Exploring the K-Adapter Framework

Zhaokun Wang
Institute of Computational Linguistics
Heidelberg University, Heidelberg, Germany
zhaokun.wang@stud.uni-heidelberg.de

Abstract

This report presents an exploration of the K-Adapter framework, combining a brief review with an external evaluation of its parameter efficiency, transferability, and deployment trade-offs. We reproduce and extend prior results on OpenEntity and FIGER, showing that factual adapters outperform baselines but offer limited zero-shot transfer. Through ablation experiments, we find that adapter performance is largely insensitive to size and internal complexity but highly sensitive to insertion position, with middle-layer placement proving most effective. We also show that adapter-tuning matches full fine-tuning accuracy with far fewer trainable parameters, albeit with modest latency and memory overhead. Finally, pretraining on large-scale, general-purpose knowledge sources yields far stronger transfer than small, task-specific datasets. These findings provide practical guidelines for building parameter-efficient, knowledge-enhanced language models.[1]

1 Introduction

Pre-trained Language Models (PLMs) such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) have transformed natural language processing, yet they often struggle to reliably store and retrieve structured knowledge. For instance, a base PLM may fail to consistently infer a person's profession from their workplace when such information is not explicitly present in its training corpus. Early approaches like ERNIE (Zhang et al., 2019) and KnowBERT (Peters et al., 2019) addressed this by fine-tuning the entire model on knowledge-infusion tasks. However, this monolithic strategy is parameter-inefficient, risks catastrophic forgetting, and produces entangled representations that are difficult to update or interpret.

Parameter-Efficient Fine-Tuning (PEFT) methods offer a compelling alternative. By freezing the backbone PLM and only training a small number of additional parameters, methods like Adapters (Houlsby et al., 2019), LoRA (Hu et al., 2022), and Prefix-Tuning (Li and Liang, 2021) achieve strong performance with a fraction of the training cost. The K-Adapter framework (Wang et al., 2020) specifically applies this modular philosophy to knowledge infusion, decoupling knowledge from the backbone model. It introduces external "knowledge adapters" that can be independently trained and plugged into a frozen PLM.

While the original K-Adapter work established its effectiveness, many practical questions about its design and behavior remain unanswered. Key design choices—such as the adapter's size, its internal complexity, and its placement within the PLM—can critically impact both performance and efficiency, yet these have not been systematically studied. Furthermore, the framework's zero-shot transferability and the true cost of its deployment in terms of latency and memory are not fully understood.

To address these gaps, this project presents an empirical study of the framework. We examine the influence of architectural hyperparameters, evaluate its zero-shot transferability and inference trade-offs, and analyze how different pre-training data sources affect downstream performance. Our results aim to provide practical insights that may help guide the effective and efficient use of K-Adapters.

2 Background: The K-Adapter Framework

2.1 Architecture and Method

K-Adapter maintains a frozen RoBERTa-Large backbone while introducing external adapter modules alongside it. Each adapter specializes in a particular type of knowledge and is trained inde-

[1] Codes are publicly available at https://github.com/BufferHund/K-Adapter_SemesterProject

pendently on a corresponding pre-training task. For instance, the **factual adapter** employs a relation-classification objective built from a filtered T-REx alignment of Wikipedia text and Wikidata triples. The **linguistic adapter** is trained on dependency-relation prediction using a parsed subset of BookCorpus.

Architecturally, each adapter comprises projection layers bracketing a small stack of Transformer layers with skip connections and concatenation to the backbone's hidden states. During downstream fine-tuning, the outputs of the backbone and selected adapters are concatenated and passed into task-specific heads for classification or span prediction. This design naturally supports continual knowledge infusion: adding new knowledge merely involves introducing another adapter without retraining the backbone or existing adapters.

3 Experimental Setup

3.1 Tasks and Datasets

We evaluate K-Adapter on two standard entity typing datasets:

- **OpenEntity** (Choi et al., 2018), which features a mix of coarse- and fine-grained entity types. Performance is measured using F1-A (all mentions) and F1-B (mentions with at least one fine-grained type).
- **FIGER** (Ling and Weld, 2012), which contains 113 fine-grained entity types. Performance is measured using Micro-F1 and Macro-F1 scores.

