Total Time Consumption in Generating Answers: 66.12 seconds

Total Time Consumption in Evaluating Answers: 173.39 seconds

LLM Evaluation Summary

Evaluation 1:

Evaluation of the Answer:

Relevance: **Excellent**

The answer directly addresses the question by explaining what a GPT pre-trained model is, its architecture, training process, and key capabilities.

Faithfulness: **Excellent**

The answer is strictly supported by the provided context, accurately describing the GPT model, its transformer architecture, generative pretraining method, and applications as outlined in the given information.

Completeness: **Excellent**

The answer covers all major aspects of the question, including the model's architecture, training process, its general functionality, and examples of applications.

Clarity: **Excellent**

The answer is well-structured, easy to follow, and clearly explains the concepts without ambiguity.

Conciseness: **Excellent**

The answer is concise, avoiding unnecessary repetition or verbosity while providing all relevant details.

Final Ratings:

- **Relevance: Excellent**

- **Faithfulness: Excellent**

- **Completeness: Excellent**

- **Clarity: Excellent**

- **Conciseness: Excellent**

Weighted Score: 100.0

Question 1: What is a GPT pre-trained model?

Answer 1: A GPT pre-trained model is a type of large language model (LLM) based on the transformer deep learning architecture. It is trained using generative pretraining, which is a form of unsupervised learning. During this training process, the model is fed large amounts of unlabeled data and learns to detect patterns, structures, and relationships within the text. This enables the model to generate human-like responses, predict text, and perform various natural language processing tasks. The pre-training phase equips the model with a general understanding of language, which can later be fine-tuned for specific applications such as chatbots, text summarization, translation, or other AI-driven tasks.

Response Latency: 4.19 seconds

Evaluation Latency: 4.36 seconds

[Prompt used for Q1]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

[Web] A generative pre-trained transformer (GPT) is a type of large language model (LLM) and a prominent framework for generative artificial intelligence. It is an artificial neural network that is used in natural language processing. It is based on the transformer deep learning architecture, pre-trained ...

[Web] Generative pretraining is a form of unsupervised learning, where the model is fed unlabeled data and forced to make sense of it on its own. By learning to detect patterns in unlabeled datasets, machine learning models gain the ability to draw similar conclusions when exposed to new inputs, ...

[Web] How was GPT-3 trained? What are examples of some applications that use GPT? How can AWS help you run large language models like GPT-3? Generative Pre-trained Transformers, commonly known as GPT, are a family of neural network models that uses the transformer architecture and is a key advancement ...

Current Question:

What is a GPT pre-trained model?

Answer:

Evaluation 2:

Evaluation of the Answer

Relevance: Excellent

The answer directly addresses the question by outlining the advantages of pre-trained models compared to other options. Each point is focused on explaining specific benefits, ensuring relevance to the topic.

Faithfulness: Excellent

The answer is strictly supported by the provided context, which discusses the advantages of pre-trained models like GPT and BERT. It does not introduce unsupported claims and remains consistent with the given information.

Completeness: Excellent

The answer thoroughly covers all major aspects of the question, listing multiple distinct advantages such as efficiency, cost-effectiveness, generalization, ease of fine-tuning, state-of-the-art performance, and community support.

Clarity: Excellent

The answer is well-structured, with numbered points and clear explanations for each advantage. It is easy to follow and understand.

Conciseness: Excellent

The answer avoids unnecessary repetition or verbosity, succinctly presenting each advantage in a clear and straightforward manner.

Final Ratings:

- **Relevance:** Excellent

- **Faithfulness:** Excellent

- **Completeness:** Excellent

- **Clarity:** Excellent

*- **Conciseness:** Excellent*

Weighted Score: 100.0

Question 2: What advantages does it have compared with other pre-trained models?

Answer 2: Pre-trained models offer several advantages compared to building an AI system from scratch or using other approaches. Here are the key advantages of pre-trained models, particularly when comparing them to other options:

*1. **Efficiency and Reduced Training Time**:*

Pre-trained models already have calculated weights from extensive training on large datasets. This eliminates the need for training the model from scratch, saving significant time and computational resources.

*2. **Cost-Effectiveness**:*

Developing a model in-house requires substantial investment in data collection, labeling, infrastructure, and computational power. Pre-trained models minimize these costs as they are already trained on diverse datasets.

*3. **Generalization**:*

Pre-trained models, especially large-scale ones like GPT and BERT, are trained on vast amounts of data, enabling them to generalize better across various tasks. Their robustness allows them to adapt well to new tasks with minimal fine-tuning.

*4. **Accessibility of Weights**:*

Many pre-trained models share weights, particularly for initial layers, which are better at generalizing across different domains. This enables developers to leverage high-quality features without needing to retrain those layers.

*5. **Ease of Fine-Tuning**:*

Pre-trained models can be fine-tuned for specific tasks with relatively small datasets. This makes them highly versatile and suitable for applications requiring customization.

6. **State-of-the-Art Performance**:

Large-scale pre-trained models have achieved state-of-the-art results in various benchmarks and tasks, making them an excellent choice for applications requiring cutting-edge performance.

7. **Community and Ecosystem**:

Many pre-trained models come with extensive documentation, support, and community contributions. This makes them easier to integrate into products and allows developers to benefit from shared knowledge and tools.

In summary, pre-trained models provide significant advantages in terms of efficiency, cost, generalization, and ease of use, making them a preferred choice for incorporating language AI into products.

Response Latency: 7.03 seconds

Evaluation Latency: 6.07 seconds

[Prompt used for Q2]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

[Web] Adding language AI to your product? Let's explore whether it makes more sense to build it in-house or use a pre-trained LLM.

[Web] Large-scale pre-trained models (PTMs) such as BERT and GPT have recently achieved great success and become a milestone in the field of artificial inte?

[Web] Answer (1 of 2): Trained and pre-trained is usually the same thing. It?s just a model that was already trained and has calculated *weights with it. Sometimes people also share weights of particular layers, usually first ones, as they generalize better. Main advantage - you don?t need to ...*

Current Question:

What advantages does it have compared with other pre-trained models?

Answer:

Evaluation 3:

Evaluation of the Answer:

Relevance: Excellent

The answer directly addresses the question by listing and explaining the key advantages of LoRA as mentioned in the provided context.

Faithfulness: Fair

While the answer captures some key points from the context, it introduces additional details that are not strictly supported by the provided context (e.g., "Preservation of Pre-Trained Knowledge" and "Ease of Experimentation and Fine-Tuning") and omits explicit mentions of some advantages in the context, such as reduced task-switching overhead and lower hardware requirements.

Completeness: Fair

The answer provides a reasonably detailed list of advantages but misses some major aspects explicitly stated in the context, such as storage reduction and efficient task-switching via replacing matrices.

Clarity: Excellent

The answer is well-structured, logically organized, and easy to understand.

Conciseness: Fair

The answer includes unnecessary verbosity and introduces extraneous details that are not strictly relevant to the context, making it less concise.

Final Assessment:

The answer demonstrates good understanding but lacks strict adherence to the context and introduces unsupported information, leading to "Fair" ratings for faithfulness and completeness.

Weighted Score: 84.0

Question 3: what's the key advantages of LoRA?

Answer 3: LoRA (Low-Rank Adaptation) possesses several key advantages that make it an efficient and effective method for adapting large language models (LLMs) to specific tasks. Here are the main advantages of LoRA:

*1. **Parameter Efficiency**:*

LoRA introduces trainable low-rank matrices into the weight updates, significantly reducing the number of parameters required for fine-tuning. This allows for task-specific customization without the need to update or store the full set of model parameters.

*2. **Reduced Computational Overhead**:*

By focusing on low-rank updates, LoRA minimizes the computational cost associated with fine-tuning large models. This makes it a practical choice for resource-constrained environments.

*3. **Preservation of Pre-Trained Knowledge**:*

LoRA adjusts only specific components of the model (via rank decomposition) while preserving the original pre-trained weights. This ensures that the model retains its general knowledge base and can be effectively adapted to new tasks.

*4. **Orthogonality to Other Techniques**:*

LoRA can be combined with other adaptation methods, such as prefix tuning or embedding tuning, often providing complementary improvements. This makes it a versatile tool for enhancing model performance.

*5. **Flexibility in Scenarios with Latency Constraints**:*

LoRA offers the option to merge weights into the pre-trained model to eliminate inference latency. Alternatively, in scenarios where latency is not critical, weights can remain separate, allowing dynamic selection of modules.

*6. **Ease of Experimentation and Fine-Tuning**:*

LoRA simplifies the process of investigating how features learned during pre-training are transformed for downstream tasks. This tractability can lead to deeper insights into model behavior and adaptation mechanisms.

*7. **Cost-Effectiveness for Multi-Task Learning**:*

While batching inputs for tasks with different A and B matrices has limitations, LoRA enables modular adaptations that can be tailored to specific tasks without requiring extensive retraining.

In summary, LoRA is a parameter-efficient, computationally lightweight, and versatile method for fine-tuning large language models, making it a powerful tool for adapting pre-trained models to diverse tasks.

Response Latency: 10.46 seconds

Evaluation Latency: 7.85 seconds

[Prompt used for Q3]

You are a helpful assistant. Use the following history and context to answer.

History:

Q: What is a GPT pre-trained model?

A: A GPT pre-trained model is a type of large language model (LLM) based on the transformer deep learning architecture. It is trained using generative pretraining, which is a form of unsupervised learning. During this training process, the model is fed large amounts of unlabeled data and learns to detect patterns, structures, and relationships within the text. This enables the model to generate human-like responses, predict text, and perform various natural language processing tasks. The pre-training phase equips the model with a general understanding of language, which can later be fine-tuned for specific applications such as chatbots, text summarization, translation, or other AI-driven tasks.

Q: What advantages does it have compared with other pre-trained models?

A: Pre-trained models offer several advantages compared to building an AI system from scratch or using other approaches. Here are the key advantages of pre-trained models, particularly when comparing them to other options:

*1. **Efficiency and Reduced Training Time**:*

Pre-trained models already have calculated weights from extensive training on large datasets. This eliminates the need for training the model from scratch, saving significant time and computational resources.

2. **Cost-Effectiveness**:

Developing a model in-house requires substantial investment in data collection, labeling, infrastructure, and computational power. Pre-trained models minimize these costs as they are already trained on diverse datasets.

3. **Generalization**:

Pre-trained models, especially large-scale ones like GPT and BERT, are trained on vast amounts of data, enabling them to generalize better across various tasks. Their robustness allows them to adapt well to new tasks with minimal fine-tuning.

4. **Accessibility of Weights**:

Many pre-trained models share weights, particularly for initial layers, which are better at generalizing across different domains. This enables developers to leverage high-quality features without needing to retrain those layers.

5. **Ease of Fine-Tuning**:

Pre-trained models can be fine-tuned for specific tasks with relatively small datasets. This makes them highly versatile and suitable for applications requiring customization.

