Total Time Consumption in Generating Answers: 98.41 seconds

Total Time Consumption in Evaluating Answers: 161.61 seconds

LLM Evaluation Summary

Evaluation 1:

Evaluation:

Relevance: Excellent

The answer directly addresses the question by defining what a GPT pre-trained model is and explaining its core aspects, including its architecture and training methodology.

Faithfulness: Excellent

The answer is strictly supported by the given context, accurately describing a GPT pre-trained model, the transformer architecture, and the generative pretraining process.

Completeness: Excellent

The answer covers all major aspects of the question, including the model type, training process, and capabilities.

Clarity: Excellent

The answer is well-structured, clear, and easy to understand, with no ambiguity.

Conciseness: Excellent

The answer is concise and avoids unnecessary repetition or verbosity, providing all relevant information succinctly.

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) that utilizes the transformer

deep learning architecture. It is "pre-trained" through generative pretraining, an unsupervised learning process where the model is exposed to vast amounts of unlabeled data and learns to detect patterns, understand context, and generate coherent text. This pretraining equips the model with foundational knowledge, enabling it to handle a wide range of natural language processing tasks when fine-tuned or applied to specific applications.

Response Latency: 4.14 seconds

Evaluation Latency: 4.09 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 providing a detailed list of advantages specific to pre-trained models like BERT and GPT. It stays focused on the topic and does not deviate.

Faithfulness: Good

The answer is mostly supported by the provided context, which mentions the generalization capabilities and advantages of pre-trained models. However, some claims (like "state-of-the-art performance" and details about layer-specific advantages) are not explicitly mentioned in the context and may rely on external knowledge.

Completeness: Excellent

The answer thoroughly covers major aspects of the question, including generalization, efficiency, cost-effectiveness, customization, community support, and layer-specific advantages. No significant dimension of the advantages is left unexplored.

Clarity: Excellent

The answer is well-structured and easy to understand, with clear headings and concise explanations for each point. Technical terms are appropriately used and explained.

Conciseness: Excellent

The answer is free of unnecessary repetition or verbosity. Each point is succinctly presented, ensuring the response remains direct and focused.

Final Ratings

- Relevance: Excellent

- Faithfulness: Good

- Completeness: Excellent

- Clarity: Excellent

- Conciseness: Excellent

Weighted Score: 96.0

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

Answer 2: Pre-trained models, such as BERT and GPT, have several advantages compared to other types of pre-trained models, depending on the specific use case and design. These advantages include:

*1. **Generalization**: Pre-trained models, especially large-scale ones, are trained on vast datasets and can generalize well across a variety of tasks. They often outperform traditional machine learning models that are trained from scratch on smaller, task-specific datasets.*

*2. **Time and Resource Efficiency**: Since pre-trained models already have established weights and parameters, they save significant time and computational resources. Developers only need to fine-tune them for specific tasks, rather than training from the ground up.*

*3. **Access to State-of-the-Art Performance**: Models like BERT and GPT are built using cutting-edge techniques and architectures, which allow them to achieve state-of-the-art results in various NLP tasks such as text classification, sentiment analysis, and language generation.*

*4. **Flexibility in Customization**: Pre-trained models can be fine-tuned to adapt to specific domains or tasks, ensuring that they meet particular requirements while still leveraging the general knowledge they were trained on.*

*5. **Cost-Effective Development**: Using pre-trained models eliminates the need for extensive data collection and model training,*

which can be expensive. Instead, developers can focus resources on fine-tuning and integration.

*6. **Community and Ecosystem Support**: Pre-trained models like GPT and BERT have large communities and ecosystems, including pre-built libraries, tools, and documentation. This makes integration and troubleshooting more straightforward.*

*7. **Layer-Specific Advantages**: Some pre-trained models share weights for certain layers (e.g., initial layers), which are highly generalized and transferable across tasks. This can be particularly useful for tasks requiring foundational language understanding.*

In comparison with other pre-trained models, the specific advantages depend on the architecture (e.g., transformer-based models like GPT vs. other frameworks) and the scale of training data used. Large-scale models tend to have broader applicability and better performance due to their extensive training datasets and sophisticated architectures.

