approaches, the knowledge graph language model (KGLM) and KnowBERT, for endowing language models with the means to condition on information from entity and knowledge graph embeddings. We then use the prompts developed in the previous section to show that conditioning on this information improves language models' recall of facts. We will conclude our treatment of this research direction by studying whether the converse is true, i.e., whether languages models can be used to help maintain consistency in knowledge bases when their content is updated to reflect new information. Taken together, our work on these research topics provides a collection of insights into the nature and applications of knowledge in neural language models.

1

Introduction

"Isn't this where we came in?"

– Pink Floyd, *The Wall*

The landscape of natural language processing (NLP) has been drastically changed over the past decade by parallel advances in both the theory of neural networks and specialized hardware and software for training them. Models now routinely achieve state-of-the-art accuracy on supposedly challenging benchmarks, often matching or exceeding human performance baselines.[1][2] Surprisingly, little expert knowledge is required to obtain such results. In many cases, anybody with an internet connection and access to sufficiently powerful hardware can train a state-of-the-art model on a task of interest by merely downloading and finetuning model weights provided by other researchers.

Central to this development are *neural language models*, which, in their most general formulation, are probabilistic models trained to restore truncated or distorted pieces of text. Originally used for language generation (e.g., in summarization and dialogue systems) and scoring (e.g., in automatic

[1] See: https://gluebenchmark.com/leaderboard, https://super.gluebenchmark.com/leaderboard, https://leaderboard.allenai.org/ for examples.

[2] The author would like to carefully to note that this is not to say that the models have "superhuman abilities," only that model accuracy on such benchmarks has drastically improved.

speech recognition and statistical machine translation), these models now widely serve as the universal starting point for transfer learning on almost all downstream NLP tasks. This development is largely due to how well neural language models scale. Because they only require unlabeled text for supervision, they can be trained using the massive amounts of text readily available on the web. Accordingly, when they are trained on a large enough dataset, the models are exposed to knowledge that subsequently improves their performance when they are transferred to downstream tasks.

As these models become increasingly prevalent in modern NLP, it is important to understand the nature of this knowledge and its uses. This leads us to the following research questions that we will seek to answer in this dissertation:

1. Can we develop techniques to measure whether a language model has learned a particular fact or type of knowledge?

2. If these techniques reveal that the language model's knowledge is deficient in some way, e.g., incorrect or missing, can we incorporate information from an external knowledge base to address this deficiency? Conversely, if the information in a knowledge base is incomplete or inconsistent, can language models be used to correct it?

1.1 Contributions and Key Findings

This dissertation makes the following contributions:

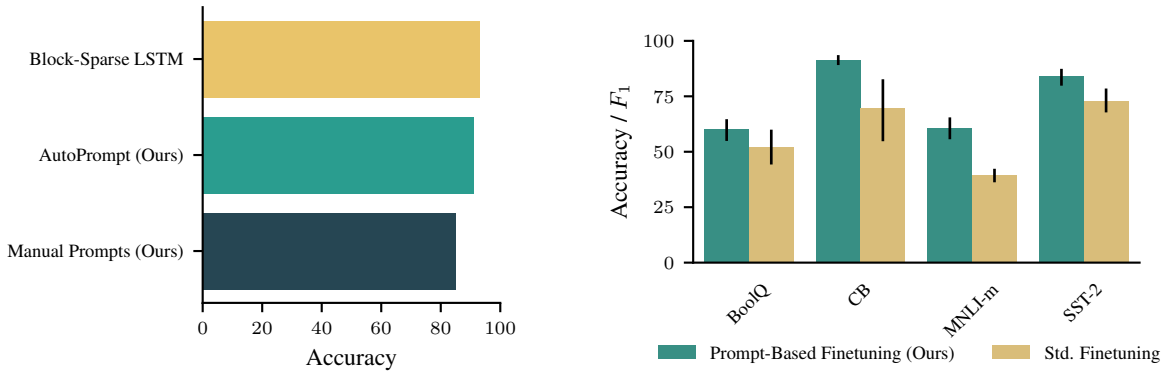
1.1.1 Prompting to Elicit Knowledge from Language Models

We develop the method of prompting to evaluate the knowledge that language models learn from pretraining. Prompting reformulates tasks as fill-in-the-blanks and complete-the-sentence questions

that are naturally solvable using language models. For example, to determine whether a language model knows who Barack Obama's wife is, we can test whether it correctly fills the blank in the prompt "Barack Obama is married to [BLANK]" with the token "Michelle." Similarly, to determine whether a language model is knowledgeable about sentiment, we can take an instance from a sentiment analysis dataset such as SST-2 (Socher et al., 2013b), e.g., "This is movie of the year material." and append the text "Overall, I thought this movie was [BLANK]" then see whether the language model correctly assigns a higher probability to words associated with positive sentiment (e.g., "good," "great," etc.) than words associated with negative sentiment (e.g., "bad," "terrible," etc.). Through the usage of prompting, we find that while language models struggle to memorize facts about the world (Figure 1.2), they nonetheless possess surprisingly powerful capabilities to perform certain tasks (Figure 1.1a; manual).

1.1.2 Automatic Prompt Construction

One major drawback of prompting is that writing effective prompts requires manual effort and can be unintuitive for certain types of knowledge (e.g., testing for language model's knowledge of entailment). We address this by introducing AUTOPROMPT, an data-driven approach that automates prompt construction. Given a number of instances that test for a specific type of knowledge, AUTOPROMPT employs a gradient-guided search procedure that iteratively updates the words in the prompt to elicit better performance on those instances. We find that AUTOPROMPT produces prompts that are substantially more effective than manually written prompts at eliciting both factual and task-specific knowledge from language models. In fact, for some classification tasks, the results produced using AUTOPROMPT are competitive with formerly state-of-the-art classifiers, despite the fact that prompting does not make any updates to the model weights (Figure 1.1a; AUTOPROMPT).



(a) (b)

Figure 1.1: **Prompting results (a)** . We use prompting to measure RoBERTa-large's "natural" ability to perform sentiment analysis on SST-2. We find that, *without any parameter tuning*, RoBERTa performs comparably to the Block-Sparse LSTM, the state-of-the-art model in 2017. **Prompt-based finetuning (b)**. Using prompt-based finetuning significantly outperforms standard finetuning in few-shot settings (16 examples per class).

1.1.3 Applications of Prompting to Few-Shot Learning

Based on the aforementioned results, we hypothesize that prompting may also be useful for improving language model accuracy in few-shot learning settings, where it is important to leverage the knowledge already known by the model in order to learn as much information as possible from a small number of training examples. Accordingly, we explore how the method of prompting can be best used in conjunction with model finetuning. We find that, in low-data settings, prompt-based approaches to finetuning significantly outperform standard language model finetuning approaches. We additionally show that, when using prompt-based finetuning, models are relatively robust to variation in the prompt (unlike in the frozen model setting), and that only a small fraction (approximately 0.1%) of the model's parameters need to be updated in order to achieve this performance (Figure 1.1b).

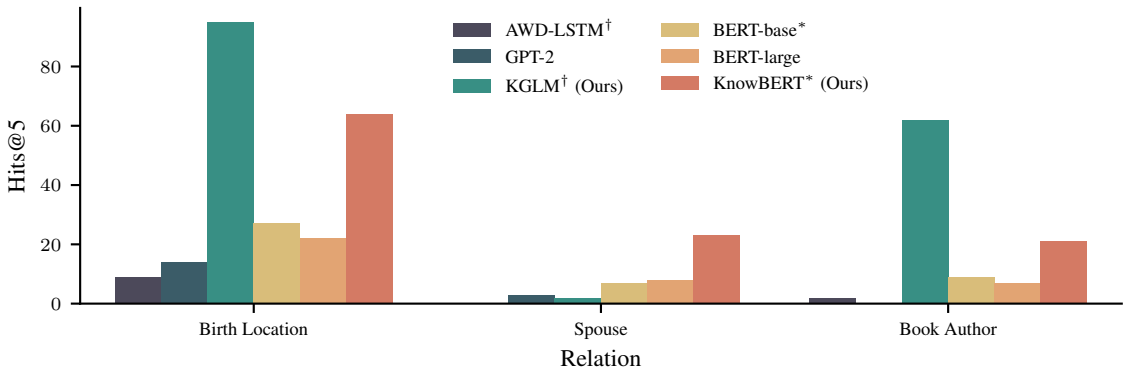


Figure 1.2: **Factual recall** capabilities of different language models. KGLM and KnowBERT, the knowledge-backed language models introduced in this dissertation, have substantially better factual recall abilities than existing language models. Autoregressive language models are colored blue and masked language models are colored orange. Superscripts denote comparable model architectures. Hits@5 measures whether the correct prompt completion occurs within the top 5 predictions of the language model.

1.1.4 Injecting Knowledge into Language Models

Although our prompting results show that language models possess task-specific knowledge, we also find their knowledge of real-world facts to be lacking. To address this, we investigate whether factual knowledge can be "injected" into a language model, by allowing the model to condition on representations of information in knowledge graphs and textual knowledge bases. We introduce two different approaches: 1) the knowledge graph language model (KGLM), that iteratively traverses and incorporates information from a knowledge graph into an autoregressive language model, and 2) KnowBERT, that incorporates information from an entity linker into a masked language model. We show that leveraging these external sources of knowledge substantially improves language models' ability to complete prompts requiring factual knowledge (Figure 1.2; KGLM and KnowBERT).

1.1.5 Using Language Models to Improve Consistency in Knowledge Bases

Knowledge bases themselves are often incomplete (Nickel et al., 2011), and can be difficult to maintain as new information is added that may conflict with the information already present. In the final part of this dissertation, we introduce a novel generation task—faithfully reflecting updated information in text (FRUIT)—that seeks to automatically update existing entries in a textual knowledge base so that they are consistent with new (potentially conflicting) information. In addition to collecting training and test data for this task, we find that, with some modification, neural language models serve as strong baselines for this task, producing completely correct updates 50% of the time. These results suggest a degree of complementarity between the knowledge stored in language models and knowledge bases.

1.2 Declaration of Previous Work and Collaborations

The content of this dissertation is based on a number previously published works, all of which were written under the guidance of Sameer Singh. Specifically:

- The approach for evaluating autoregressive language models using prompts derived from knowledge graphs (Chapter 3) and knowledge graph language model (KGLM) architecture (Chapter 6) were published in the 57th Annual Meeting of the Association of Computational Linguists (ACL 2019) in collaboration with Nelson F. Liu, Matthew E. Peters, and Matt Gardner (Logan et al., 2019).

- The approach for evaluating masked language models using prompts derived from knowledge graphs (Chapter 3) and KnowBERT architecture (Chapter 6) were published in the Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP

2020) in collaboration with Matthew E. Peters, Mark Neumann, Roy Schwartz, Vidur Joshi, and Noah A. Smith (Peters et al., 2019).

- Additional results pertaining to the application of importance sampling to estimate the perplexity of latent language models (Chapter 7) were published in ACL 2020 in collaboration with Matt Gardner (Logan IV et al., 2020).

- AutoPrompt (Chapter 4) was published in the Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020) in collaboration with Taylor Shin, Yasaman Razeghi, and Eric Wallace (Shin et al., 2020).

- The application of prompting to few-shot learning (Chapter 5) will appear in the Findings of the Association for Computational Linguistics 2022 (ACL Findings 2022) in collaboration with Ivana Balažević, Eric Wallace, Fabio Petroni, and Sebastian Riedel (Logan IV et al., 2021a).

- Faithfully reflecting updated information in text (FRUIT; Chapter 8) will appear in the Proceedings of the 2022 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2022) and was written under the mentorship of Alexandre Passos and Ming-Wei Chang during an internship at Google (Logan IV et al., 2021b).

The author of this manuscript has or shares the lead author role on all of these works except Peters et al. (2019), and a detailed account of his contributions is provided at the end of each chapter.

1.3 Thesis Outline

In the next chapter, we provide necessary background information about the learning objectives and neural architectures used by modern language models, as well as the different types of knowledge bases considered in this work.

The remainder of the dissertation is divided into two parts:

The first part focuses on the method and applications of prompting. Chapter 3 provides a primer on how prompting can be used to elicit knowledge from language models. In the next chapter, Chapter 4, we introduce AUTOPROMPT, a data-driven method for automating prompt construction. Then, in Chapter 5, we investigate the utility of prompting in few-shot learning scenarios.

The second part of this dissertation focuses on methods for integrating the knowledge in neural language models and knowledge bases. Chapter 6 introduces the knowledge graph language model (KGLM) and KnowBERT, neural language models capable of leveraging knowledge in external knowledge bases to improve their ability to model factual language. In Chapter 7, we take a slight detour to investigate the application of importance sampling to estimating the perplexity of the KGLM and related latent language models that additionally model some form of latent structure underlying a piece of text. Then, in Chapter 8, we introduce a dataset and benchmark models for the novel task of faithfully reflecting updated information in text (FRUIT), which looks at how language models can be applied to maintain consistency in textual knowledge bases as information is updated over time.

Finally, Chapter 9 wraps up this dissertation with a discussion of the limitations of the approaches discussed, and proposals of promising areas for future work on these topics.

2

Background

"We barely remember who or what came before this precious moment."

– Tool, *Parabola*

In this chapter, we provide background material on language modeling and knowledge bases, as well as establish notation used throughout the rest of this dissertation. Although we will provide relevant details about these subjects and recent modeling developments, we assume the reader has an undergraduate-level familiarity with linear algebra and machine learning, as well as basic familiarity with natural language processing and neural networks (e.g., recurrent and convolutional neural networks, dropout as a form of regularization, etc.). For an introduction to these topics we recommend:

- "Introduction to Linear Algebra" (Strang, 1993),
- "Pattern Recognition and Machine Learning" (Bishop and Nasrabadi, 2007),
- "Deep Learning" (Salakhutdinov, 2014), and

- "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition" (Jurafsky and Martin, 2000).

2.1 Notation

Matrices are denoted using uppercase bolded letters (e.g., $\boldsymbol{W}$), vectors using lowercase bolded letters (e.g., $\boldsymbol{b}$), sets using uppercase calligraphic/script letters, (e.g., $\mathcal{S}$ or $\mathscr{S}$), and scalars using lowercase regular weight Greek letters[1] (e.g., α). As sequences are a core focus of this work, we denote the space of all possible sequences of elements from a set using an asterisk superscript (e.g., $\mathcal{S}^*$ is the set of all possible sequences of elements in $\mathcal{S}$), and denote elements of $\mathcal{S}^*$ using lowercase regular weight letters (e.g., $s \in \mathcal{S}^*$) to distinguish them from vectors, where it is understood that $s = \left(s_1, s_2, \ldots, s_{|s|}\right)$. Lastly, we will adopt the notation $s_{<i} = \left(s_1, s_2, \ldots, s_{i-1}\right)$ to denote the subsequence of the first $i-1$ elements of s, and $s_{[i,j]} = \left(s_i, s_{i+1}, \ldots, s_{j-1}, s_j\right)$ to denote the subsequence of tokens whose index is in the closed set of integers $[i, j]$.

To simplify notation when working with random variables (e.g., U and V), we will use $p(u)$ in place of $p(U = u)$ to denote the probability density/mass function of U evaluated at a value u, $p(u|v)$ in place of $p(U = u|V = v)$ to denote the conditional probability density/mass function, and will generally opt to suppress the symbols U and V. In cases where we are working with parameterized probability distributions and need to refer their parameters, we may do so using a subscript (e.g., $p_{\boldsymbol{\theta}}(u)$ denotes a probability distribution with parameters $\boldsymbol{\theta}$).

[1] The only exception to this convention is that we will use $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to refer to collections of model parameters.

Input	A groan of tedium escapes me
Character	A, , g, r, o, a, n, , o, f, , t, e, d, i, u, m, . . .
Whitespace	A, groan, of, <UNK>, escapes, me
BPE	A, gro, #an, of, ted, #ium, escapes, me

Figure 2.1: **Illustration of different tokenization schemes.** Comparison of how the same piece of text is segmented into separate tokens using character-level tokenization, whitespace tokenization, and byte pair encoding (BPE). For BPE tokens prefaced with the character "#" indicate continuations are concatenated with the previous token.

2.2 Language Models

In this section, we will provide a brief primer on the task of language modeling, with our primary goal being to introduce the neural language models (e.g., AWD-LSTM-LM, BERT, GPT, and T5) and associated vocabulary (e.g., pretraining, finetuning, mask tokens) that appear in this dissertation.

2.2.1 Non-Neural Language Models (1940-2000)

The task of language modeling has had a prominent place in the history of machine learning and natural language processing, with the *n*-gram language model appearing as early as Claude Shannon's seminal paper "*A Mathematical Theory of Communication*" (Shannon, 1948), and serving as a crucial component of automatic speech recognition (Jelinek et al., 1975) and statistical machine translation systems (Brown et al., 1988).

Until recently, the term language model was used exclusively in adherence with the following definition:

DEFINITION 1 (Generative Formulation)**.** A *language model* is a generative model of text, i.e., a probability distribution $p(x)$ over sequences of tokens $x \in \mathscr{V}^*$ coming from a vocabulary $\mathscr{V}$.

An important element of this definition is the requirement that the text being modeled can be represented as a sequence of tokens. *Tokenization*, or the process of demarcating sections of text

into tokens, is a common preprocessing step applied in natural language processing. Different tokenization schemes include character-level tokenization (which breaks the text into sequences of characters), whitespace tokenization (which breaks the text into sequences of words), and methods such as WordPiece segmentation (Wu et al., 2016a) and Byte Pair Encoding (BPE) (Sennrich et al., 2016) (which attempt to strike a balance between character-level and word-level tokenization by breaking text into subword units). An illustration of how these different schemes tokenize the same piece of text is provided in Figure 2.1.

It is important to recognize that the choice of tokenization scheme is a part of a language model; although two models may be trained on the same corpus of text, they may fundamentally differ in terms of how that text is tokenized, which, as we will see in Chapter 3, can complicate evaluation. It is also important to know that most vocabularies include a special *unknown token*, <UNK>, that all tokens outside of the vocabulary, $u \notin \mathscr{V}$, are mapped to during preprocessing.

***n*-gram Language Model** Prior to the 2000s, language modeling was predominantly performed using n-gram language models. In an n-gram model the probability distribution $p(x)$ is factorized autoregressively using the chain rule of probability:

$$p(x) = \prod_{i=1}^{|x|} p(x_i | x_{<i}), \tag{2.1}$$

an nth order Markov assumption is made:

$$p(x_i | x_{<i}) = p(x_i | x_{[i-n:i-1]}), \tag{2.2}$$

and the transition probabilities are estimated using co-occurrence statistics. Note: Language models that generate text by modeling the distribution $p(x_i | x_{<i})$ are commonly referred to as *autoregressive language models*.

A common flaw of n-gram language models is that transition probabilities can become sparse as the value of n increases. This issue is typically addressed using one of a number of different smoothing techniques (Jelinek and Mercer, 1980; Katz, 1987; Ney et al., 1994), the discussion of which is outside the scope of this work.

2.2.2 Early Neural Language Models (2000-2018)

Feed-Forward Neural Language Model The earliest work applying neural networks to language modeling is that of Bengio et al. (2000), who, similar to the n-gram model, make a nth order Markov assumption, but parameterize the distributions $p(x_i|x_{[i-n:i-1]})$ using a multi-layer perceptron (MLP):

$$p(x_i|x_{[i-n:i-1]}) = \mathrm{softmax}(\boldsymbol{W}\tanh(\boldsymbol{x}_{i-n}, \ldots, \boldsymbol{x}_{i-1})) \tag{2.3}$$

where each $\boldsymbol{x}_j \in \mathbb{R}^d$ is a d-dimensional *word embedding* of the word x_j, and $\boldsymbol{W} \in \mathbb{R}^{|\mathscr{V}|\times d(n-1)}$ is a projection matrix that ensures the resulting logit vector is the same size as the vocabulary $\mathscr{V}$.

RNN/CNN Language Models The feed-forward neural language model was subsequently improved by Collobert and Weston (2008), who modify the architecture to use a convolutional neural network (CNN) and use max-pooling over the sequence dimension to allow for arbitrarily long sequences of tokens to be conditioned on. They additionally train the network on NLP tasks other than language modeling in a multi-task setup.

Under this approach, words that function similarly will have similar embeddings, allowing the model to assign reasonable probabilities to sequences unobserved in the training data without resorting to smoothing. For example, Collobert and Weston (2008) show that the nearest neighbors of the embedding of the word "France" are the embeddings of "Spain", "Italy", and "Russia". Thus, even if the sequence "I would like to visit France" never occurs in the training data, the model still

has a reasonable chance of generating this sequence provided it has seen similar sequences, e.g., "I would like to visit Spain".[2]

The following decade of work on neural language modeling introduced numerous engineering improvements such as: usage of recurrent neural network architectures (RNNs) (Mikolov et al., 2010; Zaremba et al., 2014), improved regularization schemes (Merity et al., 2018), and innovations for dealing with rare words (Merity et al., 2017b; Grave et al., 2017). Of these works we would like to highlight the AWD-LSTM-LM in particular as it was the state-of-the-art neural language model at the time that writing began on this manuscript and this model will appear in later chapters.

AWD-LSTM-LM (Merity et al., 2018) The AWD-LSTM-LM is an LSTM-based (Hochreiter and Schmidhuber, 1997) recurrent neural language model which uses the following enhancements:

- DropConnect (Wan et al., 2013): A variant of dropout applied to weight matrices instead of layer activations.
- Weight tying (Inan et al., 2017): The weights in the input embedding layer and output projection are shared, substantially reducing the number of trainable variables.
- Non-monotonically triggered averaged stochastic gradient descent: An optimization algorithm that switches from stochastic gradient descent (SGD) to averaged stochastic gradient descent (ASGD), a variant of SGD with better theoretical convergence properties (Polyak and Juditsky, 1992).
- Activation Regularization and Temporal Activation Regularization (Merity et al., 2017a): Variants of L_2 regularization applied to network activations instead of weights.

[2]Historical context: Embeddings have been of interest in NLP even prior to neural language modeling. For a treatment of pre-neural embedding methods refer to: https://web.stanford.edu/~jurafsky/li15/lec3.vector.pdf.

2.2.3 Parallel Developments (2010-2018)

In this subsection, we will provide prerequisite background information on developments in machine learning that happened outside the field of neural language modeling but are needed to understand the models covered in the next section.

Denoising Autoencoders (Vincent et al., 2010) Denoising autoencoders learn feature representations from data without the need of labels. Generally speaking, an autoencoder is a form of model that learns a deterministic mapping of inputs x into representations $\boldsymbol{y} = f_{\boldsymbol{\theta}}(x)$ and a subsequent deterministic mapping of representations back to the input space $x' = g_{\boldsymbol{\phi}}(\boldsymbol{y})$.[3] The model is trained with the objective of minimizing some form of reconstruction error $\mathcal{L}(x, x')$ (e.g., Euclidean distance, cross-entropy, etc.). Because autoencoders require only input data x, they can learn feature representations without requiring the collection of additional labels, i.e., they are entirely *self-supervised*.

Denoising autoencoders build upon the vanilla autoencoder setup by introducing an additional step where the input x is corrupted into $\tilde{x}$ by a stochastic mapping $\tilde{x} \sim q_{\text{noise}}(\tilde{x}|x)$. The objective is then modified to learn to reconstruct the original input from its corrupted form, e.g., minimize $\mathcal{L}(x, \tilde{x}')$ where $\tilde{x}' = g_{\boldsymbol{\phi}}(f_{\boldsymbol{\theta}}(\tilde{x}))$. For example, an image may be corrupted using Gaussian noise or rotation and cropping, while text may be corrupted by masking out spans of tokens or truncating sequences.

Sequence-to-Sequence Models (Sutskever et al., 2014a) As the name suggests, sequence-to-sequence (Seq2Seq) models were developed to solve problems that require mapping an input sequence x into an output sequence y. Each Seq2Seq model has two primary components: a neural *encoder* network and a neural *decoder* network (see Figure 2.2 for illustration). The encoder

[3] Typically $f_{\boldsymbol{\theta}}$ and $g_{\boldsymbol{\phi}}$ are referred to as an encoder and decoder (respectively). However, we will not use these terms to avoid conflating $f_{\boldsymbol{\theta}}$ and $g_{\boldsymbol{\phi}}$ with the encoder and decoder in a sequence-to-sequence model.

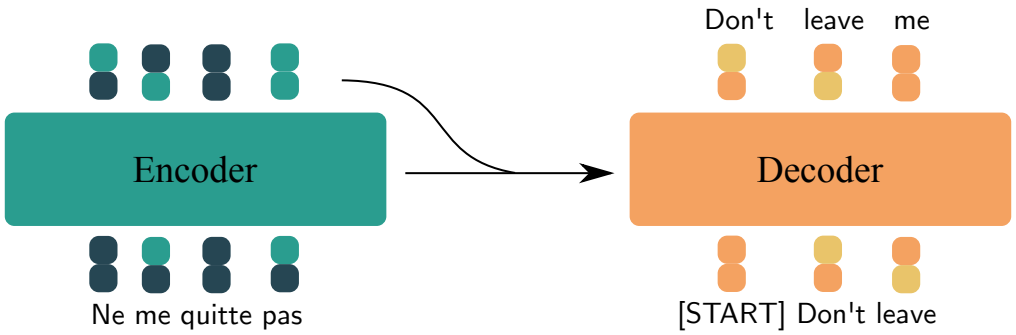


Figure 2.2: **Illustration of a sequence-to-sequence model** used for machine translation. Input text is encoded using the encoder on the left, producing a vector representation of each token. These representations are then fed to the decoder on the right which generates a candidate translation along with vector representations of each token in the output sequence.

processes the input sequence into a sequence of vector representations of its elements:

$$\left(\boldsymbol{h}_1^x, \ldots, \boldsymbol{h}_{|x|}^x\right) = \text{Enc}\left(x_1, \ldots, x_{|x|}\right) \tag{2.4}$$

and the decoder then conditions on these vectors (as well as its past outputs) to produce "forward-looking" representations:

$$\boldsymbol{h}_i^y = \text{Dec}(\boldsymbol{h}_1^x, \ldots, \boldsymbol{h}_{|x|}^x, \boldsymbol{h}_1^y, \ldots, \boldsymbol{h}_{i-1}^y, y_{i-1}), \tag{2.5}$$

that are used to obtain a distribution over the next output element:

$$p(y_i|x, y_{<i}) = \text{softmax}(\boldsymbol{W}\boldsymbol{h}_i^y). \tag{2.6}$$

The details of how the encoder and decoder process, condition on, and generate sequences vary across models. For example, in the original Seq2Seq model of Sutskever et al. (2014a), both the encoder and decoder are LSTMs, and conditioning is achieved by feeding the hidden representation of the last input sequence element produced by the encoder as the initial hidden representation in

the decoder. Formally,

$$\boldsymbol{h}_j^x = \text{LSTM}(\boldsymbol{h}_{j-1}^x, \boldsymbol{x}_j) \tag{2.7}$$

$$\boldsymbol{h}_i^y = \text{LSTM}(\boldsymbol{h}_{i-1}^y, \boldsymbol{y}_{i-1}) \tag{2.8}$$

where $\boldsymbol{h}_0^y$ is the last hidden state output by the encoder (i.e., $\boldsymbol{h}_0^y = \boldsymbol{h}_{|x|}^x$), and $\boldsymbol{x}_j$ and $\boldsymbol{y}_i$ denote learned embeddings of the sequence elements x_j and y_i, respectively. Note that, in this setup, the only information the decoder receives about the input sequence is from the single vector $\boldsymbol{h}_{|x|}^x$. This "bottleneck" has been observed to negatively impact performance as the length of the input sequence grows (Cho et al., 2014).

Attention Mechanism (Bahdanau et al., 2015) The attention mechanism was originally introduced to provide an alternative way for the decoder to condition on the outputs of the encoder while avoiding the aforementioned bottleneck. Conceptually, at each decoding step, the mechanism computes a relevance score for each of the representations produced by the encoder and then uses these scores to take a weighted sum of the encoder representations. While there are a number of different variations (Luong et al., 2015), the most commonly used is *dot product attention*, which is computed as follows:

$$\text{Attn}(\boldsymbol{q}_i, \boldsymbol{h}_1^x, \ldots, \boldsymbol{h}_{|x|}^x) = \sum_j \alpha_{i,j} \boldsymbol{h}_j^x \tag{2.9}$$

where:

$$\alpha_{i,j} = \frac{\boldsymbol{q}_i \cdot \boldsymbol{h}_j^x}{\sum_k \boldsymbol{q}_i \cdot \boldsymbol{h}_k^x} \tag{2.10}$$

are referred to as *attention weights*, and $\boldsymbol{q}_i$ is a *query* vector. Importantly, because the query vector $\boldsymbol{q}_i$ can differ at each decoding step, different parts of the input sequence can be attended to during decoding.

To provide a concrete example of how the attention mechanism could be incorporated into the decoder, Equation 2.8 could be replaced with:

$$\boldsymbol{q}_i = \text{LSTM}(\boldsymbol{h}^y_{i-1}, \boldsymbol{y}_{i-1}) \tag{2.11}$$

$$\boldsymbol{h}^y_i = \boldsymbol{q}_i + \text{Attn}(\boldsymbol{q}_i, \boldsymbol{h}^x_1, \ldots, \boldsymbol{h}^x_{|x|}). \tag{2.12}$$

In practice, the output of the attention mechanism is typically modified (e.g., via a linear projection) and incorporated into the decoder by more complicated means (e.g., via a MLP), however these are auxiliary modeling decisions unrelated to the mechanism itself.

Transformer (Vaswani et al., 2017a) The Seq2Seq architectures described thus far all rely on RNNs to handle processing variable-length input and output sequences. While well suited to this task, a major drawback of RNNs is that each hidden state is computed as a function of the previous hidden state, thereby precluding parallelization over the sequence dimension during training. In contrast, the attention mechanism introduced in the previous paragraph also provides a means for processing variable-length sequences, however, it is not necessarily constrained to be recurrent, i.e., while $\boldsymbol{h}^x$ and $\boldsymbol{q}_i$ were obtained using RNNs, the mechanism itself relies only dot products and sums, operations that are easily parallelizable.

Leveraging this insight, Vaswani et al. (2017a) introduce the Transformer architecture, a Seq2Seq model that processes sequences entirely using attention and, accordingly, is highly parallelizable during training. The core operation in the Transformer is *scaled dot product attention*, which makes the following improvements to the attention mechanism in the previous paragraph.

First, Vaswani et al. (2017a) observe that the vectors summed over in Equation 2.13 do not necessarily need to be the same as those used compute the attention weights in Equation 2.10, there only need to be equally many of them to ensure that the sum over the index j is well-defined.

Thus, the first way that the attention mechanism is modified is that so-called *value* vectors $\boldsymbol{v}_j$ are substituted for $\boldsymbol{h}_j^x$ in Equation 2.9, and *key* vectors $\boldsymbol{k}_j$ are substituted for $\boldsymbol{h}_j^x$ in Equation 2.10.

Second, Vaswani et al. (2017a) observe that the dot-product used to compute the attention weights naturally becomes larger as the dimension of the query and key vectors increases and hypothesize that this causes stability issues during training. To adjust for the increase in magnitude, they propose scaling the dot products in Equation 2.10 by a factor of $\sqrt{d}$, where d is the dimension of the query and key vectors.

By stacking the query, key and value vectors into matrices $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$, respectively, their new form of *scaled dot-product attention* can be compactly written as:

$$\text{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right)\boldsymbol{V}. \tag{2.13}$$

When formulated in terms of these matrix operations, this equation emphasizes that the attention outputs are simultaneously computed for all of the query vectors.

The Transformer architecture uses the attention operation in three different ways. First, similar to the previously covered Seq2Seq architecture, attention is used to incorporate the outputs of the encoder into the decoder. In terms of Equation 2.13, this corresponds to using the decoder to produce the query vectors, and the encoder to produce the key and value vectors. Second, *self-attention* is used to encode the input sequence, which uses the same inputs for both the query vectors, and the key and value vectors. Lastly, *masked self-attention* is used to decode output sequence. Just like in self-attention, the same inputs are used for the query, key and value vectors, however, a mask is applied to the attention weights to prevent attending to future parts of the sequence, e.g.:

$$\text{MaskedAttn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \left(\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right) \odot \boldsymbol{M}\right)\boldsymbol{V}, \tag{2.14}$$

where $\boldsymbol{M}$ is a lower triangular matrix of ones, and $\odot$ denotes the Hadamard (i.e., elementwise) product.

One drawback of using attention to encode sequences is that it is unaffected by re-ordering of its inputs; permuting the order of the key and value vectors does not change the output at all, and permuting the query vectors has the same effect as applying the same permutation to the outputs of the attention mechanism. This is not ideal for most tasks, such as processing language, since re-ordering sequences can change their meaning (e.g., "I eat meat not vegetables" has a different meaning than "I eat vegetables not meat"). In the Transformer architecture, this issue is addressed by adding a learned positional encoding vector $\boldsymbol{p}_i$ to the ith input embedding.

In terms of our mathematical description of Seq2Seq models, the encoder of a single-layer Transformer can be written as follows:

$$\boldsymbol{H}^x = \text{Attn}(\boldsymbol{X}', \boldsymbol{X}', \boldsymbol{X}'), \tag{2.15}$$

$$\text{where} \quad \boldsymbol{X}' = \boldsymbol{X} + \boldsymbol{P} \tag{2.16}$$

and $\boldsymbol{X}$, $\boldsymbol{P}$, and $\boldsymbol{H}^x$ are the matrices formed by stacking the embeddings $\boldsymbol{x}_j$, corresponding positional encoding vectors $\boldsymbol{p}_j$, and encoder outputs $\boldsymbol{h}^x_j$, respectively. The decoder can be written as:

$$\boldsymbol{H}^y = \text{Attn}(\boldsymbol{H}^h, \boldsymbol{H}^x, \boldsymbol{H}^x), \tag{2.17}$$

$$\text{where} \quad \boldsymbol{H}^h = \text{MaskedAttn}(\boldsymbol{Y}', \boldsymbol{Y}', \boldsymbol{Y}'), \tag{2.18}$$

$$\boldsymbol{Y}' = \boldsymbol{Y} + \boldsymbol{P}, \tag{2.19}$$

and $\boldsymbol{Y}$, $\boldsymbol{P}$, and $\boldsymbol{H}^y$ are the matrices formed by stacking the embeddings $\boldsymbol{y}_i$, corresponding positional encoding vectors $\boldsymbol{p}_i$, and encoder outputs $\boldsymbol{h}^x_i$, respectively.