We also use the **TACRED** (Zhang et al., 2017) relation classification dataset as a pre-training source in one experiment to study the impact of pre-training data.

3.2 Model and Baselines

Our primary model is RoBERTa-Large (Liu et al., 2019) with K-Adapters. The baseline configuration is the frozen RoBERTa-Large model with a trainable classification head, without any adapters. For efficiency comparison, we also evaluate a fully fine-tuned RoBERTa-Large model.

3.3 Implementation Details

All experiments followed a consistent training setup with standard hyperparameter choices. Experiments were run on a single NVIDIA P100 GPU provided by Kaggle. The factual adapter is pre-trained on a 1% subset of the original T-REx dataset due to computational constraints.

4 Experiments and Results

We structure our empirical evaluation into three parts: first, we assess the baseline fine-tuning performance and zero-shot transferability; second, we perform a series of ablation studies on the adapter's architecture; finally, we analyze its efficiency and the impact of pre-training data.

4.1 Fine-tuning vs. Zero-shot Transfer

4.1.1 Fine-tuning Performance on OpenEntity

We first fine-tune different adapter configurations on OpenEntity to establish baseline performance. As shown in Table 1 and Figure 1, the factual-only adapter achieves the highest scores (F1-A: 0.762, F1-B: 0.629), outperforming both the no-adapter baseline and the linguistic-only adapter. Combining both adapters via concatenation slightly underperforms the factual-only adapter, while additive combination performs worse. This confirms that factual knowledge is most beneficial for this task and that fine-tuning is necessary to adapt this knowledge.

Table 1: Test F1 scores on OpenEntity with different adapter configurations after fine-tuning.

Setup	F1-A	F1-B
Baseline	0.749	0.579
Linguistic-only	0.755	0.620
Factual-only	**0.762**	**0.629**
Fac+Lin (Add)	0.754	0.616
Fac+Lin (Concat)	0.758	0.629

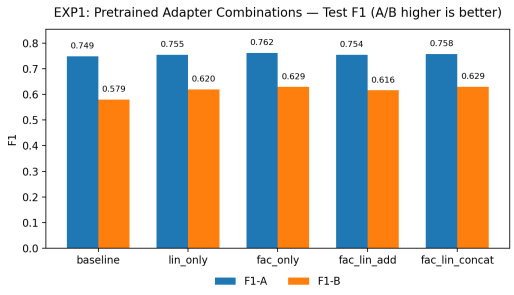
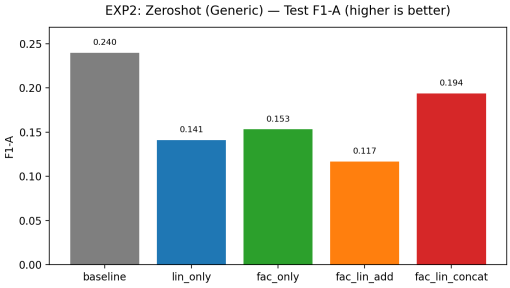


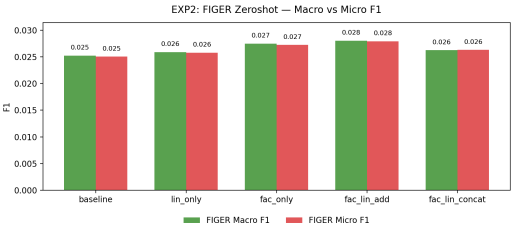
Figure 1: Test F1-A and F1-B scores on OpenEntity for different fine-tuned adapter combinations.

4.1.2 Zero-shot Transferability

Next, we evaluated the out-of-the-box performance of pre-trained adapters without any fine-tuning. As seen in Figure 2, the results are starkly different. On OpenEntity, all adapter configurations perform worse than the RoBERTa-large baseline, indicating negative transfer. On FIGER, while the absolute scores are near random guessing, the additive combination shows a marginal gain over the baseline. Overall, these experiments demonstrate that **pre-trained adapters do not directly improve zero-shot performance** on these tasks. The knowledge they contain must be unlocked and adapted via fine-tuning.