Response Latency: 10.69 seconds

Evaluation Latency: 8.60 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:

*Relevance: **Excellent***

The answer directly addresses the question by listing and explaining the key advantages of LoRA.

*Faithfulness: **Good***

The answer is mostly supported by the given context, but it introduces additional claims (e.g., scalability with GPT-3, competitive performance on benchmarks, interpretability) that are not explicitly mentioned in the context provided.

*Completeness: **Good***

The answer covers most major advantages of LoRA mentioned in the context but adds extra points (such as task performance and interpretability) that are not in the context, potentially omitting some details provided in the source text.

*Clarity: **Excellent***

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

*Conciseness: **Fair***

The answer includes verbose explanations and additional details that are not directly supported by the context, which could have been omitted for more concise delivery.

Weighted Score: 90.0

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

*Answer 3: The key advantages of **LoRA (Low-Rank Adaptation)** are:*

*1. **Efficiency in Training**:*

- LoRA significantly reduces the number of trainable parameters by injecting low-rank matrices (A and B) into the model.

This makes training more efficient and lowers the hardware barrier to entry by up to 3 times compared to full fine-tuning.

- Only the smaller low-rank matrices are optimized, eliminating the need to calculate gradients or maintain optimizer states for most parameters.

*2. **No Inference Latency**:*

*- LoRA?s design allows the trainable matrices to be merged with the frozen weights during deployment. This means LoRA introduces **no additional inference latency** when compared to a fully fine-tuned model.*

*3. **Scalability**:*

- LoRA exhibits excellent scalability, as demonstrated in experiments on large-scale models like GPT-3 (175B parameters). It performs competitively across a wide range of tasks, even with a substantial reduction in trainable parameters (up to $10{,}000\times$).

*4. **Storage Efficiency**:*

- A pre-trained model can be shared across tasks, while small LoRA modules (low-rank matrices A and B) are task-specific. This reduces storage requirements and allows efficient task-switching by simply replacing the LoRA modules.

*5. **Orthogonality to Other Methods**:*

- LoRA is orthogonal to many prior adaptation methods (e.g., prefix-tuning or prefix-embedding tuning) and can be combined with them to provide additional improvements in downstream task performance.

*6. **Lower Hardware Requirements**:*

- By focusing on low-rank updates, LoRA lowers the computational and memory requirements, enabling experiments to run on less expensive hardware like NVIDIA Tesla V100 GPUs.

*7. **Interpretability**:*

- The low-rank structure of LoRA provides better insight into how the update weights (ΔW) are correlated with pre-trained

weights. This makes the process of adaptation more interpretable compared to full fine-tuning.

*8. **Task Performance**:*

- LoRA achieves competitive or superior performance on various natural language understanding (NLU) and generation (NLG) tasks, as seen in evaluations on benchmarks like GLUE, WikiSQL, and SAMSum.

In essence, LoRA combines efficiency, scalability, and adaptability, making it a powerful tool for fine-tuning large-scale pre-trained models without compromising task performance.

Response Latency: 8.76 seconds

Evaluation Latency: 6.41 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) that utilizes the transformer deep learning architecture. It is "pre-trained" through generative pretraining, an unsupervised learning process where the model is exposed to vast amounts of unlabeled data and learns to detect patterns, understand context, and generate coherent text. This pretraining equips the model with foundational knowledge, enabling it to handle a wide range of natural language processing tasks when fine-tuned or applied to specific applications.
Q: What advantages does it have compared with other pre-trained models?
A: Pre-trained models, such as BERT and GPT, have several advantages compared to other types of pre-trained models, depending on the specific use case and design. These advantages include:

*1. **Generalization**: Pre-trained models, especially large-scale ones, are trained on vast datasets and can generalize well across a variety of tasks. They often outperform traditional machine learning models that are trained from scratch on smaller, task-specific datasets.*

*2. **Time and Resource Efficiency**: Since pre-trained models already have established weights and parameters, they save significant time and computational resources. Developers only need to fine-tune them for specific tasks, rather than training from*

the ground up.