As in our descriptions of previous Seq2Seq models, these equations are simplified for the purpose of illustration. In practice, implementations of Transformers typically make use of a number of

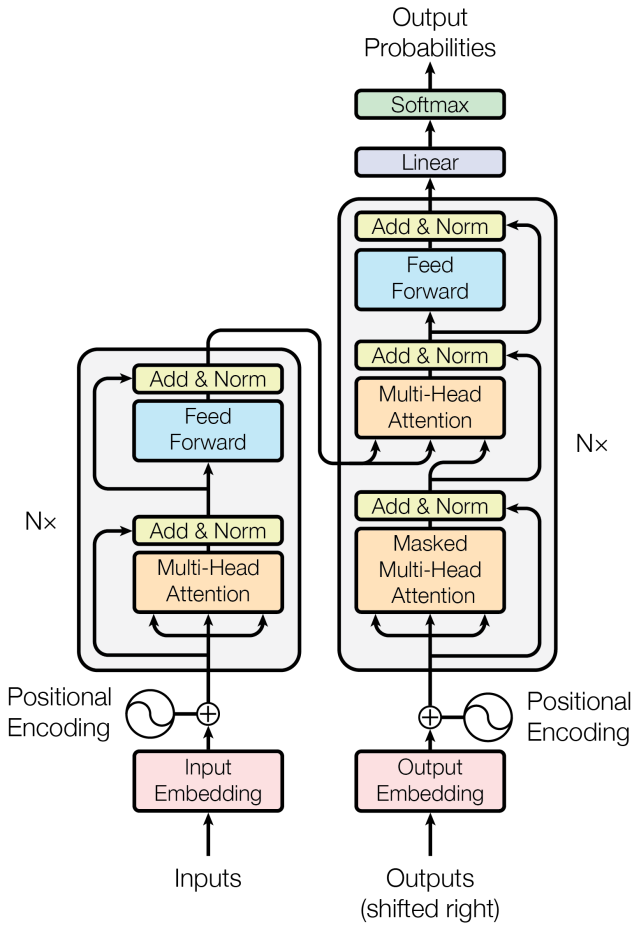


Figure 2.3: **Illustration of the Transformer architecture** from Vaswani et al. (2017a).

modifications to improve performance. Notably, the computational cost of the matrix multiplications in Equation 2.13 can be prohibitive when the dimension of the input vectors is too large. Vaswani et al. (2017a) use *multi-head attention* as a means of reducing this cost, where the basic idea is to concatenate together the outputs of multiple attention mechanisms applied to lower dimensional projections of the inputs, as opposed to applying a single mechanism directly to the high dimensional inputs themselves. Formally,

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \left[\text{head}_1; \ldots; \text{head}_h\right], \tag{2.20}$$

$$\text{where} \quad \text{head}_i = \text{Attn}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V), \tag{2.21}$$

and $\boldsymbol{W}_i^Q$, $\boldsymbol{W}_i^K$, and $\boldsymbol{W}_i^V$ are learned projections specific to each head. Other improvements include: stacking multiple attention layers, and using layer normalization (Ba et al., 2016) and residual connections (He et al., 2016). An illustration of a standard Transformer model with all of these improvements is provided in Figure 2.3 for reference.

2.2.4 Massive Neural Language Models (2018-2022)

We now move on to describing the "massive" neural language models that are used in this dissertation. In essence, these models simply combine the ideas presented in the previous section—Transformers and denoising autoencoders—however, they do so at massive scale, with architectures consisting of up to hundreds of billions of parameters trained on terabytes of data.

To begin our discussion of these models, we will first highlight that not all massive neural language models satisfy the traditional definition of a language model presented at the start of this chapter, as they are not all generative models of text. Instead, we need to adopt the following, more generalized definition:

DEFINITION 2 (Denoising Autoencoder Formulation)**.** Provided a sequence of tokens $x \in \mathscr{V}^*$ coming from a vocabulary $\mathscr{V}$ and a corrupted sequence $\tilde{x} \sim q_{\text{noise}}(\tilde{x}|x)$, a *language model* is a denoising autoencoder f_θ, g_ϕ trained to restore the corrupted text, i.e., minimize $\mathcal{L}(x, \tilde{x}')$ for some reconstruction loss $\mathcal{L}$ where $\tilde{x}' = g_\phi(f_\theta(\tilde{x}))$.

Note that, under this definition, the *autoregressive language models* described in Sections 2.2.1 and 2.2.2 are still language models—they can be viewed a denoising autoencoders where corruption is performed by truncating the sequence of tokens and the reconstruction loss is the cross-entropy between the predicted and observed next token. However, this new definition also allows for a wider variety of noise distributions and reconstruction objectives to be used during training. For instance, for the class of *masked language models* (MLMs, e.g., BERT), the input text is corrupted

by masking out and randomly replacing tokens in the input. Similarly, for *sequence-to-sequence language models* (e.g., T5), the input text is corrupted by masking out entire spans. For both of these classes of models, the model is trained to predict the entire original sequence as an output and the reconstruction loss is the cross-entropy between the original and reconstructed sequence. Examples of corrupted inputs for all of the models considered in this work are provided in Table 2.1.

As we mentioned in the introduction, these models are widely used due to their seemingly universal ability to achieve state-of-the-art performance on NLP tasks. Generally speaking, when neural architectures are *pretrained* to perform language modeling on a large corpus, they tend to obtain much better accuracy than randomly initialized models using the same architecture when *finetuned* on downstream tasks (Howard and Ruder, 2018; Peters et al., 2018). Accordingly, the reason that Transformers are used over models such as RNNs, is that, as discussed in Section 2.2.3, their forward passes can be computed in parallel during training, which allows them to be pretrained on considerably larger amounts of data. To provide a sense of how much these models improve performance: a BiLSTM-based model trained from scratch obtains an average accuracy of 65.6% on the GLUE natural language understanding benchmark, finetuning a BiLSTM language model increases the score to 71.0%, and switching to using a transformer-based architecture (i.e., BERT) further increases the score to 82.1% (Wang et al., 2019b; Devlin et al., 2019a).

In the following paragraphs, we will describe the different types of massive neural language models considered in this paper. This information is summarized in Table 2.1.

Autoregressive LMs As the name suggests these models are trained using the autoregressive language modeling objective, i.e., to maximize $p(x_i|x_{<i})$, and these models are fully generative. Accordingly, these models only make use of the decoder portion of the Transformer architecture, and are sometimes referred to as *decoder-only*. The autoregressive massive language models we will study in this paper are GPT-2 (Radford et al., 2018) and GPT-3 (Brown et al., 2020).

Masked LMs Masked language models are trained to reconstruct sequences of tokens that have been corrupted by masking and random replacement.[4] Typically, these models are fed the entire corrupted sequence as an input, and accordingly use only the encoder portion of the Transformer architecture, i.e., are *encoder-only*. As a consequence, these models are typically only used for sequence classification or labeling tasks, as they are not designed to produce variable length outputs.[5]

As a preprocessing step, masked LMs typically insert special delimiter tokens [CLS] and [SEP] at the start of the input and ends of sequences, respectively. The [CLS] token has particular importance in classification settings; masked language models are converted to classifiers by training linear projections into the label space over the representation of the [CLS] token. The masked language models we will study in this paper are BERT (Devlin et al., 2019a), RoBERTa (Liu et al., 2019), and ALBERT (Lan et al., 2020).

Seq2Seq LMs The final class of transformer language models are sequence-to-sequence language models (Seq2Seq LMs) which use both the encoder and the decoder. The training objective of Seq2Seq LMs is similar to that of masked language models in that the model is tasked with reconstructing masked out portions of the input, however, the output is generated autoregressively (using the decoder), entire spans of text may be masked out for Seq2Seq LMs (instead of individual tokens), and the unmasked text does not factor into the reconstruction loss (whereas the reconstruction loss for masked language models is computed over all of the tokens). The only sequence-to-sequence language model we will study in this work is T5 (Roberts et al., 2020).

[4]Some masked language models are also trained on additional auxiliary loss objectives such as next sentence prediction (Devlin et al., 2019a), however the benefit of these objectives is subject to debate (Liu et al., 2019), and so we avoid discussing them here.

[5]This is not to say that masked language model cannot be used to generate text (see Wang and Cho (2019) and Ghazvininejad et al. (2019) for examples), only that they are not "naturally" trained to do so.

Class	Model	Transformer	Noise Distribution	Corrupted Input	Target
Autoregressive	GPT	Decoder-Only	Truncate	The cat in the	hat
Masked	BERT	Encoder-Only	Mask / Replace	[CLS] The [MASK] in the bat [SEP]	The cat in the hat
Seq2Seq	T5	Encoder-Decoder	Span Corruption	The [X] in [Y]	[X] cat [Y] the hat

Table 2.1: **Overview of massive transformer language models considered in this work.** Example corruptions of the input "The cat in the hat" are provided along with the target output of the model.

2.3 Knowledge Bases

The core focus of this dissertation is to diagnose and address knowledge deficiencies in language models. One of the ways we will do this, is by leveraging knowledge bases.

DEFINITION 3 (Knowledge Base)**.** A *knowledge base* is a store of structured or unstructured information that captures knowledge (i.e., facts and information) about the world.

In this section, we will provide a basic overview of the two types of knowledge bases considered in this work: knowledge graphs and textual knowledge bases.

2.3.1 Knowledge Graphs

Knowledge graphs are structured knowledge bases that describe relations between entities. They are typically represented as a collection of triples $\mathcal{KG} = \{(s, r, o) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}\}$, where $\mathcal{E}$ is the set of entities or nodes in the graph, $\mathcal{R}$ is the set of relations or edge types, and (s, r, o) is a directed edge denoting that entities s and o are related by relation r, where s is the subject of the relation and o is the object.

Each edge in the graph captures a known fact, e.g., the fact that "Barack Obama is married to Michelle Obama" may be captured by the edge (Barack Obama, *Spouse*, Michelle Obama). Depending on the set of relations supported, knowledge graphs capture different types of knowledge. For instance, Freebase (Bollacker et al., 2008) and Wikidata (Vrandečić and Krötzsch, 2014) capture factual knowledge about real world entities, e.g., the previous example about Barack Obama. In

contrast, other knowledge graphs such as ConceptNet (Speer et al., 2017) and ATOMIC (Sap et al., 2019) capture commonsense knowledge, e.g., (alcoholic, *desires*, liquor).

The graphical nature of knowledge graphs makes them convenient objects to work with from a computational perspective. For example, they can easily be traversed to find complex "multi-hop" relations between entities, e.g., the fact that "Barack Obama's mother-in-law is Marian Shields Robinson" can be attained by following the path (Barack Obama, *spouse*, Michelle Obama) → (Michelle Obama, *mother*, Marian Shields Robinson). In Chapter 6 we will show how this structure can be leveraged by a generative language model to improve its ability to produce factually correct language about a given entity. However, this convenience comes at a cost; knowledge graphs are sparse relative to other knowledge bases in the sense that they are missing many of the known relations between entities (Nickel et al., 2016). This sparseness has spurred a large body of research into the tasks of *link prediction* and *knowledge graph completion* which develop methods that add new edges to the graph based on the ones already present. A common approach for these tasks is to learn vector representations of the entity and relations that can be used to compute a score for whether or not a relation should exist (Bordes et al., 2013; Balazevic et al., 2019a). In Chapter 6, we will demonstrate how these representations can leveraged by neural language models.

2.3.2 Textual Knowledge Bases

Textual knowledge bases represent knowledge as a set of textual excerpts. Conceptually, they can be viewed as a form of encyclopedia, and in many cases are often constructed from encyclopedia such as Wikipedia (Wikipedia contributors, 2004). In contrast to knowledge graphs, textual knowledge bases are mostly unstructured, although some structure may exist in the form of metadata (e.g., page titles, categories, etc.) and grouping of the text (e.g., into sections, subsections, etc.). While this lack of structure can make textual knowledge bases more difficult to navigate, it can also allow textual knowledge bases to be more expressive and easier to maintain. Although work has also

been conducted into creating neural language models that leverage textual knowledge bases (e.g., Guu et al. (2020) and Lewis et al. (2020)), we do not explore this topic in this work. Instead, in Chapter 8, we explore how neural language models can be used to help enforce consistency in textual knowledge bases as information in the knowledge base is updated.

Part I

Prompting

3

Eliciting Knowledge from Language Models

"Take all of your so-called problems. Better put 'em in quotations. Say what you need to say. Say what you need to say."

– John Mayer, *Say*

The Merriam-Webster dictionary defines knowledge as "the fact or condition of knowing something with familiarity gained through experience or association" (Merriam-Webster, 2022). As language models essentially learn by association during training (i.e., they are trained to model the association between words/phrases and the contexts in which they appear), it is thus reasonable to ask what kinds of knowledge they learn from their training data, and, given a pair of language models, whether one is more knowledgeable than the other. In this chapter we will introduce the method of prompting language models as an approach for diagnosing their knowledge.

3.1 Perplexity: The Traditional Approach

To motivate the need for prompting, we will first discuss the traditional approach for assessing language model performance—estimating perplexity on a held out test corpus. Perplexity (PPL) is a metric from information theory that estimates how well a probabilistic model predicts a collection of samples. For an autoregressive language model, perplexity can be estimated using the formula:

$$\text{perplexity} = \exp\left(-\frac{1}{|x|}\sum_{i=1}^{|x|}\log p(x_i|x_{<i})\right), \tag{3.1}$$

where $x = (x_1, \ldots, x_{|x|})$ is an observed sequence of tokens. Perplexity values range from 1 to $+\infty$ and lower values indicate that a model assigns higher probability to the sequence of tokens. When models are compared using perplexity on a test set, models that achieve lower scores are considered better. Accordingly, one way to compare the relative knowledge of two language models is to take a corpus that contains knowledge, e.g., a textual knowledge base such as Wikipedia, and compare the models' perplexities on it.

Although this is approach is reasonable for determining which model assigns a higher average likelihood to the corpus, it is lacking as a measurement of language model knowledge for a number of reasons:

- *Perplexity comparisons are difficult to interpret.* For instance, on the WikiText-2 benchmark (which is derived from Wikipedia), AWD-LSTM has a perplexity of 65.8 (Merity et al., 2018) while GPT-2 has a perplexity of 18.3 (Radford et al., 2019). It is unclear how these numbers reflect how much more confident we should be that GPT-2 will generate factually correct statements than the AWD-LSTM.

- *Perplexity values are heavily influenced by choice of tokenization scheme.* To illustrate this, a language model with a vocabulary of n tokens that assigns equal probability to each token will always have a perplexity of n. This example illustrates the general trend that perplexities of

language models with larger vocabularies tends to be higher than the perplexities of language models with smaller vocabularies.

- *Perplexity is only well defined for generative models.* Accordingly, it is not directly useful for comparing models such as BERT and T5 that are trained to perform fill-in-the-blanks style tasks as opposed to sentence completion.

In the next section, we will introduce prompting as an alternative approach for assessing language models, and discuss how it addresses these issues.

3.2 Prompting: An Alternative Approach

Instead of assessing knowledge using model likelihood, prompting treats pieces of knowledge as propositional statements, and measures the model's probability that such statements are true. The key idea is to assess language model's knowledge using the same input and output format used during training, by reformulating test instances as fill-in-the-blanks or complete-the-sentence questions. This approach for testing knowledge closely resembles the usage of Cloze tests (Taylor, 1953) to assess human reading comprehension.

The application of prompting requires three components:

1. a *diagnostic dataset* $\mathcal{D} = (x_1^{\text{inp}}, y_1), \ldots, (x_N^{\text{inp}}, y_N)$ that tests for a specific type of knowledge, where each $x_i^{\text{inp}} \subset \mathscr{V}^*$ is an input and each $y_i \in \mathcal{Y}$ is a label,
2. a *template* $\lambda : \mathcal{X} \rightarrow \mathscr{V}^*$ that maps task inputs x^{inp} into *prompts* $x^{\text{prompt}} = \lambda(x^{\text{inp}})$, and
3. a set of *label tokens* $\mathscr{V}_y \subset \mathscr{V}$ representing each label $y \in \mathcal{Y}$.

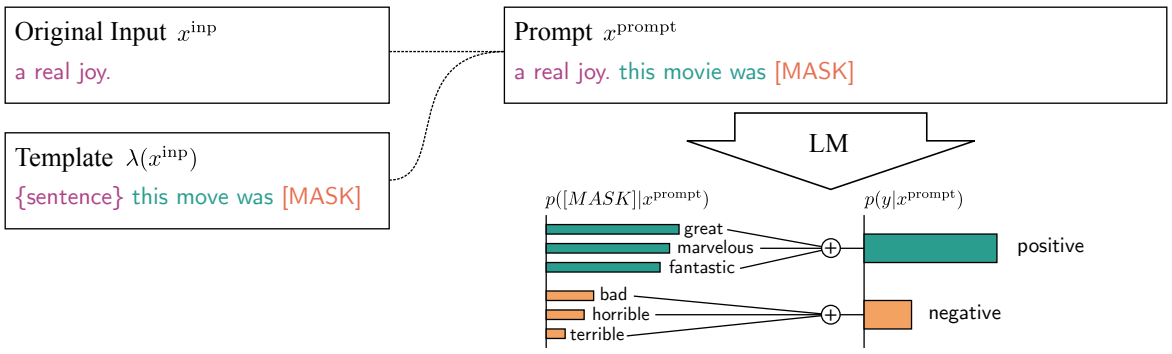


Figure 3.1: **Illustration of prompting** applied to measure a masked language model's ability to perform sentiment analysis. Each input, x^{inp}, is placed into a natural language prompt, x^{prompt}, using a template, λ. Probabilities for each class label, y, are obtained by marginalizing the language model probabilities, $p([\text{MASK}]|x^{\text{prompt}})$, over sets of label tokens. Note that prompting works similarly for autoregressive language models, however class label probabilities are obtained by aggregating over the next token predictions instead of [MASK] predictions.

Provided an input x^{prompt}, we can then obtain label probabilities from the language model by marginalizing over the set of label tokens:

$$p(y|x^{\text{prompt}}) = \sum_{w \in \mathcal{V}_y} p([\text{MASK}] = w|x^{\text{prompt}}) \tag{3.2}$$

and then use these probabilities to measure the model's knowledge using a suitable metric (e.g., accuracy, F_1, hits-at-k, etc.).

Prompting for Task-Specific Knowledge An example illustration of how prompting can be used to measure a masked language model's knowledge of sentiment is provided in Figure 3.1. Provided an instance from the SST-2 (Socher et al., 2013a) sentiment analysis dataset, e.g., x^{inp} = "a real joy.", we use a template to convert it into the masked language model prompt, x^{prompt} = "a real joy. this movie was [MASK]", and then see whether the language correctly assigns a higher probability to words associated with a positive sentiment (e.g., "great," "marvelous,", and "fantastic") than words associated with a negative sentiment (e.g., "bad," "horrible,", and "terrible"). Note that in this

case, this template could be adapted for autoregressive language models by simply removing the mask token from the end of the template.

By aggregating results over the entire SST-2 dataset, we can assess how well different language models know sentiment. For example, we have included results for BERT-base and RoBERTa-large using the template from the previous paragraph in Figure 3.2. These results suggest that RoBERTa has a fairly good in-built knowledge of sentiment (for reference, the original baseline provided in Socher et al. (2013a) has similar accuracy). However, we should be somewhat careful about how we interpret the result for BERT. Although the prompt we tried yielded worse results, we need to entertain the possibility that our template is bad or that we chose bad sets of label tokens. Accordingly, in order to conclude that a model lacks a particular type of knowledge from a negative results, we need to be confident that our prompt is effective. This confidence could be based on intuition, human accuracy on the prompt, or that the prompt has been optimized to best elicit knowledge from the model (we will present a technique for performing such optimization in the next chapter).

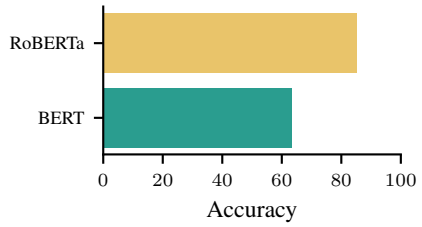


Figure 3.2: **Measurement of BERT-base and RoBERTa-large's knowledge of sentiment** on SST-2 using the prompt from Figure 3.1.

Prompting for Factual Knowledge In addition to using prompts to diagnose whether language models have task-specific knowledge, we also use prompts to diagnose whether language models have factual knowledge by using the Wikidata knowledge graph. Given a relation $r \in \mathcal{R}$, we sample a collection of 100 edges $(s, r, o) \in \mathcal{KG}$ between entities $s, o \in \mathit{entities}$ that are related by r, as well as write a template for that relation that tests whether the language model can predict the tail entity given the head entity. For example, to test for the spouse relation we use the template "{s} is married to". A complete list of templates is provided in Table 3.1, and our label sets consist of the

nation-capital	The capital of {s} is
bithloc	{s} was born in
bithdate	{s} was born on
spouse	{s} is married to
city-state	{s} is a city in the state of
book-author	{s} was written by

Table 3.1: **Prompt templates used in fact completion experiments**.

set of tokens appearing in any alias of the tail entity, e.g., the label token set for "Michelle Obama" is { "Michelle," "LaVaughn," "Robinson," "Obama", "First," "Lady"}.

We report hits-at-k (H@k, $k \in \{1, 5\}$) for the AWD-LSTM and GPT-2 medium language models in Table 3.2, which measures whether a label token appeared in the top-k predictions. We observe that, in general, these language models appear to lack factual knowledge of most relations. In particular, the top predictions of the AWD-LSTM are never correct, and GPT-2 medium performs poorly on all relations except for the city-state relation. In Chapter 6, we will revisit this analysis and show that language models' factual recall can be significantly improved by augmenting language models with the capability to condition on information from external knowledge bases.

Prompting Pros and Cons Observe that prompting addresses many of the issues we raised about perplexity. First, because results are stated in terms of familiar evaluation metrics, they are much easier to interpret. Second, prompting is robust to variation in tokenization; as long as models are capable of generating the label tokens we can assess whether or not the model outputs are correct. Finally, as we've shown in our SST-2 example, prompting can be used to evaluate masked language models in addition to autoregressive language models.

Prompting does however have its own issues. One is that prompting results are not necessarily comparable across different types of language models. In particular, prompting tend to slightly favor masked language models since the input to these models contains context on both the left and right of the [MASK] token as well as information about the length of the prompt. Our formulation

	AWD-LSTM		GPT-2-medium	
	H@1	H@5	H@1	H@5
nation-capital	0	0	6	7
birthloc	0	9	14	14
birthdate	0	25	8	9
spouse	0	0	2	3
city-state	0	13	62	62
book-author	0	2	0	0
Average	0.0	8.2	15.3	15.8

Table 3.2: **Fact completion results**. The table reports hits-at-k (H@k, $k \in \{1, 5\}$) for the AWD-LSTM and GPT-2 language models on our fact completion prompts derived from WikiData.

of prompting is also restrictive in that it does not support label token sets consisting of sequences of tokens. Although supporting label sequences is straightforward for autoregressive language models and sequence-to-sequence language models (since they both are capable of generating variable length outputs), masked language models are difficult to support since they receive a fixed number of [MASK] tokens in their input, and there are many different approaches that have been proposed for decoding from them (Wang and Cho, 2019; Ghazvininejad et al., 2019).

3.3 Note to the Reader

Unlike the remaining chapters in this dissertation, this content in this chapter is not based upon a single publication, but rather compiled from multiple publications. These publications all present slightly different descriptions of prompting, and so this chapter was added to provide a unified overview of the technique and introduce terminology that will be used throughout the rest of this text. Accordingly, this chapter is somewhat shorter than the rest as its purpose is to introduce an idea that is usually described in a few brief paragraphs.

Prompting has received substantial attention from the NLP community during the time that this dissertation was written (Liu et al., 2021a, provides a comprehensive survey). Notably, concurrent

with two works appearing in this dissertation (Logan et al. (2019) and Peters et al. (2019)), Petroni et al. (2019) introduced the LAMA probe, which also uses prompts constructed from knowledge graphs to assess language models' factual knowledge. In the next chapter, we perform our analysis on the LAMA probe as it subsequently became widely used by the community.

3.4 Summary of Contributions

In this chapter, we discussed different methods for evaluating knowledge in language models. In particular, we introduced the method of prompting as an alternative to perplexity that directly measures the degree to which a language model can generate correct answers when provided contexts that require specific forms of knowledge (e.g., factual or task-specific) to complete. The experiments and discussion in this chapter are taken from the following:

- The results measuring autoregressive language models' knowledge of facts from WikiData are from: *Barack's Wife Hillary: Using Knowledge-Graphs for Fact-Aware Language Modeling* (Logan et al., 2019, ACL 2019)
- The general description of prompting and results measuring masked language models' ability to perform sentiment analysis are from: *AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts* (Shin et al., 2020, EMNLP 2020)

The author was a primary contributor on both of these publications, and is responsible for designing all of the experiments described in this chapter. However, the author would also like to specially acknowledge Nelson Liu, who painstakingly performed a manual evaluation of the fact completion results.

4

AutoPrompt

"Won't you please tell me what we've learned. I know it sounds absurd."

– Supertramp, *The Logical Song*

In the previous chapter, we demonstrated the efficacy of prompting as a means of eliciting knowledge from language models. One drawback of prompting is that it requires manually crafting the template to feed into the model. Not only is this time consuming and non-intuitive for many tasks (e.g., textual entailment), more importantly, models are highly sensitive to this template: improperly-constructed templates cause artificially low performance (Jiang et al., 2020). Overcoming the need to manually specify templates would make prompting a more widely useful analysis tool.

In this chapter, we introduce AUTOPROMPT—an *automated* method for generating prompts for any task, illustrated in Figure 4.1. Given a task, e.g., sentiment analysis, AUTOPROMPT creates a prompt by combining the original task inputs (e.g. reviews) with a collection of learned *trigger tokens* that serve as an additional input to the template. The same set of trigger tokens is used for all inputs, and is learned using a variant of the gradient-based search strategy proposed in Wallace et al. (2019). The LM predictions for the prompt are converted to class probabilities by marginalizing

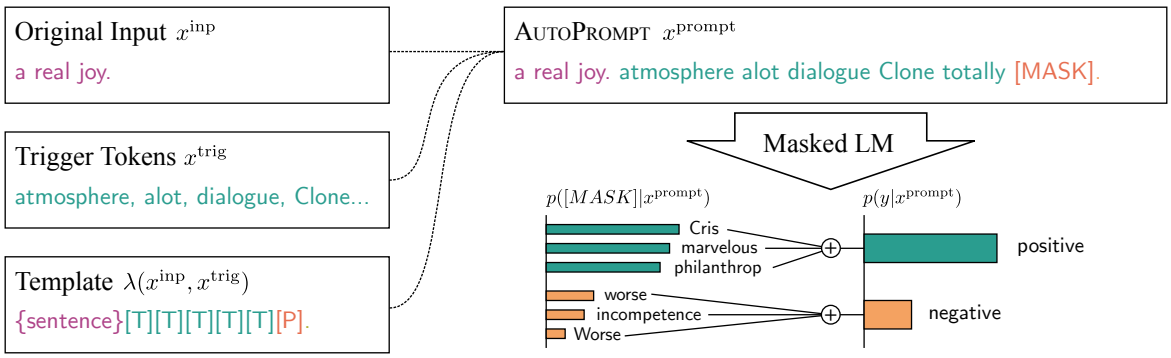


Figure 4.1: **Illustration of AUTOPROMPT** applied to probe a masked language model's (MLM's) ability to perform sentiment analysis. Each input, x^{inp}, is placed into a natural language prompt, x^{prompt}, which contains a single [MASK] token. The prompt is created using a template, λ, which combines the original input with a set of trigger tokens, x^{trig}. The trigger tokens are shared across all inputs and determined using a gradient-based search (Section 4.1.1). Probabilities for each class label, y, are then obtained by marginalizing the MLM predictions, $p([\text{MASK}]|x^{\text{prompt}})$, over sets of automatically detected label tokens (Section 4.1.2).

over a set of associated label tokens, which can either be learned or specified ahead of time, enabling the LM to be evaluated the same as one would any other classifier.

We validate the effectiveness of AUTOPROMPT in numerous experiments. First, we use AUTOPROMPT to construct prompts that test pretrained masked language models (MLMs) on sentiment analysis and natural language inference (NLI). Our tests reveal that, without any finetuning, MLMs perform well on both of these tasks—a properly-prompted RoBERTa achieves 91% accuracy on SST-2 (better than a finetuned ELMo model (Peters et al., 2018)), and 69% accuracy on a balanced variant of the SICK-E dataset (Marelli et al., 2014). Next, we apply AUTOPROMPT to the fact retrieval tasks of LAMA (Petroni et al., 2019), where we are able to construct prompts that more effectively elicit MLM's factual knowledge than existing prompts generated using manual and corpus-mining methods. Concretely, we achieve 43.3% hits-at-1, compared to the current best single-prompt result of 34.1% (Jiang et al., 2020). We also introduce a variant of this task, similar to relation extraction (RE), that tests whether MLMs can extract knowledge from a given piece of text. We show that MLMs can actually *outperform* existing RE models when context sentences with real facts are provided, however, they struggle when context sentences are artificially falsified.

Finally, although the goal of AUTOPROMPT is to analyze models, we find that it provides certain practical advantages over finetuning. First, AUTOPROMPT achieves higher average- and worst-case accuracy than finetuning in low-data regimes. Moreover, unlike finetuning, prompting LMs does not require large amounts of disk space to store model checkpoints; once a prompt is found, it can be used on off-the-shelf pretrained LMs. This is beneficial when serving models for multiple tasks.

4.1 Overview of AUTOPROMPT

Writing prompts is not only time consuming, but it is not clear that the same phrasing will be effective for every model, nor is it clear what criteria determine whether a particular phrasing is the *best* at eliciting the desired information. In light of this, we introduce AUTOPROMPT, a method that constructs customized prompts for a specific task and MLM of interest, to cause the MLMs to produce the desired knowledge.[1] An illustration of AUTOPROMPT is provided in Figure 4.1. The prompt is constructed by taking the original task inputs—a collection of one or more sequences of tokens (e.g., the review in Figure 4.1)—and mapping them to a sequence of tokens using a template. In the following sections, we describe how AUTOPROMPT uses labeled training data to construct prompts, and how it uses the output of the MLM as a prediction for the task.

4.1.1 Gradient-Based Prompt Search

In the previous chapter, we introduced a template-based approach for prompt construction. In this section, we will demonstrated how to extend this method to perform *automatic prompt construction* using an approach based on Wallace et al. (2019). The idea is to add a number of "trigger" tokens that are shared across all prompts (denoted by [T] in the example template in Figure 4.1). These tokens

[1] Although we focus only on MLMs in this work, our method is trivially extendable to autoregressive LMs. The only adjustment is that the predict token must occur at the end of the prompt.

are initialized to [MASK] tokens, and then iteratively updated to maximize the label likelihood (Equation (3.2)) over batches of examples.

Formally, at each step, we compute a first-order approximation of the change in the log-likelihood that would be produced by swapping the jth trigger token x_j^{trig} with another token $w \in \mathcal{V}$. Then we identify a candidate set $\mathcal{V}^{\text{cand}}$ of the top-k tokens estimated to cause the greatest increase:

$$\mathcal{V}^{\text{cand}} = \underset{w \in \mathcal{V}}{\text{top-}k} \left[\mathbf{w}^{\text{in}} \cdot \nabla_{\boldsymbol{x}_j^{\text{trig}}} \log p(y | x^{\text{prompt}}) \right] \tag{4.1}$$

where $\mathbf{w}^{\text{in}}$ is the input embedding of w, and the gradient is taken with respect to the input embedding of x_j^{trig}. Note that computing this candidate set is roughly as expensive as a single forward pass and backward pass of the model (the dot-products require the same amount of multiplications as computing the LM output projection). For each candidate in this set, we then re-evaluate Equation (3.2) on the updated prompt, and retain the prompt with the highest probability in the next step—this requires k forward passes of the model. An example prompt produced by this method for the task of sentiment analysis is shown in Figure 4.1.

4.1.2 Automating Label Token Selection

While in some settings the choice of label tokens is obvious (e.g., when class labels directly correspond to words in the vocabulary), it is less clear what label tokens are appropriate for problems involving more abstract class labels (e.g., NLI). In this section, we develop a general two-step approach to automate the selection of the sets of label tokens $\mathcal{V}_y$. In the first step, we train a logistic classifier to predict the class label using the contextualized embedding of the [MASK] token as input:

$$\boldsymbol{h} = \text{Encoder}(x^{\text{prompt}}) \tag{4.2}$$

We write the output of this classifier as:

$$p(y|\mathbf{h}_i) \propto \exp(\mathbf{h}_i \cdot \mathbf{y} + \beta_y) \tag{4.3}$$

where $\mathbf{y}$ and β_y are the learned weight and bias terms for the label y, and i represents the index of the [MASK] token.

In the second step, we substitute $\mathbf{h}_i$ with the MLM's output word embeddings $\mathbf{w}^{\text{out}}$ to obtain a score $s(y, w) = p(y|\mathbf{w}^{\text{out}})$. Intuitively, because $\mathbf{w}^{out} \cdot \mathbf{h}$ and $\mathbf{y} \cdot \mathbf{h}$ are large for words and labels that are relevant to a particular context, the label probability, $\exp(\mathbf{w}^{\text{out}} \cdot \mathbf{y} + \beta_y)$, should be large for words that are typically associated with a given label. The sets of label tokens are then constructed from the k-highest scoring words:

$$\mathcal{V}_y = \underset{w \in \mathcal{V}}{\text{top-}k} \left[s(y, w) \right] \tag{4.4}$$

4.1.3 Evaluation Setup

In the following sections, we apply AUTOPROMPT to probe $\text{BERT}_{\text{BASE}}$[2] (110M parameters) and $\text{RoBERTa}_{\text{LARGE}}$'s (355M parameters) knowledge of the following tasks: sentiment analysis, natural language inference (NLI), fact retrieval, and relation extraction. We use the PyTorch implementations and pretrained weights provided by the transformers Python library (Wolf et al., 2019). For sentiment analysis and NLI, we find label tokens using the logistic-regression-based heuristic described in Section 4.1.2. For fact retrieval and relation extraction, we skip this step as the labels (entities) directly correspond to tokens in the vocabulary. For all tasks, we perform the prompt search described in Section 4.1.1 for multiple iterations. In each iteration, we use a batch of training data to identify the candidate set $\mathcal{V}^{\text{cand}}$ of replacement trigger tokens. We then evaluate the label likelihoods of the updated prompts on a separate batch of data, and we retain the best trigger token in the next iteration of the search. At the end of every iteration, we measure the label likelihood on

[2]For brevity, we will omit subscripts in the model names.

withheld development data, and return the best prompt found during the entire search as the final output. Performance is evaluated using the appropriate task-specific metrics—e.g., accuracy for sentiment analysis and NLI, and mean reciprocal rank and hits-at-k for fact retrieval—on a separate withheld test set.

4.2 Sentiment Analysis

Sentiment analysis is a fundamental task in NLP, both for natural language understanding research and real-world applications. It is also difficult to probe the extent to which MLMs understand sentiment without finetuning.

Setup We apply our method to convert instances from the binary Stanford Sentiment Treebank (Socher et al., 2013b, SST-2) into prompts, using the standard train/test splits. We find label tokens using a prompt based on the template in Table 4.3. For our gradient-based prompt search, we perform a grid search over the following hyperparameters: $|\mathcal{V}^{cand}| \in \{10, 100\}$, $|\mathcal{V}_y| \in \{1, 3, 5\}$, $|x^{\text{trig}}| \in [3, 6]$.[3] All prompts are initialized with the same template used to find the label set.

We also repeat the results for the prompt from the last chapter, which uses "{sentence} this movie was [P]." as the template, and "terrible" and "fantastic" for the negative and positive label tokens, respectively.

Results We show results in Table 4.1, along with reference scores from the GLUE (Wang et al., 2019b) SST-2 leaderboard, and scores for a linear probe trained over the elementwise average of the LM token representations. Prompts generated by AUTOPROMPT reveal that both BERT and RoBERTa have a strong knowledge of sentiment analysis: without any finetuning, BERT performs comparably to a supervised BiLSTM, and RoBERTa achieves an accuracy on-par with finetuned

[3]Required 2 days to run with 8 NVIDIA 2080Ti GPUs.

Model	Dev	Test
BiLSTM	-	82.8†
BiLSTM + ELMo	-	89.3†
BERT (linear probing)	85.2	83.4
BERT (finetuned)	-	93.5†
RoBERTa (linear probing)	87.9	88.8
RoBERTa (finetuned)	-	96.7†
BERT (manual)	63.2	63.2
BERT (AUTOPROMPT)	80.9	82.3
RoBERTa (manual)	85.3	85.2
RoBERTa (AUTOPROMPT)	91.2	91.4

Table 4.1: **Sentiment analysis** performance on the SST-2 test set of supervised classifiers (top) and fill-in-the-blank MLMs (bottom). Scores marked with † are from the GLUE leaderboard: http://gluebenchmark.com/leaderboard.