6. **State-of-the-Art Performance**:

Large-scale pre-trained models have achieved state-of-the-art results in various benchmarks and tasks, making them an excellent choice for applications requiring cutting-edge performance.

7. **Community and Ecosystem**:

Many pre-trained models come with extensive documentation, support, and community contributions. This makes them easier to integrate into products and allows developers to benefit from shared knowledge and tools.

In summary, pre-trained models provide significant advantages in terms of efficiency, cost, generalization, and ease of use, making them a preferred choice for incorporating language AI into products.

Context:

LoRA possesses several key advantages.

LoRA possesses several key advantages.

LoRA possesses several key advantages.

In Table 15, we show the evaluation results of $_\mathrm{L oRA+PE}$ and $_\mathrm{LoRAA+PL}$ on WikiSQL and MultiNLI. First of all, $_\mathrm{L oRA+PE}$ signi?cantly outperforms both LoRA and pre?x-embedding tuning on WikiSQL, which indicates that LoRA is somewhat orthogonal to pre?x-embedding tuning. On MultiNLI, the combination of $_\mathrm{L oRA+PE}$ doesn?t perform better than LoRA, possibly because LoRA on its own already achieves performance comparable to the human baseline. Secondly, we notice that $_\mathrm{L oRA+PL}$ performs slightly worse than LoRA even with more trainable parameters. We at- tribute this to the fact that pre?x-layer tuning is very sensitive to the choice of learning rate and thus makes the optimization of LoRA weights more dif?cult in $_\mathrm{L oRA+PL}$.

In Table 15, we show the evaluation results of $_\mathrm{L oRA+PE}$ and $_\mathrm{LoRAA+PL}$ on WikiSQL and MultiNLI. First of all, $_\mathrm{L oRA+PE}$ signi?cantly outperforms both LoRA and pre?x-embedding tuning on WikiSQL, which indicates that LoRA is somewhat orthogonal to pre?x-embedding tuning. On MultiNLI, the combination of $_\mathrm{L oRA+PE}$ doesn?t perform better than LoRA, possibly because LoRA on its own already achieves performance comparable to the human baseline. Secondly, we notice that $_\mathrm{L oRA+PL}$ performs slightly worse than LoRA even with more trainable parameters. We at- tribute this to the fact that pre?x-layer tuning is very sensitive to the choice of learning rate and thus makes the optimization of LoRA weights more dif?cult in $_\mathrm{L oRA+PL}$.

In Table 15, we show the evaluation results of $_\mathrm{L oRA+PE}$ and $_\mathrm{LoRAA+PL}$ on WikiSQL and MultiNLI. First of all, $_\mathrm{L oRA+PE}$ signi?cantly outperforms both LoRA and pre?x-embedding tuning on WikiSQL, which indicates that LoRA is somewhat orthogonal to pre?x-embedding tuning. On MultiNLI, the combination of $_\mathrm{L oRA+PE}$ doesn?t perform better than LoRA, possibly because LoRA on its own already achieves performance comparable to the human baseline. Secondly, we notice that $_\mathrm{L oRA+PL}$ performs slightly worse than LoRA even with more trainable parameters. We at- tribute this to the fact that pre?x-layer tuning is very sensitive to the choice of learning rate and thus makes the optimization of LoRA weights more dif?cult in $_\mathrm{L oRA+PL}$.

LoRA also has its limitations. For example, it is not straightforward to batch inputs to different tasks with different A and B in a single forward pass, if one chooses to absorb A and B into W to eliminate additional inference latency. Though it is possible to not merge the weights and dynamically choose the LoRA modules to use for samples in a batch for scenarios where latency is not critical.

5 E MPIRICAL E XPERIMENTS

LoRA also has its limitations. For example, it is not straightforward to batch inputs to different tasks with different A and B in a single forward pass, if one chooses to absorb A and B into W to eliminate additional inference latency. Though it is possible to not merge the weights and dynamically choose the LoRA modules to use for samples in a batch for scenarios where latency is not critical.

5 E MPIRICAL E XPERIMENTS

LoRA also has its limitations. For example, it is not straightforward to batch inputs to different tasks with different A and B in a single forward pass, if one chooses to absorb A and B into W to eliminate additional inference latency. Though it is possible to not merge the weights and dynamically choose the LoRA modules to use for samples in a batch for scenarios where latency is not critical.

 # 5 E MPIRICAL E XPERIMENTS

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

Current Question:

what's the key advantages of LoRA?

Answer:

Evaluation 4:

Evaluation of the Answer

Relevance: Excellent

The answer directly addresses the question, detailing both the practical benefits, advantages, and limitations of LoRA as requested. It remains on-topic throughout.

Faithfulness: Excellent

The answer is strictly supported by the context provided, accurately summarizing the benefits, advantages, and limitations mentioned in the passage without introducing unsupported claims.

Completeness: Excellent

The answer thoroughly covers all major aspects of the question. It includes practical benefits, advantages, limitations, and even proposes future directions, ensuring a comprehensive response.

Clarity: Excellent

The answer is well-structured, dividing the information into clear sections (Benefits, Advantages, Limitations, Future Directions). It uses simple and precise language, making the content accessible and understandable.

Conciseness: Good

While the answer is detailed and informative, it could be slightly more concise, particularly in the sections discussing advantages and future directions, to avoid redundancy.

Final Ratings

*- **Relevance:** Excellent*

*- **Faithfulness:** Excellent*

*- **Completeness:** Excellent*

- **Clarity:** Excellent

- **Conciseness:** Good

Weighted Score: 98.0

Question 4: Lora's practical benefits, advantages and limitations.

Answer 4: ### LoRA's Practical Benefits

1. **Efficient Task Switching**:

A pre-trained model can be shared across many tasks by creating small LoRA modules specific to each task. By freezing the shared model and replacing only the low-rank matrices \(A\) and \(B\), LoRA reduces storage requirements and task-switching overhead significantly.

2. **Reduced Hardware Barrier**:

LoRA lowers the computational and hardware requirements for training by up to 3 times when using adaptive optimizers. Since most parameters of the model remain frozen, gradients and optimizer states only need to be calculated for the smaller, injected low-rank matrices.

3. **No Inference Latency**:

LoRA's design allows for merging the trainable matrices with the frozen weights once deployed. This ensures no additional inference latency compared to a fully fine-tuned model.

4. **Minimal Storage Requirements**:

Due to its low-rank parameterization, LoRA requires significantly less storage for saving task-specific modules, making it ideal for situations with constrained memory or storage.

5. **Orthogonality to Other Methods**:

LoRA is orthogonal to many prior adaptation methods and can be combined with them, such as prefix-tuning or other fine-tuning techniques, offering additional flexibility and performance improvements.

6. **Simple Design**:

LoRA's approach involves injecting low-rank trainable matrices into the existing model architecture. This simplicity ensures ease of implementation and compatibility with dense layers in deep learning models.

Advantages of LoRA

1. **Cost-Effective Fine-Tuning**:

LoRA enables fine-tuning with fewer resources by optimizing only a small subset of parameters, making it highly practical for organizations with limited computational budgets.

2. **Scalability**:

The modular design of LoRA allows scaling to multiple downstream tasks without retraining or modifying the frozen base model.

3. **Flexibility**:

LoRA's ability to combine with other methods and adapt to various architectures provides flexibility in implementing diverse AI applications.

4. **Improved Interpretability**:

By focusing on low-rank updates, LoRA makes it easier to study and understand how features learned during pre-training are transformed for downstream tasks compared to full fine-tuning.

Limitations of LoRA

1. **Heuristic-Based Matrix Selection**:

LoRA currently relies on heuristics to select which weight matrices to apply the low-rank updates to. This may result in suboptimal choices and could benefit from more principled approaches.

2. **Rank Deficiency Concerns**:

The rank-deficiency of the update matrices ΔW suggests that the original weight matrices W might also be rank-deficient. Understanding the implications and addressing this limitation may require further research.

3. **Limited Understanding of Mechanisms**:

The process by which pre-trained features adapt to downstream tasks through LoRA is not fully understood. This presents an opportunity for further exploration to enhance its theoretical foundation.

4. **Focus on Transformers**:

While LoRA has primarily been tested on Transformer models, its applicability to other architectures and domains is less explored, potentially limiting its broader adoption.

Future Directions for LoRA

1. **Combination with Other Methods**:

Exploring combinations of LoRA with other adaptation techniques may yield orthogonal improvements.

2. **Principled Matrix Selection**:

Developing systematic methods for selecting weight matrices to apply LoRA updates could enhance its effectiveness.

3. **Understanding Fine-Tuning Mechanisms**:

Investigating how pre-trained features evolve during fine-tuning with LoRA could provide insights into improving adaptation strategies.

*4. **Rank Deficiency Exploration**:*

Analyzing the rank-deficiency of weight matrices and its implications could inspire new approaches to model optimization.

In summary, LoRA's practical benefits center around efficiency, scalability, and simplicity, while its limitations highlight areas for further research and refinement.

Response Latency: 12.57 seconds

Evaluation Latency: 5.65 seconds

[Prompt used for Q4]

You are a helpful assistant. Use the following history and context to answer.

History:

Q: What is a GPT pre-trained model?

A: A GPT pre-trained model is a type of large language model (LLM) based on the transformer deep learning architecture. It is trained using generative pretraining, which is a form of unsupervised learning. During this training process, the model is fed large amounts of unlabeled data and learns to detect patterns, structures, and relationships within the text. This enables the model to generate human-like responses, predict text, and perform various natural language processing tasks. The pre-training phase equips the model with a general understanding of language, which can later be fine-tuned for specific applications such as chatbots, text summarization, translation, or other AI-driven tasks.

Q: What advantages does it have compared with other pre-trained models?

A: Pre-trained models offer several advantages compared to building an AI system from scratch or using other approaches. Here are the key advantages of pre-trained models, particularly when comparing them to other options:

*1. **Efficiency and Reduced Training Time**:*

Pre-trained models already have calculated weights from extensive training on large datasets. This eliminates the need for training the model from scratch, saving significant time and computational resources.

*2. **Cost-Effectiveness**:*

Developing a model in-house requires substantial investment in data collection, labeling, infrastructure, and computational power. Pre-trained models minimize these costs as they are already trained on diverse datasets.

3. **Generalization**:

Pre-trained models, especially large-scale ones like GPT and BERT, are trained on vast amounts of data, enabling them to generalize better across various tasks. Their robustness allows them to adapt well to new tasks with minimal fine-tuning.

4. **Accessibility of Weights**:

Many pre-trained models share weights, particularly for initial layers, which are better at generalizing across different domains. This enables developers to leverage high-quality features without needing to retrain those layers.

5. **Ease of Fine-Tuning**:

Pre-trained models can be fine-tuned for specific tasks with relatively small datasets. This makes them highly versatile and suitable for applications requiring customization.