(a) Zero-shot F1-A on OpenEntity.



(b) Macro and Micro F1 on FIGER (Zero-shot).

Figure 2: Zero-shot evaluation results.

4.2 Ablation Studies on Adapter Architecture

We conducted a series of ablations to understand the sensitivity of K-Adapter to its core architectural hyperparameters.

4.2.1 Impact of Adapter Size

We varied the bottleneck dimension (adapter_size) from 16 to 768. As shown in Table 2 and Figure 3, performance is remarkably stable, with F1-A scores hovering between 0.686 and 0.690. This demonstrates the high parameter efficiency of the adapter approach and suggests that for OpenEntity, a very small adapter is sufficient.

4.2.2 Impact of Insertion Position

We investigated how adapter insertion position influences performance by placing adapters in early (layers 0-2), middle (10-12), late (21-23), and dispersed (0, 11, 22) layers of RoBERTa. The results in Table 3 show a decisive impact. **Placing adapters in the middle layers yields the best performance** (Micro F1: 0.706), significantly outperforming all other configurations. In contrast, late-layer insertion leads to a catastrophic performance drop, suggesting that middle layers provide the ideal mix of low-level features and semantic abstractions for knowledge injection.

Table 2: Test F1-A on OpenEntity across different adapter sizes.

Adapter Size	Test F1-A
16	0.690
64	0.686
256	0.690
768	0.688

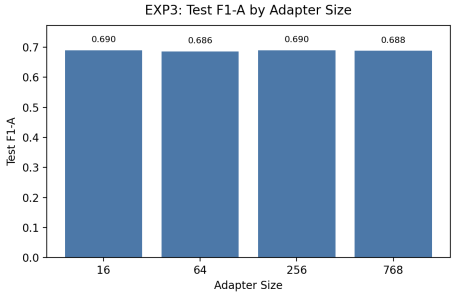


Figure 3: Test F1-A by adapter size.

Table 3: Performance on OpenEntity across different adapter insertion strategies.

Strategy	Adapter List	Micro F1	Macro F1
Early	0,1,2	0.611	0.456
Middle	10,11,12	**0.706**	**0.585**
Dispersed (Baseline)	0,11,22	0.686	0.563
Late	21,22,23	0.389	0.371

4.2.3 Impact of Internal Complexity

Finally, we varied the number of internal Transformer layers within the adapter (1, 2, or 4). As shown in Table 4, this had almost no effect on performance. The simplest adapter with one layer performed on par with more complex configurations. This indicates that a simple projection is sufficient and increasing internal depth brings no clear benefit for this task.

Table 4: Performance on OpenEntity with varying adapter internal layers.

Adapter Layers	Micro F1	Macro F1
1 (Simpler)	**0.689**	0.567
2 (Baseline)	0.686	0.563
4 (Deeper)	0.679	**0.569**

4.3 Efficiency and Pre-training Data Analysis

4.3.1 Parameter and Inference Efficiency

We compared adapter-tuning with full fine-tuning. While adapter-tuning trains far fewer parameters (millions vs. hundreds of millions), it introduces inference overhead. Figure 4 shows that adding adapters increases latency and GPU memory usage. For example, at batch size 1, latency rises by 35% with one adapter and 60% with two. This highlights the trade-off: adapter-tuning is highly parameter-efficient for training but incurs a predictable cost at inference time.

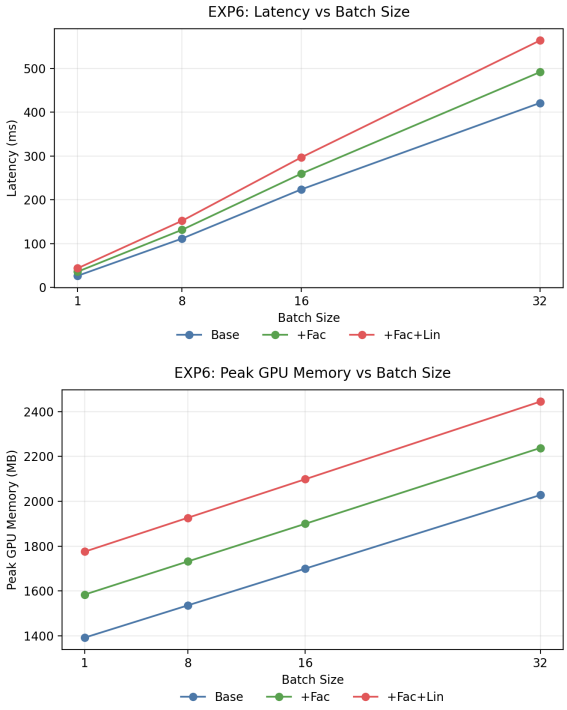


Figure 4: Inference latency and GPU memory vs. batch size.