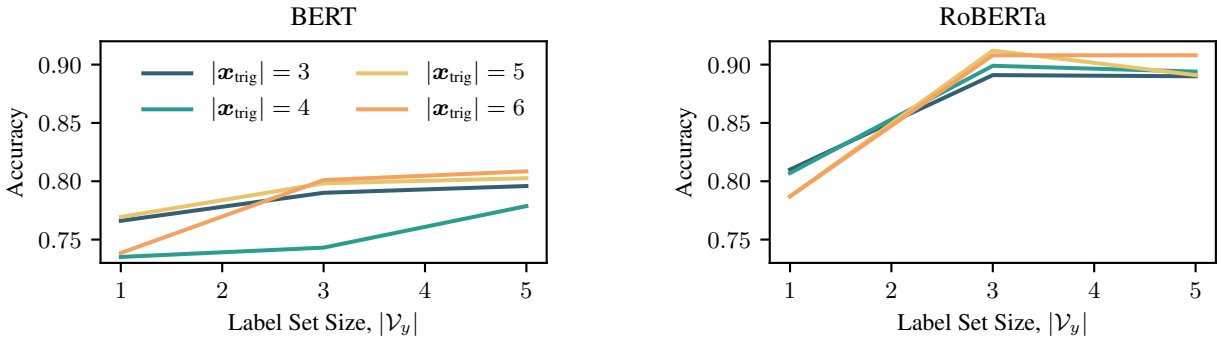
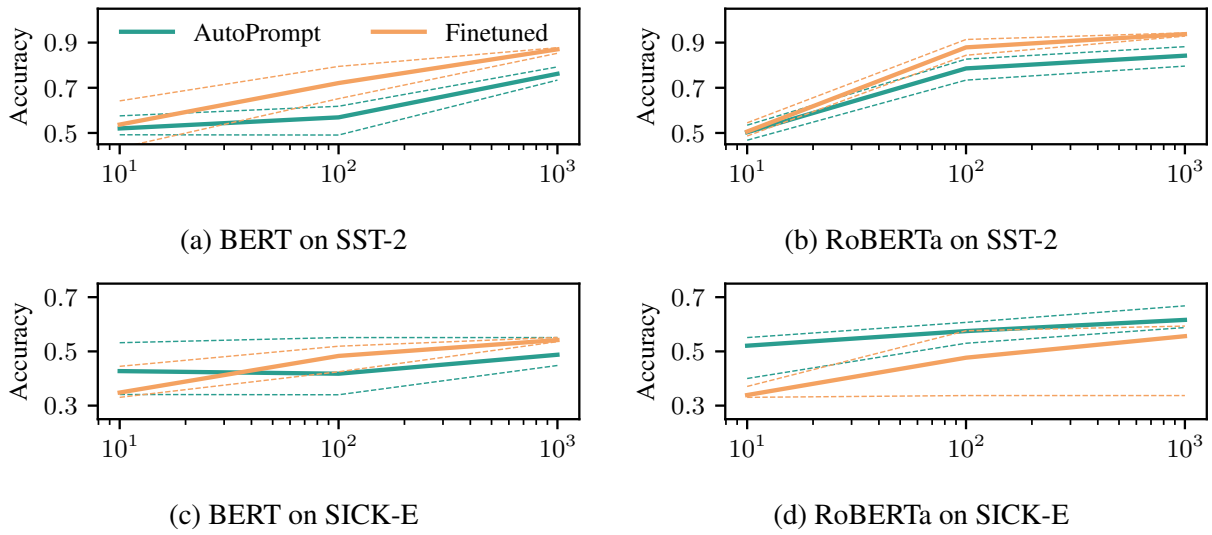


Figure 4.2: **Effect of label and trigger set sizes** on sentiment analysis. The number of candidate replacements is fixed at $|\mathcal{V}_{\text{cand}}| = 100$. Increasing the label set size improves performance, while changing the trigger length does not have much impact.

BERT and ELMo models. In addition, we observe that our automatically constructed prompts are more effective than manual prompts, however, are also typically ungrammatical, e.g., "{sentence} atmosphere alot dialogue Clone totally [P]." We suspect that this lack of grammaticality is due to the fact that AutoPrompt's loss objective does not include any term that accounts for the likelihood of the prompt.

We additionally study the effect of the AUTOPROMPT hyperparameters. We plot the validation accuracy as a function of label set size $|\mathcal{V}_y|$ and the number of trigger tokens $|x^{\text{trig}}|$ in Figure 4.2. We fix the number of candidates at $|\mathcal{V}_{\text{cand}}| = 100$. We observe similar trends when $|\mathcal{V}_{\text{cand}}| = 10$.



(a) BERT on SST-2

(b) RoBERTa on SST-2

(c) BERT on SICK-E

(d) RoBERTa on SICK-E

Figure 4.3: **Effect of training data** on sentiment analysis and NLI for AUTOPROMPT vs. finetuning. X-axis is the number of data points used during training. Error bars plot the max. and min. accuracies observed over 10 independent runs.

Varying the number of trigger tokens generally has little effect. On the other hand, there is a substantial increase in accuracy when increasing the label set size from 1 to 3 (approximately +5% for BERT, and +10% for RoBERTa). After analyzing the label sets, we find that our method generally produces intuitive results—"marvelous" and "philanthrop" are associated with positive sentiment, whereas "worse" and "incompetence" are associated with negative sentiment for RoBERTa.

Accuracy in Low-Data Settings Although the goal of AUTOPROMPT is to probe a model's knowledge, we also find that it may be a viable alternative to finetuning in the low-data regime. To show this, we measure the development set accuracy of AUTOPROMPT prompts when using random subsets of 10, 100, and 1000 instances from the training data. We run our prompt search with $|x^{\text{trig}}| = 10$, $|\mathcal{V}_y| = 3$, and $|\mathcal{V}^{\text{cand}}| = 10$. We compare to the performance of BERT and RoBERTa finetuned on the same data. For fair comparison between AUTOPROMPT and finetuning, we use Mosbach et al. (2021)'s recommended parameters for finetuning on small datasets: trained for 20 epochs, using AdamW (Loshchilov and Hutter, 2019) with bias correction and a learning rate that linearly increases to 2×10^{-5} in the first 10% of iterations, and linearly decreases to 0 afterwards.

Model	SICK-E Datasets		
	standard	3-way	2-way
Majority	56.7	33.3	50.0
BERT (finetuned)	86.7	84.0	95.6
BERT (linear probing)	68.0	49.5	91.9
RoBERTa (linear probing)	72.6	49.4	91.1
BERT (AUTOPROMPT)	62.3	55.4	85.7
RoBERTa (AUTOPROMPT)	65.0	69.3	87.3

Table 4.2: **Natural language inference** performance on the SICK-E test set and variants. (Top) Baseline classifiers. (Bottom) Fill-in-the-blank MLMs.

Experiments are repeated 10 times on random subsets of data (and seeds for the finetuned models). Best-case, worst-case, and average performance are shown in Figure 4.3.

We observe that while finetuning outperforms AUTOPROMPT on sentiment analysis, AUTOPROMPT can perform better than finetuning on NLI. Notably, AUTOPROMPT elicits better average performance from both BERT and RoBERTa given only 10 training examples. Furthermore, results for RoBERTa are more stable across all sample sizes whereas finetuning can result in "failed runs" (consistent with Dodge et al. 2020). This behavior in the low-data regime is an interesting phenomenon, and suggests that there are barriers that MLMs must surmount when they are converted to finetuned classifiers that are not encountered when the task is presented as masked language modeling. We will investigate the application of prompting in few-shot settings in further detail in the next chapter.

4.3 Natural Language Inference

To evaluate the *semantic* understanding of MLMs, we experiment on Natural Language Inference (NLI). NLI is crucial in many tasks such as reading comprehension and commonsense reasoning (Bowman et al., 2015), and it is used as a common benchmark for language understanding.

Task	Prompt Template	Prompt found by AUTOPROMPT	Label Tokens
Sentiment Analysis	{sentence}[T]. . . [T][P].	unflinchingly bleak and desperate Writing academicswhere overseas will appear [MASK].	**pos**: partnership, extraordinary, ##bla **neg**: worse, persisted, unconstitutional
NLI	{prem}[P][T]. . . [T]{hyp}	Two dogs are wrestling and hugging [MASK] concretepathic workplace There is no dog wrestling and hugging	**con**: Nobody, nobody, nor **ent**: ##found, ##ways, Agency **neu**: ##ponents, ##lary, ##uated
Fact Retrieval	*X plays Y music* {sub}[T]. . . [T][P].	Hall Overton fireplacemade antique son alto [MASK] .	
Relation Extraction	*X is a Y by profession* {sent}{sub}[T]. . . [T][P].	Leonard Wood (born February 4, 1942) is a former Canadian politician. Leonard Wood gymnasium brotherdicative himself another [MASK].	

Table 4.3: **Example prompts** by AUTOPROMPT for each task. On the left, we show the prompt template, which combines the input, a number of trigger tokens [T], and a prediction token [P]. For classification tasks (sentiment analysis and NLI), we make predictions by summing the model's probability for a number of automatically selected label tokens. For fact retrieval and relation extraction, we take the most likely token predicted by the model.

Setup We use the entailment task from the SICK dataset (Marelli et al., 2014, SICK-E) which consists of around 10,000 pairs of human-annotated sentences labeled as entailment, contradiction, and neutral. The standard dataset is biased toward the neutral class which represent 56.7% of instances. We also experiment on an unbiased variant with 2-way classification of contradiction vs. entailment (*2-way*), as well as an unbiased 3-way classification variant (*3-way*). The template used for AUTOPROMPT is provided in Table 4.3. We search over the following parameters: $|\mathcal{V}^{cand}| \in \{10, 50\}$, $|\mathcal{V}_y| \in \{1, 3, 5, 10\}$, $|x^{\text{trig}}| \in [1, 5]$, and choose the best prompt according to development set accuracy.

Results Table 4.2 shows that AUTOPROMPT considerably outperforms the majority baseline in all experiments. For example, on the 2-way SICK-E dataset, AUTOPROMPT is comparable to a supervised finetuned BERT. We also test linear probes—linear classifiers trained on top of frozen MLM representations with average pooling —and find AUTOPROMPT has comparable or higher accuracy, despite linear probes being susceptible to false positives. Overall, these results demonstrate that both BERT and RoBERTa have some inherent knowledge of natural language inference.

We also examine the efficacy of AUTOPROMPT in the low-data regime (using the same procedure as SST-2) on the unbiased 3-way SICK-E data. The results in Figure 4.3 show that AUTOPROMPT performs on par with finetuned BERT and significantly better than finetuned RoBERTa in low data settings.

MLMs Excel on Contradiction We find that the label tokens are more interpretable for *contradiction* compared to *entailment* or *neutral* (examples in Table 4.3). We investigate if this hurts the model performance on entailment and neutral classes. We measure the precision for each label in the 3-way balanced SICK-E dataset. BERT achieves 74.9%, 54.4%, and 36.8% precision for contradiction, entailment, and neutral cases, respectively, while RoBERTa obtains 84.9%, 65.1%, and 57.3%. These results suggest that AUTOPROMPT may be more accurate for concepts that can be easily expressed using natural label tokens.

4.4 Fact Retrieval

In this section, we will revisit the question of whether pretrained MLMs *know* facts about real-world entities. The LAMA dataset (Petroni et al., 2019) evaluates this using cloze tests that consist of (sub, rel, obj) triples, e.g. (Obama, bornIn, Hawaii), and *manually* created prompts with missing objects, e.g. "Obama was born in [MASK].". LPAQA (Jiang et al., 2020) extends this idea by *systematically*

creating prompts that are generated by mining Wikipedia, paraphrasing, and crowdsourcing. In this section, we use the same cloze-style setup but *automatically* generate prompts in order to better evaluate the factual knowledge of MLMs. We compare our approach against LAMA and LPAQA, which are explicitly designed for the task of fact retrieval.

Setup We reformulate fact retrieval by mapping (sub,rel,obj) triples to a prompt using the template "{sub}[T]...[T][P].", where the trigger tokens are specific to the relation, rel, and the correct object, obj, is the label token. We use the original test set from LAMA (Petroni et al., 2019), henceforth *Original*. To collect training data for AUTOPROMPT, we gather at most 1000 facts for each of the 41 relations in LAMA from the T-REx dataset (Elsahar et al., 2018). For the relations that still have less than 1000 samples, we gather extra facts straight from Wikidata. We ensure that none of the T-REx triples are present in the test set, and we split the data 80-20 into train and development sets. Moreover, because the collected T-REx data is from a slightly different distribution than the LAMA test set, we also consider a separate evaluation where we split the T-REx triples into a 60-20-20 train/dev/test split and evaluate on the test set. This *T-REx* dataset is used to measure the performance of our prompts when the train and test data is from the same distribution.

We use AUTOPROMPT with 5 or 7 tokens, and select the search parameters using the T-REx development set. We prevent proper nouns and tokens that appear as gold objects in the training data from being selected as trigger tokens. This is done to prevent AUTOPROMPT from "cheating" by embedding common answers inside the prompt. To evaluate, we observe the rank of the true object in label token distribution of the MLM, and use standard ranking metrics: mean reciprocal rank (MRR), hits-at-1 (H@1), and hits-at-10 (H@10).

Results Table 4.4 shows the performance of MLMs with different prompting methods, and we show qualitative examples in Table 4.3. Prompts generated using AUTOPROMPT can extract factual knowledge from BERT more effectively than their manual and mined counterparts: we improve

Prompt Type	Original			T-REx		
	MRR	H@10	H@1	MRR	H@10	H@1
LAMA	40.27	59.49	31.10	35.79	54.29	26.38
LPAQA (Top1)	43.57	62.03	34.10	39.86	57.27	31.16
AUTOPROMPT 5 Tokens	53.06	72.17	42.94	54.42	70.80	45.40
AUTOPROMPT 7 Tokens	53.89	73.93	43.34	54.89	72.02	45.57

Table 4.4: **Fact retrieval.** We evaluate BERT on fact retrieval using the *Original* LAMA dataset from Petroni et al. (2019). For all three metrics (mean reciprocal rank, mean hits-at-10 (H@10), and mean hits-at-1(H@1)), AUTOPROMPT significantly outperforms past prompting methods. We also report results on a *T-REx* version of the data (see text for details). We compare BERT versus RoBERTa on a subset of the LAMA data using AUTOPROMPT with 5 tokens.

Model	MRR	H@10	H@1
BERT	55.22	74.01	45.23
RoBERTa	49.90	68.34	40.01

Table 4.5: **BERT versus RoBERTa** on a subset of the LAMA data using AUTOPROMPT with 5 tokens.

H@1 by up to 12 points. Moreover, despite AUTOPROMPT using only one prompt per relation, it still outperforms LPAQA's ensemble method (which averages predictions for up to 30 prompts) by approximately 4 points. Using 7 trigger tokens achieves slightly higher scores than 5 trigger tokens, although the difference is not substantial. This indicates that our approach is stable to the choice of trigger length, which is consistent with our sentiment analysis results. Overall, these results show that AUTOPROMPT can retrieve facts more effectively than past prompting methods, thus demonstrating that BERT contains more factual knowledge than previously estimated.

Relation Breakdown We also provide a detailed breakdown of the prompts found by Petroni et al. (2019) and AUTOPROMPT, and their associated accuracies in Table 4.6. Manual prompts are competitive when the prompt is *easy* to specify, e.g., the prompt *"was born in"* for the PLACE OF BIRTH relation. On the other hand, AUTOPROMPT performs especially well for relations that are difficult to specify in a natural language prompt. For example, Petroni et al. (2019)'s prompt for the POSITION PLAYED ON TEAM relation is "{sub} plays in [MASK] position", which is not as specific

Relation	Method	Prompt	H@1
P101	Manual	[X] works in the field of [Y]	11.52
	AUTOPROMPT BERT	[X] probability earliest fame totaled studying [Y]	15.01
	AUTOPROMPT RoBERTa	[X] 1830 dissertation applying mathsucci [Y]	0.17
P103	Manual	The native language of [X] is [Y]	74.54
	AUTOPROMPT BERT	[X]PA communerug speaks proper [Y]	84.87
	AUTOPROMPT RoBERTa	[X]neau optionally fluent!?ťraditional [Y]	81.61
P106	Manual	[X] is a [Y] by profession	0.73
	AUTOPROMPT BERT	[X] supporters studied politicians musician turned [Y]	15.83
	AUTOPROMPT RoBERTa	[X] (), astronomers businessman·former [Y]	19.24
P127	Manual	[X] is owned by [Y]	36.67
	AUTOPROMPT BERT	[X] is hindwings mainline architecture within [Y]	47.01
	AUTOPROMPT RoBERTa	[X] picThom unwillingness officially governs [Y]	39.58
P1303	Manual	[X] plays [Y]	18.91
	AUTOPROMPT BERT	[X] playingdrum concertoative electric [Y]	42.69
	AUTOPROMPT RoBERTa	[X]Trump learned soloKeefe classical [Y]	44.44
P136	Manual	[X] plays [Y] music	0.7
	AUTOPROMPT BERT	[X] freaking genre orchestra fiction acid [Y]	59.95
	AUTOPROMPT RoBERTa	[X] blends postwar hostage drama sax [Y]	52.97
P1376	Manual	[X] is the capital of [Y]	81.11
	AUTOPROMPT BERT	[X] boasts native territory traditionally called [Y]	63.33
	AUTOPROMPT RoBERTa	[X] limestone depositedati boroughDepending [Y]	28.33
P178	Manual	[X] is developed by [Y]	62.76
	AUTOPROMPT BERT	[X] is memory arcade branding by [Y]	64.45
	AUTOPROMPT RoBERTa	[X] 1987 floppy simulator users sued [Y]	69.56
P20	Manual	[X] died in [Y]	32.07
	AUTOPROMPT BERT	[X] reorganizationotype photographic studio in [Y]	33.53
	AUTOPROMPT RoBERTa	[X].. enigmatic twentieth nowadays near [Y]	31.33
P27	Manual	[X] is [Y] citizen	0.0
	AUTOPROMPT BERT	[X] m³ badminton pieces internationally representing [Y]	46.13
	AUTOPROMPT RoBERTa	[X] offic organise forests statutes northwestern [Y]	42.07
P276	Manual	[X] is located in [Y]	43.73
	AUTOPROMPT BERT	[X] consists kilograms centred neighborhoods in [Y]	44.64
	AUTOPROMPT RoBERTa	[X] manoeuv constructs whistleblowers hills near [Y]	37.47
P279	Manual	[X] is a subclass of [Y]	31.04
	AUTOPROMPT BERT	[X] is î adequately termed coated [Y]	55.65
	AUTOPROMPT RoBERTa	[X],formerly prayers unstaceous [Y]	52.55
P37	Manual	The official language of [X] is [Y]	56.89
	AUTOPROMPT BERT	[X]inen dialects resembled officially exclusively [Y]	54.44
	AUTOPROMPT RoBERTa	[X]onen tribes descending speak mainly [Y]	53.67
P407	Manual	[X] was written in [Y]	60.21
	AUTOPROMPT BERT	[X] playedić every dialect but [Y]	69.31
	AUTOPROMPT RoBERTa	[X] scaven pronunciation.*Wikipedia speaks [Y]	72.0
P413	Manual	[X] plays in [Y] position	0.53
	AUTOPROMPT BERT	[X] played colors skier ↔ defensive [Y]	41.71
	AUTOPROMPT RoBERTa	[X]," (), ex-,Liverpool [Y]	23.21

Table 4.6: **Examples of manual prompts and prompts generated via AUTOPROMPT** for Fact Retrieval.

as the relation requires. Although the prompt from AUTOPROMPT is not grammatical ("*{sub} ediatric striker ice baseman defensive {obj}*"), it does contain tokens that are directly related to sports.

BERT Outperforms RoBERTa We finally directly compare BERT and RoBERTa. To do so, we subsample the LAMA test set to consist of examples where the object is a single token for both BERT and RoBERTa (*Original-RoBERTa*).[4] BERT actually slightly outperforms RoBERTa, and we find that the prompts generated for RoBERTa tend to contain more irrelevant words (see Table 4.6). For example, the prompt generated by RoBERTa for the PLAYS INSTRUMENT relation contains words such as "Trump" and symbols such as "," ()," for the POSITION PLAYED ON TEAM relation. It is surprising that RoBERTa does not perform better than BERT, and it is worthy of investigating this further in future work. Additionally, recall that prompting is a *lower bound* on a model's knowledge: the lower relative performance does not mean that the model actually knows less.

4.5 Relation Extraction

Apart from evaluating whether MLMs *know* facts, it is also important to evaluate whether they can *extract knowledge* from text. In this section, we use the task of relation extraction (RE)—to identify how entities are related in a given sentence—an important task in information extraction. We create RE prompts in a similar fashion as fact retrieval: for a given triple (subj,rel,obj) and sentence, sent, that expresses this relation, we construct a prompt as "{sent}{sub}[T]...[T][P].", where the trigger tokens are specific to the relation, and label token is the correct object obj (see Table 4.3 for an example).

[4]The original dataset consists of examples where the object is a single token for BERT.

Setup We use the T-Rex dataset for RE because each T-REx fact comes with context sentences that mention the subject and object surface forms. We compare AUTOPROMPT to LAMA and LPAQA (their prompts are still useful here), as well as a recent supervised relation extraction model (Sorokin and Gurevych, 2017) that was also used by Petroni et al. (2019). To make the evaluation fair for the supervised RE model, we modify the standard RE evaluation. We give the model credit as long as it does not predict a different relation for the subject and object, i.e. we ignore the "no relation" prediction and all other relations. We also drop all sentences from evaluation for which the model's named entity extractor failed to identify the subject and the object as entities. For the evaluation of all systems, we treat a prediction as correct if it is either the canonical version of the object (e.g., "USA") or the rendered surface form (e.g., "American") for *any* of the context sentences in a given triple.

Results Table 4.8 shows the results for BERT and RoBERTa. MLMs can extract relational information *more effectively* than the supervised RE model, providing up to a 33% increase on the task when using AUTOPROMPT. RoBERTa also outperforms the supervised RE model, although it is worse than BERT (likely for similar reasons as we outline in Section 4.4). For both BERT and RoBERTa, we notice that the trigger tokens consist of words related to their corresponding relations (see Table 4.7 for full list), e.g. RoBERTa selects "defy trademarks of namesake manufacturer" for relation MANUFACTURER/PRODUCER OF PRODUCT.

Perturbed Sentence Evaluation A possible explanation for the strong results of MLMs in the RE setting is that they may *already* know many of the relations. Thus, they may directly predict the objects instead of *extracting* them. To separate this effect, we synthetically perturb the relation extraction dataset by replacing each object in the test data with a random other object and making the same change to the prompt. For example, *"Ryo Kase (born November 9, 1974 in ~~Yokohama~~→Yorkshire) is a Japanese actor"* where Ryo Kase is the subject, Yokohama is the

Relation	Model	Context and Prompt	Prediction
P103 (native language)	BERT	Alexandra Lamy (born 14 October 1971) is a French actress. Alexandra Lamy speaks airfield dripping % of [MASK].	French
P36 (capital)	RoBERTa	Kirk was born in Clinton County, Ohio, and he entered service in Wilmington, Ohio. Clinton County famously includes the zoo influencing [MASK].	Wilmington
P530 (diplomatic relation)	BERT	The Black Sea forms in an east-west trending elliptical depression which lies between Bulgaria, Georgia, Romania, Russia, Turkey, and Ukraine. Ukraine qualified some immigration actually entered [MASK].	Russia
P106 (occupation)	RoBERTa	Spencer Treat Clark (born September 24, 1987) is an American actor who has appeared in several films, including Gladiator, Mystic River, and Unbreakable. Spencer Treat Clark famously the famously handsome the [MASK].	Hulk
P276 (location)	BERT	The Immortal Game was a chess game played by Adolf Anderssen and Lionel Kieseritzky on 21 June 1851 in ~~London~~Seoul, during a break of the first international tournament. The Immortal Game locatedstered regardless streets in [MASK].	Seoul
P176 (manufacturer)	RoBERTa	The Honda Civic del Sol is a 2-seater front-engined, front wheel drive, targa top car manufactured by ~~Honda~~Toyota in the 1990s. Honda Civic del Sol defy trademarks of namesake manufacturer [MASK].	Toyota
P279 (subclass of)	BERT	Mizeria is a Polish ~~salad~~sandwich consisting of thinly sliced or grated cucumbers, often with sour cream though in some cases oil. Mizeria is calls direcend altitude [MASK].	food
P463 (member of)	RoBERTa	~~Rush~~Aerosmith was a Canadian rock band consisting of Geddy Lee (bass, vocals, keyboards), Alex Lifeson (guitars), and Neil Peart (drums, percussion, lyricist). Alex Lifeson affiliatedalach the internationally initials [MASK].	Kiss

Table 4.7: **Examples prompts generated using AUTOPROMPT for relation extraction.** Underlined words represent the gold object. The bottom half of the Table shows examples of our augmented evaluation where the original objects (represented by crossed-out words) are replaced by new objects.

original object, and Yorkshire is the new object. We regenerate the prompts using the perturbed version of the data.

The accuracy of the RE model does not change significantly on the perturbed data (Table 4.8), however, the accuracy of the MLMs decreases significantly. This indicates that a significant portion

Model	Original	Perturbed
Supervised RE LSTM	57.95	58.81
BERT (LAMA)	69.06	28.02
BERT (LPAQA)	76.55	30.79
BERT (AUTOPROMPT)	90.73	56.43
RoBERTa (AUTOPROMPT)	60.33	28.95

Table 4.8: **Relation extraction:** We use prompts to test pretrained MLMs on relation extraction. Compared to a state-of-the-art LSTM model from 2017, MLMs have higher mean hits-at-1 (H@1), especially when using prompts from AUTOPROMPT. We also test models on sentences that have been edited to contain incorrect facts. The accuracy of MLMs drops significantly on these sentences, indicating that their high performance stems from their factual knowledge.

of MLM accuracy comes from background information rather than relation extraction. Nevertheless, our prompts for BERT outperform their LAMA and LPAQA counterparts, which provides further evidence that AUTOPROMPT produces better probes.

4.6 Discussion

Prompting as an Alternative to Finetuning The goal of prompting a language model is to probe the knowledge that the model acquired from pretraining. Nevertheless, prompting has some practical advantages over finetuning for solving real-world tasks. First, as shown in Section 4.2, prompts generated using AUTOPROMPT can achieve higher accuracy than finetuning in the *low-data regime*. Moreover, prompting has advantages over finetuning when trying to solve *many different tasks* (e.g., the many users of the OpenAI GPT-3 API (Brown et al., 2020)). In particular, finetuning requires storing large language model checkpoints for each individual task, and drastically increases system cost and complexity because it requires deploying many different models at the same time. Prompting alleviates both of these issues. Only prompts are stored for each individual task, while the same pretrained model is used across all of the tasks.

Limitations of Prompting There are certain phenomena that are difficult to elicit from pretrained language models via prompts. In our preliminary evaluation on datasets such as QQP (Iyer et al., 2017) and RTE (Dagan et al., 2005), prompts generated manually and with AUTOPROMPT did not perform considerably better than chance. However, we cannot conclude that BERT does not know paraphrasing or entailment from these results. In general, different probing methods have different tasks and phenomena they are suitable for: AUTOPROMPT makes *prompt-based probes* more generally applicable, but, it still remains just one tool for diagnosing knowledge in language models.

Limitations of AUTOPROMPT One downside of AUTOPROMPT is that it requires labeled training data. Although this is also required for other probing techniques (e.g., linear probing classifiers), manual prompts rely on domain/language insights instead of labeled data. Compared to human-designed prompts, AUTOPROMPT generated prompts lack interpretability, which is similar to other probing techniques, such as linear probing classifiers. Another limitation of AUTOPROMPT is that it can sometimes struggle when the training data is highly imbalanced. For example, in Sections 4.3 and 4.4 we show that the prompts often just increase the likelihood of the majority label. Rebalancing the training data can help to mitigate this problem. Finally, due to the greedy search over the large discrete space of phrases, AUTOPROMPT is sometimes brittle; we leave more effective crafting techniques for future directions.

4.7 Summary of Contributions

In this chapter, we introduced AUTOPROMPT, an approach for automatically constructing prompts. We found that automatically generated prompts are better at eliciting knowledge from language models than manually written ones, thereby establishing that language models know even more than we were able to show in the previous chapter. Notably, in our sentiment analysis experiments

we saw that prompted language models can be competitive with finetuned models without requiring any finetuning of model weights, and in our NLI experiments we also saw evidence that prompting may be more stable in low data regimes. The text in this chapter is based on the publication:

- *AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts* (Shin et al., 2020, EMNLP 2020)

which has 175 citations at the time of writing, and has spurred a considerable amount of follow-up work investigating further applications of prompting, including the material presented in the next chapter.

Primary authorship of the source publication is shared between the author of this dissertation, Taylor Shin, and Yasaman Razeghi. The author played an advisory role in early stages of the project, and proposed the approach for performing label token selection, however credit for the idea to apply universal adversarial triggers to prompt search belongs to Sameer Singh and Eric Wallace. The author also takes credit for writing the majority of the code used to run experiments, as well as running the sentiment analysis experiments, and writing a substantial portion of the text. Yasaman Razeghi deserves credit for running the NLI experiments, and Taylor Shin for running the fact retrieval and relation extraction experiments. The results presented in this chapter can be replicated using the code and instructions provided at: https://github.com/ucinlp/autoprompt.

5

Few-Shot Learning with Prompts

"If you had, one shot, or one opportunity, to seize everything you ever wanted, in one moment, would you capture it, or just let it slip?"

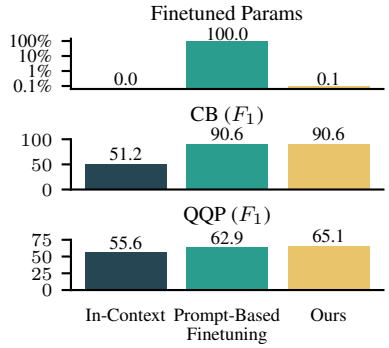
– Eminem, *Lose Yourself*

In the previous two chapters, we observed that prompted language models can innately perform some classification tasks without requiring any parameter updates. For example, in Chapter 3, we were able to obtain 85% accuracy performing sentiment analysis on SST-2 using prompt templates crafted using only intuition, and, in Chapter 4, we were able to further improve this number to 91% using AUTOPROMPT. We additionally observed that these methods can be data efficient: virtually zero training data is needed to write prompts using intuition, and AUTOPROMPT performed substantially better than finetuning on SICK-E using only dozens of training examples. These results suggest that prompting may be an efficient approach for improving language model performance in few-shot (i.e., low data) settings. In this chapter, we explore this hypothesis, seeking to address efficiency along three fronts: 1) prompt search efficiency, 2) parameter efficiency, and 3) data efficiency.

We improve prompt search efficiency by identifying a class of simple prompts that are effective across many tasks for masked language models (LMs). We find that, when using an approach called prompt-based finetuning (Schick and Schütze, 2021a; Gao et al., 2021, Section 5.1.2), the prompt requires less optimization than previously thought; in fact, the prompt template and training examples can be completely cut out (e.g., Figure 5.1, right). *Null prompts*—simple concatenations of the inputs and the [MASK] token—achieve comparable accuracy to manually-written templates while drastically simplifying prompt design: users only need to decide the label tokens (a.k.a. the verbalizer) and where to place the [MASK] token. The effectiveness of null prompts also challenges the notion that the success of few-shot learning is due to inductive biases present in the prompt.

A key drawback of prompt-based finetuning is that it has large memory requirements for each new downstream task at inference time (Figure 5.1, left). In contrast, in-context learning (Brown et al., 2020) allows reusing the large-scale LM across tasks, but it requires significant prompt engineering. To determine whether memory efficiency and simple prompt selection can be *simultaneously* achieved, we experiment with either: 1) making prompts for in-context learning similarly easy to create, or 2) making prompt-based finetuning more memory efficient. For 1), we simplify prompt engineering for in-context learning by automatically tuning the prompt's tokens or embeddings, an approach that has been successful in the non-few-shot setting (Shin et al., 2020; Lester et al., 2021). For 2), we study lightweight finetuning alternatives that update a smaller set of parameters: BitFit (Ben-Zaken et al., 2021), Adapters (Houlsby et al., 2019), and calibration layers (Zhao et al., 2021).

We show that the latter approach—prompt-based finetuning with lightweight updates—is considerably more successful. In particular, learning only the model's bias terms (BitFit) can achieve competitive or better few-shot accuracy than standard finetuning while only requiring switching out 0.1% of the parameters at inference time to perform different tasks. On the other hand, automated prompt tuning for in-context learning generally fails to find prompts that are competitive with manual ones. Taken together, our results show that prompt-based finetuning is preferable because it



In-Context	{What does it feel like to be on Xanax?}$_1$ and {Do 4mg Xanax bars exist?}$_2$ have different meanings. {How do you know if you're unconditionally in love with someone?}$_1$ and {How do you know if you're in love with someone and might only be denying the fact to yourself?}$_2$ have similar meanings. {Will GST affect the price level in India?}$_1$ and {Will GST effect the price level in India?}$_2$ have [MASK] meanings.
Prompt-Based Finetuning	{Will GST affect the price level in India?}$_1$? [MASK] , I want to know {Will GST effect the price level in India?}$_2$
Null Prompts (Ours)	{Will GST affect the price level in India?}$_1$ {Will GST effect the price level in India?}$_2$ [MASK]

Figure 5.1: **Different methods of few-shot learning.** Right: We visualize different types of prompts for QQP. We denote the input fields using curly brackets {}, the manually-written prompt template using magenta, and the label tokens using green. We show that *null prompts*, ones that do not contain training examples or task-specific prompt templates, can achieve competitive accuracy. Left: We compare different methods for model finetuning. Unlike standard prompt-based finetuning, we propose to update only the masked LM's bias terms (BitFit). This achieves competitive accuracy while only updating 0.1% of the parameters.

is more accurate, works well for different types of prompts, and can be made nearly as efficient as using frozen LMs.

5.1 Prompting Language Models

In this chapter, we study masked LMs for few-shot learning. Recalling our setup from Chapter 3, we have:

- A pre-trained masked LM, with $\mathscr{V}$ denoting its vocabulary and $\mathscr{V}^*$ the set of all token sequences.
- A small set of training inputs $x_i \in X \subset \mathscr{V}^*$ and their corresponding labels $y_i \in Y$.
- A *template* $\lambda : X \to T^*$ that maps inputs to prompts containing a single [MASK] token. Additionally, a collection of *label tokens* $\mathscr{V}^y \subset T$ associated with each label y.

Method	Finetuned Params	Prompt Design	Few-shot
AUTOPROMPT (Shin et al., 2020)	None	Learned (Discrete)	✗
Prompt Tuning (Lester et al., 2021)	Prompt Token Embeds	Learned (Continuous)	✗
OPTIPROMPT (Zhong et al., 2021)	Prompt Token Embeds	Learned (Continuous)	✗
Soft Prompts (Qin and Eisner, 2021)	All Contextualized Embeds	Learned (Continuous)	✗
GPT-3 (Brown et al., 2020)	None	Manual	✓
PET (Schick and Schütze, 2021a)	All	Manual	✓
LM-BFF (Gao et al., 2021)	All	Learned (Discrete)	✓
P-Tuning (Liu et al., 2021b)	All + Prompt Token Embeds	Learned (Continuous)	✓
Null Prompts + Bitfit (This Chapter)	Bias Terms	None	✓

Table 5.1: **Overview of existing work on prompting**. *Finetuned Params* indicates the parameters altered during training. *Prompt Design* indicates how prompts are created; we use *null prompts*. *Few-Shot* indicates using few-shot training and validation sets.

We consider different ways of constructing the templates (Section 5.1.1) and updating the masked LM's parameters (Section 5.1.2). Table 5.1 contains an overview of contemporary prompting methods and the settings they are evaluated in.

5.1.1 Constructing the Prompt

In some settings, different prompts can cause accuracy to vary from near chance to near state-of-the-art (Zhao et al., 2021). However, finding good prompts can be difficult. Prompt construction requires a non-trivial combinatorial search over the prompt's wording, whether to include training examples, or how to convert LM probabilities to class predictions. As a consequence, prompts are either designed using human intuition that is hard to replicate and apply in a principled manner (Perez et al., 2021), or using automated methods (e.g., AUTOPROMPT). These methods search for elements such as: (1) the text of the prompt template, (2) the label tokens, and (3) whether and how training examples are prepended before the test input. Although automated prompt search can match the accuracy of manual tuning, it introduces its own complexities. For example, the prompts from Gao et al. (2021) achieve comparable results to manually-designed prompts but are found using

generative models and careful validation. In this chapter, we show that prompt-based finetuning (see Section 5.1.2) can considerably reduce the importance of the prompt.