6. **State-of-the-Art Performance**:

Large-scale pre-trained models have achieved state-of-the-art results in various benchmarks and tasks, making them an excellent choice for applications requiring cutting-edge performance.

7. **Community and Ecosystem**:

Many pre-trained models come with extensive documentation, support, and community contributions. This makes them easier to integrate into products and allows developers to benefit from shared knowledge and tools.

In summary, pre-trained models provide significant advantages in terms of efficiency, cost, generalization, and ease of use, making them a preferred choice for incorporating language AI into products.

Context:

LoRA possesses several key advantages.

LoRA possesses several key advantages.

LoRA possesses several key advantages.

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

? A pre-trained model can be shared and used to build many small LoRA modules for dif- ferent tasks. We can freeze the shared model and ef?ciently switch tasks by replacing the matrices A and B in Figure 1, reducing the storage requirement and task-switching over- head signi?cantly. ? LoRA makes training more ef?cient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most parameters. Instead, we only optimize the injected, much smaller low-rank matrices. ? Our simple linear design allows us to merge the trainable matrices with the frozen weights when deployed, introducing no inference latency compared to a fully ?ne-tuned model, by construction. ? LoRA is orthogonal to many prior methods and can be combined with many of them, such as pre?x-tuning. We provide an example in Appendix E.

? A pre-trained model can be shared and used to build many small LoRA modules for dif- ferent tasks. We can freeze the shared model and ef?ciently switch tasks by replacing the matrices A and B in Figure 1, reducing the storage requirement and task-switching over- head signi?cantly. ? LoRA makes training more ef?cient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most parameters. Instead, we only optimize the injected, much smaller low-rank matrices. ? Our simple linear design allows us to merge the trainable matrices with the frozen weights when deployed, introducing no inference latency compared to a fully ?ne-tuned model, by construction. ? LoRA is orthogonal to many prior methods and can be combined with many of them, such as pre?x-tuning. We provide an example in Appendix E.

? A pre-trained model can be shared and used to build many small LoRA modules for dif- ferent tasks. We can freeze the shared model and ef?ciently switch tasks by replacing the matrices A and B in Figure 1, reducing the storage requirement and task-switching over- head signi?cantly. ? LoRA makes training more ef?cient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most

parameters. Instead, we only optimize the injected, much smaller low-rank matrices. ? Our simple linear design allows us to merge the trainable matrices with the frozen weights when deployed, introducing no inference latency compared to a fully ?ne-tuned model, by construction. ? LoRA is orthogonal to many prior methods and can be combined with many of them, such as pre?x-tuning. We provide an example in Appendix E.

4 O UR M ETHOD

We describe the simple design of LoRA and its practical bene?ts. The principles outlined here apply to any dense layers in deep learning models, though we only focus on certain weights in Transformer language models in our experiments as the motivating use case.

4.1 L OW -R ANK -P ARAMETRIZED U PDATE M ATRICES

Current Question:

Lora's practical benefits, advantages and limitations.

Answer:

Evaluation 5:

Evaluation of the Provided Answer:

Relevance: **Good**

The response addresses the question about which weight matrices in the Transformer should be adapted with LoRA. However, it introduces additional information (e.g., mentioning MLP modules, which are not explicitly relevant to the context) and does not fully align with the specific parameter budget discussed in the context.

Faithfulness: **Fair**

The answer mentions several weight matrices (W_q, W_k, W_v, W_o), but the context specifically emphasizes adapting matrices in the self-attention module and highlights the focus on W_q and W_v. The inclusion of W_k and W_o without explicit justification from the context makes the response partially faithful.

Completeness: **Fair**

While the answer lists weight matrices in the self-attention module and identifies W_q and W_v as key candidates, it does not explicitly reference the parameter budget or the empirical findings presented in the context (e.g., the rank values and the results in Table 5). This leaves the explanation incomplete.

Clarity: **Excellent**

The response is structured clearly with bullet points, technical terms are well-defined, and the explanation is easy to follow.

Conciseness: **Good**

The answer is relatively concise but includes minor extraneous details (e.g., mentioning MLP modules when the focus is solely on self-attention weight matrices). Removing these would improve conciseness.

Final Ratings:

- **Relevance:** Good

- **Faithfulness:** Fair

- **Completeness:** Fair

- **Clarity:** Excellent

- **Conciseness:** Good

Weighted Score: 76.0

Question 5: which WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Answer 5: In the Transformer architecture, LoRA can be applied to the weight matrices in the self-attention module and potentially other modules. Specifically, the weight matrices in the self-attention module are:

1. **Query weight matrix** (W_q)
2. **Key weight matrix** (W_k)
3. **Value weight matrix** (W_v)
4. **Output weight matrix** (W_o)

Based on the context provided, LoRA is most commonly applied to the attention weight matrices, particularly W_q and W_v. This choice is made because adapting these matrices often strikes a good balance between parameter efficiency and performance. For simplicity and efficiency, the MLP modules are often frozen during downstream tasks, meaning their weights are not adapted using LoRA.

Furthermore, empirical findings suggest that even with a very small rank (r), LoRA performs competitively, especially when applied to the combined set $\{W_q, W_v\}$. This indicates that a low-rank adaptation matrix is sufficient to capture meaningful updates for downstream tasks, without requiring the full rank of the original weight matrices.

In summary, the focus of LoRA application should be on the attention weight matrices (W_q, W_k, W_v, W_o), with *emphasis on W_q and W_v for optimal parameter efficiency and downstream task performance.*

Response Latency: 6.74 seconds

Evaluation Latency: 10.98 seconds

[Prompt used for Q5]

You are a helpful assistant. Use the following history and context to answer.

History:

Q: What is a GPT pre-trained model?

A: A GPT pre-trained model is a type of large language model (LLM) based on the transformer deep learning architecture. It is trained using generative pretraining, which is a form of unsupervised learning. During this training process, the model is fed large amounts of unlabeled data and learns to detect patterns, structures, and relationships within the text. This enables the model to generate human-like responses, predict text, and perform various natural language processing tasks. The pre-training phase equips the model with a general understanding of language, which can later be fine-tuned for specific applications such as chatbots, text summarization, translation, or other AI-driven tasks.

Q: What advantages does it have compared with other pre-trained models?

A: Pre-trained models offer several advantages compared to building an AI system from scratch or using other approaches. Here are the key advantages of pre-trained models, particularly when comparing them to other options:

*1. **Efficiency and Reduced Training Time**:*

Pre-trained models already have calculated weights from extensive training on large datasets. This eliminates the need for training the model from scratch, saving significant time and computational resources.

*2. **Cost-Effectiveness**:*

Developing a model in-house requires substantial investment in data collection, labeling, infrastructure, and computational power. Pre-trained models minimize these costs as they are already trained on diverse datasets.

*3. **Generalization**:*

Pre-trained models, especially large-scale ones like GPT and BERT, are trained on vast amounts of data, enabling them to generalize better across various tasks. Their robustness allows them to adapt well to new tasks with minimal fine-tuning.

*4. **Accessibility of Weights**:*

Many pre-trained models share weights, particularly for initial layers, which are better at generalizing across different domains. This enables developers to leverage high-quality features without needing to retrain those layers.

5. **Ease of Fine-Tuning**:
Pre-trained models can be fine-tuned for specific tasks with relatively small datasets. This makes them highly versatile and suitable for applications requiring customization.

6. **State-of-the-Art Performance**:
Large-scale pre-trained models have achieved state-of-the-art results in various benchmarks and tasks, making them an excellent choice for applications requiring cutting-edge performance.

7. **Community and Ecosystem**:
Many pre-trained models come with extensive documentation, support, and community contributions. This makes them easier to integrate into products and allows developers to benefit from shared knowledge and tools.

In summary, pre-trained models provide significant advantages in terms of efficiency, cost, generalization, and ease of use, making them a preferred choice for incorporating language AI into products.

Context:

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the number of trainable parameters. In the Transformer architecture, there are four weight matrices in the self-attention module $(W_{q},W_{k},W_{v},W_{o})$ and two in the MLP module. We treat W_{q} (or W_{k}, W_{v}) as a single matrix of dimension $d_{m o d e l}\times d_{m o d e l}$, even though the output dimension is usually sliced into attention heads. We limit our study to only adapting the attention weights for downstream tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity and parameter-ef?ciency.We further study the effect on adapting different types of attention weight matrices in a Transformer in Section 7.1. We leave the empirical investigation of adapting the MLP layers, LayerNorm layers, and biases to a future work.

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the number of trainable parameters. In the Transformer architecture, there are four weight matrices in the self-attention module $(W_{q},W_{k},W_{v},W_{o})$ and two in the MLP module. We treat W_{q} (or W_{k}, W_{v}) as a single matrix of dimension $d_{m o d e l}\times d_{m o d e l}$, even though the output dimension is usually sliced into attention heads. We limit our study to only adapting the attention weights for downstream tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity and parameter-ef?ciency.We further study the effect on adapting different types of attention weight matrices in a

Transformer in Section 7.1. We leave the empirical investigation of adapting the MLP layers, LayerNorm layers, and biases to a future work.

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the number of trainable parameters. In the Transformer architecture, there are four weight matrices in the self-attention module $(W_{q},W_{k},W_{v},W_{o})$ and two in the MLP module. We treat W_{q} (or W_{k} , W_{v}) as a single matrix of dimension $d_{m o d e l}\times d_{m o d e l}$, even though the output dimension is usually sliced into attention heads. We limit our study to only adapting the attention weights for downstream tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity and parameter-ef?ciency.We further study the effect on adapting different types of attention weight matrices in a Transformer in Section 7.1. We leave the empirical investigation of adapting the MLP layers, LayerNorm layers, and biases to a future work.

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

Table 6 shows that, surprisingly, LoRA already performs competitively with a very small r (more so for $\{W_{q},W_{v}\}$ than just $W_{q,i}$). This suggests the update matrix ΔW could have a very small ?intrinsic rank?. To further support this ?nding, we check the overlap of the subspaces learned by different choices of r and by different random seeds. We argue that increasing r does not cover a more meaningful subspace, which suggests that a low-rank adaptation matrix is suf?cient.

Table 6 shows that, surprisingly, LoRA already performs competitively with a very small r (more so for $\{W_{q},W_{v}\}$

than just $W_{q,i}$). This suggests the update matrix ΔW could have a very small ?intrinsic rank?. To further support this ?nding, we check the overlap of the subspaces learned by different choices of r and by different random seeds. We argue that increasing r does not cover a more meaningful subspace, which suggests that a low-rank adaptation matrix is suf?cient.

Table 6 shows that, surprisingly, LoRA already performs competitively with a very small r (more so for $\{W_{q},W_{v}\}$ than just $W_{q,i}$). This suggests the update matrix ΔW could have a very small ?intrinsic rank?. To further support this ?nding, we check the overlap of the subspaces learned by different choices of r and by different random seeds. We argue that increasing r does not cover a more meaningful subspace, which suggests that a low-rank adaptation matrix is suf?cient.