4.3.2 Impact of Pre-training Data

We compared a factual adapter pre-trained on the large, general-purpose T-REx dataset with one pre-trained on the smaller, task-specific TACRED dataset. As shown in Table 5, the T-REx-trained adapter achieves a test F1-A of 0.762 (from Exp. 1), while the TACRED-trained adapter only reaches 0.397. Although part of this gap may be attributed to the limited scale and the additional preprocessing required for TACRED, the results highlight a broader issue: pre-training on small or mismatched datasets can significantly hinder knowledge transfer. This finding suggests that adapters built from insufficient or domain-incompatible corpora are unlikely to generalize well, underscoring the importance of large, high-coverage and well-aligned pre-training data.

Table 5: Performance of an adapter pre-trained on the task-specific TACRED dataset.

Split	Precision	Recall	F1-A
Dev	0.572	0.270	0.367
Test	**0.610**	**0.295**	**0.397**

5 Discussion and Conclusion

This study highlights a hierarchy of design considerations for effective and efficient knowledge infusion. The results show that an adapter's performance is primarily affected by two factors: insertion position and the quality of its pre-training data. In particular, placing adapters in the middle layers of the PLM tends to yield stronger performance, while pre-training on large-scale, general-purpose knowledge sources is important for successful transfer.

At the same time, performance appears relatively less sensitive to the adapter's internal architecture; compact and simple adapters achieve results comparable to larger, more complex ones, underscoring their parameter efficiency. This approach, however, comes with certain limitations. Adapters generally require downstream fine-tuning to reach their full potential, as their zero-shot capability is limited. They also introduce predictable overhead in inference latency and memory usage.

Overall, these findings point to practical guidelines for practitioners: prioritize middle-layer placement and high-quality, general pre-training data, while keeping adapters compact and simple to maintain efficiency. This modular approach offers a promising direction for building more knowledgeable and maintainable language models.

Future work may investigate automated methods for identifying optimal adapter configurations and explore more advanced fusion mechanisms.

References

Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-fine entity typing. In *ACL*, pages 87–96.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186.

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 *ICML*, pages 2790–2799.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Xiao Ling and Daniel S. Weld. 2012. Fine-grained entity recognition. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press.

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 arXiv:1907.11692*.

Matthew E Peters, Mark Neumann, IV Logan, L Robert, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. 2019. Knowledge enhanced contextual word representations. In *EMNLP*, pages 43–54.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Guihong Cao, Daxin Jiang, Ming Zhou, and 1 others. 2020. K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv preprint arXiv:2002.01808*.

Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. 2017. Position-aware Attention and Supervised Data Improve Slot Filling. In *EMNLP*, pages 35–45.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced Language Representation with Informative Entities. In *ACL*, pages 1441–1451.

Appendix

A Use of AI-Based Tools

This appendix documents the use of artificial intelligence (AI)-based tools in the preparation of this academic work.

List of Steps Involving AI-Based Tools

- **DeepSeek**: I consulted DeepSeek models to learn more formal organization of an conclusion chapter. The suggested frameworks were adapted and rewritten entirely in my own words. I also referred to DeepSeek during debugging to understand and resolve specific error messages.
- **QuillBot**: QuillBot was used sparingly to rephrase sentences for improved readability and flow. All suggestions were manually reviewed and edited to ensure alignment with my original intent and academic style.
- **DeepL and Youdao Translation**: DeepL and Youdao Translation assisted in translating a small number of technical terms and short phrases from Chinese to English to clarify meaning during drafting. These translations were verified and incorporated into my own text.