5.1.2 Prompting Approaches for Few-Shot Learning

In-Context Learning An increasingly popular strategy for few-shot learning is prompting frozen LMs (Brown et al., 2020). This strategy relies solely on *in-context learning* (a.k.a. priming), where the LM learns by conditioning on the prompt rather than updating its parameters. In-context learning has been shown to be successful when using very large (e.g., billions of parameters) LMs, as these models better leverage the prompt.

Prompt-Based Finetuning Rather than using frozen LMs, *prompt-based finetuning* methods finetune all of the LM's parameters (Schick and Schütze, 2021a; Le Scao and Rush, 2021; Gao et al., 2021). For masked LMs, this is done by constructing training examples that contain a [MASK] token and finetuning the masked LM to generate the correct label token in that position.

The main advantage of prompt-based finetuning over in-context learning is that it achieves higher accuracy, especially when the LM is relatively small, e.g., millions of parameters (Schick and Schütze, 2021b). The main downside is that the same model can no longer be reused across different tasks, thus reducing efficiency. The efficiency is impacted in two ways. First, it requires large amounts of disk space at test time because numerous model checkpoints must be stored. Second, during training time, it requires large amounts of GPU memory to perform updates on massive LMs.

In this chapter, we will show an additional benefit to prompt-based finetuning—it makes prompt engineering easier. We will also show that the memory inefficiency of prompt-based finetuning can be drastically mitigated using lightweight finetuning alternatives. These lightweight methods allow one to switch out only a small subset of model parameters at inference time in order to solve multiple tasks, and also drastically reduce training-time memory costs. Moreover, in many cases

these lightweight methods also improve model accuracy. Our work is related to Le Scao and Rush (2021), who show that different manually-written prompt templates lead to similar accuracy for prompt-based finetuning.

5.2 Experimental Setup

5.2.1 Datasets and Hyperparameter Tuning

We use the following classification datasets from GLUE (Wang et al., 2019b) and SuperGLUE (Wang et al., 2019a): BoolQ, CB, MNLI, MRPC, QNLI, QQP, RTE, and SST-2.[1]

To build few-shot datasets, past work collects K examples from each label for training and K examples from each label for development (Gao et al., 2021). Despite this setup often being denoted as K-shot learning, it effectively uses $2K$ examples and splits the examples evenly into train and development. We instead propose to use cross validation to perform more principled model selection. Concretely, we sample $2K$ examples from each label and use 4-fold cross validation to determine the best hyperparameters. After finding the best hyperparameters, we train on the first K examples and early stop on the second K examples. We use $K = 16$ following past work (Gao et al., 2021).

We sample our examples from each dataset's original training set. Since transformers' performance in few-shot settings can be highly dependent on weight initialization (Dodge et al., 2020), we initialize the weights with 10 different random seeds and report the mean and variance of the model performance. We use each dataset's original development set for our final evaluation and use the standard evaluation metrics (accuracy or F_1) associated with each dataset. We do not check the final evaluation metrics during any tuning of the hyperparameters to ensure that we are doing "true" few-shot learning (Perez et al., 2021).

[1] We also evaluated on WiC and WNLI. We omit these results because all models achieved near-random accuracy.

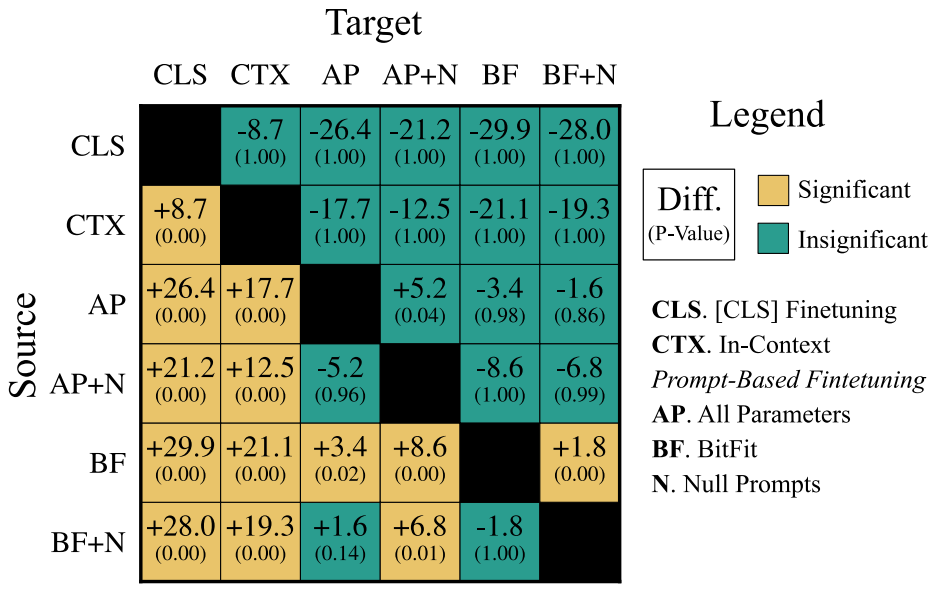


Figure 5.2: **How # Wins are computed**. For a given dataset, we perform a Welch's *t*-test to determine if there is a significant difference in accuracy for each pair of methods. The method which performs better than most other methods (i.e., the row with the most yellow squares; BitFit in this case) is considered the "winner" of the task, and its *# Wins* is incremented by 1. In the figure above, we show a subset of methods evaluated on a single dataset.

5.2.2 Masked Language Models

Following Schick and Schütze (2021b), we use the RoBERTa (large, 330M params, Liu et al., 2019) and ALBERT (xxl-v2, 223M params, Lan et al., 2020) masked LMs provided by the HuggingFace transformers library (Wolf et al., 2019). Training and evaluation were performed on a heterogeneous compute cluster with the following minimum specs: 2xNVIDIA GeForce GTX 1080 Ti's, 8-core Intel Core i7 CPU, 64 GB RAM.

5.2.3 Comparing Few-Shot Methods by # Wins

The results for different few-shot learning methods can be quite different across datasets and seeds for the training set (Zhao et al., 2021; Schick and Schütze, 2021a). To compare different methods at a high level, we use a metric denoted as *# Wins*: the number of datasets that a given method

performs significantly better than all other methods on. We compute this metric for a given dataset by first performing a Welch's t-test to determine if there is a significant difference in accuracy for each pair of methods. The method which performs better than most other methods is considered the "winner" of the task and its *# Wins* is incremented by 1. There are multiple winners in the case of a tie. See Figure 5.2 for illustration.

5.3 Simplifying Prompt Engineering

In this section, we run prompt-based finetuning and ablate different elements of the prompt. We consider the following ablations:

- **Manual Prompt (Prior)**: We use manually-written prompts from Schick and Schütze (2021a,b), and Gao et al. (2021). We show the prompt templates and label tokens in Appendix A.1.
- **Manual Prompt (w/o Engineering)**: We simulate standard prompt design by manually writing one prompt for each task using our intuition. We show the prompts in Appendix A.2.
- **Prompt Tuning**: Inspired by Liu et al. (2021b) and Lester et al. (2021), we use the prompt templates from Manual Prompt (Prior) but randomly initialize the embeddings of the prompt tokens and learn them using gradient-based optimization. This ablates the gains from human-designed prompts.
- **Null Prompt**: We use the same label tokens as Manual Prompt (Prior) but use a prompt template that consists of only the input fields and a [MASK] token (Appendix A.3). This ablates the prompt template entirely.
- **Random Label Tokens**: We use the same prompt template as Manual Prompt (Prior) but—following Opitz (2019) and Le Scao and Rush (2021)—select random label tokens. This ablates the gains from human-chosen label tokens.

- **Null Prompt + Random Label Tokens**: We use both null prompts and random label tokens.

In all cases, we finetune all of the masked LM parameters. We show the accuracy of the above prompts as well as traditional finetuning (using a [CLS] token and a classification head) in Figure 5.3.[2]

Manual Prompts Perform Best The manually-written prompts from prior work perform best on average for both models. On the other hand, our manual prompts (w/o Engineering) are noticeably worse than the ones from prior work and are outperformed by many other methods.

Null Prompts Are Competitive In many cases, prompt tuning and null prompts perform comparably to manually-written prompts, especially for RoBERTa. For instance, both of these methods outperform manual prompts (w/o Engineering) in terms of # Wins. These results are exciting from a practical perspective as they show that one can achieve competitive few-shot results without resorting to any tuning of the prompt.

From an analysis perspective, these results also show that effective few-shot learning can be accomplished without any inductive bias from a manually-written prompt template. In fact, combining null prompts with random label tokens, which involves no human design at all, still significantly outperforms standard [CLS] finetuning for numerous tasks (3 for RoBERTa and 5 for ALBERT at $p = 0.05$). This shows that some of the effectiveness of prompt-based finetuning is due to its basic setup, i.e., predicting on a [MASK] token with an MLM head.

Null Prompts or Prompt Tuning? Both null prompts and prompt tuning achieve competitive results without resorting to manual prompt design. We advocate for using null prompts over prompt tuning because they are easier to use. Null prompts only require choosing which order to concatenate

[2]For fair comparison we use the finetuning recommendations of Mosbach et al. (2021) to improve stability.

the input fields and the [MASK] token. Prompt tuning requires choosing the number of embeddings, their placement, their initialization, etc.

Null Prompts Simplify Prompt Search One complication that arises in standard prompt-based finetuning is that prompts become a hyperparameter of the finetuning procedure, and have a combinatorially large search space. On the other hand, determining the concatenation order for null prompts is trivial by just trying all of the few possible options and choosing which one works best on the validation set. To see this, in Figure 5.4 we plot the accuracy on the few-shot development set and the full test set for different concatenation orders for RoBERTa on MNLI.[3] The development and test accuracy is strongly correlated ($R^2 = 79.05$), which demonstrates that tuning the concatenation order is easy even when validation data is scarce.

Impact of Dataset Size We next investigate whether the observations made in the previous paragraphs hold across different dataset sizes. Intuitively, when the amount of data is small, manual prompts may outperform other approaches because the inductive bias provided by the prompt has the most impact when there is little data to learn the task at hand. In Figure 5.5 we compare the accuracy of prompt-based finetuning using manually-written prompts and null prompts to traditional finetuning, using the same setup described in Section 5.2.1 but varying $K \in \{4, 8, 16, 32\}$. Although there is some instability at lower values of K, we find that the accuracy of both prompt-based finetuning approaches tends to be similar, and is either substantially better or on-par with traditional finetuning. In other words, null prompts are competitive with manual prompts, even when K is small.

5.4 Achieving Simplicity *and* Efficiency

Thus far, we have shown that prompt-based finetuning can simplify prompt engineering at the cost of memory inefficiency—a new set of parameters must be learned for each task. This is in contrast to in-context learning, which holds all model weights fixed but is heavily influenced by small prompt modifications (Zhao et al., 2021; Lu et al., 2021). In this section, we investigate how to achieve *both* memory efficiency and simple prompts. Concretely, in Section 5.4.1 we try to simplify prompt engineering for in-context learning by tuning the prompt, and in Section 5.4.2, we reduce the number of learned parameters for prompt-based finetuning.

5.4.1 Simplifying In-Context Learning With Prompt-Only Tuning

Here, we try to make prompt engineering for in-context learning as simple as prompt-based finetuning by automatically finding the prompt. Concretely, we focus on the emerging class of methods that do *prompt-only tuning*: learning the prompt while keeping the rest of the model fixed (Shin et al., 2020; Lester et al., 2021). We consider:

- **AUTOPROMPT**: Following Shin et al. (2020), we search for discrete tokens to use in the input instead of manually-designed prompt templates. Search is performed using the original hyperparameters.
- **Prompt Tuning (Short)**: We use the same prompt tuning approach described in the previous section but we keep the masked LM fixed.
- **Prompt Tuning (Long)**: Based on the advice of Lester et al. (2021), we increase the number of learned prompt embeddings to 20 in order to expand the learning capacity.

[3]We use MNLI because the concatenation order has a large impact on performance.

For reference, we also report the results from prompt-based finetuning with null prompts. We show the results for RoBERTa in Figure 5.6. We find that only tuning the prompt is relatively unsuccessful. First, on average it fails to match the performance of manually-designed prompts. Second, all methods struggle to match the accuracy of prompt-based finetuning. In fact, for many of the datasets, prompt-only methods perform worse by a wide margin (e.g., 40% absolute difference in F_1 score on CB). This shows that finetuning masked LMs in the few-shot setting leads to substantially higher accuracy than prompt-only tuning.

Our Results versus Recent Prompt Tuning Work We find that only tuning the prompt performs substantially worse than finetuning the entire LM. This is in contrast to recent work, which argues that prompt-only tuning is competitive with finetuning (Lester et al., 2021; Li and Liang, 2021). We believe these are not contradictions but rather differences in the models and settings. Li and Liang (2021) focus on *left-to-right* LMs for *generation* tasks, whereas we focus on masked LMs for classification tasks. They also finetune additional parameters in intermediate layers of the model. These differences may explain the difference in prompting accuracies. Moreover, Lester et al. (2021) show that prompt-only tuning becomes less competitive as models get smaller; we use even smaller models than evaluated in their work. Consequently, although we find that finetuning a masked LM is superior to prompt-only tuning, there may be other settings in which they fare similarly.

5.4.2 Memory-Efficient Finetuning

Given the inadequacies of prompt-only tuning, we next study if prompt-based finetuning can be made memory-efficient. To do so, we focus on reducing the number of trainable parameters, taking inspiration from recent work in the non-few-shot setting. The benefits of these methods is that they: (1) reduce storage costs at test time when running many tasks (one can store only the modified parameters for each task), and (2) reduce memory costs at training time, as fewer

		BoolQ (acc)	CB (F_1)	MNLI (acc)	MRPC (F_1)	QNLI (acc)	QQP (F_1)	RTE (acc)	SST-2 (acc)	Wins (#)
RoBERTa	In-context	49.2	51.2	48.0 / 48.1	28.0	55.2	55.6	60.7	84.1	0
	[CLS] finetuning	51.0	74.3	39.4 / 38.6	**77.8**	58.2	61.9	54.5	72.9	1
	Prompt-based Finetuning									
	All Parameters	63.9	**90.6**	66.5 / 61.6	74.1	57.4	62.9	**68.8**	**92.6**	3
	+ Null Prompt	59.9	**91.2**	61.6 / 57.8	**76.1**	65.8	**65.9**	54.6	83.8	3
	BitFit	**66.7**	**89.8**	**69.3 / 70.0**	69.7	62.3	**66.3**	64.9	**92.1**	6
	+ Null Prompt	**67.2**	**90.6**	67.5 / 62.9	68.2	**66.4**	65.1	65.4	89.6	3
ALBERT	In-context	68.0	19.9	35.4 / 35.2	20.7	50.1	0.3	53.1	49.1	0
	[CLS] finetuning	53.3	56.5	36.0 / 38.6	**76.9**	66.6	**58.5**	54.1	62.9	2
	Prompt-based Finetuning									
	All Parameters	**73.5**	**91.1**	**65.0 / 56.0**	**75.2**	**73.9**	**59.9**	**61.4**	**93.2**	8
	+ Null Prompt	53.7	**89.4**	58.2 / 53.7	**78.5**	67.3	**62.0**	59.2	91.5	3
	BitFit	**77.2**	**86.7**	**64.6 / 61.6**	**79.7**	**73.1**	**61.4**	58.6	**92.0**	8
	+ Null Prompt	52.8	**86.3**	55.3 / 58.0	65.5	63.8	52.7	57.2	89.7	1

Table 5.2: **Final few-shot results** from representative methods. Wins are computed on a per-datasets basis and the "winners" of the different approaches are highlighted in bold. Prompt-based finetuning significantly outperforms in-context learning and traditional [CLS] finetuning, even without any tuning of the prompt (*null prompt*). Moreover, prompt-based finetuning can be highly memory efficient using bias-only finetuning (*BitFit*). We show matched and mismatched results for MNLI.

optimized parameters means much smaller statistics in optimizers like Adam. We consider four lightweight finetuning methods:

- **Adapters**: We use Adapters (Houlsby et al., 2019), neural networks layers that are inserted between the feedforward portion of the Transformer architecture. We use the default Adapters hyperparameters from Houlsby et al. (2019) ($\approx 10^7$ parameters per task).
- **BitFit**: Following Ben-Zaken et al. (2021), we only update the bias terms inside the Transformer ($\approx 10^5$ parameters per task).
- **LM Head Tuning**: We update the embeddings in the MLM output layer that are associated with the label tokens ($\approx 10^3$ parameters per task).
- **Calibration**: Following Zhao et al. (2021), we learn an affine transformation on top of the logits associated with the label tokens ($\approx 10^1$ parameters per task).

We run prompt-based finetuning for each method with the prompts from Manual Prompts (Prior). We also report the accuracy of finetuning all of the parameters for reference.

Results We show the results in Figure 5.7. There are diminishing returns as the parameter count is increased. In particular, substantial gains are made when going from calibration to LM head tuning to BitFit, however, there is either a marginal improvement or even a decrease in performance when going to Adapters or All Parameters. The BitFit method provides the best accuracy-efficiency trade-off, and even outperforms finetuning all of the parameters in terms of # Wins. This suggests that updating all of the LM's hundreds of millions of parameters on only 16 data points is suboptimal.

5.4.3 Putting Everything Together

We finally combine null prompts and memory-efficient finetuning. We show the results from this method, as well as the other best few-shot methods, in Table 5.2. Overall, we recommend finetuning with null prompts and BitFit: it achieves competitive accuracy, is simple to set up, and introduces small memory costs for each new task.

5.5 Summary of Contributions

In this chapter, we studied how to best apply prompting in few-shot settings. We found that prompt-based finetuning is not only effective, but also fairly robust to variations in the prompt template. In fact, there is a simple class of prompts—null prompts—that can be flexibly applied to different tasks without degrading performance relative to manually-written and learned prompts. We additionally demonstrated that prompt-based finetuning can be made memory efficient: finetuning only the bias terms (BitFit) achieves comparable or better accuracy than finetuning all the parameters while being 1000x more memory efficient. Taken together, using null prompts with BitFit is an approach that

is efficient, simple-to-tune, and competitive in accuracy. The text in this chapter is based on the publication:

- *Cutting Down on Prompts and Parameters: Simple Few-Shot Learning with Language Models* (Logan IV et al., 2021a, ACL Findings 2022)

which has 25 citations at the time of writing.

The author of this dissertation was the primary author on this publication, and is primarily responsible for the decision to investigate few-shot learning scenarios, and null prompts as a means to ablate the effect of manually written prompts. The author also wrote the majority of code used to run and analyze experiments, and shares credit with Sebastian Riedel and Fabio Petroni for actually running the experiments.

Although prompt tuning was concurrently proposed in a number of works (see Table 5.1), the idea was novel at the time that work on this chapter began, and credit for it belongs to Ivana Balažević, Sebastian Reidel, and Sameer Singh. Ivana Balažević also deserves acknowledgment for her active role running experiments the early stages of this project. The results presented in this chapter can be replicated using the code and instructions provided at: https://github.com/ucinlp/null-prompts.

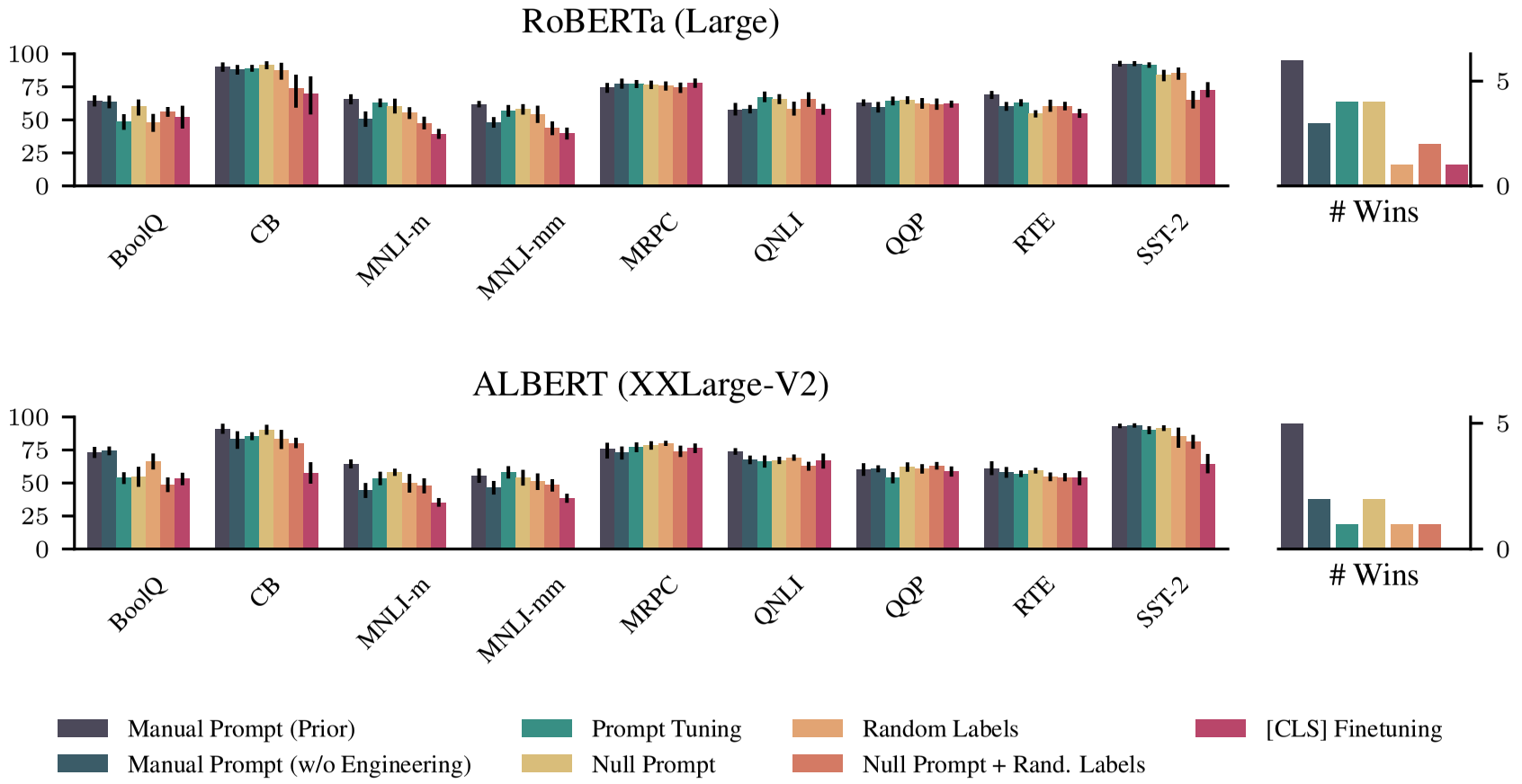


Figure 5.3: **Simplifying the selection of prompts**. We apply prompt-based finetuning in conjunction with six different types of prompts. We report accuracy or F_1 on each dataset. Manually-designed prompts from prior work achieve the best accuracy but require manual tuning on validation sets. On the other hand, null prompts and prompt tuning both perform competitively without requiring any tuning of the prompt template.

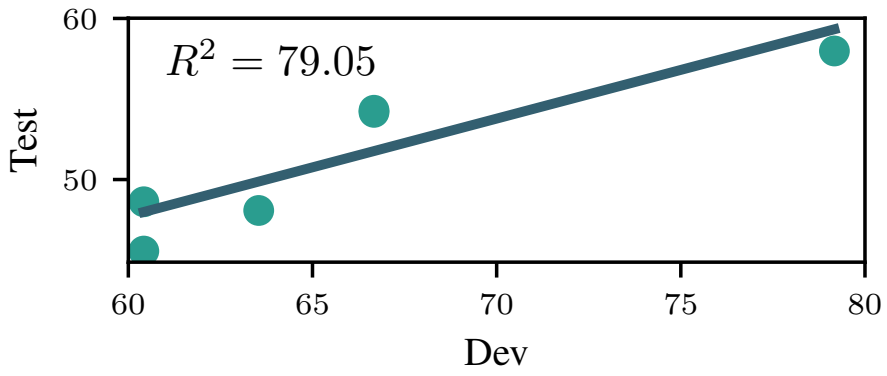


Figure 5.4: **Correlation of development and test set performance of null prompts on MNLI**. The only decision to make when using null prompts is which order to concatenate the mask token and the input fields. One can choose the best option using a tiny held-out development set. We show the results for MNLI, with the few-shot development set accuracy on the x-axis.

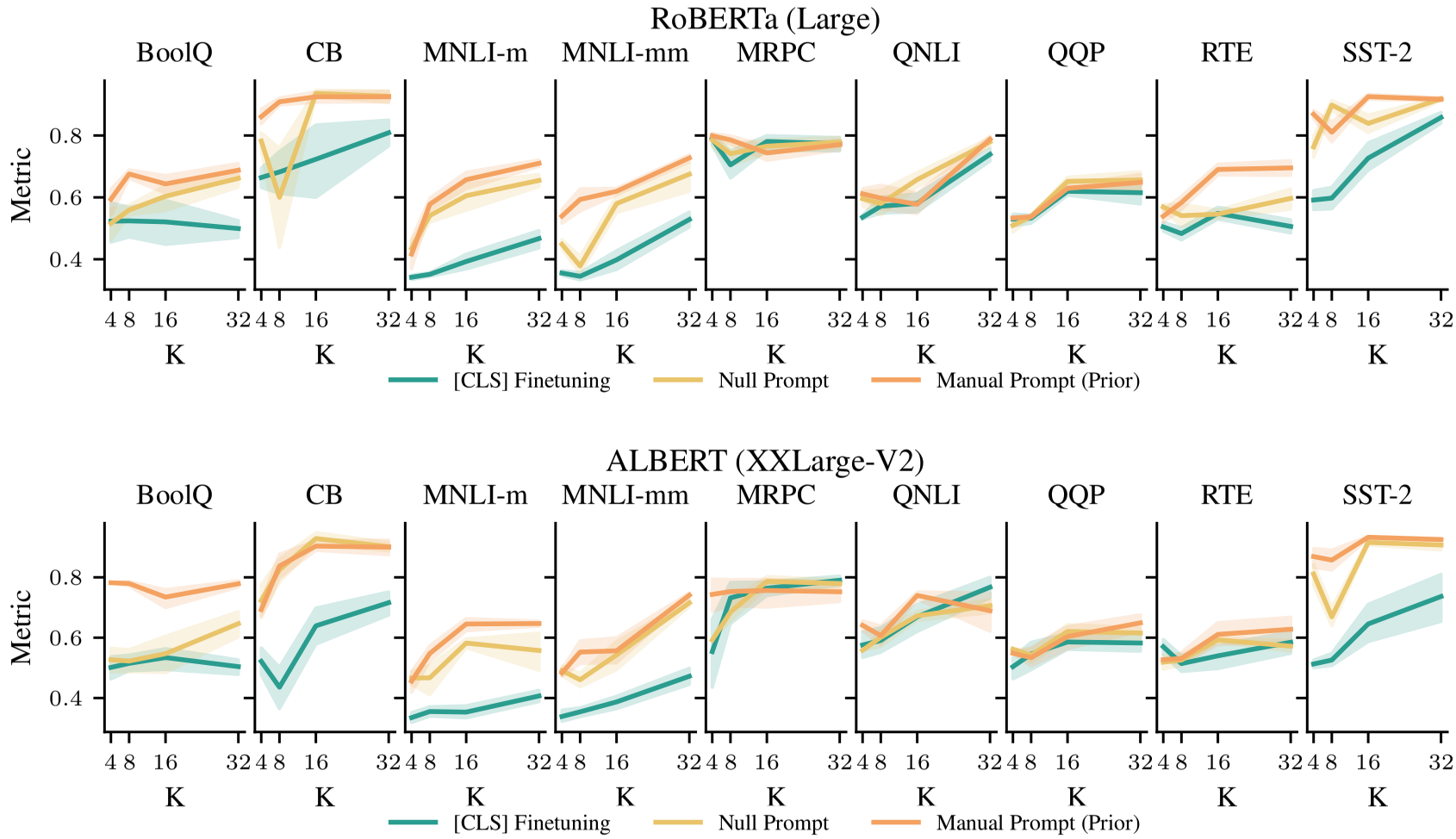


Figure 5.5: **Impact of dataset size**. We plot learning curves for $K \in \{4, 8, 16, 32\}$. Shaded regions indicate the range of performance across 10 different random seeds. In general, we find that as K increases the accuracy of prompt tuning with null prompts tends to be close to that of manual prompts, and substantially better than traditional finetuning.

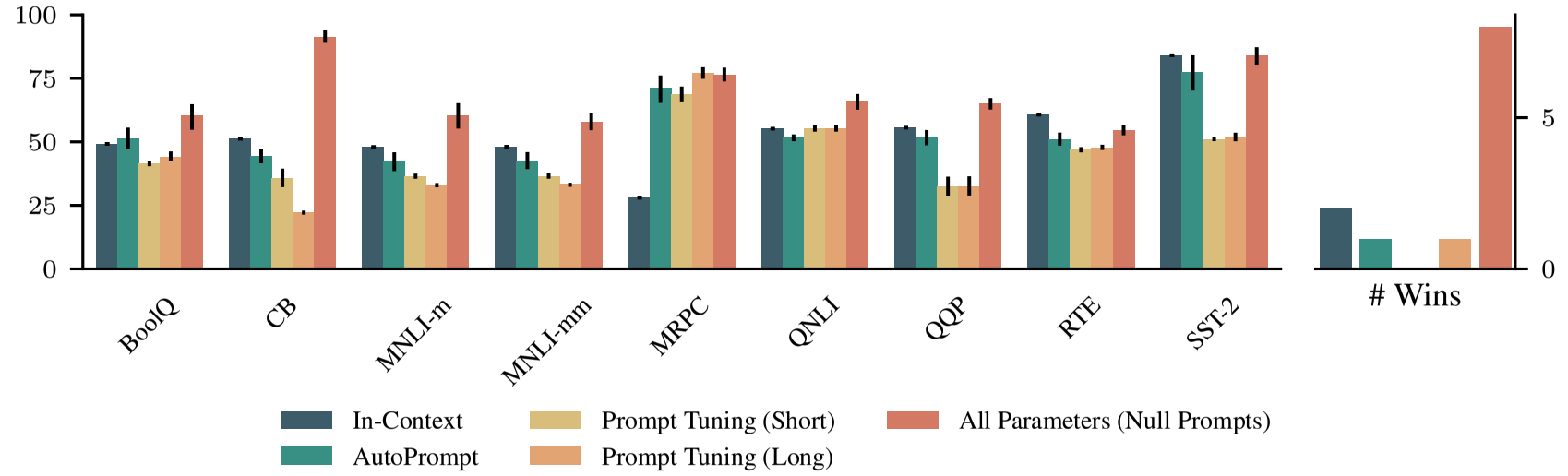


Figure 5.6: **Prompt-only tuning**. We try to simplify prompt engineering for in-context learning (i.e., using frozen models) by directly learning the prompt. The performance (accuracy/F_1) for prompt-only tuning is substantially lower than finetuning the LM parameters for RoBERTa-large. Thus, we recommend finetuning over in-context learning in the few-shot setting.

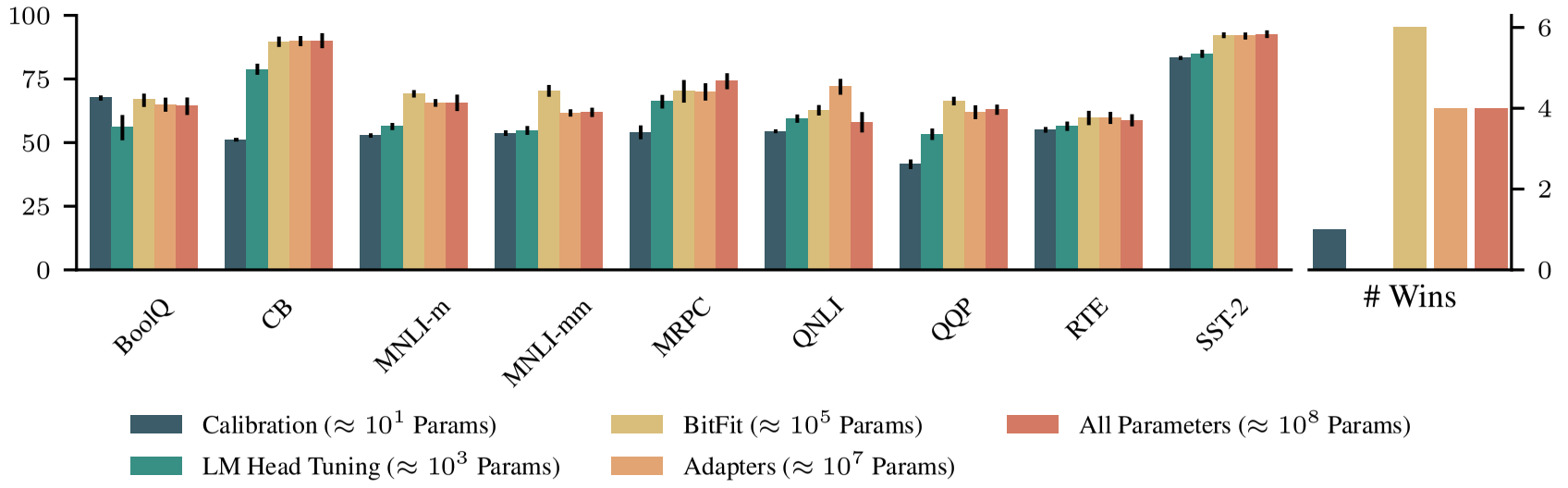


Figure 5.7: **Parameter-efficient prompt-based finetuning**. We perform prompt-based finetuning using different lightweight finetuning schemes. We show the accuracy or F_1 on each dataset for RoBERTa-large. BitFit achieves the highest accuracy on average and only modifies 0.1% of the parameters.

Part II

Integrating Knowledge Bases and Language Models

6

Leveraging Knowledge Bases to Improve Language Models

How 'bout ya go check your facts and Wiki-wik-wik-Wikipedia that!

– BitByBitBot, *Vinylly (Wiki-Wikipedia That)*

In Chapter 3, we observed that language models appear to struggle with factual knowledge. One of the driving factors behind this is that language models, at best, can only memorize facts observed during training. Furthermore, even when facts have been seen before, models can still fail to generate correct statements. For instance, when conditioned on the text at the top of Figure 6.1, an AWD-LSTM language model (Merity et al., 2018) trained on *Wikitext-2* assigns higher probability to the word "*PlayStation*" than "*Game Boy*", even though this sentence appears verbatim in the training data.

In this chapter, we investigate whether factual knowledge can be "injected" into a language model, by allowing the model to condition on representations of information in knowledge graphs and textual knowledge bases. We introduce two different approaches: 1) the knowledge graph language

model (KGLM), that iteratively traverses and incorporates information from a knowledge graph into an autoregressive language model, and 2) KnowBERT, that incorporates information from an entity linker into a masked language model. We show that leveraging these external sources of knowledge substantially improves language models' ability to complete prompts requiring factual knowledge.

6.1 The Knowledge Graph Language Model

In this section, we introduce the *knowledge graph language model* (KGLM), a neural language model with mechanisms for selecting and copying information from an external knowledge graph. The KGLM maintains a dynamically growing *local knowledge graph*, a subset of the knowledge graph that contains entities that have already been mentioned in the text, and their related entities. When generating entity tokens, the model either decides to render a new entity that is absent from the local graph, thereby growing the local knowledge graph, or to render a fact from the local graph. When rendering, the model combines the standard vocabulary with tokens available in the knowledge graph, thus supporting numbers, dates, and other rare tokens.