4 OUR METHOD

We describe the simple design of LoRA and its practical bene?ts. The principles outlined here apply to any dense layers in deep learning models, though we only focus on certain weights in Transformer language models in our experiments as the motivating use case.

4.1 LOW-RANK-PARAMETRIZED UPDATE MATRICES

Current Question:

which WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Answer:

Evaluation 6:

Evaluation of the Answer

Relevance: Excellent

The answer directly addresses the question "Why Self-Attention?" by explaining its purpose, benefits, and characteristics. It stays focused on the advantages and mechanisms of self-attention.

Faithfulness: Excellent

The answer is strictly supported by the provided context. It accurately describes self-attention, its applications, interpretability, and the comparison between positional embeddings as mentioned in the text.

Completeness: Excellent

The answer thoroughly covers all major aspects of self-attention discussed in the context, including its representation capabilities, interpretability, and flexibility in positional embeddings.

Clarity: Excellent

The response is well-structured, logically organized, and easy to understand. It uses clear language and breaks down the advantages of self-attention into distinct points.

Conciseness: Excellent

The answer is concise and avoids unnecessary repetition or verbosity, efficiently summarizing the key points about self-attention.

Final Ratings:

- **Relevance**: Excellent

- **Faithfulness**: Excellent

- **Completeness**: Excellent

- **Clarity**: Excellent

*- **Conciseness**: Excellent*

Weighted Score: 100.0

Question 6: Why Self-Attention?

Answer 6: Self-attention is an attention mechanism that relates different positions within a single sequence to compute a representation of that sequence. It offers several advantages:

*1. **Effective Representation of Sequences**: By relating positions within the sequence to each other, self-attention can capture both local and global dependencies. This makes it effective for tasks such as reading comprehension, abstractive summarization, textual entailment, and learning task-independent sentence representations.*

*2. **Interpretable Models**: Self-attention can make models more interpretable. Attention distributions from models reveal that individual attention heads learn to perform different tasks, with many exhibiting behavior related to the syntactic and semantic structure of sentences. This provides insights into how models understand and process text.*

*3. **Flexibility in Positional Embeddings**: Experiments comparing learned positional embeddings and sinusoidal positional embeddings showed nearly identical results. However, the sinusoidal version was chosen because it allows the model to extrapolate to sequence lengths longer than those encountered during training, making it more versatile.*

These characteristics make self-attention a powerful mechanism for modeling sequences and understanding complex relationships within textual data.

Response Latency: 4.60 seconds

Evaluation Latency: 5.49 seconds

[Prompt used for Q6]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 22, 23, 19].

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 22, 23, 19].

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 22, 23, 19].

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 22, 23, 19].

As side bene?t, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

As side bene?t, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear

to exhibit behavior related to the syntactic and semantic structure of the sentences.

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

As side bene?t, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

As side bene?t, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

We also experimented with using learned positional embeddings [8] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

4 Why Self-Attention

We also experimented with using learned positional embeddings [8] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

4 Why Self-Attention

Current Question:

Why Self-Attention?

Answer:

Evaluation 7:

Evaluation of the Answer:

Relevance: Excellent

The answer directly addresses the question by identifying the model with the highest average score and provides the score itself.

Faithfulness: Excellent

The answer is strictly supported by the given context. It accurately identifies **BERT LARGE** as the model with the highest average score and states the score as **90**, which aligns with the data in Table 1.

Completeness: Excellent

The answer covers all major aspects of the question, including both the name of the model and its corresponding score.

Clarity: Excellent

The answer is well-structured and easy to understand, with no ambiguity in the response.

Conciseness: Excellent

The answer is concise and avoids unnecessary repetition or verbosity.

Overall Rating: Excellent

Weighted Score: 100.0

Question 7: Among the five models in Table 1, which model has the highest average score and what is the highest average score?

Answer 7: In Table 1, the model with the highest average score is **BERT LARGE**, with an average score of **90**.

Response Latency: 4.04 seconds

Evaluation Latency: 62.35 seconds

[Prompt used for Q7]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

[Sheet: Sheet1] Row 25: R.M.Reader(Ensemble)

NaN

NaN

81.2

87.9

82.3

88.5

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 29: BERT LARGE(Ensemble)

NaN

NaN

85.8

91.8

-

-

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 45: unet(Ensemble)

NaN

NaN

-

-

71.4

74.9

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 19: Top Leaderboard Systems (Dec 10th, 2018)

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1]

Unnamed: 0 Unnamed: 1 Unnamed: 2 Unnamed: 3 Unnamed: 4 Unnamed: 5 Unnamed: 6 Unnamed: 7 Unnamed: 8 Unnamed: 9 Unnamed: 10 Unnamed: 11

System NaN NaN MNLI-(m/mm) QQP QNLI SST-2 CoLA STS-B MRPC RTE Average

NaN	NaN	NaN	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	NaN	NaN	80.6/80.1	66.1	82.3	93.2	35	81	86	61.7	74
BiLSTM+ELMo+Attn	NaN	NaN	76.4/76.1	64.8	79.8	90.4	36	73.3	84.9	56.8	71
OpenAI GPT	NaN	NaN	82.1/81.4	70.3	87.4	91.3	45.4	80	82.3	56	75.1
BERT BASE(Single)	NaN	NaN	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT BLARGE	NaN	NaN	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	90
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples.The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components. NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

NaN

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

System *NaN* *NaN* *Dev* *NaN* *Test* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN NaN NaN EM F1 EM F1

NaN NaN NaN NaN NaN

Top Leaderboard Systems (Dec 10th, 2018) NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

Human NaN NaN - - 82.3 91.2

NaN NaN NaN NaN NaN

#1 Ensemble - nlnet NaN NaN - - 86 91.7

NaN NaN NaN NaN NaN

#2 Ensemble - QANet NaN NaN - - 84.5 90.5

NaN NaN NaN NaN NaN

Published NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

BiDAF+ELMo(Single) NaN NaN - 85.6 - 85.8

NaN NaN NaN NaN NaN

R.M.Reader(Ensemble) NaN NaN 81.2 87.9 82.3

88.5 NaN NaN NaN NaN NaN

Ours	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT BASE(Single)	*NaN*	*NaN*	*80.8*	*88.5*	-	-	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Single)	*NaN*	*NaN*	*84.1*	*90.9*	-	-	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Ensemble)	*NaN*	*NaN*	*85.8*	*91.8*	-	-	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Shl.+TriviaQA)	*NaN*	*NaN*	*84.2*	*91.1*	*85.1*	*91.8*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Ens.+TriviaQA)	*NaN*	*NaN*	*86.2*	*92.2*	*87.4*	*93.2*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
	NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*

Table 2:

SQuAD 1.1 results. The BERT ensemble is 7x systems\nwhich use different pre-training checkpoints and fine-tuning\nseeds.

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

System NaN NaN Dev NaN Test NaN

NaN NaN NaN NaN NaN

NaN NaN NaN EM F1 EM F1

NaN NaN NaN NaN NaN

Top Leaderboard Systems (Dec 10th, 2018) NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

Human NaN NaN 86.3 89 86.9 89.5

NaN NaN NaN NaN NaN

#1 Single-MIR-MRC(F-Net) NaN NaN - - 74.8 78

NaN NaN NaN NaN NaN

#2 Single-nlnet NaN NaN - - 74.2 77.1

NaN NaN NaN NaN NaN

Published NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

unet(Ensemble) NaN NaN - - 71.4 74.9

NaN NaN NaN NaN NaN

SLQA+(Single) NaN NaN - - 71.4 74.4

NaN NaN NaN NaN NaN

Ours NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

BERT LARGE(Single) NaN NaN 78.7 81.9 80

83.1 NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components. NaN NaN

NaN NaN NaN NaN NaN NaN NaN NaN NaN

[Sheet: Sheet1]

Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11
System	NaN	NaN	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
NaN	NaN	NaN	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	NaN	NaN	80.6/80.1	66.1	82.3	93.2	35	81	86	61.7	74
BiLSTM+ELMo+Attn	NaN	NaN	76.4/76.1	64.8	79.8	90.4	36	73.3	84.9	56.8	71
OpenAI GPT	NaN	NaN	82.1/81.4	70.3	87.4	91.3	45.4	80	82.3	56	75.1
BERT BASE(Single)	NaN	NaN	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT BLARGE	NaN	NaN	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	90

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples.The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components. *NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN*

NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

System NaN NaN Dev NaN Test NaN

NaN NaN NaN NaN NaN

NaN NaN NaN EM F1 EM F1

NaN NaN NaN NaN NaN

Top Leaderboard Systems (Dec 10th, 2018) NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

Human NaN NaN - - 82.3 91.2

NaN NaN NaN NaN NaN

#1 Ensemble - nlnet NaN NaN - - 86 91.7

NaN NaN NaN NaN NaN

#2 Ensemble - QANet NaN NaN - - 84.5 90.5

NaN NaN NaN NaN NaN

Published NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

BiDAF+ELMo(Single)	NaN	NaN	-	85.6	-	85.8	NaN	NaN	NaN	NaN	NaN
R.M.Reader(Ensemble)	NaN	NaN	81.2	87.9	82.3	88.5	NaN	NaN	NaN	NaN	NaN
Ours	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BERT BASE(Single)	NaN	NaN	80.8	88.5	-	-	NaN	NaN	NaN	NaN	NaN
BERT LARGE(Single)	NaN	NaN	84.1	90.9	-	-	NaN	NaN	NaN	NaN	NaN
BERT LARGE(Ensemble)	NaN	NaN	85.8	91.8	-	-	NaN	NaN	NaN	NaN	NaN
BERT LARGE(Shl.+TriviaQA)	NaN	NaN	84.2	91.1	85.1	91.8	NaN	NaN	NaN	NaN	NaN
BERT LARGE(Ens.+TriviaQA)	NaN	NaN	86.2	92.2	87.4	93.2	NaN	NaN	NaN	NaN	NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

Table 2:

SQuAD 1.1 results. The BERT ensemble is 7x systems\nwhich use different pre-training checkpoints and fine-tuning\nseeds.

NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

System NaN NaN Dev NaN Test NaN

NaN NaN NaN NaN NaN

NaN NaN NaN EM F1 EM F1

NaN NaN NaN NaN NaN

Top Leaderboard Systems (Dec 10th, 2018) *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

Human *NaN* *NaN* *86.3* *89* *86.9* *89.5*

NaN *NaN* *NaN* *NaN* *NaN*

#1 Single-MIR-MRC(F-Net) *NaN* *NaN* *-* *-* *74.8* *78*

NaN *NaN* *NaN* *NaN* *NaN*

#2 Single-nlnet *NaN* *NaN* *-* *-* *74.2* *77.1*

NaN *NaN* *NaN* *NaN* *NaN*

Published *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

unet(Ensemble) *NaN* *NaN* *-* *-* *71.4* *74.9*

NaN *NaN* *NaN* *NaN* *NaN*

SLQA+(Single) *NaN* *NaN* *-* *-* *71.4* *74.4*

NaN *NaN* *NaN* *NaN* *NaN*

Ours *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

BERT LARGE(Single) *NaN* *NaN* *78.7* *81.9* *80*

83.1 NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components. NaN NaN

NaN NaN NaN NaN NaN NaN NaN NaN NaN

[Sheet: Sheet1]

Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11
System	NaN	NaN	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
NaN	NaN	NaN	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	NaN	NaN	80.6/80.1	66.1	82.3	93.2	35	81	86	61.7	74
BiLSTM+ELMo+Attn	NaN	NaN	76.4/76.1	64.8	79.8	90.4	36	73.3	84.9	56.8	71
OpenAI GPT	NaN	NaN	82.1/81.4	70.3	87.4	91.3					

							45.4	80	82.3	56	75.1
BERT BASE(Single)	NaN	NaN	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT BLARGE	NaN	NaN	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	90
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples.The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

System	*NaN*	*NaN*	*Dev*	*NaN*	*Test*	*NaN*
NaN	*NaN*	*NaN*	*EM*	*F1*	*EM*	*F1*
Top Leaderboard Systems (Dec 10th, 2018)	*NaN*	*NaN*	*NaN*	*NaN*		
Human	*NaN*	*NaN*	-	-	*82.3*	*91.2*
#1 Ensemble - nlnet	*NaN*	*NaN*	-	-	*86*	*91.7*

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

#2 Ensemble - QANet NaN NaN - - 84.5 90.5

NaN NaN NaN NaN NaN

Published NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

BiDAF+ELMo(Single) NaN NaN - 85.6 - 85.8

NaN NaN NaN NaN NaN

R.M.Reader(Ensemble) NaN NaN 81.2 87.9 82.3

88.5 NaN NaN NaN NaN NaN

Ours NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

BERT BASE(Single) NaN NaN 80.8 88.5 - -

NaN NaN NaN NaN NaN

BERT LARGE(Single) NaN NaN 84.1 90.9 - -

NaN NaN NaN NaN NaN

BERT LARGE(Ensemble) NaN NaN 85.8 91.8 -

- NaN NaN NaN NaN NaN

BERT LARGE(Shl.+TriviaQA) *NaN* *NaN* *84.2* *91.1* *85.1*

91.8 *NaN* *NaN* *NaN* *NaN* *NaN*

BERT LARGE(Ens.+TriviaQA) *NaN* *NaN* *86.2* *92.2* *87.4*

93.2 *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

Table 2:

SQuAD 1.1 results. The BERT ensemble is 7x systems\nwhich use different pre-training checkpoints and fine-tuning\nseeds.

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN *NaN* *NaN* *NaN* *NaN*

System	*NaN*	*NaN*	*Dev*	*NaN*	*Test*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*EM*	*F1*	*EM*	*F1*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Top Leaderboard Systems (Dec 10th, 2018)	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Human	*NaN*	*NaN*	*86.3*	*89*	*86.9*	*89.5*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
#1 Single-MIR-MRC(F-Net)	*NaN*	*NaN*	*-*	*-*	*74.8*	*78*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
#2 Single-nlnet	*NaN*	*NaN*	*-*	*-*	*74.2*	*77.1*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Published	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
unet(Ensemble)	*NaN*	*NaN*	*-*	*-*	*71.4*	*74.9*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*

SLQA+(Single) NaN NaN - - 71.4 74.4
NaN NaN NaN NaN NaN

Ours NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN NaN

BERT LARGE(Single) NaN NaN 78.7 81.9 80
83.1 NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN NaN

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components. NaN NaN
NaN NaN NaN NaN NaN NaN NaN NaN NaN
[Sheet: Sheet1]

Unnamed: 0 Unnamed: 1 Unnamed: 2 Unnamed: 3 Unnamed: 4
Unnamed: 5 Unnamed: 6 Unnamed: 7 Unnamed: 8 Unnamed: 9 Unnamed: 10 Unnamed: 11

System NaN NaN MNLI-(m/mm) QQP QNLI
SST-2 CoLA STS-B MRPC RTE Average

NaN NaN NaN 392k 363k 108k 67k
8.5k 5.7k 3.5k 2.5k -

Pre-OpenAI SOTA	*NaN*	*NaN*	*80.6/80.1*	*66.1*	*82.3*	*93.2*	*35*	*81*	*86*	*61.7*	*74*
BiLSTM+ELMo+Attn	*NaN*	*NaN*	*76.4/76.1*	*64.8*	*79.8*	*90.4*	*36*	*73.3*	*84.9*	*56.8*	*71*
OpenAI GPT	*NaN*	*NaN*	*82.1/81.4*	*70.3*	*87.4*	*91.3*	*45.4*	*80*	*82.3*	*56*	*75.1*
BERT BASE(Single)	*NaN*	*NaN*	*84.6/83.4*	*71.2*	*90.5*	*93.5*	*52.1*	*85.8*	*88.9*	*66.4*	*79.6*
BERT BLARGE	*NaN*	*NaN*	*86.7/85.9*	*72.1*	*92.7*	*94.9*	*60.5*	*86.5*	*89.3*	*70.1*	*90*
NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples.The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components. *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN* *NaN*

NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*

					NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*							
					NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*							
					NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*							
					NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*							
					NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*							
					NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*							
					System	*NaN*	*NaN*	*Dev*	*NaN*	*Test*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*							
					NaN	*NaN*	*NaN*	*EM*	*F1*	*EM*	*F1*
NaN	*NaN*	*NaN*	*NaN*	*NaN*							

Top Leaderboard Systems (Dec 10th, 2018)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Human	NaN	NaN	-	-	82.3	91.2	NaN	NaN	NaN	NaN	NaN
#1 Ensemble - nlnet	NaN	NaN	-	-	86	91.7	NaN	NaN	NaN	NaN	NaN
#2 Ensemble - QANet	NaN	NaN	-	-	84.5	90.5	NaN	NaN	NaN	NaN	NaN
Published	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BiDAF+ELMo(Single)	NaN	NaN	-	85.6	-	85.8	NaN	NaN	NaN	NaN	NaN
R.M.Reader(Ensemble)	NaN	NaN	81.2	87.9	82.3	88.5	NaN	NaN	NaN	NaN	NaN
Ours	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BERT BASE(Single)	NaN	NaN	80.8	88.5	-	-					

NaN NaN NaN NaN NaN

BERT LARGE(Single) NaN NaN 84.1 90.9 - -

NaN NaN NaN NaN NaN

BERT LARGE(Ensemble) NaN NaN 85.8 91.8 -

- NaN NaN NaN NaN NaN

BERT LARGE(Shl.+TriviaQA) NaN NaN 84.2 91.1 85.1

91.8 NaN NaN NaN NaN NaN

BERT LARGE(Ens.+TriviaQA) NaN NaN 86.2 92.2 87.4

93.2 NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

Table 2:

SQuAD 1.1 results. The BERT ensemble is 7x systems\nwhich use different pre-training checkpoints and fine-tuning\nseeds.

NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
System	NaN	NaN	Dev	NaN	Test	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	EM	F1	EM	F1	NaN	NaN	NaN	NaN	NaN
Top Leaderboard Systems (Dec 10th, 2018)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Human	NaN	NaN	86.3	89	86.9	89.5	NaN	NaN	NaN	NaN	NaN
#1 Single-MIR-MRC(F-Net)	NaN	NaN	-	-	74.8	78	NaN	NaN	NaN	NaN	NaN
#2 Single-nlnet	NaN	NaN	-	-	74.2	77.1	NaN	NaN	NaN	NaN	NaN

Published NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN NaN

unet(Ensemble) NaN NaN - - 71.4 74.9
NaN NaN NaN NaN NaN

SLQA+(Single) NaN NaN - - 71.4 74.4
NaN NaN NaN NaN NaN

Ours NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN NaN

BERT LARGE(Single) NaN NaN 78.7 81.9 80
83.1 NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN NaN

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components. NaN NaN
NaN NaN NaN NaN NaN NaN NaN NaN NaN
[Sheet: Sheet1]

Unnamed: 0 Unnamed: 1 Unnamed: 2 Unnamed: 3 Unnamed: 4
Unnamed: 5 Unnamed: 6 Unnamed: 7 Unnamed: 8 Unnamed: 9 Unnamed: 10 Unnamed: 11

System	NaN	NaN	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
NaN	NaN	NaN	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	NaN	NaN	80.6/80.1	66.1	82.3	93.2	35	81	86	61.7	74
BiLSTM+ELMo+Attn	NaN	NaN	76.4/76.1	64.8	79.8	90.4	36	73.3	84.9	56.8	71
OpenAI GPT	NaN	NaN	82.1/81.4	70.3	87.4	91.3	45.4	80	82.3	56	75.1
BERT BASE(Single)	NaN	NaN	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT BLARGE	NaN	NaN	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	90
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples.The "Average" column is slightly different than the official GLUE score, since we exclude

the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components. *NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN*

NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

System	*NaN*	*NaN*	*Dev*	*NaN*	*Test*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*		
NaN	*NaN*	*NaN*	*EM*	*F1*	*EM*	*F1*
NaN	*NaN*	*NaN*	*NaN*	*NaN*		
Top Leaderboard Systems (Dec 10th, 2018)	*NaN*	*NaN*	*NaN*	*NaN*		
NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Human	*NaN*	*NaN*	-	-	*82.3*	*91.2*
NaN	*NaN*	*NaN*	*NaN*	*NaN*		
#1 Ensemble - nlnet	*NaN*	*NaN*	-	-	*86*	*91.7*
NaN	*NaN*	*NaN*	*NaN*	*NaN*		
#2 Ensemble - QANet	*NaN*	*NaN*	-	-	*84.5*	*90.5*
NaN	*NaN*	*NaN*	*NaN*	*NaN*		
Published	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*		
BiDAF+ELMo(Single)	*NaN*	*NaN*	-	*85.6*	-	*85.8*
NaN	*NaN*	*NaN*	*NaN*	*NaN*		

R.M.Reader(Ensemble)	*NaN*	*NaN*	*81.2*	*87.9*	*82.3*	*88.5*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Ours	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT BASE(Single)	*NaN*	*NaN*	*80.8*	*88.5*	-	-	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Single)	*NaN*	*NaN*	*84.1*	*90.9*	-	-	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Ensemble)	*NaN*	*NaN*	*85.8*	*91.8*	-	-	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Shl.+TriviaQA)	*NaN*	*NaN*	*84.2*	*91.1*	*85.1*	*91.8*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Ens.+TriviaQA)	*NaN*	*NaN*	*86.2*	*92.2*	*87.4*	*93.2*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems\nwhich use different pre-training checkpoints and fine-tuning\nseeds.

NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

System NaN NaN Dev NaN Test NaN

NaN NaN NaN NaN NaN

NaN NaN NaN EM F1 EM F1

NaN NaN NaN NaN NaN

Top Leaderboard Systems (Dec 10th, 2018) NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

Human NaN NaN 86.3 89 86.9 89.5

NaN NaN NaN NaN NaN

#1 Single-MIR-MRC(F-Net)	*NaN*	*NaN*	-	-	*74.8*	*78*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
#2 Single-nlnet	*NaN*	*NaN*	-	-	*74.2*	*77.1*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Published	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
unet(Ensemble)	*NaN*	*NaN*	-	-	*71.4*	*74.9*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
SLQA+(Single)	*NaN*	*NaN*	-	-	*71.4*	*74.4*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Ours	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Single)	*NaN*	*NaN*	*78.7*	*81.9*	*80*	*83.1*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
NaN	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components. NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

[Sheet: Sheet1]

Unnamed: 0 Unnamed: 1 Unnamed: 2 Unnamed: 3 Unnamed: 4 Unnamed: 5 Unnamed: 6 Unnamed: 7 Unnamed: 8 Unnamed: 9 Unnamed: 10 Unnamed: 11

System NaN NaN MNLI-(m/mm) QQP QNLI SST-2 CoLA STS-B MRPC RTE Average

NaN NaN NaN 392k 363k 108k 67k 8.5k 5.7k 3.5k 2.5k -

Pre-OpenAI SOTA NaN NaN 80.6/80.1 66.1 82.3 93.2 35 81 86 61.7 74

BiLSTM+ELMo+Attn NaN NaN 76.4/76.1 64.8 79.8 90.4 36 73.3 84.9 56.8 71

OpenAI GPT NaN NaN 82.1/81.4 70.3 87.4 91.3 45.4 80 82.3 56 75.1

BERT BASE(Single) NaN NaN 84.6/83.4 71.2 90.5 93.5 52.1 85.8 88.9 66.4 79.6

BERT BLARGE NaN NaN 86.7/85.9 72.1 92.7

94.9 60.5 86.5 89.3 70.1 90

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples.The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components. *NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN*

NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

System NaN NaN Dev NaN Test NaN

NaN NaN NaN NaN NaN

NaN NaN NaN EM F1 EM F1

NaN NaN NaN NaN NaN

Top Leaderboard Systems (Dec 10th, 2018) NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

Human NaN NaN - - 82.3 91.2

NaN NaN NaN NaN NaN

#1 Ensemble - nlnet NaN NaN - - 86 91.7

NaN NaN NaN NaN NaN

#2 Ensemble - QANet NaN NaN - - 84.5 90.5

NaN NaN NaN NaN NaN

Published	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BiDAF+ELMo(Single)	*NaN*	*NaN*	-	*85.6*	-	*85.8*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
R.M.Reader(Ensemble)	*NaN*	*NaN*	*81.2*	*87.9*	*82.3*	*88.5*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
Ours	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT BASE(Single)	*NaN*	*NaN*	*80.8*	*88.5*	-	-	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Single)	*NaN*	*NaN*	*84.1*	*90.9*	-	-	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Ensemble)	*NaN*	*NaN*	*85.8*	*91.8*	-	-	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*
BERT LARGE(Shl.+TriviaQA)	*NaN*	*NaN*	*84.2*	*91.1*	*85.1*	*91.8*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*

BERT LARGE(Ens.+TriviaQA)	NaN	NaN	86.2	92.2	87.4	93.2	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems\nwhich use different pre-training checkpoints and fine-tuning\nseeds.

NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
System	NaN	NaN	Dev	NaN	Test	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	EM	F1	EM	F1					

NaN NaN NaN NaN NaN

Top Leaderboard Systems (Dec 10th, 2018) NaN NaN NaN NaN

NaN NaN NaN NaN NaN NaN NaN

Human NaN NaN 86.3 89 86.9 89.5

NaN NaN NaN NaN NaN

#1 Single-MIR-MRC(F-Net) NaN NaN - - 74.8 78

NaN NaN NaN NaN NaN

#2 Single-nlnet NaN NaN - - 74.2 77.1

NaN NaN NaN NaN NaN

Published NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

unet(Ensemble) NaN NaN - - 71.4 74.9

NaN NaN NaN NaN NaN

SLQA+(Single) NaN NaN - - 71.4 74.4

NaN NaN NaN NaN NaN

Ours NaN NaN NaN NaN NaN NaN

NaN NaN NaN NaN NaN

BERT LARGE(Single)   NaN   NaN   78.7   81.9   80

83.1   NaN   NaN   NaN   NaN   NaN

NaN   NaN   NaN   NaN   NaN   NaN   NaN

NaN   NaN   NaN   NaN   NaN

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components.   NaN   NaN

NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

Current Question:

Among the five models in Table 1, which model has the highest average score and what is the highest average score?

Answer:

Evaluation 8:

Evaluation of the Answer:

Relevance: **Excellent**

The answer directly addresses the question by providing a clear and detailed explanation of LoRA fine-tuning, including its mechanism, features, and applications. It is highly relevant to the context and question.

Faithfulness: **Excellent**

The answer is strictly supported by the provided context, accurately describing all relevant aspects of LoRA fine-tuning, including its efficiency, modularity, and orthogonality to other methods.

Completeness: **Excellent**

The answer covers all major aspects of LoRA fine-tuning, such as its key features, comparison to other methods, and applications. It provides a well-rounded explanation without omitting significant details.

Clarity: **Excellent**

The answer is well-structured and easy to understand, with clear headings and concise explanations for each feature of LoRA fine-tuning.

Conciseness: **Good**

While the answer is detailed and comprehensive, some sections (e.g., "Key Features of LoRA Fine-Tuning") could be slightly condensed without losing critical information. However, this issue is minor.

Final Ratings:

- **Relevance:** Excellent

- **Faithfulness:** Excellent

- **Completeness:** Excellent

- **Clarity:** Excellent

- **Conciseness:** Good

Weighted Score: 98.0

Question 8: what's LoRA fine-tuning?

Answer 8: LoRA (Low-Rank Adaptation) fine-tuning is an efficient method for adapting pre-trained models to downstream tasks by introducing small, trainable low-rank matrices into the model's weight structure, without needing to fine-tune the entire set of parameters in the model.

Key Features of LoRA Fine-Tuning:

1. **Low-Rank Updates**: LoRA injects low-rank matrices into the weight update process during training. This allows the model to adapt without requiring the gradient updates to have full rank, significantly reducing the number of trainable parameters.

2. **Efficient Training**: By optimizing only the injected low-rank matrices, LoRA reduces computational overhead and storage requirements. This makes training more hardware-efficient and lowers the barrier to entry, especially when using adaptive optimizers.

3. **No Inference Latency**: LoRA's design allows the trainable matrices to be merged with the pre-trained weights during deployment, ensuring that the model's inference speed remains unchanged compared to a fully fine-tuned model.

4. **Orthogonality**: LoRA is orthogonal to many other adaptation methods, meaning it can be combined with approaches like prefix-tuning to potentially achieve further improvements.

5. **Expressiveness**: LoRA approximates the expressiveness of full fine-tuning as the number of trainable parameters increases. By setting the LoRA rank \(r \) to the rank of the pre-trained weight matrices, LoRA can recover the capabilities of full fine-tuning.

6. **Task Modularity**: LoRA enables modular task-switching. A shared pre-trained model can be frozen, and small LoRA modules (matrices \(A \) and \(B \)) can be swapped in and out to efficiently adapt the model to multiple tasks, reducing storage requirements and overhead.

Comparison to Other Methods:

- **Adapter-based methods** converge to an MLP structure with limited expressiveness as trainable parameters increase.
- **Prefix-based methods** struggle with long input sequences.
- LoRA provides a more efficient and scalable approach, especially for tasks where full fine-tuning is impractical due to computational or storage constraints.

Applications:

LoRA fine-tuning is particularly useful for scenarios requiring efficient adaptation of large pre-trained models, such as language models, with minimal hardware and storage requirements. It is also beneficial for multi-task setups where switching between tasks quickly is necessary.

Response Latency: 10.85 seconds

Evaluation Latency: 5.00 seconds

[Prompt used for Q8]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

In Table 15, we show the evaluation results of $_\mathrm{L oRA+PE}$ and $_\mathrm{LoRAA+PL}$ on WikiSQL and MultiNLI. First of all, $_\mathrm{L oRA+PE}$ signi?cantly outperforms both LoRA and pre?x-embedding tuning on WikiSQL, which indicates that LoRA is somewhat orthogonal to pre?x-embedding tuning. On MultiNLI, the combination of $_\mathrm{L oRA+PE}$ doesn?t perform better than LoRA, possibly because LoRA on its own already achieves performance comparable to the human baseline. Secondly, we notice that $_\mathrm{L oRA+PL}$ performs slightly worse than LoRA even with more trainable parameters. We at- tribute this to the fact that pre?x-layer tuning is very sensitive to the choice of learning rate and thus makes the optimization of LoRA weights more dif?cult in $_\mathrm{L oRA+PL}$.

In Table 15, we show the evaluation results of $_\mathrm{L oRA+PE}$ and $_\mathrm{LoRAA+PL}$ on WikiSQL and MultiNLI. First of all, $_\mathrm{L oRA+PE}$ signi?cantly outperforms both LoRA and pre?x-embedding tuning on WikiSQL, which indicates that LoRA is somewhat orthogonal to pre?x-embedding tuning. On MultiNLI, the combination of $_\mathrm{L oRA+PE}$ doesn?t perform better than LoRA, possibly because LoRA on its own already achieves performance comparable to the human baseline. Secondly, we notice that $_\mathrm{L oRA+PL}$ performs slightly worse than LoRA even with more trainable parameters. We at- tribute this to the fact that pre?x-layer tuning is very sensitive to the choice of learning rate and thus makes the optimization of LoRA weights more dif?cult in $_\mathrm{L oRA+PL}$.

In Table 15, we show the evaluation results of $_\mathrm{L oRA+PE}$ and $_\mathrm{LoRAA+PL}$ on WikiSQL and MultiNLI. First of all, $_\mathrm{L oRA+PE}$ signi?cantly outperforms both LoRA and pre?x-embedding tuning on WikiSQL, which indicates that LoRA is somewhat orthogonal to pre?x-embedding tuning. On MultiNLI, the combination of $_\mathrm{L oRA+PE}$ doesn?t perform better than LoRA, possibly because LoRA on its own already achieves performance comparable to the human baseline. Secondly, we notice that $_\mathrm{L oRA+PL}$ performs slightly worse than LoRA even with more trainable parameters. We at- tribute this to the fact that pre?x-layer tuning is very sensitive to the choice of learning rate and thus makes the optimization of LoRA weights more dif?cult in $_\mathrm{L oRA+PL}$.