*3. **Access to State-of-the-Art Performance**: Models like BERT and GPT are built using cutting-edge techniques and architectures, which allow them to achieve state-of-the-art results in various NLP tasks such as text classification, sentiment analysis, and language generation.*

*4. **Flexibility in Customization**: Pre-trained models can be fine-tuned to adapt to specific domains or tasks, ensuring that they meet particular requirements while still leveraging the general knowledge they were trained on.*

*5. **Cost-Effective Development**: Using pre-trained models eliminates the need for extensive data collection and model training, which can be expensive. Instead, developers can focus resources on fine-tuning and integration.*

*6. **Community and Ecosystem Support**: Pre-trained models like GPT and BERT have large communities and ecosystems, including pre-built libraries, tools, and documentation. This makes integration and troubleshooting more straightforward.*

*7. **Layer-Specific Advantages**: Some pre-trained models share weights for certain layers (e.g., initial layers), which are highly generalized and transferable across tasks. This can be particularly useful for tasks requiring foundational language understanding.*

In comparison with other pre-trained models, the specific advantages depend on the architecture (e.g., transformer-based models like GPT vs. other frameworks) and the scale of training data used. Large-scale models tend to have broader applicability and better performance due to their extensive training datasets and sophisticated architectures.

Context:

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 $_\mathrm{L oRA+PL}$.

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.

? 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

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.

Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance. See Section F.2 for more details on the plotted data points.

 # 6 R ELATED W ORKS

Given the empirical advantage of LoRA, we hope to further explain the properties of the low-rank adaptation learned from downstream tasks. Note that the low-rank structure not only lowers the hardware barrier to entry which allows us to run multiple experiments in parallel, but also gives better interpret ability of how the update weights are correlated with the pre-trained weights. We focus our study on GPT-3 175B, where we achieved the largest reduction of trainable parameters (up to $10{,}000\times)$ without adversely affecting task performances.

We evaluate the downstream task performance of LoRA on RoBERTa (Liu et al., 2019), De- BERTa (He et al., 2021), and GPT-2 (Radford et al., b), before scaling up to GPT-3 175B (Brown et al., 2020). Our experiments cover a wide range of tasks, from natural language understanding (NLU) to generation (NLG). Speci?cally, we evaluate on the GLUE (Wang et al., 2019) benchmark for RoBERTa and DeBERTa. We follow the setup of Li & Liang (2021) on GPT-2 for a direct com- parison and add WikiSQL (Zhong et al., 2017) (NL to SQL queries) and SAMSum (Gliwa et al., 2019) (conversation summarization) for large-scale experiments on GPT-3. See Appendix C for more details on the datasets we use. We use NVIDIA Tesla V100 for all experiments.

Current Question:

what's the key advantages of LoRA?

Answer:

Evaluation 4:

Evaluation of the Answer

***Relevance:** Excellent*

The answer directly addresses the question by discussing the practical benefits, advantages, and limitations of LoRA, aligning closely with the provided context.

***Faithfulness:** Excellent*

The answer is strictly supported by the given context, accurately reflecting the details about LoRA's efficiency, reduced storage, scalability, and limitations (e.g., batching across tasks and dynamic task selection).

***Completeness:** Excellent*

The answer covers all major aspects of the question, including a detailed breakdown of practical benefits, advantages, and limitations. It is thorough and includes nuanced points, such as LoRA's modularity and interpretability.

***Clarity:** Excellent*

The answer is well-structured with clear headings and bullet points, making it easy to follow and understand. Technical terms are explained adequately without compromising readability.

***Conciseness:** Good*

While the answer is detailed and informative, it could be slightly more concise by removing redundant points or rephrasing some sections to reduce verbosity.