Figure 6.1 illustrates how the KGLM works. Initially, the graph is empty and the model uses the entity Super Mario Land to render the first three tokens, thus adding it and its relations to the local knowledge graph. After generating the next two tokens ("*is*", "*a*") using the standard language model, the model selects Super Mario Land as the parent entity, *Publication Date* as the relation to render, and copies one of the tokens of the date entity as the token ("*1989*" in this case).

To train the KGLM, we collect the distantly supervised *Linked WikiText-2* dataset. The underlying text closely matches *WikiText-2* (Merity et al., 2017b), a popular benchmark for language modeling, allowing comparisons against existing models. The tokens in the text are linked to entities in Wikidata (Vrandečić and Krötzsch, 2014a) using a combination of human-provided links and off-the-shelf linking and coreference models. We also use relations between these entities in Wikidata

[*Super Mario Land*] is a [*1989*] [*side-scrolling*] [*platform video game*] developed and published by [*Nintendo*] as a [*launch title*] for their [*Game Boy*] [*handheld game console*].

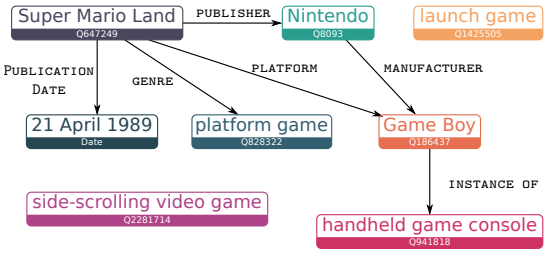


Figure 6.1: ***Linked WikiText-2* example**. A localized knowledge graph containing facts that are (possibly) conveyed in the sentence above. The graph is built by iteratively linking each detected entity to Wikidata, then adding any relations to previously mentioned entities. Note that not all entities are connected, potentially due to missing relations in Wikidata.

to construct plausible reasons for why an entity may have been mentioned: it could either be related to an entity that is already mentioned (including itself) or a brand new, unrelated entity for the document.

We train and evaluate the KGLM on *Linked WikiText-2*. When compared against AWD-LSTM, a recent and performant language model, KGLM obtains not only a lower overall perplexity, but also a substantially lower *unknown-penalized* perplexity (Ueberla, 1994; Ahn et al., 2016), a metric that allows fair comparisons between models that accurately model rare tokens and ones that predict them to be *unknown*. We also compare *factual completion* capabilities of these models, where they predict the next word after a factual sentence (e.g., "*Barack is married to* ___") and show that KGLM is significantly more accurate. Lastly, we show that the model is able to generate accurate facts for rare entities, and can be *controlled* via modifications the knowledge graph.

6.1.1 Problem Setup and Notation

At a high level, the KGLM architecture augments an LSTM language model:

$$p(x_t|x_{<t}) = \text{softmax}(\boldsymbol{W}_h \boldsymbol{h}_t + \boldsymbol{b}), \tag{6.1}$$

$$\boldsymbol{h}_t = \text{LSTM}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_{t-1}), \tag{6.2}$$

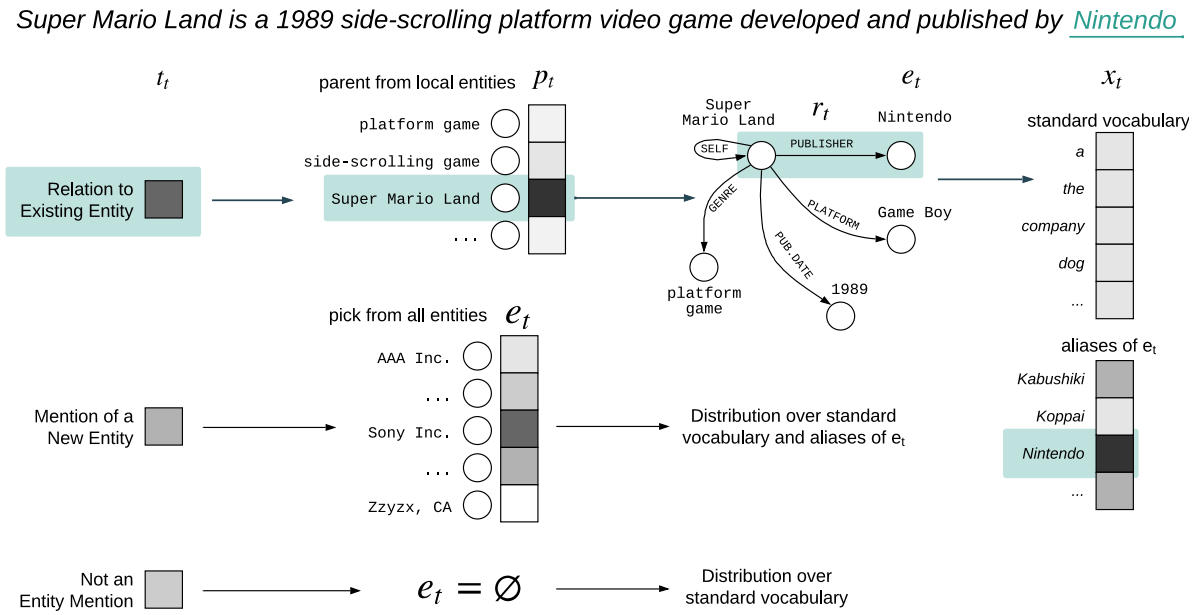


Figure 6.2: **KGLM illustration.** When trying to generate the token following "*published by*", the model first decides the type of the mention (t_t) to be a related entity (darker indicates higher probability), followed by identifying the parent (p_t), relation (r_t), and entity to render (e_t) from the local knowledge graph as (Super Mario Land, *Publisher*, Nintendo). The final distribution over the words includes the standard vocabulary along with aliases of Nintendo, and the model selects "*Nintendo*" as the token x_t. Facts related to Nintendo will be added to the local graph.

with the capability to incorporate information from an external knowledge graph.

To remind the reader, a knowledge graph is a directed, labeled graph consisting of entities $\mathcal{E}$ as nodes, with edges defined over a set of relations $\mathcal{R}$, i.e. $\mathcal{KG} = \{(s, r, o) \mid s \in \mathcal{E}, r \in \mathcal{R}, o \in \mathcal{E}\}$, where s is a subject entity with relation r to another entity o. Practical KGs have other aspects that make this formulation somewhat inexact: some relations are to *literal values*, such as numbers and dates, facts may be expressed as *properties* on relations, and entities have *aliases* as the set of strings that can refer to the entity. We also define a *local knowledge graph* for a subset of entities $\mathcal{E}_{<t}$ as $\mathcal{KG}_{<t} = \left\{(s, r, o) \mid s \in \mathcal{E}_{<t}, r \in \mathcal{R}, o \in \mathcal{E}\right\}$, i.e. contains entities $\mathcal{E}_{<t}$ and all facts they participate in.

Generative KG Language Model

The primary goal of the knowledge graph language model (KGLM) is to enable a neural language model to generate entities and facts from a knowledge graph. To encourage the model to generate facts that have appeared in the context already, KGLM will maintain a local knowledge graph containing all facts involving entities that have appeared in the context. As the model decides to refer to entities that have not been referred to yet, it will grow the local knowledge graph with additional entities and facts to reflect the new entity.

Formally, we will compute $p(x_t, \mathcal{E}_t | x_{<t}, \mathcal{E}_{<t})$ where $x_{<t}$ is the sequence of observed tokens, $\mathcal{E}_{<t}$ is the set of entities mentioned in $x_{<t}$, and $\mathcal{KG}_{<t}$ is the local knowledge graph determined by $\mathcal{E}_{<t}$, as described above. The generative process is:

- Decide the *type* of x_t, which we denote by t_t: whether it is a reference to an entity in $\mathcal{KG}_{<t}$ (related), a reference to an entity not in $\mathcal{KG}_{<t}$ (new), or not an entity mention ($\varnothing$).
- If $t_t = \text{new}$ then choose the upcoming entity o_t from the set of all entities $\mathcal{E}$.
- If $t_t = \text{related}$ then:
  - Choose a subject entity s_t from $\mathcal{E}_{<t}$.
  - Choose a factual relation r_t to render,

    $r_t \in \left\{(s, r, e) \in \mathcal{KG}_{<t} | s = s_t\right\}$.
  - Choose o_t as one of the object entities,

    $o_t \in \left\{e | (s_t, r_t, o) \in \mathcal{KG}_{<t}\right\}$.
- If $t_t = \varnothing$ then $o_t = \varnothing$.
- Generate x_t conditioned on o_t, potentially copying one of o_t's aliases.
- If $o_t \notin \mathcal{E}_{<t}$, then $\mathcal{E}_{<(t+1)} \leftarrow \mathcal{E}_{<t} \cup \left\{o_t\right\}$,

  else $\mathcal{E}_{<(t+1)} \leftarrow \mathcal{E}_{<t}$.

For the model to refer to an entity it has already mentioned, we introduce a *Reflexive* relation that self-relates, i.e. $s = o$ for $(s, \textit{Reflexive}, o)$.

An illustration of this process and the variables is provided in Figure 6.2, for generating a token in the middle of the same sentence as in Figure 6.1. Amongst the three mention types (t_t), the model chooses a reference to existing entity, which requires picking a fact to render. As the subject entity of this fact (s_t), the model picks Super Mario Land, and then follows the *Publisher* relation (r_t) to select Nintendo as the entity to render (o_t). When rendering Nintendo as a token x_t, the model has an *expanded* vocabulary available to it, containing the standard vocabulary along with all word types in any of the aliases of o_t.

Marginalizing out the KG There is a mismatch between our initial task requirement, $p(x_t|x_{<t})$, and the model we describe so far, which computes $p(x_t, \mathcal{E}_t|x_{<t}, \mathcal{E}_{<t})$. We will essentially *marginalize* out the local knowledge graph to compute the probability of the tokens, i.e. $p(\boldsymbol{x}) = \sum_{\mathcal{E}} p(\boldsymbol{x}, \boldsymbol{\mathcal{E}})$. We will clarify this, along with describing the training and the inference/decoding algorithms for this model and other details of the setup, in Section 6.1.3.

Parameterizing the Distributions

The parametric distributions used in the generative process above are defined as follows. We begin by computing the hidden state $\boldsymbol{h}_t$ using the formula in Eqn (6.1). We then split the vector into three components: $\boldsymbol{h}_t = \left[\boldsymbol{h}_{t,x}; \boldsymbol{h}_{t,s}; \boldsymbol{h}_{t,r}\right]$, which are respectively used to predict words, parents, and relations. The type of the token, t_t, is computed using a single-layer softmax over $\boldsymbol{h}_{t,x}$ to predict one of $\{\mathsf{new}, \mathsf{related}, \varnothing\}$.

Picking an Entity We also introduce pretrained embeddings for all entities and relations in the knowledge graph, denoted by $\boldsymbol{v}_e$ for entity e and $\boldsymbol{v}_r$ for relation r. To select e_t from all entities in case $t_t = \mathsf{new}$, we use:

$$p(e_t) = \mathrm{softmax}(\boldsymbol{v}_e \cdot (\boldsymbol{h}_{t,s} + \boldsymbol{h}_{t,r}))$$

over all $e \in \mathcal{E}$. The reason we add $\boldsymbol{h}_{t,s}$ and $\boldsymbol{h}_{t,r}$ is to mimic the structure of TransE, which we use to obtain entity and relation embeddings. Details on TransE will be provided in Section 6.1.3. For mention of a related entity, $t_t = \texttt{related}$, we pick a subject entity s_t using

$$p(s_t) = \text{softmax}(\boldsymbol{v}_s \cdot \boldsymbol{h}_{t,s})$$

over all $s \in \mathcal{E}_t$, then pick the relation r_t using

$$p(r_t) = \text{softmax}(\boldsymbol{v}_r \cdot \boldsymbol{h}_{t,r})$$

over all $r \in \{r|(s_t, r, e) \in \mathcal{KG}_t\}$. The combination of s_t and r_t determine the entity e_t (which must satisfy $(s_t, r_t, e_t) \in \mathcal{KG}_t$; if there are multiple options one is chosen at random).

Rendering the Entity If $e_t = \emptyset$, i.e. there is no entity to render, we use the same distribution over the vocabulary as in Eqn (6.1) - a softmax using $\boldsymbol{h}_{t,x}$. If there is an entity to render, we construct the distribution over the original vocabulary *and* a vocabulary containing all the tokens that appear in aliases of e_t. This distribution is conditioned on e_t in addition to x_t. To compute the scores over the original vocabulary, $\boldsymbol{h}_{t,x}$ is replaced by $\boldsymbol{h}'_{t,x} = \boldsymbol{W}_{\text{proj}}[\boldsymbol{h}_{t,x}; \boldsymbol{v}_{e_t}]$ where $\boldsymbol{W}_{\text{proj}}$ is a learned weight matrix that projects the concatenated vector into the same vector space as $\boldsymbol{h}_{t,x}$.

To obtain probabilities for words in the alias vocabulary, we use a copy mechanism Gu et al. (2016). The token sequences comprising each alias $\{a_j\}$ are embedded then encoded using an LSTM to form vectors $\boldsymbol{a}_j$. Copy scores are computed as:

$$p(x_t = a_j) \propto \exp\left[\sigma\left(\left(\boldsymbol{h}'_{t,x}\right)^T \boldsymbol{W}_{\text{copy}}\right)\boldsymbol{a}_j\right]$$

Tokens	x_t	Super	Mario	Land	is	a	1989			side	-	scrolling	platform	video	game	developed									
Mention Type	t_t	new			∅	∅	related			new			related			∅								
Entity Mentioned	e_t	SML			∅	∅	04-21-1989			SIDE_SCROLL			PVG			∅								
Relation	r_t	∅			∅	∅	pub date			∅			genre			∅								
Parent Entity	p_t	∅			∅	∅	SML			∅			SML			∅								
x_t	and	published	by	Nintendo	as	a	launch	title	for	their	Game	Boy	handheld	game	console	.									
t_t	∅	∅	∅	related	∅	∅	new		∅	∅	related		related			∅									
e_t	∅	∅	∅	NIN	∅	∅	LT		∅	∅	GAME_BOY		HGC			∅									
r_t	∅	∅	∅	pub	∅	∅	∅		∅	∅	R:manu / platform		instance of			∅									
p_t	∅	∅	∅	SML	∅	∅	∅		∅	∅	NIN	SML	GAME_BOY			∅									

Table 6.1: **Example annotation** of the sentence from Figure 6.1, including corresponding variables from Figure 6.2. Note that *Game Boy* has multiple parent and relation annotations, as the platform for Super Mario Land and as manufactured by Nintendo. Wikidata identifiers are made human-readable (e.g., SML is Q647249) for clarity.

6.1.2 *Linked WikiText-2*

Modeling aside, one of the primary barriers to incorporating factual knowledge into language models is that training data is hard to obtain. Standard language modeling corpora consist only of text, and thus are unable to describe which entities or facts each token is referring to. In contrast, while relation extraction datasets link text to a knowledge graph, the text is made up of disjoint sentences that do not provide sufficient context to train a powerful language model. Our goals are much more aligned to the *data-to-text* task (Ahn et al., 2016; Lebret et al., 2016; Wiseman et al., 2017; Yang et al., 2017; Gardent et al., 2017; Castro Ferreira et al., 2018), where a small table-sized KB is provided to generate a short piece of text; we are interested in language models that dynamically decide the facts to incorporate from the knowledge graph, guided by the discourse.

For these reasons we introduce the *Linked WikiText-2* dataset, consisting of (approximately) the same articles appearing in the *WikiText-2* language modeling corpus, but linked to the Wikidata (Vrandečić and Krötzsch, 2014a) knowledge graph. Because the text closely matches, models trained on *Linked WikiText-2* can be compared to models trained on *WikiText-2*. Furthermore, because many of the facts in Wikidata are derived from Wikipedia articles, the knowledge graph has a good coverage of facts expressed in the text. The dataset is available for download at: https://rloganiv.github.io/

linked-wikitext-2. Our system annotates one document at a time, and consists of entity linking, relation annotations, and post-processing. The following paragraphs describe each step in detail.

Initial Entity Annotations We begin by identifying an initial set of entity mentions within the text. The primary source of these mentions is the human-provided links between Wikipedia articles. Whenever a span of text is linked to another Wikipedia article, we associate its corresponding Wikidata entity with the span. While article links provide a large number of gold entity annotations, they are insufficient for capturing all of the mentions in the article since entities are only linked the first time they occur. Accordingly, we use the neural-el (Gupta et al., 2017) entity linker to identify additional links to Wikidata, and identify coreferences using Stanford CoreNLP[1] to cover pronouns, nominals, and other tokens missed by the linker.

Local Knowledge Graph The next step iteratively creates a generative story for the entities using relations in the knowledge graph as well as identifies new entities. To do this, we process the text token by token. Each time an entity is encountered, we add all of the related entities in Wikidata as candidates for matching. If one of these related entities is seen later in the document, we identify the entity as a parent for the later entity. Since multiple relations may appear as *explanations* for each token, we allow a token to have multiple facts.

Expanding the Annotations Since there may be entities that were missed in the initial set, as well as non-entity tokens of interest such as dates and quantities we further expand the entity annotations using string matching. For entities, we match the set of aliases provided in Wikidata. For dates, we create an exhaustive list of all of the possible ways of expressing the date (e.g. "*December 7, 1941*", "*7-12-1941*", "*1941*", ...). We perform a similar approach for quantities, using the pint library in Python to handle the different ways of expressing units (e.g. "*g*", "*gram*", ...). Since there are many ways to express a numerical quantity, we only render the quantity at the level of precision supplied by Wikidata, and do not perform unit conversions.

[1] https://stanfordnlp.github.io/CoreNLP/

	Train	Dev	Test
Documents	600	60	60
Tokens	2,019,195	207,982	236,062
Vocab. Size	33,558	-	-
Mention Tokens	207,803	21,226	24,441
Mention Spans	122,983	12,214	15,007
Unique Entities	41,058	5,415	5,625
Unique Relations	1,291	484	504

Table 6.2: ***Linked WikiText-2*** **corpus statistics.**

Example Annotation An example annotation is provided in Table 6.1 corresponding to the instance in Figure 6.1, along with the variables that correspond to the generative process of the knowledge graph language model (KGLM). The entity mentioned for most tokens here are human-provided links, apart from "*1989*" that is linked to 04-21-1989 by the string matching process. The annotations indicate which of the entities are *new* and *related* based on whether they are reachable by entities linked so far, clearly making a mistake for side-scrolling game and platform video game due to missing links in Wikidata. Finally, multiple plausible reasons for Game Boy are included: it's the platform for Super Mario Land and it is manufactured by Nintendo, even though only the former is more relevant here. Even with these omissions and mistakes, it is clear that the annotations are rich and detailed, with a high coverage, and thus should prove beneficial for training knowledge graph language models.

Dataset Statistics Statistics for *Linked WikiText-2* are provided in Table 6.2. In this corpus, more than 10% of the tokens are considered entity tokens, i.e. they are generated as factual references to information in the knowledge graph. Each entity is only mentioned a few times (less than 5 on average, with a long tail), and with more than thousand different relations. Thus it is clear that regular language models would not be able to generate factual text, and there is a need for language models to be able to refer to external sources of information.

Differences from *WikiText-2* Although our dataset is designed to closely replicate *WikiText-2*, there are some differences that prevent direct comparison. Firstly, there are minor variations in

text across articles due to edits between download dates. Secondly, according to correspondence with Merity et al. (2017b), *WikiText-2* was collected by querying the Wikipedia Text API. Because this API discards useful annotation information (e.g. article links), *Linked WikiText-2* instead was created by directly from the article HTML.

6.1.3 Training and Inference for KGLM

In this section, we describe the training and inference algorithm for KGLM.

Pretrained KG Embeddings During evaluation, we may need to make predictions on entities and relations that have not been seen during training. Accordingly, we use fixed entity and relations embeddings pre-trained using TransE (Bordes et al., 2013) on Wikidata. Given (p, r, e), we learn embeddings $\boldsymbol{v}_p$, $\boldsymbol{v}_r$ and $\boldsymbol{v}_e$ to minimize the distance:

$$\delta(\boldsymbol{v}_p, \boldsymbol{v}_r, \boldsymbol{v}_e) = \left\| \boldsymbol{v}_p + \boldsymbol{v}_r - \boldsymbol{v}_e \right\|^2 .$$

We use a max-margin loss to learn the embeddings:

$$\mathcal{L} = \max\left(0, \gamma + \delta\left(\boldsymbol{v}_p, \boldsymbol{v}_r, \boldsymbol{v}_e\right) - \delta\left(\boldsymbol{v}'_p, \boldsymbol{v}_r, \boldsymbol{v}'_e\right)\right)$$

where γ is the margin, and either p' or e' is a randomly chosen entity embedding.

Training with *Linked WikiText-2* Although the generative process in KGLM involves many steps, training the model on *Linked WikiText-2* is straightforward. Our loss objective is the negative log-likelihood of the training data:

$$\ell(\Theta) = \sum_t \log p(x_t, \mathcal{E}_t | x_{<t}, \mathcal{E}_{<t}; \Theta),$$

where Θ is the set of model parameters. Note that if an annotation has multiple viable parents such as Game Boy in 6.1, then we marginalize over all of the parents. Since all random variables are observed, training can performed using off-the-shelf gradient-based optimizers.

Inference While observing annotations makes the model easy to train, we do not assume that the model has access to annotations during evaluation. Furthermore, as discussed in Section 6.1.1, the goal in language modelling is to measure the marginal probability $p(\boldsymbol{x}) = \sum_{\mathcal{E}} p(\boldsymbol{x}, \boldsymbol{\mathcal{E}})$ not the joint probability. However, this sum is intractable to compute due to the large combinatorial space of possible annotations. We address this problem by approximating the marginal distribution using importance sampling. Given samples from a proposal distribution $q(\boldsymbol{\mathcal{E}}|\boldsymbol{x})$ the marginal distribution is:

$$\begin{aligned} p(\boldsymbol{x}) &= \sum_{\mathcal{E}} p\left(\boldsymbol{x}, \boldsymbol{\mathcal{E}}\right) = \sum_{\mathcal{E}} \frac{p\left(\boldsymbol{x}, \boldsymbol{\mathcal{E}}\right)}{q\left(\boldsymbol{\mathcal{E}}|\boldsymbol{x}\right)} q\left(\boldsymbol{\mathcal{E}}|\boldsymbol{x}\right) \\ &\approx \frac{1}{N} \sum_{\mathcal{E} \sim q} \frac{p\left(\boldsymbol{x}, \boldsymbol{\mathcal{E}}\right)}{q\left(\boldsymbol{\mathcal{E}}|\boldsymbol{x}\right)} \end{aligned}$$

This approach is used to evaluate models in Ji et al. (2017) and Dyer et al. (2016). Following Ji et al. (2017), we compute $q\left(\boldsymbol{\mathcal{E}}|\boldsymbol{x}\right)$ using a discriminative version of our model that predicts annotations for the current token instead of for the next token.

6.1.4 Experiments

To evaluate the proposed language model, we first introduce the baselines, followed by an evaluation using perplexity of held-out corpus, accuracy on fact completion, and an illustration of how the model uses the knowledge graph.

Evaluation Setup

Baseline Models We compare KGLM to the following baseline models:

- **AWD-LSTM** (Merity et al., 2018): strong LSTM-based model used as the foundation of most state-of-the-art models on *WikiText-2*.
- **EntityNLM** (Ji et al., 2017): an LSTM-based language model with the ability to track entity mentions. Embeddings for entities are created dynamically, and are not informed by any external sources of information.
- **EntityCopyNet:** a variant of the KGLM where $t_t = \texttt{new}$ for all mentions, i.e. entities are selected from $\mathcal{E}$ and entity aliases are copied, but relations in the knowledge graph are unused.

Hyperparameters We pre-train 256 dimensional entity and relation embeddings for all entities within two hops of the set of entities that occur in *Linked WikiText-2* using TransE with margin $\gamma = 1$. Weights are tied between all date embeddings and between all quantity embeddings to save memory. Following Merity et al. (2018) we use 400 dimensional word embeddings and a 3 layer LSTM with hidden dimension 1150 to encode tokens. We also employ the same regularization strategy (DropConnect (Wan et al., 2013) + Dropout(Srivastava et al., 2014)) and weight tying approach. However, we perform optimization using Adam (Kingma and Ba, 2015) with learning rate 1e-3 instead of NT-ASGD, having found that it is more stable.

Results

Perplexity We present the model perplexities in Table 6.3. To marginalize over annotations, perplexities for the EntityNLM, EntityCopyNet, and KGLM are estimated using the importance sampling approach described in Section 6.1.3. We observe that the KGLM attains substantially lower perplexity than the other entity-based language models (44.1 vs. 76.1/85.4), providing strong

	PPL	UPP
ENTITYNLM* (Ji et al., 2017)	85.4	189.2
EntityCopyNet*	76.1	144.0
AWD-LSTM (Merity et al., 2018)	74.8	165.8
KGLM*	**44.1**	**88.5**

Table 6.3: **Perplexity results** on *Linked WikiText-2*. Results for models marked with * are obtained using importance sampling.

	AWD-LSTM	GPT-2	KGLM	
			Oracle	NEL
nation-capital	0 / 0	**6 / 7**	0 / 0	0 / 4
birthloc	0 / 9	14 / 14	**94 / 95**	85 / 92
birthdate	0 / 25	8 / 9	**65 / 68**	61 / 67
spouse	0 / 0	**2** / 3	**2** / 2	1 / **19**
city-state	0 / 13	**62 / 62**	9 / 59	4 / 59
book-author	0 / 2	0 / 0	**61 / 62**	25 / 28
Average	0.0/8.2	15.3/15.8	**38.5/47.7**	29.3/44.8

Table 6.4: **Fact completion results**. Top-k accuracy (@1/@5,%) for predicting the next token for an incomplete factual sentence. See examples in Table 6.5.

evidence that leveraging knowledge graphs is crucial for accurate language modeling. Furthermore, KGLM significantly outperforms all models in unknown penalized perplexity, demonstrating its ability to generate rare tokens.

Fact Completion

We revisit our prompt-based approach for diagnosing language model's fact completion capabilities from Chapter 3. To remind the reader of the setup, we sample a collection of 100 edges $(x, r, y) \in \mathcal{KG}$ for 6 different relations r, as well as write a template for each relation. The complete list of templates was provided in Table 3.1. Table 6.4 presents performance of AWD-LSTM, GPT-2-medium, and KGLM on the relations. The *oracle* KGLM is given the correct entity annotation for the head entity, x, while the *NEL* KGLM uses the discriminative model used for importance sampling combined with the NEL entity linker to produce an entity annotation for x.

	Input Sentence	Gold	GPT-2	KGLM
Both correct	Paris Hilton was born in ___	New York City	New	1981
	Arnold Schwarzenegger was born on ___	1947-07-30	July	30
KGLM correct	Bob Dylan was born in ___	Duluth	New	Duluth
	Barack Obama was born on ___	1961-08-04	January	August
	Ulysses is a book that was written by ___	James Joyce	a	James
GPTv2 correct	St. Louis is a city in the state of ___	Missouri	Missouri	Oldham
	Richard Nixon was born on ___	1913-01-09	January	20
	Kanye West is married to ___	Kim Kardashian	Kim	the
Both incorrect	The capital of India is ___	New Delhi	the	a
	Madonna is married to ___	Carlos Leon	a	Alex

Table 6.5: **Completion examples**. Examples of fact completion by KGLM and GPT-2, which has been trained on a much larger corpus. GPT-2 tends to produce very common and general tokens, such as one of a few popular cities to follow "*born in*". KGLM sometimes makes mistakes in linking to the appropriate fact in the KG, however, the generated facts are more specific and contain rare tokens. We omit AWD-LSTM from this figure as it rarely produced tokens apart from the generic "*the*" or "*a*", or "⟨*UNK*⟩".

Amongst models trained on the same data, both KGLM variants significantly outperform AWD-LSTM; they produce accurate facts, while AWD-LSTM produced generic, common words. The KGLM variants are also competitive with models trained on orders of magnitude more data, producing factual completions that require specific knowledge, such as birthplaces, dates, and authors. However, they do not capture facts or relations that frequently appear in large corpora, like the cities within states.[2] It is encouraging to see that the KGLM with automatic linking performs comparably to oracle linking.

We provide examples in Table 6.5 to highlight qualitative differences between KGLM, trained on 600 documents, and the recent state-of-the-art language model, GPT-2, trained on the WebText corpus with over 8 million documents (Radford et al., 2019). For examples that both models get factually correct or incorrect, the generated tokens by KGLM are often much more specific, as opposed to selection of more popular/generic tokens (GPT-2 often predicts "New York" as the birthplace, even for popular entities). KGLM, in particular, gets factual statements correct when the

[2]This is not a failure of the KG, but of the model's ability to pick the correct relation from the KG given the prompt.

head or tail entities are rare, while GPT-2 can only complete facts for more-popular entities while using more-generic tokens (such as "*January*" instead of "*20*").

Effect of Changing the KG For most language models, it is difficult to control their generation since *factual* knowledge is entangled with generation capabilities of the model. For KGLM, an additional benefit of its use of an external source of knowledge is that KGLM is directly controllable via modifications to the KG. To illustrate this capability with a simple example, we create completion of "*Barack Obama was born on* _" with the original fact (Barack Obama, *birthDate*, 1961-08-04), resulting in the top three decoded tokens as "*August*", "*4*", "*1961*". After changing the birth date to 2013-03-21, the top three decoded tokens become "*March*", "*21*", "*2013*". Thus, changing the fact in the knowledge graph directly leads to a corresponding change in the model's prediction.

6.2 KnowBERT

In this section we introduce KnowBERT. In contrast to the previous section, KnowBERT incorporates information from knowledge bases into masked language models as opposed to autoregressive language models. We start by describing the BERT and knowledge base components of KnowBERT. We then move to introducing the Knowledge Attention and Recontextualization component (KAR) that is used to in inject entity embeddings into BERT. Next, we describe the training procedure, including the multitask training regime for jointly training KnowBERT and an entity linker. Finally, we evaluate KnowBERT on similar fact completion experiments to those used in the previous section and Chapter 3.

6.2.1 Pretrained BERT

We describe KnowBERT as an extension to (and candidate replacement for) BERT, although the method is general and can be applied to any deep pretrained model including left-to-right and right-to-left LMs such as ELMo and GPT. Formally, BERT accepts as input a sequence of N WordPiece tokens (Sennrich et al., 2016; Wu et al., 2016b), $(x_1, \ldots, x_N)$, and computes L layers of D-dimensional contextual representations $\mathbf{H}_i \in \mathbb{R}^{N \times D}$ by successively applying non-linear functions $\mathbf{H}_i = \mathrm{F}_i(\mathbf{H}_{i-1})$. The non-linear function is a multi-headed self-attention layer followed by a position-wise multilayer perceptron (MLP) (Vaswani et al., 2017b):

$$\mathrm{F}_i(\mathbf{H}_{i-1}) = \mathrm{TransformerBlock}(\mathbf{H}_{i-1}) = \mathrm{MLP}(\mathrm{MultiHeadAttn}(\mathbf{H}_{i-1}, \mathbf{H}_{i-1}, \mathbf{H}_{i-1})).$$

The multi-headed self-attention uses $\mathbf{H}_{i-1}$ as the query, key, and value to allow each vector to attend to every other vector.

BERT is trained to minimize an objective function that combines both next-sentence prediction (NSP) and masked LM log-likelihood (MLM):

$$\mathcal{L}_{\mathrm{BERT}} = \mathcal{L}_{\mathrm{NSP}} + \mathcal{L}_{\mathrm{MLM}}.$$

Given two inputs $\mathbf{x}_A$ and $\mathbf{x}_B$, the next-sentence prediction task is binary classification to predict whether $\mathbf{x}_B$ is the next sentence following $\mathbf{x}_A$. The masked LM objective randomly replaces a percentage of input word pieces with a special [MASK] token and computes the negative log-likelihood of the missing token with a linear layer and softmax over all possible word pieces.

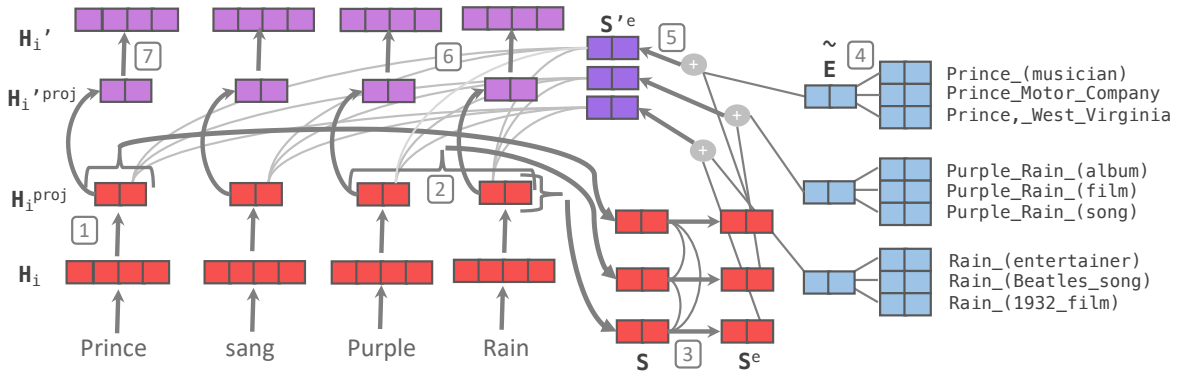


Figure 6.3: **The knowledge attention and recontextualization (KAR) component.** BERT word piece representations ($\mathbf{H}_i$) are first projected to $\mathbf{H}_i^{\text{proj}}$ (1), then pooled over candidate mentions spans (2) to compute $\mathbf{S}$, and contextualized into $\mathbf{S}^e$ using mention-span self-attention (3). An integrated entity linker computes weighted average entity embeddings $\tilde{\mathbf{E}}$ (4), which are used to enhance the span representations with knowledge from the KB (5), computing $\mathbf{S}'^e$. Finally, the BERT word piece representations are recontextualized with word-to-entity-span attention (6) and projected back to the BERT dimension (7) resulting in $\mathbf{H}'_i$.

6.2.2 Knowledge Bases

The key contribution of this paper is a method to incorporate knowledge bases (KB) into a pretrained BERT model. To encompass as wide a selection of prior knowledge as possible, we adopt a broad definition for a KB in the most general sense as fixed collection of K entity nodes, e_k, from which it is possible to compute entity embeddings, $\mathbf{e}_k \in \mathbb{R}^E$. This includes KBs with a typical $(\text{subj}, \text{rel}, \text{obj})$ graph structure, KBs that contain only entity metadata without a graph, and those that combine both a graph and entity metadata, as long as there is some method for embedding the entities in a low dimensional vector space. We also do not make any assumption that the entities are typed. As we show in the next section, this flexibility is beneficial, where we compute entity embeddings from WordNet using both the graph and synset definitions, but link directly to Wikipedia pages without a graph by using embeddings computed from the entity description.

We also assume that the KB is accompanied by an entity candidate selector that takes as input some text and returns a list of C potential entity links, each consisting of the start and end indices of the

potential mention span and M_m candidate entities in the KG:

$$C = \{\langle(\text{start}_m, \text{end}_m), (e_{m,1}, \ldots, e_{m,M_m})\rangle \mid m \in 1 \ldots C, e_k \in 1 \ldots K\}.$$

In practice, these are often implemented using precomputed dictionaries (e.g., CrossWikis; Spitkovsky and Chang, 2012), KB specific rules (e.g., a WordNet lemmatizer), or other heuristics (e.g., string match; Mihaylov and Frank, 2018). Ling et al. (2015) showed that incorporating candidate priors into entity linkers can be a powerful signal, so we optionally allow for the candidate selector to return an associated prior probability for each entity candidate. In some cases, it is beneficial to over-generate potential candidates and add a special NULL entity to each candidate list, thereby allowing the linker to discriminate between actual links and false positive candidates. In this work, the entity candidate selectors are fixed but their output is passed to a learned context dependent entity linker to disambiguate the candidate mentions.