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

There are many directions for future works. 1) LoRA can be combined with other ef?cient adapta- tion methods, potentially providing orthogonal improvement. 2) The mechanism behind ?ne-tuning or LoRA is far from clear ? how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full ?ne- tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways

to do it? 4) Finally, the rank-de?ciency of ΔW suggests that W could be rank-de?cient as well, which can also be a source of inspiration for future works.

A Generalization of Full Fine-tuning. A more general form of ?ne-tuning allows the training of a subset of the pre-trained parameters. LoRA takes a step further and does not require the accumu- lated gradient update to weight matrices to have full-rank during adaptation. This means that when applying LoRA to all weight matrices and training all biases 2 , we roughly recover the expressive- ness of full ?ne-tuning by setting the LoRA rank r to the rank of the pre-trained weight matrices. In other words, as we increase the number of trainable parameters 3 , training LoRA roughly converges to training the original model, while adapter-based methods converges to an MLP and pre?x-based methods to a model that cannot take long input sequences.

A Generalization of Full Fine-tuning. A more general form of ?ne-tuning allows the training of a subset of the pre-trained parameters. LoRA takes a step further and does not require the accumu- lated gradient update to weight matrices to have full-rank during adaptation. This means that when applying LoRA to all weight matrices and training all biases 2 , we roughly recover the expressive- ness of full ?ne-tuning by setting the LoRA rank r to the rank of the pre-trained weight matrices. In other words, as we increase the number of trainable parameters 3 , training LoRA roughly converges to training the original model, while adapter-based methods converges to an MLP and pre?x-based methods to a model that cannot take long input sequences.

A Generalization of Full Fine-tuning. A more general form of ?ne-tuning allows the training of a subset of the pre-trained parameters. LoRA takes a step further and does not require the accumu- lated gradient update to weight matrices to have full-rank during adaptation. This means that when applying LoRA to all weight matrices and training all biases 2 , we roughly recover the expressive- ness of full ?ne-tuning by setting the LoRA rank r to the rank of the pre-trained weight matrices. In other words, as we increase the number of trainable parameters 3 , training LoRA roughly converges to training the original model, while adapter-based methods converges to an MLP and pre?x-based methods to a model that cannot take long input sequences.

? A pre-trained model can be shared and used to build many small LoRA modules for dif- ferent tasks. We can freeze the shared model and ef?ciently switch tasks by replacing the matrices A and B in Figure 1, reducing the storage requirement and task-switching over- head signi?cantly. ? LoRA makes training more ef?cient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most parameters. Instead, we only optimize the injected, much smaller low-rank matrices. ? Our simple linear design allows us to merge the trainable matrices with the frozen weights when deployed, introducing no inference latency compared to a fully ?ne-tuned model, by construction. ? LoRA is orthogonal to many prior methods and can be combined with many of them, such as pre?x-tuning. We provide an example in Appendix E.

Current Question:

what's LoRA fine-tuning?

Answer:

Evaluation 9:

Evaluation of the Answer

Relevance: **Excellent**

The answer directly addresses the question, providing a comparison between full fine-tuning and LoRA, focusing on the advantages and challenges of both approaches.

Faithfulness: **Excellent**

The explanation of full fine-tuning and LoRA aligns closely with the provided context, especially the challenges mentioned for full fine-tuning and the efficiency of LoRA. No unsupported claims are made.

Completeness: **Excellent**

The answer comprehensively discusses both full fine-tuning and LoRA, covering their key features, challenges, and advantages. It addresses all major aspects of the comparison.

Clarity: **Excellent**

The answer is well-structured and easy to understand. It uses headings and bullet points to organize information, making it accessible.

Conciseness: **Excellent**

The answer avoids unnecessary repetition and verbosity while providing sufficient detail to fully explain the concepts.

Final Ratings:

- **Relevance**: Excellent

- **Faithfulness**: Excellent

- **Completeness**: Excellent

- **Clarity**: Excellent

- **Conciseness**: Excellent

Weighted Score: 100.0

Question 9: compare it to full fine-tuning

Answer 9: Full fine-tuning involves updating all the parameters of a pre-trained model for a specific downstream task. While this approach can yield high performance, it comes with significant drawbacks:

Challenges in Full Fine-Tuning:
1. **Storage Requirements**: For each downstream task, a separate set of parameters is learned (\(\Delta \Phi\)), which is equal to the size of the original model (\(\left|\Phi_0\right|\)). For large models like GPT-3 (\(\left|\Phi_0\right| \approx 175\) billion parameters), storing and deploying multiple fine-tuned versions becomes impractical.
2. **Scalability Issues**: Managing numerous independent instances of fine-tuned models for different tasks is highly resource-intensive and challenging to deploy.

Alternative: LoRA (Low-Rank Adaptation)
LoRA provides a more efficient way to adapt large language models to downstream tasks by introducing adapter layers. Instead of fine-tuning all parameters, LoRA updates only a small subset, drastically reducing the storage and computational requirements.

Advantages of LoRA Over Full Fine-Tuning:
1. **Parameter Efficiency**: LoRA updates only low-rank matrices, resulting in significantly fewer parameters to store and manage compared to full fine-tuning.
2. **Modularity**: LoRA enables the use of a shared pre-trained model across multiple tasks with lightweight task-specific parameter updates.
3. **Reduced Latency**: Although LoRA introduces some inference latency due to the adapter layers, this is much smaller compared to the overhead of storing and loading separate fine-tuned models.

Summary:

While full fine-tuning achieves strong performance, it faces scalability and storage challenges for large-scale models like GPT-3. LoRA provides a more practical alternative by focusing on efficient parameter updates, making it more feasible for deploying large language models across multiple tasks.

Response Latency: 5.63 seconds

Evaluation Latency: 4.99 seconds

[Prompt used for Q9]

You are a helpful assistant. Use the following history and context to answer.

History:

Context:

![Table 8: Fine-tuning signi?cantly outperforms few-shot learning on GPT-3 (Brown et al., 2020).
](images/51a82a62001af34f2413052810f8d26eddcbd5a2b368851a2b87288daf1ccc12.jpg)

B I NFERENCE L ATENCY I NTRODUCED BY A DAPTER L AYERS

![Table 8: Fine-tuning signi?cantly outperforms few-shot learning on GPT-3 (Brown et al., 2020).
](images/51a82a62001af34f2413052810f8d26eddcbd5a2b368851a2b87288daf1ccc12.jpg)

B I NFERENCE L ATENCY I NTRODUCED BY A DAPTER L AYERS

![Table 8: Fine-tuning signi?cantly outperforms few-shot learning on GPT-3 (Brown et al., 2020).
](images/51a82a62001af34f2413052810f8d26eddcbd5a2b368851a2b87288daf1ccc12.jpg)

B I NFERENCE L ATENCY I NTRODUCED BY A DAPTER L AYERS

One of the main drawbacks for full ?ne-tuning is that for each downstream task, we learn a different set of parameters $\Delta\Phi$ ension $|\Delta\Phi|$ equals $\left|\Phi_{0}\right|$. Thus, if the pre-trained model is large (such as GPT-3 with | $|\Phi_{0}|\,\approx\,175$ | ? Billion), storing and deploying many independent instances of ?ne-tuned models can be challenging, if at all feasible.

One of the main drawbacks for full ?ne-tuning is that for each downstream task, we learn a different set of parameters $\Delta\Phi$ ension $|\Delta\Phi|$ equals $\left|\Phi_{0}\right|$. Thus, if the pre-trained model is large (such as GPT-3 with | $|\Phi_{0}|\,\approx\,175$ | ? Billion), storing and deploying many independent instances of ?ne-tuned models can be challenging, if at all feasible.

One of the main drawbacks for full ?ne-tuning is that for each downstream task, we learn a different set of parameters $\Delta\Phi$ ension $|\Delta\Phi|$ equals $\left|\Phi_{0}\right|$. Thus, if the pre-trained model is large (such as GPT-3 with | $|\Phi_{0}|\,\approx\,175$ | ? Billion), storing and deploying many independent instances of ?ne-tuned models can be challenging, if at all feasible.

4.2 A PPLYING L O RA TO T RANSFORMER

4.2 A PPLYING L O RA TO T RANSFORMER

4.2 A PPLYING L O RA TO T RANSFORMER

A L ARGE L ANGUAGE M ODELS S TILL N EED P ARAMETER U PDATES

Current Question:

compare it to full fine-tuning

Answer:

LLM Evaluator Robustness Test Results

[Fake Test 1] Question: What is a GPT pre-trained model?

Fake Answer: GPT pretraining is a supervised learning technique where the model is trained on labeled data to directly solve classification tasks like sentiment analysis or spam detection. Unlike language models, it does not require large amounts of text data and typically uses decision trees instead of neural networks.

LLM Evaluation: Evaluation of the provided answer:

Relevance: **Fair**
The answer partially addresses the question by discussing what GPT pretraining entails but includes inaccurate and irrelevant information about decision trees and supervised learning, which are not core to GPT pretraining.

Faithfulness: **Worse**
The answer is not supported by the provided context. It incorrectly describes GPT pretraining as supervised learning focused on classification tasks and mentions decision trees instead of neural networks, which contradicts the context discussing large-scale pre-trained language models like GPT.

Completeness: **Fair**
The answer fails to adequately cover the key aspects of GPT pre-trained models, such as their reliance on large-scale text data, unsupervised learning, and adaptation to downstream tasks. It only touches on pretraining in a superficial and incorrect manner.

Clarity: **Bad**
The explanation is confusing and misrepresents the concept of GPT pretraining. The inclusion of unrelated terms like decision trees adds to the lack of clarity.

Conciseness: **Good**

The answer is concise and free of unnecessary verbosity, but this is overshadowed by its inaccuracies and lack of relevance.

Final Assessment:

The answer demonstrates significant issues in faithfulness, clarity, and completeness, with only minor strengths in conciseness.

Weighted Score: 52.0

Response Latency: 0.00 seconds

Evaluation Latency: 5.93 seconds

[Fake Test 2] Question: What advantages does it have compared with other pre-trained models?

Fake Answer: GPT has no advantage over other models because it cannot process sequential data and lacks any form of attention mechanism. Additionally, GPT is a bidirectional model just like BERT, and it was designed primarily for image classification rather than text-based tasks.

LLM Evaluation: ### Evaluation:

Relevance: Worse

The answer does not address the advantages of GPT compared to other pre-trained models. Instead, it provides incorrect and irrelevant information about GPT's capabilities and design.

Faithfulness: Worse

The statements in the answer are not supported by the provided context. Key claims, such as GPT lacking attention mechanisms and being designed for image classification, are outright false and contradict established facts about GPT.

Completeness: Worse

The answer fails to cover any relevant aspects of the advantages of GPT compared to other pre-trained models, as requested in the question.

Clarity: Bad

The structure and phrasing of the answer are reasonably clear, but the content is misleading and incorrect, which negatively impacts understanding.

Conciseness: Good

While the answer is concise, brevity cannot compensate for the lack of correctness and relevance.