Finally, by restricting the number of candidate entities to a fixed small number (we use 30), KnowBERT's runtime is independent of the size the KB, as it only considers a small subset of all possible entities for any given text. As the candidate selection is rule-based and fixed, it is fast and in our implementation is performed asynchronously on CPU. The only overhead for scaling up the size of the KB is the memory footprint to store the entity embeddings.

6.2.3 KAR

The Knowledge Attention and Recontextualization component (KAR) is the heart of KnowBERT. The KAR accepts as input the contextual representations at a particular layer, $\mathbf{H}_i$, and computes knowledge enhanced representations $\mathbf{H}'_i = \text{KAR}(\mathbf{H}_i, C)$. This is fed into the next pretrained layer, $\mathbf{H}_{i+1} = \text{TransformerBlock}(\mathbf{H}'_i)$, and the remainder of BERT is run as usual.

In this section, we describe the KAR's key components: mention-span representations, retrieval of relevant entity embeddings using an entity linker, update of mention-span embeddings with retrieved information, and recontextualization of entity-span embeddings with word-to-entity-span attention. We describe the KAR for a single KB, but extension to multiple KBs at different layers is straightforward. See Fig. 6.3 for an overview.

Mention-Span Representations The KAR starts with the KB entity candidate selector that provides a list of candidate mentions which it uses to compute mention-span representations. $\mathbf{H}_i$ is first projected to the entity dimension (E, typically 200 or 300) with a linear projection,

$$\mathbf{H}_i^{\text{proj}} = \mathbf{H}_i \mathbf{W}_1^{\text{proj}} + \mathbf{b}_1^{\text{proj}}. \tag{6.3}$$

Then, the KAR computes C mention-span representations $\mathbf{s}_m \in \mathbb{R}^E$, one for each candidate mention, by pooling over all word pieces in a mention-span using the self-attentive span pooling from Lee et al. (2017). The mention-spans are stacked into a matrix $\mathbf{S} \in \mathbb{R}^{C \times E}$.

Entity Linker The entity linker is responsible for performing entity disambiguation for each potential mention from among the available candidates. It first runs mention-span self-attention to compute

$$\mathbf{S}^e = \text{TransformerBlock}(\mathbf{S}). \tag{6.4}$$

The span self-attention is identical to the typical transformer layer, exception that the self-attention is between mention-span vectors instead of word piece vectors. This allows KnowBERT to incorporate global information into each linking decision so that it can take advantage of entity-entity co-occurrence and resolve which of several overlapping candidate mentions should be linked.[3]

[3]We found a small transformer layer with four attention heads and a 1024 feed-forward hidden dimension was sufficient, significantly smaller than each of the layers in BERT. Early experiments demonstrated the effectiveness of this layer with improved entity linking performance.

Following Kolitsas et al. (2018), $\mathbf{S}^e$ is used to score each of the candidate entities while incorporating the candidate entity prior from the KB. Each candidate span m has an associated mention-span vector $\mathbf{s}_m^e$ (computed via Eq. 6.4), M_m candidate entities with embeddings $\mathbf{e}_{mk}$ (from the KB), and prior probabilities p_{mk}. We compute M_m scores using the prior and dot product between the entity-span vectors and entity embeddings,

$$\psi_{mk} = \text{MLP}(p_{mk}, \mathbf{s}_m^e \cdot \mathbf{e}_{mk}), \tag{6.5}$$

with a two-layer MLP (100 hidden dimensions).

If entity linking (EL) supervision is available, we can compute a loss with the gold entity e_{mg}. The exact form of the loss depends on the KB, and we use both log-likelihood,

$$\mathcal{L}_{\text{EL}} = -\sum_m \log\left(\frac{\exp(\psi_{mg})}{\sum_k \exp(\psi_{mk})}\right), \tag{6.6}$$

and max-margin,

$$\begin{aligned}\mathcal{L}_{\text{EL}} = &\max(0, \gamma - \psi_{mg}) \; + \\ &\sum_{e_{mk} \neq e_{mg}} \max(0, \gamma + \psi_{mk}),\end{aligned} \tag{6.7}$$

formulations.

Knowledge Enhanced Entity-Span Representations KnowBERT next injects the KB entity information into the mention-span representations computed from BERT vectors ($\mathbf{s}_m^e$) to form entity-span representations. For a given span m, we first disregard all candidate entities with score

ψ below a fixed threshold, and softmax normalize the remaining scores:

$$\tilde{\psi}_{mk} = \begin{cases} \dfrac{\exp(\psi_{mk})}{\sum_{\psi_{mk} \geq \delta} \exp(\psi_{mk})}, & \psi_{mk} \geq \delta \\ 0, & \psi_{mk} < \delta. \end{cases}$$

Then the weighted entity embedding is

$$\tilde{\mathbf{e}}_m = \sum_k \tilde{\psi}_{mk} \mathbf{e}_{mk}.$$

If all entity linking scores are below the threshold δ, we substitute a special NULL embedding for $\tilde{\mathbf{e}}_m$. Finally, the entity-span representations are updated with the weighted entity embeddings

$$\mathbf{s}'^e_m = \mathbf{s}^e_m + \tilde{\mathbf{e}}_m, \tag{6.8}$$

which are packed into a matrix $\mathbf{S}'^e \in \mathbb{R}^{C \times E}$.

Recontextualization After updating the entity-span representations with the weighted entity vectors, KnowBERT uses them to recontextualize the word piece representations. This is accomplished using a modified transformer layer that substitutes the multi-headed self-attention with a multi-headed attention between the projected word piece representations and knowledge enhanced entity-span vectors. As introduced by Vaswani et al. (2017b), the contextual embeddings $\mathbf{H}_i$ are used for the query, key, and value in multi-headed self-attention. The word-to-entity-span attention in KnowBERT substitutes $\mathbf{H}^{\text{proj}}_i$ for the query, and $\mathbf{S}'^e$ for both the key and value:

$$\mathbf{H}'^{\text{proj}}_i = \text{MLP}(\text{MultiHeadAttn}(\mathbf{H}^{\text{proj}}_i, \mathbf{S}'^e, \mathbf{S}'^e)).$$

This allows each word piece to attend to all entity-spans in the context, so that it can propagate entity information over long contexts. After the multi-headed word-to-entity-span attention, we run a position-wise MLP analogous to the standard transformer layer.[4]

Finally, $\mathbf{H'}_i^{\text{proj}}$ is projected back to the BERT dimension with a linear transformation and a residual connection added:

$$\mathbf{H'}_i = \mathbf{H'}_i^{\text{proj}}\mathbf{W}_2^{\text{proj}} + \mathbf{b}_2^{\text{proj}} + \mathbf{H}_i \tag{6.9}$$

Alignment of BERT and Entity Vectors As KnowBERT does not place any restrictions on the entity embeddings, it is essential to align them with the pretrained BERT contextual representations. To encourage this alignment we initialize $\mathbf{W}_2^{\text{proj}}$ as the matrix inverse of $\mathbf{W}_1^{\text{proj}}$ (Eq. 6.3). The use of dot product similarity (Eq. 6.5) and residual connection (Eq. 6.9) further aligns the entity-span representations with entity embeddings.

6.2.4 Training Procedure

Our training regime incrementally pretrains increasingly larger portions of KnowBERT before finetuning all trainable parameters in a multitask setting with any available EL supervision. It is similar in spirit to the "chain-thaw" approach in Felbo et al. (2017).

We assume access to a pretrained BERT model and one or more KBs with their entity candidate selectors. To add the first KB, we begin by pretraining entity embeddings (if not already provided from another source), then freeze them in all subsequent training, including task-specific finetuning. If EL supervision is available, it is used to pretrain the KB specific EL parameters, while freezing

[4] As for the multi-headed entity-span self-attention, we found a small transformer layer to be sufficient, with four attention heads and 1024 hidden units in the MLP.

the remainder of the network. Finally, the entire network is finetuned to convergence by minimizing

$$\mathcal{L}_{\text{KnowBERT}} = \mathcal{L}_{\text{BERT}} + \mathcal{L}_{\text{EL}}.$$

We apply gradient updates to homogeneous batches randomly sampled from either the unlabeled corpus or EL supervision.

To add a second KB, we repeat the process, inserting it in any layer above the first one. When adding a KB, the BERT layers above it will experience large gradients early in training, as they are subject to the randomly initialized parameters associated with the new KB. They are thus expected to move further from their pretrained values before convergence compared to parameters below the KB. By adding KBs from bottom to top, we minimize disruption of the network and decrease the likelihood that training will fail.

The entity embeddings and selected candidates contain lexical information (especially in the case of WordNet), that will make the masked LM predictions significantly easier. To prevent leaking into the masked word pieces, we adopt the BERT strategy and replace all entity candidates from the selectors with a special [MASK] entity if the candidate mention span overlaps with a masked word piece.[5] This prevents KnowBERT from relying on the selected candidates to predict masked word pieces.

6.2.5 Experiments

Experimental Setup We used the English uncased $\text{BERT}_{\text{BASE}}$ model (Devlin et al., 2019b) to train three versions of KnowBERT: KnowBERT-Wiki, KnowBERT-WordNet, and KnowBERT-W+W that includes both Wikipedia and WordNet.

[5]Following BERT, for 80% of masked word pieces all candidates are replaced with [MASK], 10% are replaced with random candidates and 10% left unmasked.

System	Fact. Recall MRR	# params. masked LM	# params. KAR	# params. entity embed.	Fwd. / Bwd. time
$\text{BERT}_{\text{BASE}}$	0.09	110	0	0	0.25
$\text{BERT}_{\text{LARGE}}$	0.11	336	0	0	0.75
KnowBERT-Wiki	0.26	110	2.4	141	0.27
KnowBERT-WordNet	0.22	110	4.9	265	0.31
KnowBERT-W+W	**0.31**	110	7.3	406	0.33

Table 6.6: **Comparison of factual recall MRR, and number of parameters (in millions) for BERT and KnowBERT.** The table also includes the total time to run one forward and backward pass (in seconds) on a TITAN Xp GPU (12 GB RAM) for a batch of 32 sentence pairs with total length 80 word pieces. Due to memory constraints, the $\text{BERT}_{\text{LARGE}}$ batch is accumulated over two smaller batches.

KnowBERT-Wiki The entity linker in KnowBERT-Wiki borrows both the entity candidate selectors and embeddings from Ganea and Hofmann (2017). The candidate selectors and priors are a combination of CrossWikis, a large, precomputed dictionary that combines statistics from Wikipedia and a web corpus (Spitkovsky and Chang, 2012), and the YAGO dictionary (Hoffart et al., 2011). The entity embeddings use a skip-gram like objective (Mikolov et al., 2013) to learn 300-dimensional embeddings of Wikipedia page titles directly from Wikipedia descriptions without using any explicit graph structure between nodes. As such, nodes in the KB are Wikipedia page titles, e.g., Prince_(musician). Ganea and Hofmann (2017) provide pretrained embeddings for a subset of approximately 470K entities. Early experiments with embeddings derived from Wikidata relations[6] did not improve results.

We used the AIDA-CoNLL dataset (Hoffart et al., 2011) for supervision, adopting the standard splits. This dataset exhaustively annotates entity links for named entities of person, organization and location types, as well as a miscellaneous type. It does not annotate links to common nouns or other Wikipedia pages. At both train and test time, we consider all selected candidate spans and the top 30 entities, to which we add the special NULL entity to allow KnowBERT to discriminate between actual links and false positive links from the selector. As such, KnowBERT models both entity mention detection and disambiguation in an end-to-end manner. Eq. 6.7 was used as the objective.

[6] https://github.com/facebookresearch/PyTorch-BigGraph

KnowBERT-WordNet Our WordNet KB combines synset metadata, lemma metadata and the relational graph. To construct the graph, we first extracted all synsets, lemmas, and their relationships from WordNet 3.0 using the nltk interface. After disregarding certain symmetric relationships (e.g., we kept the hypernym relationship, but removed the inverse hyponym relationship) we were left with 28 synset-synset and lemma-lemma relationships. From these, we constructed a graph where each node is either a synset or lemma, and introduced the special lemma_in_synset relationship to link synsets and lemmas. The candidate selector uses a rule-based lemmatizer without part-of-speech (POS) information.[7]

Our embeddings combine both the graph and synset glosses (definitions), as early experiments indicated improved perplexity when using both vs. just graph-based embeddings. We used TuckER (Balazevic et al., 2019b) to compute 200-dimensional vectors for each synset and lemma using the relationship graph. Then, we extracted the gloss for each synset and used an off-the-shelf state-of-the-art sentence embedding method (Subramanian et al., 2018) to produce 2048-dimensional vectors. These are concatenated to the TuckER embeddings. To reduce the dimensionality for use in KnowBERT, the frozen 2248-dimensional embeddings are projected to 200-dimensions with a learned linear transformation.

For supervision, we combined the SemCor word sense disambiguation (WSD) dataset (Miller et al., 1994) with all lemma example usages from WordNet[8] and link directly to synsets. The loss function is Eq. 6.6. At train time, we did not provide gold lemmas or POS tags, so KnowBERT must learn to implicitly model coarse grained POS tags to disambiguate each word. At test time when evaluating we restricted candidate entities to just those matching the gold lemma and POS tag, consistent with the standard WSD evaluation.

[7] https://spacy.io/

[8] To provide a fair evaluation on the WiC dataset which is partially based on the same source, we excluded all WiC train, development and test instances.

Training Details To control for the unlabeled corpus, we concatenated Wikipedia and the Books Corpus (Zhu et al., 2015) and followed the data preparation process in BERT with the exception of heavily biasing our dataset to shorter sequences of 128 word pieces for efficiency. Both KnowBERT-Wiki and KnowBERT-WordNet insert the KB between layers 10 and 11 of the 12-layer $\text{BERT}_{\text{BASE}}$ model. KnowBERT-W+W adds the Wikipedia KB between layers 10 and 11, with WordNet between layers 11 and 12. Earlier experiments with KnowBERT-WordNet in a lower layer had worse perplexity. We generally followed the finetuning procedure in Devlin et al. (2019b); see supplemental materials for details.

Fact Completion We test KnowBERT's ability to recall facts from the KBs, using 90K tuples from Wikidata (Vrandečić and Krötzsch, 2014b) for 17 different relationships (shown in Table 6.7). Models are evaluated on whether they can predict the correct entity by greedily decoding all of the masked word pieces in parallel.

Table 6.6 displays a summary of the results, and a per-relation breakdown is provided in Figure 6.4. We observe that all of the KnowBERT models achieve better performance than BERT models on almost all of the tested relations (the exception being that KnowBERT-Wiki appears to be slightly worse than $\text{BERT}_{\text{LARGE}}$ on the personMemberOfSportsTeam and videoGamePlatform relations). Among KnowBERT models, we find that KnowBERT-Wiki tends to have better hits-at-1 than KnowBERT-WordNet , presumably due to overlapping information in Wikipedia and Wikidata. However, somewhat surprisingly, the KnowBERT-WordNet model still performs substantially better than the BERT models, and KnowBERT-W+W tends to perform better than KnowBERT-Wiki even though WordNet does not typically contain factual information. One plausible explanation for this is that, by providing an external source of information about synonymy and hypernymy, more model capacity can be dedicated to memorizing facts during pretraining.

companyFoundedBy	{s} was founded by {o}. {o} is the founder of {s}.
movieDirectedBy	{s} was directed by {o}. {o} is the director of {s}.
movieStars	{s} stars {o}. {o} starred in {s}.
personCityOfBirth	{s} was born in the city of {o}.
personCountryOfBirth	{s} was born in the country of {o}.
personCountryOfDeath	{s} died in the country of {o}.
personEducatedAt	{s} was educated at {o}. {s} studied at {o}.
personEmployer	{s} works for {o}. {s} is an employee of {o}.
personFather	{s} is the father of {o}. {o} is the child of {s}.
personMemberOfBand	{s} is a member of {o}.
personMemberOfSportsTeam	{s} played on {o}.
personMother	{s} is the mother of {o}. {o} is the child of {s}.
personOccupation	{s} is a {o}.
personSpouse	{s} married {o}.
songPerformedBy	{s} is performed by {o}.
videoGamePlatform	{s} is a game on the {o}.
writtenTextAuthor	{s} is written by {o}. {o} is the author of {s}.

Table 6.7: **Prompt templates used in fact completion experiments**.

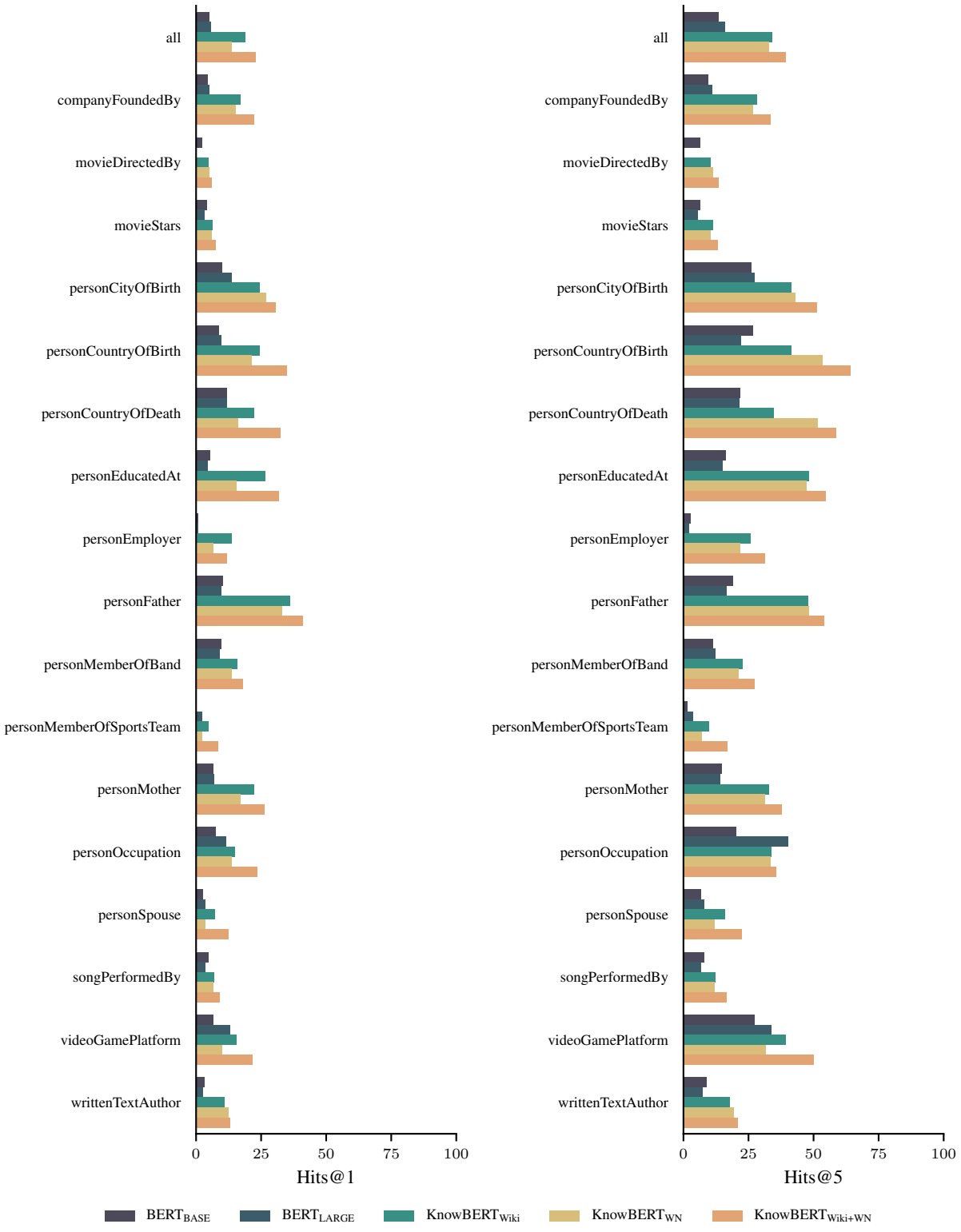


Figure 6.4: **Per-relation factual recall results**. This figure displays hits-at-k ($k \in \{1, 5\}$) for BERT and KnowBERT on the expanded set of factual completion queries listed in Table 6.7.

6.3 Summary of Contributions

In this chapter we introduced the KGLM and KnowBERT, two language models endowed with the capability to condition on information in external knowledge bases at inference time. We showed that by conditioning on this knowledge, these models were able to significantly improve the ability to correctly recall facts, as diagnosed using the prompts introduced in Chapter 3. The text in this chapter is based upon the following publications:

- *Barack's Wife Hillary: Using Knowledge Graphs for Fact-Aware Language Modeling* (Logan et al., 2019, ACL 2019)
- *Knowledge Enhanced Contextual Word Representations* (Peters et al., 2019, EMNLP 2019)

which have a combined 538 citations at the time of writing.

The author of this dissertation was the primary author of Logan et al. (2019), and takes credit for coming up with the generative story and implementation of the KGLM model (under the close guidance of Sameer Singh and Matt Gardner), as well as setting up and running most of the experiments (although as mentioned earlier Nelson Liu deserves special credit for manually verifying the fact recall results). Primary credit for KnowBERT belongs to Matt Peters, Mark Neumann, Roy Schwartz and Vidur Joshi. While the author was involved in initial discussions about how to adapt the KGLM architecture for masked language models, in terms of the content presented in this dissertation, he would only like to take credit for designing and running the factual recall results. Other results included in the original publication have been suppressed to convey to the reader the extent of his participation.

7

Importance Sampling-Based Evaluation of Latent Language Models

In the previous chapter, we demonstrated how importance sampling could be used to approximately compute the marginal probability of a sequence of tokens under the under the knowledge graph language model. In this chapter, we will take a brief detour from the topic of integrating language models and knowledge bases to further explore this application of importance sampling to estimating the perplexity of latent language models.

Generally speaking, latent language models are generative models of text that jointly represent the text and the latent structure underlying it. In the KGLM this latent structure is the underlying links between the entities mentioned in the text, and their relations in a knowledge graph. NLP researchers have also explored other structures such as syntactic parses (Dyer et al., 2016) and coreference chains between entity mentions (Ji et al., 2017).

In all of these works, importance sampling is used to approximately marginalize over the space of latent structures to compute the likelihood of held out test sequences. However, there are some questions about how valid these results are in practice. For one, although convergence of importance

	x	Kawhi	to	join	L.A.	Clippers	.	He	...
EntityNLM	t	1	0	0	1	1	0	1	...
	e	1	∅	∅	2	2	∅	1	...
	l	1	1	1	2	1	1	1	...
KGLM	t	new	∅	∅	related		∅	related	...
	s	∅	∅	∅	kawhi_leonard		∅	kawhi_leonard	...
	r	∅	∅	∅	playerFor		∅	reflexive	...
	o	kawhi_leonard	∅	∅	la_clippers		∅	kawhi_leonard	...

Figure 7.1: **EntityNLM and KGLM latent states.** For EntityNLM, $z = (t, e, l)$, where t denotes whether the token is part of a mention, e denotes the coreference cluster, and l denotes the remaining mention length. For KGLM, $z = (t, s, r, o)$, where t has the same meaning, and s, r and o associate tokens to edges in a knowledge graph.

sampled estimates of likelihood is asymptotically guaranteed, results are typically produced using a small number of samples for which this guarantee does not necessarily apply. Furthermore, these works employ a variety of heuristics—such as sampling from proposal distributions that are conditioned on future *gold* tokens the model is being evaluated on, and changing the temperature of the proposal distribution—without providing measurements of the effect these decisions have on estimated perplexity, and often omitting details crucial to replicating their results.

In this chapter, we seek to fill in this missing knowledge, and put this practice on more rigorous footing. First, we review the theory of importance sampling, providing proof that importance sampled perplexity estimates are stochastic upper bounds of the true perplexity. In addition, we compile a list of practices used by Dyer et al. (2016), Ji et al. (2017), and the material presented in the previous chapter, and uncover a difference in the granularity at which importance samples are aggregated in these works that has a substantial effect on the final estimates. We also investigate a direct marginalization alternative to importance sampling based on beam search that produces strict bounds, and in some cases, has similar performance. Lastly, we perform experiments to measure the effect of varying sample size, aggregation method, and choice of proposal distribution for these models, an analysis that is missing from previous work. From these results we conclude by describing a set of best practices to be used in future work.

7.1 Inference in Latent LMs

In this section, we provide an overview of importance sampling-based inference in latent language models, as well as some key theoretical results.

Latent LMs A *latent language model* is a generative model which estimates the joint distribution $p(x, z)$ of a sequence of text $x = (x_1, \dots, x_T)$ and its underlying latent structure z.

In this paper, we focus on three models:

- RNNG (Dyer et al., 2016) which models syntactic structure,
- EntityNLM (Ji et al., 2017) which models coreference chains, and
- KGLM (Logan et al., 2019) which models links to an external knowledge graph.

Example latent states for EntityNLM and KGLM are depicted in Figure 7.1, showing latent coreference chains and links to the knowledge graph. Other notable latent language models include the NKLM (Ahn et al., 2016) and LRLM (Hayashi et al., 2020); we do not study them since they use alternatives to importance sampling (e.g., the forward-backward algorithm).

Perplexity The standard evaluation metric for language models is *perplexity*:

$$\text{PPL} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T}\log p(x_t|x_{<t})\right), \tag{7.1}$$

where $p(x_t|x_{<t})$ is the marginal likelihood of the token x_t conditioned on the previous tokens $x_{<t}$. By the chain rule of probabilities $p(x) = \prod_{t=1}^{T} p(x_t|x_{<t})$. Perplexity can be intractable to compute for latent language models since it requires marginalizing out the latent variable (e.g., $p(x) = \sum_z p(x, z)$) whose state space is often exponential in the length of the text.

Importance Sampling Existing approaches instead use *importance sampling* (Kahn, 1950) to estimate an approximate marginal probability:

$$\hat{p}(x) = \frac{1}{K} \sum_{k=1}^{K} \frac{p(x, z_k)}{q(z_k)}, \tag{7.2}$$

where $q(z)$ is an arbitrary *proposal distribution* and $z_1, \ldots, z_K \sim q(z)$. It is well known that $\hat{p}(x)$ is an unbiased estimator:

$$\mathbb{E}_{z_k \sim q(z)} [\hat{p}(x)] = p(x), \tag{7.3}$$

provided that $q(z) > 0$ whenever $p(z) > 0$. For proof and further details on importance sampling, we refer the reader to Owen (2013).

Stochastic Upper Bound A consequence of Equation (7.3) is that, due to Jensen's inequality:

$$\mathbb{E}_{z_k \sim q(z)} \left[\log \hat{p}(x)\right] \leq \log p(x). \tag{7.4}$$

In other words, *importance sampled estimates of a model's perplexity are stochastic upper bounds of the true perplexity*. This property has not been stated in prior work on latent language modeling, yet is an important consideration since it implies that importance sampled perplexities can be reliably used to compare against existing baselines.

Limiting Behavior Another important observation is that *importance sampled estimates of perplexity are consistent*, e.g., will converge as the number of samples approaches infinity. To prove this, we first observe that $\hat{p}(x)$ is consistent, which is a well-known consequence of the strong law of large numbers (Geweke, 1989). Accordingly, $\log \hat{p}(x)$ is also consistent due to the continuous mapping theorem (Van der Vaart, 2000).

7.2 Common Practices

Implementing importance sampling for evaluating latent language models involves a number of decisions that need to be made. One needs to select the number of samples, choose the proposal distribution, and decide whether to aggregate importance sampled estimates at the instance or corpus level. We list some common practices.[1]

Sample Size Typically, only 100 samples are used for computing the perplexity. A notable exception is Kim et al. (2019)'s follow-up to RNNG that uses 1000 samples.

Proposal Distribution The most popular choice is to use a proposal distributions $q(z|x)$ that is a *discriminative* version of the generative model (e.g., they are models that predict the latent state conditioned on the text), with one key distinction: they are conditioned not only on the sequence of tokens that have been observed so far, but also on *future* tokens that the model will be evaluated on (a trait we will refer to as *peeking*). This conditioning behavior does not contradict any of the assumptions in Equation's (7.3) and (7.4), and is useful in preventing generation of invalid structures (for instance, parse trees with more leaves then there are words in the text), or ones that are inconsistent with future tokens. Dyer et al. (2016) and Kim et al. (2019) also increase the entropy of the proposal distribution by dividing logits by a temperature parameter τ (respectively using $\tau = 1.25$ and $\tau = 2.0$).

Aggregation An oft-overlooked fact is that Equation (7.2) can be substituted into Equation (7.1) in multiple ways. Letting $x_C = \{x_1, \ldots x_N\}$ denote a corpus of evaluation data comprised of instances (token sequences) x_n, estimates can be formed at the *instance level*:

$$\widehat{\text{PPL}}_{\mathcal{I}} = \exp\left(-\frac{1}{T}\sum_{n=1}^{N}\log \hat{p}(x_n)\right), \tag{7.5}$$

[1] Based both on the cited papers and available source code.

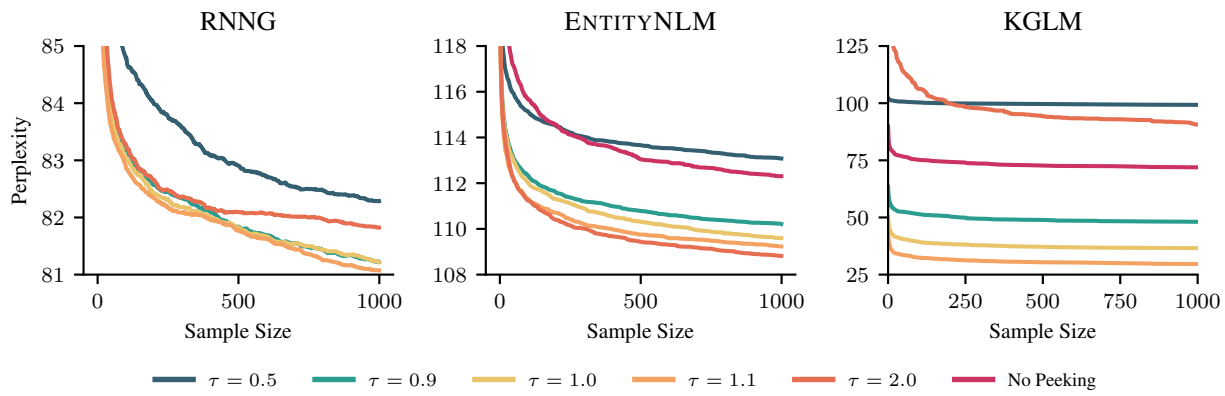


Figure 7.2: **Effect of increasing the number of samples** on instance-level perplexity estimates for different proposal distributions.

or at the *corpus level*:

$$\widehat{\text{PPL}}_{\mathcal{C}} = \exp\left(-\frac{1}{T}\log \hat{p}(x_{\mathcal{C}})\right), \tag{7.6}$$

i.e., average is either over each instance or the whole corpus.[2] RNNG and EntityNLM perform instance-level aggregation, whereas KGLM performs corpus-level aggregation. Note that these formulations are equivalent when not aggregating over samples, i.e. for non-latent language models.

7.3 Critical Evaluation

Thus far, research has neglected to measure the effectiveness of the practices detailed in Section 7.2. In the following section, we perform experiments to determine whether reporting estimates obtained from small sample sizes is warranted, as well as better understand the consequences of peeking and scaling the temperature of the proposal distribution.

Setup For our experiments, we use Kim et al. (2019)'s RNNG implementation[3], and the EntityNLM and KGLM implementations used in the previous chapter[4]. For RNNG and KGLM we use the pre-trained model weights. For EntityNLM we train the model from scratch following the

[2] One could also consider *token-level* estimates. To our knowledge, these have been unused by existing work.
[3] https://github.com/harvardnlp/urnng
[4] https://github.com/rloganiv/kglm-model

procedure described by Ji et al. (2017); results may not be directly comparable due to differences in data preprocessing and hyperparameters. We evaluate models on using the following datasets: RNNG is evaluated on the Penn Treebank corpus (Marcus et al., 1993), EntityNLM is evaluated on English data from the CoNLL 2012 shared task (Pradhan et al., 2014), and KGLM is evaluated on the Linked WikiText-2 corpus (Logan et al., 2019).

Experiments For EntityNLM and KGLM, we experiment with two kinds of proposal distributions: (1) the standard *peeking* proposal distribution that conditions on future evaluation data, and (2) a *non-peeking* variant that is conditioned only on the data observed by the model (this is akin to estimating perplexity by ancestral sampling). For RNNG we only experiment with peeking proposals, since a non-peeking variant generates invalid parse trees. For the peeking proposal distribution, we experiment with applying temperatures $\tau \in [0.5, 0.9, 1.0, 1.1, 2.0, 5.0]$. We report both corpus-level and instance-level estimates, as well as bounds produced using a direct, beam marginalization method we describe later.

Sample Size We plot instance-level perplexity estimates as sample size is varied in Figures 7.2 and 7.3a. We observe that the curves are monotonically decreasing in all settings. Consistent with our observation that importance sampled estimates of perplexity are a stochastic upper bound, this demonstrates that the bound is improved as sample size increases. Furthermore, none of the curves exhibit any signs of convergence even after drawing orders of magnitude more samples (Figure 7.3a); the estimated model perplexities continue to improve. Thus, the performance of these models is likely better than the originally reported estimates.

Aggregation Final estimates of perplexity computed using both corpus- and instance-level estimates are provided in Table 7.1. We note that instance-level estimates are uniformly lower by a wide margin. For example, using a temperature of $\tau = 1.1$ the estimated KGLM perplexity is approximately 10 nats lower using instance-level estimates. This is substantially better than the perplexity of 43 nats reported by Logan et al. (2019).

	RNNG	ENT	KGLM
Corpus-level			
$\tau = 0.5$	**94.4**	122.6	101.9
$\tau = 0.9$	96.0	122.7	59.3
$\tau = 1.0$	96.7	120.8	48.2
$\tau = 1.1$	97.9	120.7	**41.7**
$\tau = 2.0$	121.6	**120.5**	170.0
$\tau = 5.0$	734.0	152.5	7,468.7
No Peeking	-	131.7	86.8
Instance-level			
$\tau = 0.5$	85.3	113.5	99.3
$\tau = 0.9$	84.4	110.6	48.1
$\tau = 1.0$	84.2	110.0	36.6
$\tau = 1.1$	84.0	109.9	**29.6**
$\tau = 2.0$	**83.8**	**109.0**	90.7
$\tau = 5.0$	97.2	129.6	3,756.1
No Peeking	-	113.9	71.9

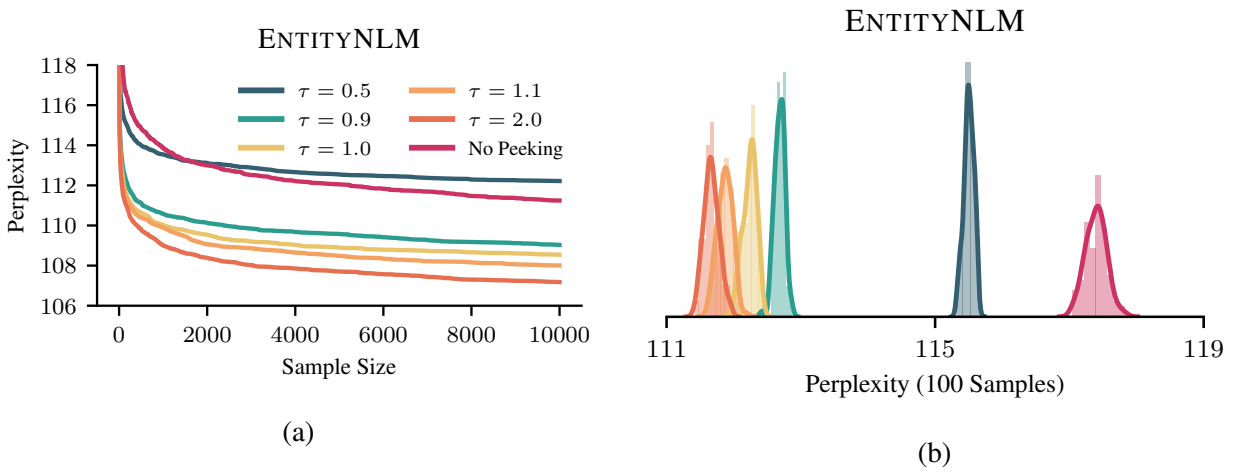
Table 7.1: **Perplexity estimates using different proposal distributions**, estimated at both the instance and corpus level. τ is temperature, and *No Peeking* refers to proposal distributions that are not conditioned on future outputs.

	RNNG	ENT	KGLM
$k = 1$	96.3	150.2	153.7
$k = 10$	87.0	147.1	152.6
$k = 100$	84.3	144.5	-

Table 7.2: **Strict perplexity upper bounds** obtained by marginalizing over the top-k states predicted by $q(z|x)$ using beam search.

Proposal Distribution These results also appear to indicate that choice of proposal distribution has a substantial effect on estimated perplexity. However, it could also be the case that the observed differences in performance across proposal distributions are due to random chance. We investigate whether this is the case for ENTITYNLM by examining the approximate density of perplexity estimates after drawing 100 importance samples (shown in Figure 7.3b).[5] Our results illustrate that the estimates are relatively stable; although there is some overlap between the better performing temperature values, the order of the modes matches the order reported in Table 7.1, and there is clear

[5]Obtained by Monte Carlo sampling 100 times.



(a) (b)

Figure 7.3: **ENTITYNLM instance-level perplexity estimates (a)** as the number of samples is increased to 10K. **Approximate density of ENTITYNLM perplexity estimates (b)** after drawing 100 importance samples.

separation from the estimates produced when $\tau = 0.5$ or by the non-peeking proposal distribution. Due to the relative cost of sampling we did not replicate this experiment for RNNG and KGLM.[6]

In general, we observe the peeking proposal distributions produce better estimates, and that better performance is obtained using temperatures that slightly increase the entropy of the proposal distribution (e.g., $\tau \in [1.1, 2.0]$), although the ideal amount varies across models. We also observe that the relative performance of proposal distributions is mostly preserved as the number of samples is increased. This suggests that good temperature parameters can be quickly identified by running many experiments with a small number of samples.

Beam Marginalization

An alternative to importance sampling is to directly marginalize over a subset of z values where we expect $p(x|z)$ is large. Specifically, we propose using the top-k most likely values of z identified by performing beam search using the proposal distribution $q(z|x)$. We will refer to this as *beam marginalization*. Because marginalization is only performed over a subset of the space, this method produces a strict upper bound of the true perplexity.

[6]Figs 7.3a & 7.3b took 1 week on a cluster of 15 NVidia 1080Tis.

Perplexity bounds obtained using beam marginalization are reported in Table 7.2. This method produces bounds close to the instance-level importance sampled estimates for RNNG, but does not perform well for the other models. This is likely due to the fact that latent space of RNNG (which operates on sentences and parse trees) is much smaller than ENTITYNLM and KGLM (which operate on documents and coreference chains/knowledge graphs).

Best Practices From these results we recommend the following practices for future work utilizing importance sampling: (1) aggregate importance samples at the instance level, (2) condition on all available information when designing proposals, (3) try increased temperatures when generating samples from the proposal distribution, good temperatures can be identified using relatively few samples, and (4) utilize as many samples as possible. In addition, consider using beam marginalization in applications where strict upper bounds are needed.

7.4 Summary of Contributions

In this chapter, we investigated the application of importance sampling to evaluating latent language models. Our contributions include: (1) showing that importance sampling produces stochastic upper bounds of perplexity, (2) a concise description of (sometimes unstated) common practices used in applying this technique, (3) a simple direct marginalization-based alternative to importance sampling, and (4) experimental results demonstrating the effect of sample size, sampling distribution, and granularity on estimates. The text in this chapter is based on the publication:

- *On Importance Sampling-Based Evaluation of Latent Language Models* (Logan IV et al., 2020, ACL 2020)

The author of this dissertation was the primary author on this publication, and is responsible for all of the results.

8

Using Language Models to Reflect Updated Information in Textual Knowledge Bases

"I keep forgettin' things will never be the same again."

– Michael McDonald, *I Keep Forgettin' (Every Time You're Near)*

Our study of language model knowledge base integration so far has looked at ways that knowledge bases can be used to improve language modeling performance. In this chapter, we will investigate the converse, i.e., whether language models can be used to improve consistency in knowledge bases. Consistency is an important issue, since information changes on a constant basis. Every day, athletes are traded to new teams, and musicians and actors produce new albums and TV shows. Maintaining textual knowledge bases to keep track of these changes requires considerable community effort. For instance, a team of 120K volunteer editors make 120 edits to English Wikipedia every minute, and write 600 new articles a day.[1] As the knowledge base grows, the amount of maintenance effort is compounded by the need to keep the knowledge base consistent; e.g., each edit may render information in one of the existing 6.3M+ articles obsolete.

[1] https://en.wikipedia.org/wiki/Wikipedia:Statistics

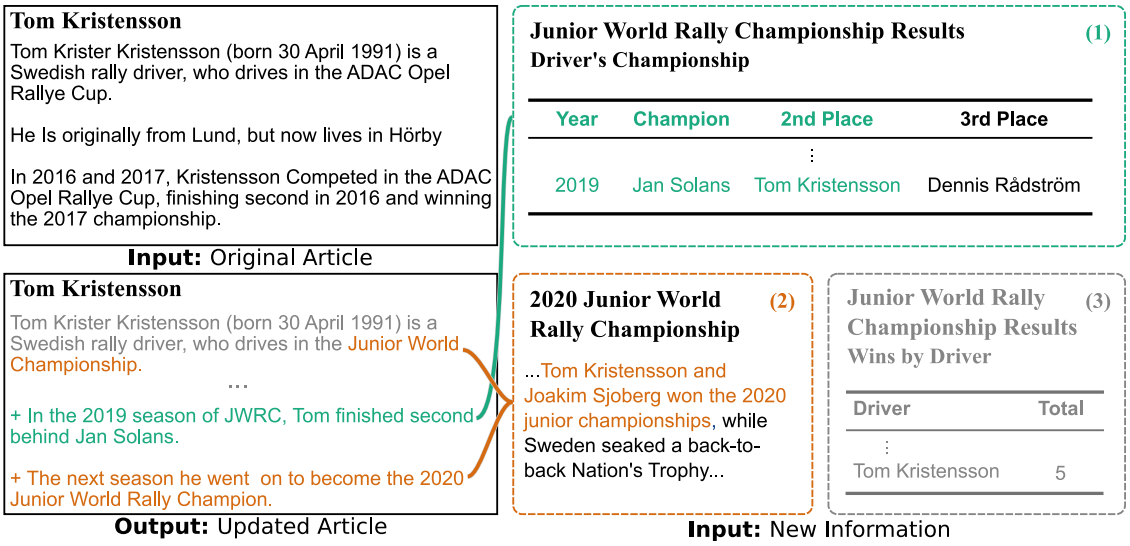


Figure 8.1: **Illustration of the FRUIT task**. An outdated *original article* and relevant *new information* are provided as inputs, and the goal is to generate the *updated article*. In this example, the original article about Tom Kristensson was written in 2020, and the new information is comprised of updated information about Tom Kristensson that has been added to other Wikipedia articles between 2020 and 2021. Given these inputs, the goal is to produce the updated 2021 version of article. Models need to identify the relevant supporting facts (orange and teal) to generate faithful updates while ignoring superfluous information (grey).

Assistive writing technologies have the potential to substantially reduce the burden of keeping text corpora up to date and consistent. However, existing work has mainly focused on correcting grammar (Wang et al., 2020), reducing repetitive typing (Chen et al., 2019), and following rhetorical directives (Sun et al., 2021), whereas the problem of producing edits grounded in external knowledge has received little attention (Kang et al., 2019). In contrast, numerous works have developed systems for distilling external knowledge into text (e.g., Wikipedia article generation) by treating the problem as multi-document summarization (Liu et al., 2018; Shi et al., 2021) or data-to-text generation (Bao et al., 2018; Parikh et al., 2020). However, these systems are not useful for updating existing texts as they can only generate text from scratch.

To help endow writing assistants with grounded editing capabilities, we introduce the novel generation task of *faithfully reflecting updated information in text* (FRUIT), where the goal is to incorporate new information into an existing piece of text. An illustration is provided in Figure 8.1. Given an outdated Wikipedia article and collection of new information about the article's subject, FRUIT

requires updating the existing text so that it is consistent with the new information, as well as adding text to reflect new salient facts, e.g., in Figure 8.1, the first sentence is updated to reflect that Tom Kristensson now drives in the Junior World Championship, and new sentences are added to reflect his achievements in 2019 and 2020.

FRUIT presents several unique challenges. First, unlike many generation tasks, models cannot obtain good performance by solely relying on their parametric world knowledge. Whenever the provided evidence contradicts parametric knowledge, the model must prefer the evidence, which recent work has shown is difficult for pretrained language models (Krishna et al., 2021; Longpre et al., 2021). Second, the generated text needs to be faithful to *both* the original article and the new evidence, *except* when evidence invalidates information in the existing article. Finally, this task requires models to jointly read and analyze evidence from both textual and tabular sources and determine which is relevant and which can be ignored, thus combining challenging aspects of both multi-document summarization and data-to-text generation.

To facilitate research on this task, we release the FRUIT-WIKI dataset, a collection of over 170K distantly supervised ("*silver*") update-evidence pairs. This dataset is produced by comparing pairs of English Wikipedia snapshots to identify updates to an article between two snapshots, and associating information from the other articles that supports these updates under a distant supervision assumption. As there is no guarantee that updates in the later Wikipedia snapshots can be supported by the collected evidence, we also collect a "*gold*" evaluation set of 914 human annotated update-evidence pairs where unsupported claims have been removed without disturbing fluency. We train and validate our models using silver data and then evaluate the final performance using gold data.

We establish initial benchmark results for a number of trivial and neural sequence-to-sequence baselines. We also introduce EDIT5, a T5-based model specially adapted for grounded editing, which establishes state-of-the-art performance on FRUIT-WIKI. Through an extensive set of analyses, we identify a number of failure modes needed to be improved upon in order to obtain

better performance on FRUIT-WIKI, as well as other interesting topics for future work on this task. We additionally release our data collection pipeline to allow researchers to produce data from future Wikipedia snapshots and other languages, which we show to produce high-quality silver data.

8.1 The FRUIT Task

8.1.1 Task Definition

In this section we introduce the task of *faithfully reflecting updated information in text* (FRUIT). Given an input piece of text focused on a topic or event, along with a collection of potentially new information about the subject of the text, the goal is to update the input text to reflect the new information. A concrete illustration of the task is provided in Figure 8.1. The original piece of text along with its updates are shown on the left, while the new information is shown on the right.

Formally, we assume access to pair of texts, A^t and $A^{t'}$, pertaining to a given subject, written at times t and t' (respectively). In addition, we assume access to a set of new information, a.k.a., evidence, $\mathcal{E}^{t \to t'} = \left\{E_1, \ldots, E_{|\mathcal{E}|}\right\}$, mentioning the subject written between times t and t'. As is shown in Figure 8.1, the evidence can contain structured objects (e.g., excerpts from tables) as well as unstructured text. Given A^t and $\mathcal{E}^{t \to t'}$ the goal is produce the updated text $A^{t'}$.

Successful completion of this task requires a number of complex and inter-related reasoning capabilities. For one, models must be able to identify which evidence contradicts existing portions of the source article, and which evidence introduces new salient information about the subject in order to correctly choose whether to alter the existing text vs. add new text. For example, in Figure 8.1 the first sentence is updated to reflect that Tom Kristensson now races in a different competition, whereas new sentences are added describing his achievements in the years 2019 and 2020. Models must also be able to determine whether a given piece of evidence should be used

at all, i.e., perform content selection. For example, in Figure 8.1, the number of rounds won by Kristennsen appears in the evidence but does not correspond to any piece of updated text. Although some evidence may not appear in the updated article, the converse is not true, the system should aim to generate an updated article where all the updates are faithful to the evidence.

8.1.2 Evaluation

In this section we introduce important considerations for evaluating FRUIT systems.

Evaluate on Updated Text There is often considerable overlap between the original and updated text. As we will see in Section 8.4 this poses a challenge for standard evaluation metrics like ROUGE (Lin, 2004) as systems can achieve high scores without making any updates. In this work, we propose to evaluate FRUIT systems using an alternative metric, UpdateROUGE, that only considers updated sentences instead full texts. For example, in Figure 8.1, the reference for UpdateROUGE only consists of the first and last two sentences.

Evaluate Faithfulness Ensuring that generations faithfully reflect information in the evidence and updated article is crucial. However measuring faithfulness of generations is an active area of research (Çelikyilmaz et al., 2020) and adapting existing metrics to the FRUIT task is non-trivial.

As a simple proxy for faithfulness, we choose to measure the token overlap between named entities appearing in the generation and the target article/evidence, where entities are identified using the named entity recognizer used by Guu et al. (2020). We specifically introduce the following measurements:

1. **Unsupported Entity Tokens.** This metric shows the average number of entity tokens appearing in generated updates that do not appear in the source article or evidence. This is intended to

capture the overall amount of unfaithful text, focusing on entities, where higher numbers indicate less faithfulness.

2. **Entity Precision and Recall.** Entity precision measures the fraction of entity tokens appearing in the generated updates that appear in target entities, whereas entity recall measures the fraction of entity tokens in the target that appear in the entities in generated updates. The latter is similar to UpdateROUGE but only evaluated on entities, and thus, potentially less sensitive to paraphrasing.

Parametric Knowledge Consideration FRUIT systems should incorporate information from the provided evidence into the update, and not information that happened to be present during training or pretraining. In this work we attempt to address this by evaluating models only on updates that were made to the text after the data used to pretrain and finetune the model was collected. As this setup precludes evaluating models trained after 2020 on FRUIT-WIKI, we release our data collection pipeline so that researchers can produce evaluation datasets from future versions of Wikipedia.

8.2 Dataset Collection and Analysis

As discussed in the introduction, keeping track of new information and then updating articles to reflect that information requires a massive amount of manual effort. Thus, in order to scalably collect sufficient data for training and evaluating FRUIT systems, some amount of automation is likely required. In this section we introduce the FRUIT-WIKI dataset and associated data collection pipeline, which allows the automatic collection of high-quality training and evaluation data for FRUIT from pairs of Wikipedia snapshots.

	Train	Test	
		Silver	Gold
Years	'19-'20	'20-'21	'20-'21
Articles	114K	54K	914
Edits	407K	182K	3.0K
Subst. Edits	135K	62K	1.3K
Evidence	720K	315K	7.7K
Content Sel.	93K	42K	913

Table 8.1: **FRUIT-WIKI Dataset statistics.** We use 10% of the training data as our validation data.

8.2.1 Pipeline

Our data collection pipeline produces distantly annotated training and evaluation data from pairs of Wikipedia snapshots. We will refer to the earlier snapshot as the *source* snapshot, and the later snapshot as the *target* snapshot.

Step 1. Collect Article Updates We compute the diff between the introductory sections of articles appearing in both the *source* and *target* snapshot to identify all of the material that has been updated (which will serve as A^t and $A^{t'}$). We also compute the diff between the non-introductory sections of articles to find new mentions of the subjects of other articles (which will serve as $\mathcal{E}^{t \to t'}$). These mentions can take the form of sentences in the text, as well as new table rows and list entries. Entities are disambiguated using Wikipedia hyperlinks.

Step 2. Filter Stylistic Updates A large number of edits to Wikipedia are stylistic (Daxenberger and Gurevych, 2012), and are therefore irrelevant to our task. In the next step of the pipeline, we attempt to filter articles that have only been superficially edited by keeping only those where at least one new *added entity* appears in the *target* snapshot.

Step 3. Identify Supporting Evidence In the last step of our pipeline, we seek to determine which pieces of evidence in $\mathcal{E}^{t \to t'}$ justify each of the updated sentences in $A^{t'}$. To do so, we make the following distant supervision assumption: an updated sentence $a \in A^{t'}$ containing an *added entity* s' is substantiated by a piece of evidence $E \in \mathcal{E}^{t \to t'}$ only if s' is also mentioned in E. The accuracy of the annotations produced by this assumption will be measured in Section 8.2.3.

Our pipeline is implemented using Apache Beam,[2] to allow for distributed processing. We plan on releasing the code upon publication to enable other users to produce FRUIT data from future Wikipedia snapshots, as well as languages other than English.

8.2.2 FRUIT-WIKI

We run our pipeline on English Wikipedia snapshots from Nov. 20, 2019 to Nov. 20, 2020 to produce the training dataset, and from Nov. 20, 2020 to June 1, 2021 to produce the evaluation dataset. Detailed statistics are provided in Table 8.1. On average, there are around 3 to 4 updates per article, and around 7 pieces of associated evidence. About 80% of updates require some form of content selection, i.e., ignoring some evidence, when performing updates.

We find that only a third of the updates are substantiated by one or more pieces of evidence according to our distant supervision assumption. Thus, the remaining updates are either: a) superficial changes to the source article, or b) additions of new unsupported claims. The latter is a particular issue as unsupported claims can cause the model to learn to hallucinate during training, and should be impossible for the model to guess during evaluation. Through the usage of human annotations and carefully selected evaluation metrics we will study the extent to which this is an issue throughout the rest of the chapter.

[2] https://beam.apache.org/

UpdateROUGE			Entity	
1	2	L	Prec.	Recall
87.4	84.6	87.1	91.8	94.6

Table 8.2: **Inter-annotator agreement**.

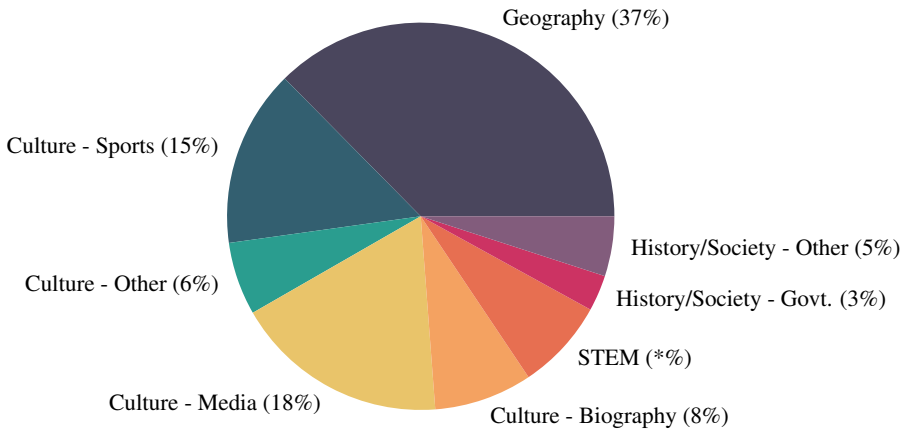


Figure 8.2: **Topic distribution of FRUIT-WIKI.**

We categorize articles in our dataset using the Wikimedia Foundation's topic model (Asthana and Halfaker, 2018). The distribution of topics is displayed in Figure 8.2. We find that the majority (approximately 50%) of updates deal with cultural topics (e.g., sports, media, personal biographies), and geographic entities (e.g., countries, states) which intuitively are likely to be affected by current events. while there are few updates to STEM- and history-related articles.

8.2.3 Gold Evaluation Data

To address the issue of unsupported claims during evaluation, we hired a team of 9 annotators to produce a *"gold"* evaluation subset of our test dataset. We collect annotations for 914 update-evidence pairs where each instance is corrected to ensure that all of the updates are supported. For the remainder of the chapter we will refer to the distantly supervised test dataset annotations as *"silver"*.

Annotation Process For each instance, annotators were shown the source article, evidence, and a marked up copy of the target article. In the marked up article, each updated sentence was highlighted and prefixed with reference labels to the supporting evidence identified by our pipeline. The correction process proceeded in two steps. In the first step, annotators were asked to highlight all of the unsupported claims and incorrect reference labels in the target article. In the second step, annotators were then asked to remove the unsupported text and minimally update the article to preserve fluency.

Annotators attended an initial 30 minute training and were provided regular feedback from the authors during the early stages of annotation. An additional annotator was hired with the sole job of checking the other annotator's work and correcting their mistakes. In total annotators spent roughly 500 hours on annotation. The annotation interface and a completed annotation are shown in Figure B.6.

Agreement We measure annotator agreement using a subset of 100 instances that were annotated by multiple annotators. Following Chen et al. (2015) and Shi et al. (2021), we quantify agreement by computing the evaluation metrics described in Section 8.1.2. The results are provided in Table 8.2. We observe high inter-annotator agreement with all scores in the 80s and 90s.

Analysis Statistics for the gold evaluation dataset are provided in Table 8.1. Overall, they closely resemble the statistics for the distantly supervised data with one exception: the fraction of substantiated updates has increased.

To measure the quality of our silver data, we re-apply the approach used to measure inter-annotator agreement to compute agreement between the gold and silver annotations. We also measure the *reference agreement*, i.e., the fraction of reference labels kept by the annotators. Results are provided in Table 8.3. We find that agreement is high with most scores in the 80s, a strong indication that the data produced by our pipeline is high quality. In particular, the high UpdateROUGE scores

UpdateROUGE			Entity		Reference Agreement
1	2	L	Prec.	Recall	
83.7	81.2	83.4	90.4	100.0	84.5

Table 8.3: **Gold and silver annotation agreement**. Quality of Silver Annotations by using the Gold.

	UpdateROUGE			Entity		Unsup.
	1	2	L	Prec.	Recall	Tokens
Copy Source	0.0	0.0	0.0	0.0	0.0	0.00
+ All Evidence	18.8	6.9	12.0	37.9	64.9	0.00
T5-Large	31.1	18.4	24.4	52.7	44.9	2.67
+ Evidence Input	44.3	29.4	36.8	62.2	50.7	2.34
EDIT5-Small	41.2	27.3	35.3	62.4	44.9	1.71
EDIT5-Base	47.0	32.1	39.7	62.2	**54.9**	2.28
EDIT5-Large	46.3	32.4	39.6	67.2	53.1	**1.54**
EDIT5-3B	**47.4**	**34.0**	**41.1**	**69.9**	52.5	1.58

(a)

Grounded Updates	50
Additional Content	15
Missing Content	22
Ungrounded Updates	35
Number/Date	21
Distorted Evidence	11
Hallucination	14
No Updates	14

(b)

Table 8.4: **Model results on gold evaluation data (a).** EDIT5 outperforms T5 models in all metrics. **Error analysis for EDIT5-3B (b)**. We find that the model makes correct, grounded updates on 50% of the inspected articles. For incorrect updates, ungrounded numbers/dates are one of the main sources of error.

provide further evidence that only a small amount of the updated text in the weakly supervised data is unsupported, while the high reference agreement indicates that our distant supervision assumption is usually accurate.

8.3 Methods

In this section we introduce baseline methods to establish initial benchmark results on FRUIT-WIKI. We consider trivial approaches that copy task inputs, as well as T5, a neural sequence-to-sequence baseline which has shown strong performance on related tasks such as summarization (Raffel et al.,

2020; Rothe et al., 2021) We additionally introduce EDIT5, a variant of T5 that produces a sequence of edits instead of the entire updated text, and employs additional tweaks to improve performance.

8.3.1 Copy Baselines

The first set of baselines we introduce are trivial methods that merely copy the input. We consider two variants:

- **Copy Source**: Generates a copy of the source article, and
- **Copy Source + Evidence**: Generates a copy of the source article concatenated with the evidence.

Our evaluation metrics only apply to unstructured text, however the evidence may contain structured tables. In order to convert these tables to text, we apply a conventional linearization scheme (Lebret et al., 2016; Wiseman et al., 2017) that separates table entries using row and column delimiters.

8.3.2 T5

T5 (Raffel et al., 2020) is a pretrained sequence-to-sequence (Sutskever et al., 2014b) model based on the transformer architecture (Vaswani et al., 2017b). Similar to the previous section we experiment with two variants:

- **T5**: Only includes the source article in its input,
- **T5 + Evidence Inputs**: Includes both the source article and evidence in the input.

Tabular inputs are linearized using the same approach described in the previous section. Experiments are performed using the JAX-based T5X library.[3] All models are trained using the AdaFactor (Shazeer and Stern, 2018) optimizer with a batch size of 128, learning rate of 1e-3, and

[3] https://github.com/google-research/t5x

dropout rate of 0.1. Models were trained for 30,000 iterations on a cluster of 16 2nd generation TPUs for <3B parameter models, and 32 TPUs for 3B parameter models.

8.3.3 EDIT5

Lastly, we introduce EDIT5, which improves upon the T5-based approach described in the previous section through the usage of a compressed output format that removes the need to write the entire update from scratch and encourages content planning. The output is modified in two ways:

First, as the majority of text in the target article is copied from the source, we replace any copied sentence with a single *copy token* identifying the sentence, e.g., if the second sentence is copied it is replaced by the token [2]. Similar to a copy mechanism (See et al., 2017), this allows the model to dedicate less capacity to repeating sequences from the input. As the resulting output resembles that produced by the diff data comparison utility, we refer to this as a diff-formatted output.

Second, before each update we insert a sequence of *reference tokens* identifying the pieces of evidence that support the update, e.g., if the first and third piece of evidence in $\mathcal{E}^{t \to t'}$ support an update then the update is prefaced by (1)(3). This approach, inspired by the use of entity chains for summarization (Narayan et al., 2021), trains the model to plan which references to use before generating an update. These reference tokens are removed from the output text of the model prior to computing the evaluation metrics.

An example of the EDIT5 output format is provided in Figure 8.3, and a comparison to the T5 output format is provided in Appendix B.1. Training details and hyperparameters match the setup described in Section 8.3.2.

(2) Tom Krister Kristensson (born 30 April 1991) is a Swedish rally driver, who drives in the Junior World Championship. [1] [2] (1) In the 2019 season of JWRC, Tom finished second behind Jan Solans. (2) The next season he went on to become the 2020 Junior World Rally champion.

Figure 8.3: **EDIT5 output format.** Instead of generating the fully updated text, EDIT5 generates sequences of edited sentences, copy tokens (e.g., [2], which means copy the second sentence), and reference tokens (e.g., (1), which means the following sentence should use the first piece of evidence).

8.4 Results and Analysis

Baseline results on the gold evaluation data are provided in Table 8.4a. In general, we find that the copy baselines perform worse than T5 and T5 performs worse than EDIT5. Notably, the copy source baseline rightfully scores zero on all metrics, while we will later find that it obtains a high ROUGE score.

Although our models are trained on silver data, they still obtain good performance on the gold evaluation set. This shows the high quality of our silver data collection pipeline, and T5's ability to generate reasonable updates based on the evidence.

For the T5 baselines, we find that adding evidence to the input results significant increase in all metrics, demonstrating that using the evidence is crucial to obtaining good performance.

EDIT5 obtains additional 3-5% absolute increase in all performance metrics compared to T5, establishing EDIT5 as a strong baseline for future systems to be compared against. The reduction of unsupported entity tokens implies that EDIT5 hallucinates less frequently than T5 models. Results are provided for different model sizes to illustrate how performance scales with parameter counts.

Ablation Study We perform an ablation study to measure the impact of the modifications made to the target output of EDIT5. The results are provided in Table 8.5 We observe that both the diff format and including reference tokens have a positive impact on the evaluation metrics, with reference tokens having the larger impact.

	UpdateROUGE			Entity		Unsupp.
	1	2	L	Prec.	Rec.	Tokens
EDIT5	**46.3**	**32.4**	**39.6**	**67.2**	**53.1**	**1.54**
- Diff	45.5	31.7	39.1	66.8	50.8	1.66
- Ref.	45.1	31.6	38.8	66.3	50.7	1.89

Table 8.5: **EDIT5 Ablations**.

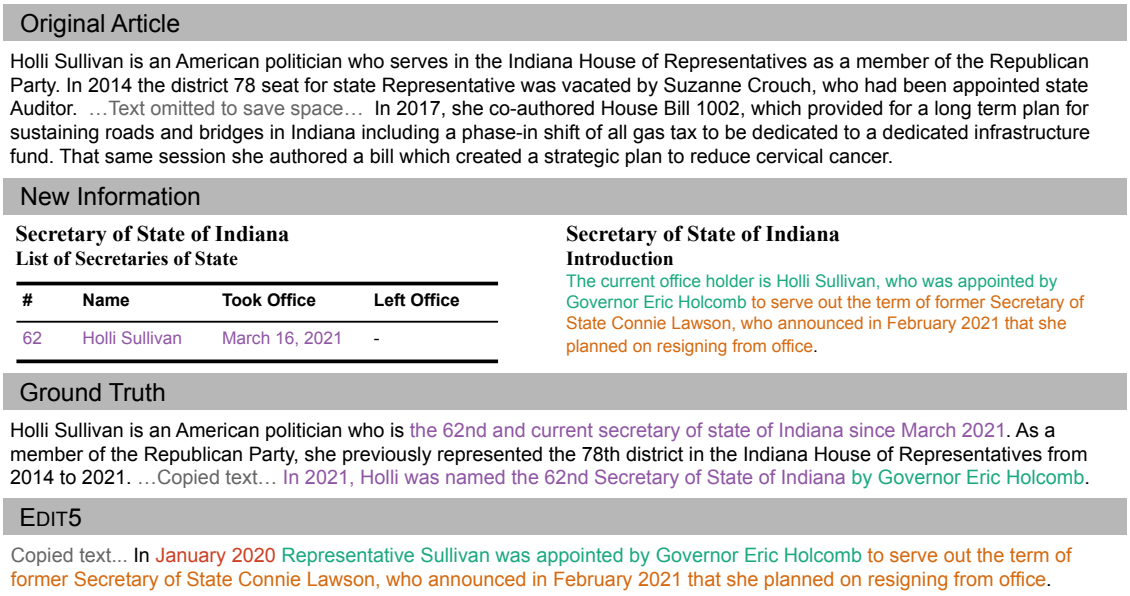


Figure 8.4: **Example model outputs.** EDIT5 updates the original article by paraphrasing sentences from the textual evidence, however misses relevant information in the table, and generates a hallucinated date.

Example Output An example EDIT5 output is provided in Figure 8.4, and additional outputs in Appendix B.2. The examples illustrate important features of the task. In Figure 8.4 the goal is to update the Wikipedia article for Holli Sullivan to reflect her new role of Secretary of State of Indiana. In the reference, this information is reflected in an updated version of the first sentence as well as in a newly added last sentence. An additional sentence is added after the first sentence paraphrasing the introduction of the source article, which describes Sullivan's previous position as a member of the Indiana House of Representatives.

In the EDIT5 output for this example, information is only added at the end of the article. While the model correctly states that Sullivan was appointed to be Secretary of State by Governor

Eric Holcomb, as well as includes additional context surrounding Sullivan's appointment that is paraphrased from the evidence, there are some issues with the output. First, because the first sentence of the article is not updated there is conflicting information about Sullivan's current position. Second, the added sentence hallucinates that Sullivan was appointed in January 2020 when she was actually appointed in March 2021, a fact that directly appears in the evidence.

Categorizing Errors To better understand the types of errors made by EDIT5, we review a random sample of 100 of its predictions on the gold evaluation data and categorize them as either: *grounded updates*, meaning all generated claims are supported, *ungrounded updates*, meaning at least one unsupported claim appears in the output, or *no updates*, meaning the model did not predict any updates. For grounded updates we additionally keep track of how many updates include *additional content* not present in the ground truth update, or are *missing content* that appears in the ground truth update. For ungrounded updates we track whether an incorrect *number/date* appears in the update, the model *distorted evidence*, i.e., paraphrased or combined claims in the evidence in a way that changed their meaning, or *hallucinated* new claims.

The results of this analysis are presented in Table 8.4b. We find that EDIT5 makes no mistakes on half of the examples, however a substantial portion of these updates had some issue with content selection. Of the incorrect updates, the most common mistake was incorrect numbers and dates, followed by hallucinations, and finally distorted evidence. This suggests that improving numeracy could be a fruitful line of study in future work on this task.

	ROUGE		
	1	2	L
Copy Source	78.1	69.3	75.0
T5-Large	57.0	44.2	49.5
EDIT5-Large	78.6	69.1	72.7

Table 8.6: **ROUGE scores are insensitive to edits.**

UpdateROUGE			Entity		Unsup.
1	2	L	Prec.	Rec.	tokens
100.0	100.0	94.3	75.4	92.8	92.8

Table 8.7: **Spearman rank correlation between gold and silver performance metrics**.

ROUGE is Problematic We provide ROUGE scores for each of the baseline models on the gold evaluation data in Table 8.6. In contrast to the previous results, we find that the simple copy source baseline attains a strong score of 77.4 despite making no updates. This is better than the T5 baseline results and comparable to the EDIT5 results. This illustrates the importance of evaluating on updates rather than the whole text.

Silver Data is Useful for Evaluation The results in Section 8.2.3 demonstrate high agreement between the silver and gold evaluation data which begs the question: can silver data be used in place of gold data for evaluation? To answer this, we measure the Spearman rank correlation between the gold baseline results in Table 8.4a and silver baseline results in Table 8.8. Rank correlations for each of the metrics are shown in Table 8.7. Overall we find high rank correlation for each of the metrics, which suggests silver evaluation performance is a reliable indicator of gold performance. Thus, models whose pretraining data overlaps FRUIT-WIKI may be evaluated and compared on data produced by running our pipeline on future Wikipedia snapshots without requiring further human evaluation.

Controllability The improvement we obtained from EDIT5 over T5 implies that more controls can be added into the model. In this section we investigate whether additional control provided by the users can improve the overall generations. We follow Keskar et al. (2019) and Narayan et al. (2021), and provide more detailed instruction by adding *control codes*, i.e., special tokens, to the *input* that instruct the model whether to add, copy, edit or remove a sentence, as well as which evidence to use when making an addition or edit. We use the target text to provide oracle labels

	UpdateROUGE			Target Entity		Evid.
	1	2	L	P	R	Acc
T5-Large	26.8	15.9	22.3	56.3	29.8	2.33
+ Evid.	39.2	27.3	34.2	66.9	42.4	1.63
EDIT5						
Small	37.8	24.9	32.6	61.4	41.2	1.53
Base	42.8	28.7	36.4	60.5	**49.2**	2.32
Large	42.7	29.9	37.2	66.1	47.5	1.47
3B	**43.8**	**31.5**	**38.6**	**68.4**	48.6	1.53

Table 8.8: **Baseline results on silver evaluation data.**

	UpdateROUGE			Entity		Unsup.
	1	2	L	Prec.	Rec.	Tokens
EDIT5	46.3	32.4	39.6	67.2	53.1	**1.54**
Control	**57.6**	**42.1**	**50.2**	**70.5**	**64.5**	2.42

Table 8.9: **Controllability**. Using control codes that indicate which sentences to delete, add or edit, and which evidence to use, can greatly improve generation.

for the control code, and see if the EDIT5 can take advantage of the codes. Example inputs and predictions are provided in Figure 8.5.

Results on the gold evaluation data are provided in Table 8.9. Including oracle control codes in the input produces a substantial 10% absolute improvement in all metrics besides unsupported tokens. This demonstrates that increased user control has the potential to produce updates that more closely resemble the desired output.

8.5 Summary of Contributions

In this chapter, we introduced FRUIT, a novel text generation task where the goal is to update an article to reflect new information about its subject. This task investigates how language models can be used to enforce consistency in textual knowledge bases. To enable research on this task, we

Figure 8.5: **Using control codes.**

formulated a pipeline for extracting weakly supervised training and evaluation data from pairs of Wikipedia snapshots, and collected data for the years 2019-2020 and 2020-2021, as well as human annotated gold evaluation data. We additionally provided results for several language models, that demonstrate both the feasibility of this task, as well as strong correlation between gold and distantly supervised data evaluation performance that establishes the trustworthiness of future data produced using our pipeline for evaluation.