Final Notes:

The answer is fundamentally flawed in all critical aspects except conciseness. It demonstrates a lack of understanding of GPT and its advantages, offering false information that contradicts the context and known facts.

Weighted Score: 28.0

Response Latency: 0.00 seconds

Evaluation Latency: 5.97 seconds

[Fake Test 3] Question: what's the key advantages of LoRA?

Fake Answer: LoRA's main advantage lies in its ability to significantly enhance internet signal strength across rural communities using low-orbit satellite relays, a method first piloted by SpaceX in 2018.

LLM Evaluation: Relevance: Worse

The answer does not address the question about the key advantages of LoRA and instead discusses unrelated information about internet signal strength and satellite relays.

Faithfulness: Worse

The answer is not supported by the provided context. The context discusses LoRA in terms of pre-trained models, task-switching, efficiency, storage, and hardware requirements, which are entirely unrelated to the answer provided.

Completeness: Worse

The answer fails to cover any of the key advantages of LoRA mentioned in the context, omitting all relevant aspects such as efficient task-switching, reduced storage requirements, lower hardware barriers, and orthogonality to prior methods.

Clarity: Worse

The answer is misleading and confusing because it introduces unrelated concepts (internet signal strength and satellite relays) that have no connection to the question or context provided.

Conciseness: Worse

The answer is concise but irrelevant, as it introduces unrelated information that adds no value to the discussion, failing to focus on the key advantages of LoRA.

Weighted Score: 20.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.03 seconds

[Fake Test 4] Question: Lora's practical benefits, advantages and limitations.

Fake Answer: While LoRA offers many practical applications in aviation control systems, especially for stabilizing autopilot algorithms during turbulent weather, its main limitation is the high energy cost associated with electromagnetic feedback loops.

LLM Evaluation: Relevance: Worse

The answer does not address the question regarding LoRA's practical benefits, advantages, and limitations. It instead discusses an unrelated application in aviation control systems.

Faithfulness: Worse

The response is completely unsupported by the provided context, which discusses VRAM reduction, checkpoint size, and training speed improvements for LoRA. The mention of "aviation control systems" and "electromagnetic feedback loops" is entirely unrelated.

Completeness: Worse

The answer fails to cover any major aspects of LoRA's benefits, advantages, or limitations as mentioned in the context.

Clarity: Worse

The response is misleading and confusing, as it introduces unrelated concepts not addressed in the context.

Conciseness: Worse

Though the answer is brief, it is irrelevant and does not contribute meaningful or concise information about LoRA's practical benefits, advantages, or limitations.

Weighted Score: 20.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.53 seconds

[Fake Test 5] Question: which WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO?

Fake Answer: LoRA is typically applied to the Fourier-transformed attention gradient maps, particularly those found in the activation buffers of the encoder's non-linear thermostat modules.

LLM Evaluation: Relevance: **Worse**

The answer does not address the question, which explicitly asks about which weight matrices in the Transformer should be adapted using LoRA. The provided response mentions "Fourier-transformed attention gradient maps" and "activation buffers" in a manner irrelevant to the context.

Faithfulness: **Worse**

The answer is not supported by the context provided. The context specifies adapting weight matrices in the self-attention module, with details about parameter budgets and specific configurations, but the answer introduces unrelated concepts such as "Fourier-transformed attention gradient maps" and "non-linear thermostat modules," which are not mentioned in the context.

Completeness: **Worse**

The answer does not cover any aspects of the question. It fails to mention any of the weight matrices in the self-attention module (e.g., query, key, or value matrices), as outlined in the context.

Clarity: **Bad**

The answer is poorly structured and introduces ambiguous and incorrect terminology like "non-linear thermostat modules," which adds confusion rather than clarity.

Conciseness: **Fair**

While the answer is concise, the information it provides is entirely irrelevant and fails to contribute meaningfully to the question.

Weighted Score: 26.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.60 seconds

[Fake Test 6] Question: Why Self-Attention?

Fake Answer: Self-attention is primarily designed to monitor the oscillation rates of neuron temperatures within GPU cooling systems. By calibrating symbolic representations of thermal drift vectors (x?,?,x?) and projecting them onto imaginary time sequences (z?,?,z?), the Transformer is able to predict quantum entanglement collapse in attention gradients. One of the key benefits of self-attention layers is that they can process holographic inputs from multilingual dolphin vocalizations without requiring recurrent fishnet layers or convolutional snorkels. The attention mechanism utilizes phase-shifted antenna beams to align token streams via echolocation, drastically improving BLEU scores in underwater wireless translation. Unlike traditional models which rely on linear algebra, self-attention operates in a post-Euclidean topology where the shortest path between tokens is determined by ambient humidity. To reduce overfitting in arid environments, the authors introduced a method called ?rotational dropout,? which spins the model weights at 3000 RPM. Furthermore, by restricting attention to a neighborhood of size ?, Transformer models can simultaneously predict protein folding patterns and stock market fluctuations, especially when trained on ancient Sumerian financial records encoded in byte-pair encoding. As a side benefit, self-attention also makes Transformers immune to electromagnetic interference caused by solar flares, enhancing performance on Mars-based inference systems.

LLM Evaluation: Relevance: Worse

The answer does not address why self-attention is used and instead introduces unrelated concepts such as GPU cooling systems, dolphin vocalizations, antenna beams, and ancient financial records.

Faithfulness: Worse

The answer is entirely unsupported by the provided context, which describes self-attention as an attention mechanism for computing a representation of a sequence and lists its applications in natural language processing tasks.

Completeness: Worse

The answer fails to cover any of the major aspects of the question. It does not explain the actual purpose or benefits of self-attention as described in the context.

Clarity: Worse

The answer is confusing and filled with nonsensical jargon, making it difficult to understand or follow. It introduces concepts that are irrelevant and incoherent.

Conciseness: Worse

The answer is verbose and elaborates on irrelevant details, such as rotational dropout and Mars-based inference systems, which detract from the focus of the question.

Weighted Score: 20.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.68 seconds

[Fake Test 7] Question: Among the five models in Table 1, which model has the highest average score and what is the highest average score?

Fake Answer: Among the five models in Table 1, the model with the highest average score is

BiLSTM+ELMo+Attn, which achieves an average of 95.3. This is significantly higher than BERT(LARGE), which only reaches an average of 74.2. The superior performance of BiLSTM+ELMo+Attn across all tasks demonstrates the continued relevance of LSTM-based architectures over Transformer-based models in most language understanding benchmarks.

LLM Evaluation: ### Evaluation of the Answer

Relevance:
Fair - The answer mentions a model and its average score, but the information provided is incorrect. BiLSTM+ELMo+Attn does not have the highest average score in Table 1. Based on the given context, BERT LARGE appears to have the highest average score (90). The answer does attempt to address the question but fails to deliver accurate information.

Faithfulness:
Worse - The answer is not supported by the context. It inaccurately claims BiLSTM+ELMo+Attn achieves an average of 95.3, which is not present in the provided data. Additionally, it misrepresents BERT LARGE's average score as 74.2, which contradicts the context.

Completeness:
Fair - The answer identifies a model and its average score but does not provide accurate or comprehensive coverage of the question. It fails to correctly identify the highest-scoring model and its actual score.

Clarity:
Good - The answer is reasonably well-structured and easy to understand. However, the clarity does not compensate for the inaccuracies in the information provided.

Conciseness:

Excellent - The answer is concise and free of unnecessary repetition or verbosity.

Final Ratings:

- **Relevance:** Fair

- **Faithfulness:** Worse

- **Completeness:** Fair

- **Clarity:** Good

- **Conciseness:** Excellent

Weighted Score: 0.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.75 seconds

[Fake Test 8] Question: what's LoRA fine-tuning?

Fake Answer: LoRA fine-tuning is a process in which the entire neural network is replaced with a low-rank matrix to reduce all model computations to simple linear projections. During LoRA fine-tuning, the pre-trained model is discarded, and instead, a new model is trained entirely from scratch using randomly initialized low-rank tensors. This approach ensures faster convergence but sacrifices the benefits of transfer learning.

LLM Evaluation: ### Evaluation of the Answer:

Relevance: Good

The answer attempts to address the concept of LoRA fine-tuning but introduces inaccuracies, such as stating that the pre-trained model is discarded, which is not supported by the given context. While it discusses aspects of fine-tuning, it doesn't fully align with LoRA's actual process as described.

Faithfulness: Bad

The explanation contradicts the context provided. LoRA does not discard the pre-trained model; rather, it builds on it by injecting low-rank matrices into the weight updates. The claim that LoRA trains entirely from scratch using randomly initialized tensors is inaccurate and not faithful to the context.

Completeness: Fair

The answer touches on the idea of reducing computations but fails to mention key aspects of LoRA fine-tuning, such as its ability to maintain the expressiveness of full fine-tuning, its efficiency, and how it lowers hardware barriers. It does not cover all major aspects provided in the context.

Clarity: Good

The answer is structured and relatively easy to understand, but some inaccuracies may confuse readers about the actual mechanism of LoRA fine-tuning.

Conciseness: Excellent

The response is concise and avoids unnecessary repetition or verbosity.

Summary of Ratings:

- **Relevance:** Good

- **Faithfulness:** Bad

- **Completeness:** Fair

- **Clarity:** Good

- **Conciseness:** Excellent

Weighted Score: 72.0

Response Latency: 0.00 seconds

Evaluation Latency: 7.50 seconds

[Fake Test 9] Question: compare it to full fine-tuning

Fake Answer: Unlike full fine-tuning, which freezes all parameters to maintain stability, LoRA fine-tuning updates every parameter in the model aggressively using reinforcement learning. Full fine-tuning requires minimal hardware resources, while LoRA consumes large amounts of memory and introduces significant inference latency. Additionally, full fine-tuning supports real-time task switching, whereas LoRA must retrain the entire model for each new task.

LLM Evaluation: ### Evaluation:

Relevance: Bad
The answer does not directly address the comparison between LoRA fine-tuning and full fine-tuning. Instead, it provides incorrect and misleading information about both techniques, failing to accurately explain the differences.

Faithfulness: Worse
The claims made about LoRA and full fine-tuning are not supported by the context provided. The information is inaccurate, as LoRA does not update every parameter aggressively using reinforcement learning, nor does full fine-tuning freeze all parameters.

Completeness: Worse
The answer fails to cover the major aspects of the comparison between LoRA and full fine-tuning. It introduces incorrect details and omits important elements such as the purpose, efficiency, and practical applications of both techniques.

Clarity: Fair

While the answer is somewhat structured, the inaccuracies and misinformation detract from its clarity. The explanation is misleading and may confuse the reader.

Conciseness: Excellent

The answer is concise and free of unnecessary repetition or verbosity, though the content itself is flawed.

Weighted Score: 42.0

Response Latency: 0.00 seconds

Evaluation Latency: 8.65 seconds