The text in this chapter is based on the publication:

- *FRUIT: Faithfully Reflecting Updated Information in Text* (Logan IV et al., 2021b, NAACL 2022)

of which the author of this manuscript is the sole primary author. The FRUIT task was conceived by the author, Ming-Wei Chang, and Alexandre Passos during the early phases of the author's

internship at Google in 2021. Primary credit for the data collection pipeline, FRUIT-WIKI dataset, EDIT5 design, experimental results, and evaluation metrics belongs to the author, however, Ming-Wei and Alexandre provided careful advice throughout the creation of these materials. Sameer Singh deserves primary credit for devising the unsupported tokens metric. The results presented in this chapter can be replicated using the code and instructions provided at: https://github.com/google-research/language/tree/master/language/fruit.

9

Conclusions and Future Work

"Yes, there are two paths you can go by, but in the long run there's still time to change the road you're on."

– Led Zeppelin, *Stairway to Heaven*

This dissertation was motivated by the crucial need to understand and leverage the knowledge within neural language models. As these models continue to scale in terms of training data and parameters, it is important to be able to measure the material benefits this scaling confers in terms that are easier to understand than likelihood on a held out test dataset. In the first part of this dissertation, we sought to instead directly test these models for specific capabilities using their natural input and output spaces via the method of prompting. Our results demonstrated that, on the one hand, neural language models attain a surprising amount of knowledge from their pretraining data, however, on the other hand, this knowledge can be incomplete or difficult to manipulate. Thus, in the second part of this dissertation, we explored the ways that knowledge in language models can be integrated with the discrete, manipulable knowledge stored in knowledge bases, laying out the groundwork for techniques that reciprocally improve these two objects using one another.

9.1 Summary of Contributions

Part I introduced the method of prompting which tests for knowledge by reformulating tasks as fill-in-the-blanks- and complete-the-sentence-style problems. In Chapter 3, we demonstrated how to create prompts using templates to test whether a language model was aware of facts present in a knowledge graph, or had acquired the capability to perform classification tasks. In Chapter 4, we introduced AUTOPROMPT which automates the process of prompt writing, and showed that automatically written prompts are better than manually written prompts at eliciting knowledge from a language model, and in some cases demonstrate that language models have enough inbuilt knowledge to obtain near state-of-the-art performance on NLP tasks without requiring finetuning. In Chapter 5, we further investigated how prompting could be used to improve accuracy of language models in few-shot learning settings.

Part II of this dissertation investigated ways to integrate the knowledge stored in language models and knowledge bases. In Chapter 6 we introduced the KGLM and KnowBERT, language models that leverage representations of entities and relations to improve their ability to produce factually correct language. Chapter 7 then took a brief detour to further investigate the application of importance sampling for perplexity-based evaluation of the KGLM, as well as other language models that incorporate latent structure at inference time. Lastly, Chapter 8 introduced the task of faithfully reflecting updated information in text (FRUIT), which investigates the application of language models towards enforcing consistency in knowledge bases that have been edited to include new information.

9.2 Impact

This work provides insight into the nature of knowledge in neural language models and how it can be measured, best used in downstream applications, and improved upon. To some extent, the impact

of these ideas is already being realized—at the time of writing, the publications covered in this manuscript have over 650 combined citations according to Google Scholar. In particular, the method of prompting described in this work provides a powerful textual interface for NLP practitioners to interact with language models and its usage is not limited to the applications considered in this dissertation. During the time that this manuscript was written, prompting has become a widely used technique for a number of tasks in NLP (Brown et al., 2020; Schick and Schütze, 2021a; Li and Liang, 2021; Lester et al., 2021; Sanh et al., 2021), and we expect that, as neural language models continue to become unwieldy for most researchers to deploy and update themselves, the popularity of this method will continue grow. Similarly, the methods we presented that enable language models to access knowledge bases during inference have also influenced a number of subsequent works (Févry et al., 2020; Verga et al., 2021; Wang et al., 2021), and exhibit the exciting potential to update language models' knowledge of the world without requiring expensive retraining.

9.3 Limitations and Future Work

Although this work provides compelling evidence regarding the efficacy of prompting and potential benefits of integrating knowledge bases and language models, it is by no means a complete treatment of these topics. In this section we identify limitations of the work presented and ideas for future research.

Consistent Prompt-Based Evaluation Across Models and Vocabularies As we discussed in Chapter 3, the results of prompt-based evaluations of knowledge in generative and masked language models are not directly comparable since masked language models are provided extra information about the number of tokens in the answer and the finiteness of the prompt while generative language models account for the infinite number of ways that a prompt could be completed. Accordingly, one important area for future investigation is designing prompt-based evaluation schemes that

put generative and masked language models on equal footing, either by limiting the information provided to masked language models (e.g., marginalizing over different answer lengths, or having the model predict pad tokens (Schick and Schütze, 2021b)), or providing additional information to generative models (e.g., employing constrained decoding approaches (Qin et al., 2022)).

On a related note, our work is limited in terms of how multi-token answers to prompts are addressed for MLMs. For the evaluation of BERT in Chapter 3 and KnowBERT in Chapter 6 we used a greedy approach of taking the arg max over all of the mask predictions independently. Thus, we also recommend looking into using alternative MLM decoding approaches such as those proposed by Wang and Cho (2019) and Ghazvininejad et al. (2019).

Pretraining Models for Better Prompting The success of the prompting approaches presented in this work hinges on a model's ability to produce output distributions over vocabulary terms that are meaningfully associated with task labels. Presumably, the reason that prompting works so well for language models is that they learn these associations from the large amount of text data that they are pretrained on. However, language model training objectives and data only roughly resemble the prompts presented to these models at inference time. Thus, it may be possible to implement training schemes that improve model performance when prompted at inference time. To some extent, this idea is already being explored, as work such as Sanh et al. (2021) has shown that finetuning language models on prompts in a multitask setting can improve their 0-shot performance on unseen tasks. However,

Investigation into Why Prompting is Useful in Few-Shot Settings Although our prompting results illustrate that language models possess some degree of task-specific knowledge, and that this knowledge can be effectively leveraged to increase accuracy in few-shot settings, it is still not clear *why* prompting is so effective. This begs the question: are language models learning reasoning skills from their pretraining data, or just memorizing answers and exploiting co-occurrence statistics? We

have seen some evidence (not published in this dissertation) that the latter appears to be the case for arithmetic problems (Razeghi et al., 2022), however, more work needs to be done to even frame how to answer these kinds of questions for more complicated tasks and forms of reasoning.

Using Common-Sense Knowledge Graphs Our work on integrating knowledge bases and language models focused solely on "factual" knowledge bases such as Wikipedia and Wikidata. However, a number of knowledge bases capturing other kinds of knowledge such as commonsense exist. It is thus interesting to ask whether there may be benefit to using the techniques we developed in order to have more sensible language models.

Better understanding of trade-offs between modular and parametric representations of knowledge Observe that counterfactual edits to KG produced a corresponding change in KGLM output, however EDIT5 had difficulty writing edits based on OOD facts (e.g., due to time split). The phenomenon of massive text-KB LMs reluctance to use external knowledge that contradicts pretraining data has been observed in other works (Krishna et al., 2021; Longpre et al., 2021). So one the one hand there appears to be some evidence that approach of copying from a modular source of knowledge, e.g., a KG, can allow more robustness to changes in information. On the other hand, KGs are sparse (Nickel et al., 2016), and there is compelling evidence that shows that finetuning an LM to perform KG completion in the text space produces more dense KG coverage (Bosselut et al., 2019). Is there a best of both worlds?

Bibliography

Sungjin Ahn, Heeyoul Choi, Tanel Pärnamaa, and Yoshua Bengio. 2016. A neural knowledge language model. *ArXiv preprint*, abs/1608.00318.

Sumit Asthana and Aaron Halfaker. 2018. With few eyes, all hoaxes are deep. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW).

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. In *NIPS Deep Learning Symposium*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Ivana Balazevic, Carl Allen, and Timothy Hospedales. 2019a. TuckER: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5185–5194, Hong Kong, China. Association for Computational Linguistics.

Ivana Balazevic, Carl Allen, and Timothy Hospedales. 2019b. TuckER: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5185–5194, Hong Kong, China. Association for Computational Linguistics.

Junwei Bao, Duyu Tang, Nan Duan, Zhao Yan, Yuanhua Lv, Ming Zhou, and Tiejun Zhao. 2018. Table-to-text: Describing table region with natural language. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5020–5027. AAAI Press.

Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *ArXiv preprint*.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 932–938. MIT Press.

Christopher M. Bishop and Nasser M. Nasrabadi. 2007. Pattern recognition and machine learning. *J. Electronic Imaging*, 16:049901.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 1247–1250, New York, NY, USA. Association for Computing Machinery.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2787–2795.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. 2019. COMET: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4762–4779, Florence, Italy. Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, R. Mercer, and P. Roossin. 1988. A statistical approach to language translation. In *Coling Budapest 1988 Volume 1: International Conference on Computational Linguistics*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Thiago Castro Ferreira, Diego Moussallem, Emiel Krahmer, and Sander Wubben. 2018. Enriching the WebNLG corpus. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 171–176, Tilburg University, The Netherlands. Association for Computational Linguistics.

Mia Xu Chen, Benjamin N. Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, Timothy Sohn, and Yonghui Wu. 2019. Gmail smart compose: Real-time assisted writing. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 2287–2295. ACM.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C. Lawrence Zitnick. 2015. Microsoft coco captions: Data collection and evaluation server. *ArXiv*, abs/1504.00325.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 160–167. ACM.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges Workshop*.

Johannes Daxenberger and Iryna Gurevych. 2012. A corpus-based study of edit categories in featured and non-featured Wikipedia articles. In *Proceedings of COLING 2012*, pages 711–726, Mumbai, India. The COLING 2012 Organizing Committee.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019a. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019b. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *ArXiv preprint*, abs/2002.06305.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Hady Elsahar, Pavlos Vougiouklis, Arslen Remaci, Christophe Gravier, Jonathon Hare, Frederique Laforest, and Elena Simperl. 2018. T-REx: A large scale alignment of natural language with knowledge base triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. 2017. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1615–1625, Copenhagen, Denmark. Association for Computational Linguistics.

Thibault Févry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski. 2020. Entities as experts: Sparse memory access with entity supervision. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4937–4951, Online. Association for Computational Linguistics.

Octavian-Eugen Ganea and Thomas Hofmann. 2017. Deep joint entity disambiguation with local neural attention. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2619–2629, Copenhagen, Denmark. Association for Computational Linguistics.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. 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 3816–3830, Online. Association for Computational Linguistics.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain. Association for Computational Linguistics.

John Geweke. 1989. Bayesian inference in econometric models using monte carlo integration. *Econometrica*, 57(6):1317–1339.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121, Hong Kong, China. Association for Computational Linguistics.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017. Improving neural language models with a continuous cache. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Nitish Gupta, Sameer Singh, and Dan Roth. 2017. Entity linking via joint encoding of types, descriptions, and context. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2681–2690, Copenhagen, Denmark. Association for Computational Linguistics.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training.

Hiroaki Hayashi, Zecong Hu, Chenyan Xiong, and Graham Neubig. 2020. Latent relation language models. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7911–7918. AAAI Press.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. 2011. Robust disambiguation of named entities in text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 782–792, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Shankar Iyer, Nikhil Dandekar, and Kornel Csernai. 2017. First quora dataset release: Question pairs.

Frederick Jelinek, Lalit R. Bahl, and Robert L. Mercer. 1975. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Trans. Inf. Theory*, 21:250–256.

Frederick Jelinek and Robert L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In *In Proceedings of the Workshop on Pattern Recognition in Practice*, pages 381–397, Amsterdam, The Netherlands: North-Holland.

Yangfeng Ji, Chenhao Tan, Sebastian Martschat, Yejin Choi, and Noah A. Smith. 2017. Dynamic entity representations in neural language models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1830–1839, Copenhagen, Denmark. Association for Computational Linguistics.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 1st edition. Prentice Hall PTR, USA.

Herman Kahn. 1950. Random sampling (monte carlo) techniques in neutron attenuation problems–i. *Nucleonics*, 6(5):27–passim.

Jun Seok Kang, Robert Logan, Zewei Chu, Yang Chen, Dheeru Dua, Kevin Gimpel, Sameer Singh, and Niranjan Balasubramanian. 2019. PoMo: Generating entity-specific post-modifiers in context. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 826–838, Minneapolis, Minnesota. Association for Computational Linguistics.

Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoust. Speech Signal Process.*, 35:400–401.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *ArXiv*, abs/1909.05858.

Yoon Kim, Alexander Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019. Unsupervised recurrent neural network grammars. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1105–1117, Minneapolis, Minnesota. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Nikolaos Kolitsas, Octavian-Eugen Ganea, and Thomas Hofmann. 2018. End-to-end neural entity linking. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 519–529, Brussels, Belgium. Association for Computational Linguistics.

Kalpesh Krishna, Aurko Roy, and Mohit Iyyer. 2021. Hurdles to progress in long-form question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4940–4957, Online. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Teven Le Scao and Alexander Rush. 2021. How many data points is a prompt worth? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2627–2636, Online. Association for Computational Linguistics.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213, Austin, Texas. Association for Computational Linguistics.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 188–197, Copenhagen, Denmark. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. 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 4582–4597, Online. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Xiao Ling, Sameer Singh, and Daniel S. Weld. 2015. Design challenges for entity linking. *Transactions of the Association for Computational Linguistics*, 3:315–328.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021a. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. In *6th International*

Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021b. GPT understands, too. *ArXiv preprint*, abs/2103.10385.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *ArXiv preprint*, abs/1907.11692.

Robert Logan, Nelson F. Liu, Matthew E. Peters, Matt Gardner, and Sameer Singh. 2019. Barack's wife hillary: Using knowledge graphs for fact-aware language modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5962–5971, Florence, Italy. Association for Computational Linguistics.

Robert L. Logan IV, Ivana Balazevic, Eric Wallace, Fabio Petroni, Sameer Singh, and Sebastian Riedel. 2021a. Cutting down on prompts and parameters: simple few-shot learning with language models. *ArXiv preprint*, abs/2106.13353.

Robert L. Logan IV, Matt Gardner, and Sameer Singh. 2020. On importance sampling-based evaluation of latent language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2171–2176, Online. Association for Computational Linguistics.

Robert L. Logan IV, Alexandre Passos, Sameer Singh, and Ming-Wei Chang. 2021b. FRUIT: faithfully reflecting updated information in text. *ArXiv preprint*, abs/2112.08634.

Shayne Longpre, Kartik Perisetla, Anthony Chen, Nikhil Ramesh, Chris DuBois, and Sameer Singh. 2021. Entity-based knowledge conflicts in question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7052–7063, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *ArXiv preprint*, abs/2104.08786.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Stephen Merity, Bryan McCann, and Richard Socher. 2017a. Revisiting activation regularization for language rnns. *ArXiv*, abs/1708.01009.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017b. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Merriam-Webster. 2022. knowledge.

Todor Mihaylov and Anette Frank. 2018. Knowledgeable reader: Enhancing cloze-style reading comprehension with external commonsense knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 821–832, Melbourne, Australia. Association for Computational Linguistics.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. of INTERSPEECH*.

Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119.

George A. Miller, Martin Chodorow, Shari Landes, Claudia Leacock, and Robert G. Thomas. 1994. Using a semantic concordance for sense identification. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2021. On the stability of fine-tuning BERT: misconceptions, explanations, and strong baselines. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Shashi Narayan, Yao Zhao, Joshua Maynez, Gonccalo Simoes, and Ryan T. McDonald. 2021. Planning with entity chains for abstractive summarization. *ArXiv*, abs/2104.07606.

Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modelling. *Comput. Speech Lang.*, 8:1–38.

Maximilian Nickel, Kevin P. Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104:11–33.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 809–816. Omnipress.

Juri Opitz. 2019. Argumentative relation classification as plausibility ranking. *ArXiv preprint*, abs/1901.11373.

Art B. Owen. 2013. Monte carlo theory, methods and examples.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186, Online. Association for Computational Linguistics.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. 2021. True few-shot learning with language models. *ArXiv preprint*, abs/2105.11447.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. 2019. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54, Hong Kong, China. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Boris Polyak and Anatoli B. Juditsky. 1992. Acceleration of stochastic approximation by averaging. *Siam Journal on Control and Optimization*, 30:838–855.

Sameer Pradhan, Xiaoqiang Luo, Marta Recasens, Eduard Hovy, Vincent Ng, and Michael Strube. 2014. Scoring coreference partitions of predicted mentions: A reference implementation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 30–35, Baltimore, Maryland. Association for Computational Linguistics.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying LMs with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212, Online. Association for Computational Linguistics.

Lianhui Qin, Sean Welleck, Daniel Khashabi, and Yejin Choi. 2022. Cold decoding: Energy-based constrained text generation with langevin dynamics. *ArXiv preprint*, abs/2202.11705.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *Technical Report*.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Yasaman Razeghi, Robert L. Logan, Matt Gardner, and Sameer Singh. 2022. Impact of pretraining term frequencies on few-shot reasoning.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online. Association for Computational Linguistics.

Sascha Rothe, Joshua Maynez, and Shashi Narayan. 2021. A thorough evaluation of task-specific pretraining for summarization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 140–145, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ruslan Salakhutdinov. 2014. Deep learning. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, page 1973. ACM.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Tali Bers, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2021. Multitask prompted training enables zero-shot task generalization.

Maarten Sap, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. 2019. ATOMIC: an atlas of machine commonsense for if-then reasoning. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference,*

IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pages 3027–3035. AAAI Press.

Timo Schick and Hinrich Schütze. 2021a. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021b. It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352, Online. Association for Computational Linguistics.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Claude E. Shannon. 1948. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27:379–423.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR.

Weijia Shi, Mandar Joshi, and Luke Zettlemoyer. 2021. DESCGEN: A distantly supervised dataset-for generating entity descriptions. 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 415–427, Online. Association for Computational Linguistics.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. 2013a. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 926–934.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Daniil Sorokin and Iryna Gurevych. 2017. Context-aware representations for knowledge base relation extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1784–1789, Copenhagen, Denmark. Association for Computational Linguistics.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4444–4451. AAAI Press.

Valentin I. Spitkovsky and Angel X. Chang. 2012. A cross-lingual dictionary for English Wikipedia concepts. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3168–3175, Istanbul, Turkey. European Language Resources Association (ELRA).

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Gilbert Strang. 1993. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA.

Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J. Pal. 2018. Learning general purpose distributed sentence representations via large scale multi-task learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Simeng Sun, Wenlong Zhao, Varun Manjunatha, Rajiv Jain, Vlad Morariu, Franck Dernoncourt, Balaji Vasan Srinivasan, and Mohit Iyyer. 2021. IGA: An intent-guided authoring assistant. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5972–5985, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014a. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014b. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112.

Wilson L. Taylor. 1953. “cloze procedure”: A new tool for measuring readability. *Journalism & Mass Communication Quarterly*, 30:415 – 433.

Joerg Ueberla. 1994. Analysing a simple language model·some general conclusions for language models for speech recognition. *Computer Speech & Language*, 8(2):153 – 176.

Aad W Van der Vaart. 2000. *Asymptotic statistics*, volume 3. Cambridge university press.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017a. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017b. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Pat Verga, Haitian Sun, Livio Baldini Soares, and William Cohen. 2021. Adaptable and interpretable neural MemoryOver symbolic knowledge. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3678–3691, Online. Association for Computational Linguistics.

Pascal Vincent, H. Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledge base. In *Communications of the ACM*.

Denny Vrandečić and Markus Krötzsch. 2014a. Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

Denny Vrandečić and Markus Krötzsch. 2014b. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for attacking and analyzing NLP. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162, Hong Kong, China. Association for Computational Linguistics.

Li Wan, Matthew D. Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1058–1066. JMLR.org.

Alex Wang and Kyunghyun Cho. 2019. BERT has a mouth, and it must speak: BERT as a Markov random field language model. In *Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation*, pages 30–36, Minneapolis, Minnesota. Association for Computational Linguistics.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3261–3275.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Jianshu Ji, Guihong Cao, Daxin Jiang, and Ming Zhou. 2021. K-Adapter: Infusing Knowledge into Pre-Trained Models with Adapters. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1405–1418, Online. Association for Computational Linguistics.

Yu Wang, Yuelin Wang, Jie Liu, and Zhuo Liu. 2020. A comprehensive survey of grammar error correction. *ArXiv*, abs/2005.06600.

Wikipedia contributors. 2004. Wikipedia, the free encyclopedia.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. HuggingFace's Transformers: State-of-the-art natural language processing. *ArXiv preprint*, abs/1910.03771.

Yonghui Wu, Mike Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason R. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. 2016a. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144.

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

Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. 2017. Reference-aware language models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1850–1859, Copenhagen, Denmark. Association for Computational Linguistics.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *ArXiv*, abs/1409.2329.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 12697–12706. PMLR.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual probing is [MASK]: Learning vs. learning to recall. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5017–5033, Online. Association for Computational Linguistics.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 19–27. IEEE Computer Society.

Asli Çelikyilmaz, Elizabeth Clark, and Jianfeng Gao. 2020. Evaluation of text generation: A survey. *ArXiv*, abs/2006.14799.

Appendix A

Few-Shot Prompts

Dataset	Prompt Template	Label Tokens
BoolQ	{passage}. Question: {question}? Answer: [MASK].	True: "Yes" False: "No"
CB	{premise}? [SEP] [MASK], {hypothesis}	entailment: "Yes" contradiction: "No" neutral: "Maybe"
MNLI	{sentence1}? [SEP] [MASK], {sentence2}	entailment: "Yes" contradiction: "No" neutral: "Maybe"
MNLI-mm	{sentence1}? [SEP] [MASK], {sentence2}	entailment: "Yes" contradiction: "No" neutral: "Maybe"
MRPC	{sentence1} and {sentence2} have [MASK] meanings.	0: "different" 1: "similar"
QNLI	{question}? [SEP] [MASK], {sentence}	entailment: "Yes" not_entailment: "No"
QQP	{question1} and {question2} have [MASK] meanings.	0: "different" 1: "similar"
RTE	{sentence1}? [SEP] [MASK], {sentence2}	entailment: "Yes" not_entailment: "No"
SST-2	{sentence} It was [MASK] .	0: "terrible" 1: "great"

Table A.1: **Prompts denoted as "Manual Prompts (Prior)"**. We use prompts inspired from past work (Schick and Schütze, 2021a; Gao et al., 2021). The fields between curly brackets indicate dataset-specific inputs. Predictions are made on the [MASK] token in each prompt. For prompt tuning, we tune the tokens in the prompt template.

Dataset	Prompt Template	Label Tokens
BoolQ	Passage: {passage} Question: {question} Answer: [MASK].	True: "true" False: "false"
CB	Premise: {premise} Hypothesis: {hypothesis} Label: [MASK]	entailment: "yes" contradiction: "no" neutral: "maybe"
MNLI	Premise: {sentence1} Hypothesis: {sentence2} Label: [MASK]	entailment: "yes" contradiction: "no" neutral: "maybe"
MNLI-mm	Premise: {sentence1} Hypothesis: {sentence2} Label: [MASK]	entailment: "yes" contradiction: "no" neutral: "maybe"
MRPC	{sentence1} and {sentence2} are the [MASK].	0: "different" 1: "same"
QNLI	Question: {question} Sentence: {sentence} Label: [MASK]	entailment: "yes" not_entailment: "no"
QQP	{question1} and {question2} are the [MASK].	0: "different" 1: "same"
RTE	Premise: {sentence1} Hypothesis: {sentence2} Label: [MASK]	entailment: "yes" not_entailment: "no"
SST-2	{sentence} Overall my impression is [MASK] .	0: "bad" 1: "good"

Table A.2: **Prompts denoted as "Manual Prompts (w/o Engineering)"**. We manually write one prompt for each task, using only our intuition, and do not tune or edit them in any way after evaluating them. Fields between curly brackets indicate dataset-specific inputs. Predictions are made on the [MASK] token in each prompt. For prompt tuning, we tune the tokens in the prompt template.

Dataset	Prompt Template	Label Tokens
BoolQ	{passage} {question} [MASK]	True: "Yes" False: "No"
CB	{premise} [MASK] {hypothesis}	entailment: "Yes" contradiction: "No" neutral: "Maybe"
MNLI	{sentence1} [MASK] {sentence2}	entailment: "Yes" contradiction: "No" neutral: "Maybe"
MNLI-mm	{sentence1} [MASK] {sentence2}	entailment: "Yes" contradiction: "No" neutral: "Maybe"
MRPC	{sentence1} {sentence2} [MASK]	0: "different" 1: "similar"
QNLI	{question} [MASK] {sentence}	entailment: "Yes" not_entailment: "No"
QQP	{question1} {question2} [MASK]	0: "different" 1: "similar"
RTE	{sentence1} [MASK] {sentence2}	entailment: "Yes" not_entailment: "No"
SST-2	{sentence} [MASK]	0: "terrible" 1: "great"

Table A.3: **Null Prompts** used for results in Sections 5.3 and 5.4.

Appendix B

Additional FRUIT Details

B.1 Input and Output Formats

(2) [0] Elizabeth Lynne Cheney (; born July 28, 1966) is an American attorney and politician serving as the U.S. Representative for since 2017. [1] Cheney is the House Republican Conference Chair, the third-highest position in GOP House leadership. [2] She is the third woman elected to that position after Deborah Pryce and Cathy McMorris Rodgers. [3] Cheney is the elder daughter of former Vice President Dick Cheney and Lynne Cheney. [4] She held several positions in the U.S. State Department during the George W. Bush administration. [5] She has been politically active on behalf of the Republican Party and is a co-founder of Keep America Safe, a nonprofit organization concerned with national security issues. [6] She was a candidate for the 2014 election to the United States Senate in Wyoming, challenging the three-term incumbent Mike Enzi, before withdrawing from the race. [7] In the House of Representatives, she holds the seat that was held by her father from 1979 to 1989. [8] She is known for her hawkish foreign policy views. [CONTEXT] (0) Andy Biggs U.S. House of Representatives - Tenure - 2021 storming of the United States Capitol On January 12, 2021, Biggs called on fellow GOP Representative Liz Cheney (R-WY) to resign from her leadership position within the Republican Caucus, after she voted in favor of Donald Trump's second impeachment. (1) 116th United States Congress Leadership - House of Representatives - Minority (Republican) leadership * House Minority Leader and Chair of the House Republican Steering Committee: Kevin McCarthy * House Minority Whip: Steve Scalise * Chair of the House Republican Conference: Liz Cheney * Vice Chair of the House Republican Conference: Mark Walker * Secretary of the House Republican Conference: Jason Smith * Chair of the House Republican Policy Committee: Gary Palmer * Chair of the National Republican Congressional Committee: Tom Emmer * House Republican Chief Deputy Whip: Drew Ferguson (2) A Call for American Renewal INTRODUCTION The manifesto was released one day after the ousting of Representative Liz Cheney as chair of the House Republican Conference, and was largely seen as a reaction against the influence of Trumpism within the Republican Party. (3) List of nicknames used by Donald Trump Domestic political figures - Table-0-11 [HEADER] [COL] Nickname [COL] Personal name [COL] Notes [ROW] id="The Warmonger" [COL] The Warmonger [COL] Liz Cheney [COL] U.S. representative for Wyoming's at-large congressional district; Chair of the House Republican Conference (4) Conscience vote Practice in various countries - United States Similarly, when House Republican leadership decided not to whip votes against the second impeachment of Donald Trump, Liz Cheney—the third-highest-ranking Republican—referred to the matter as a "vote of conscience".

Figure B.1: **Input Format.**

(2) Elizabeth Lynne Cheney (; born July 28, 1966) is an American attorney and politician who has served as the U.S. Representative for since 2017. She was the Chair of the House Republican Conference, the third-highest position in the House Republican leadership. She is the third woman elected to that position after Deborah Pryce and Cathy McMorris Rodgers. She held several positions in the U.S. State Department during the George W. Bush administration, notably as Deputy Assistant Secretary of State for Near Eastern Affairs and Coordinator for Broader Middle East and North Africa Initiatives. She promoted regime change in Iran while chairing the Iran Syria Policy and Operations Group with Elliott Abrams. In 2009 Cheney and Bill Kristol founded Keep America Safe, a nonprofit organization concerned with national security issues that advocated the positions of the former Bush administration. She was a candidate for the 2014 election to the U.S. Senate in Wyoming, challenging three-term incumbent Mike Enzi, before withdrawing from the race. In the House of Representatives, she holds the seat her father held for a decade, representing Wyoming from 1979 to 1989. Cheney is a neoconservative. She later supported the second impeachment of Donald Trump for his role in the 2021 storming of the U.S. Capitol.

Figure B.2: **T5 Output Format.**

(2) [0] [1] [2] [3] [4] [5] [6] In the House of Representatives, she holds the seat that was held by her father from 1979 to 1989. (6) She is known for her neoconservative foreign policy views, and her affiliation with the Trump campaign. (0) (1) (2) (3) (4) Cheney is under fire for her role in the second impeachment of Donald Trump in January 2021.

Figure B.3: **EDIT5 Output Format.**

B.2 More Qualitative Examples

Original Article

Mike McMeeken (born 10 May 1994) is an English rugby league footballer who plays as a forward for the Castleford Tigers in the Super League. McMeeken has also represented England at international level, playing in two games at the 2017 World Cup. He started his career in the Super League with the London Broncos, also playing on loan in League 1 at the London Skolars before joining the Tigers.

New Information

Castleford Tigers 2021
Transfers - Losses

Player	Club	Contract	Date
Mike McMeeken	Catalans Dragons	2 Year	December 2020

Catalans Dragons 2021
Transfers - Gains

Player	Club	Contract	Date
Mike McMeeken	Castleford Tigers	3 Year	June 2020

Ground Truth

Mike McMeeken (born 10 May 1994) is an English rugby league footballer who plays as a forward for the Catalans Dragons in the Super League…Copied text...He joined Catalans Dragons in December 2020, ahead of the 2021 season.

EDIT5

Mike McMeeken (born 10 May 1994) is an English rugby league footballer who plays as a forward for the Catalans Dragons in the Super League…Copied text…

Figure B.4: **Example 1.**

Original Article

Isidore Mankofsky (born September 22, 1931, in New York City, New York) is an American cinematographer. He shot more than 200 educational movies for Encyclopaedia Britannica.

New Information

2021 Deaths in the United States
Isidore Mankofsky, cinematographer ("The Muppet Movie", "Somewhere in Time", "The Jazz Singer")

Deaths in March 2021
11 - Isidore Mankofsky, 89, American cinematographer ("The Muppet Movie", "Somewhere in Time", "The Jazz Singer")

The Parent Trap (franchise)
Additional crew and production details

Film	Crew / Detail
Parent Trap III	Joel McNeely, Isidore Mankofsky, Howard Kunin & Duane Hartzell

Ground Truth

Isidore Mankofsky (September 22, 1931 – March 11, 2021) was an American cinematographer, best known for his work on films such as "The Muppet Movie" (1979) and "The Jazz Singer" (1980)...Copied text…He died at his home in Los Angeles, California in March 2021 at the age of 89.

EDIT5

Isidore Mankofsky (September 22, 1931 – March 11, 2021) was an American cinematographer…Copied text…, and worked on "The Muppet Movie", "Somewhere in Time", and "The Jazz Singer".

Figure B.5: **Example 2.**

Instructions

Overview
The goal of this task is to collect evaluation data for a system that can automatically update Wikipedia articles from new information about the article's subject. The sections below provide the text of the *original passage* to be updated, a collection of *added information* about the article subject, and the text of the *updated passage*.

What we need from you
The issue we are faced with is that some of the udpated text may not be supported by the added information section. We need you to identify all of the unsupported information, and edit the article to remove unsupported text while preserving fluency. Please make sure that your edits only remove information; while you may need to write some text to ensure that the edited passage is fluent, no new facts should be added (even if they are supported).

The original passage and added information are below this box. We request that you first read the updated passage in the green box, and highlight any unsupported text. Then copy the contents from the green box to the orange box and edit them so that all of the text is supported.

If you have questions please do not hesitate to email: REDACTED

Original Passage - DO NOT CHANGE

```
[0] Joshua Christian Kojo King (born 15 January 1992) is a Norwegian professional footballer who plays
as a forward for Championship club Bournemouth and the Norway national team.
[1] King was signed by Manchester United from Vålerenga in 2008.
[2] After loan spells with Preston North End, Borussia Mönchengladbach, Hull City and Blackburn Rovers,
he signed permanently with Blackburn in January 2013, before switching to Bournemouth in May 2015.
[3] After representing Norway at under-15, under-16, under-18, under-19 and under-21 levels, King made
his senior international debut against Iceland in 2012, and scored his first international goal against
Cyprus later that year.
```

Added Information - DO NOT CHANGE

```
(0) 2020-21_AFC_Bournemouth_season Transfers - Transfers out - Table-0-29
| Date           | Position | Nationality | Name         | To      | Fee          | Ref. |
|----------------|----------|-------------|--------------|---------|--------------|------|
| 2 February 2021 | SS       |             | Joshua King | Everton | Nominal fee |      |

(1) 2020-21_Everton_F.C._season Transfers - Transfers in - Table-0-6
| Date            | Position | Nationality | Name         | From         | Fee     | Team       | Ref. |
|-----------------|----------|-------------|--------------|--------------|---------|------------|------|
| 1 February 2021 | FW       |             | Joshua King | Bournemouth | Nominal | First team |      |

(2) Gullballen Winners - 2014-2017 - Table-0-3
| Year | Winner       | Club(s)      |
|------|--------------|--------------|
| 2017 | Joshua King | Bournemouth |

(3) 2020-21_Crawley_Town_F.C._season Review - January Nichols equalised from close range in the 59th
minute before Josh King scored Bournemouth's winner.

(4) 2020-21_Manchester_United_F.C._season Premier League McTominay restored the lead only for Dominic
Calvert-Lewin to equalise again in the final minute of stoppage time following Tuanzebe's foul on
Everton substitute and fellow United Academy graduate Joshua King.
```

Updated Passage - HIGHLIGHT UNSUPPORTED TEXT

```
(0)
(1) Joshua Christian Kojo King (born 15 January 1992) is a Norwegian professional footballer who plays
as a forward for Premier League club Everton and the Norway national team.
[1] King was signed by Manchester United from Vålerenga in 2008.
[2] After loan spells with Preston North End, Borussia Mönchengladbach, Hull City and Blackburn Rovers,
he signed permanently with Blackburn in January 2013, before switching to Bournemouth in May 2015.
(0)
(1) In February 2021, in a deadline day deal, he returned to the top flight with a move to Everton.
[3] After representing Norway at under-15, under-16, under-18, under-19 and under-21 levels, King made
his senior international debut against Iceland in 2012, and scored his first international goal against
Cyprus later that year.
```

Edited Passage - COPY FROM THE CELL ON THE LEFT AND EDIT

```
(0)
(1) Joshua Christian Kojo King (born 15 January 1992) is a Norwegian professional footballer who plays
as a forward for Premier League club Everton and the Norway national team.
[1] King was signed by Manchester United from Vålerenga in 2008.
[2] After loan spells with Preston North End, Borussia Mönchengladbach, Hull City and Blackburn Rovers,
he signed permanently with Blackburn in January 2013, before switching to Bournemouth in May 2015.
(0)
(1) In February 2021,he returned to Everton.
[3] After representing Norway at under-15, under-16, under-18, under-19 and under-21 levels, King made
his senior international debut against Iceland in 2012, and scored his first international goal against
Cyprus later that year.
```

Step 1

The section below above the text of the updated passage. Unchanged sentences from the original passage are in grey, while added or updated sentences are in black.

We have tried to automatically detect which pieces of added information justify the changed text. If a justification is detected, then the edited sentence will be prefaced with the delimiter of the added information.
For example:
(0) (1) Updated sentence
means that we think that added information 0 and 1 justify (at least some of the edit).

What to do for this column
- Highlight any text/evidence delimiters that are unsupported by the original passage or added information, using this red color (in the custom section).
- Do not edit the text.

Step 2

What to do for this column
- Copy the highlighted updated passage from the previous step.
- Edit text so that: a) any unsupported text is removed, and b) the passage is still fluent.
- Do not add any new information

Figure B.6: Annotator Interface