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: ### Practical Benefits of LoRA:

*1. **Efficiency in Training**: LoRA significantly reduces the hardware requirements for fine-tuning large-scale models by optimizing only the injected low-rank matrices (A and B). This avoids the need to compute gradients or maintain optimizer states for the majority of parameters, reducing the computational load and making training more accessible.*

*2. **Reduced Storage Requirements**: LoRA allows the pre-trained model to remain frozen and requires only small task-specific low-rank matrices (A and B) for each new task. This modular design minimizes storage overhead and enables efficient task-switching by replacing these matrices, rather than retraining or storing multiple versions of the model.*

*3. **No Additional Latency**: When deployed, the trainable matrices can be merged with the frozen weights, ensuring that LoRA introduces no inference latency compared to a fully fine-tuned model.*

*4. **Scalability**: LoRA demonstrates better scalability across tasks, as shown in experiments like WikiSQL and MNLI-matched. It achieves competitive performance even with a very small rank (r), making it highly efficient.*

*5. **Orthogonality with Other Methods**: LoRA can be combined with other efficient adaptation methods, such as prefix-tuning, to potentially achieve complementary improvements. For example, LoRA+PE (Prefix Embedding) outperforms both standalone LoRA and prefix-tuning in certain tasks, showing its ability to integrate synergistically with prior techniques.*

*6. **Improved Interpretability**: By focusing on low-rank weight updates, LoRA makes it more tractable to study and understand*

how features learned during pre-training are transformed for downstream tasks.

*7. **Parallel Experimentation**: The low-rank structure enables running multiple experiments in parallel, thanks to reduced resource requirements, thereby accelerating research and development.*

*8. **Cost-Effectiveness**: LoRA lowers the hardware barrier to entry, allowing researchers and organizations with limited resources to adapt large models like GPT-3 175B without significant infrastructure investments.*

Advantages of LoRA:

*1. **Storage Efficiency**: LoRA avoids the need to store full fine-tuned models for different tasks, instead requiring only the lightweight low-rank matrices, drastically reducing storage needs.*

*2. **Task Switching**: The modular architecture allows quick switching between tasks by simply replacing the matrices A and B, further enhancing adaptability and flexibility in multi-task scenarios.*

*3. **High Performance with Minimal Rank**: LoRA achieves competitive performance across tasks even with a very small rank (r), highlighting its ability to efficiently leverage the intrinsic low-rank structure of weight updates.*

*4. **Hardware Accessibility**: By reducing the number of trainable parameters up to 10,000x (as demonstrated with GPT-3 175B), LoRA makes fine-tuning feasible on less powerful hardware, opening up large-scale model adaptation to a broader audience.*

*5. **Flexibility Across Layers**: While LoRA is primarily applied to certain weights in transformer language models, its principles can be extended to any dense layers in deep learning models.*

Limitations of LoRA:

*1. **Batching Across Tasks**: LoRA makes it challenging to batch inputs for different tasks with different A and B matrices in a single forward pass if the weights are merged into the frozen model to eliminate latency.*

*2. **Sensitivity to Learning Rate**: In some combinations (e.g., LoRA+PL), optimization can become more difficult due to sensitivity to hyperparameters like learning rate, which may hinder task performance.*

*3. **Dynamic Task Selection**: While dynamic selection of LoRA modules is possible without merging weights, this approach may introduce latency, making it less suitable for applications requiring low-latency inference.*

*4. **Heuristic-Based Weight Selection**: The choice of which weight matrices to apply LoRA to is currently heuristic-driven, leaving room for more principled methods to improve efficiency and effectiveness.*

*5. **Intrinsic Rank Exploration**: Although LoRA exploits the low intrinsic rank of weight updates, the mechanism behind this phenomenon is not fully understood, leaving opportunities for further theoretical investigation.*

Summary:

LoRA is a practical, efficient, and scalable adaptation method for fine-tuning large pre-trained models. Its low-rank design reduces storage, computation, and latency requirements while maintaining high task performance. LoRA's modularity, orthogonality to other methods, and interpretability further enhance its versatility. However, it has limitations around batching, dynamic task handling, hyperparameter sensitivity, and heuristic-based weight selection, which present areas for future research and improvement.

Response Latency: 18.12 seconds

Evaluation Latency: 9.67 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) that utilizes the transformer deep learning architecture. It is "pre-trained" through generative pretraining, an unsupervised learning process where the model is exposed to vast amounts of unlabeled data and learns to detect patterns, understand context, and generate coherent text. This pretraining equips the model with foundational knowledge, enabling it to handle a wide range of natural language processing tasks when fine-tuned or applied to specific applications.

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

A: Pre-trained models, such as BERT and GPT, have several advantages compared to other types of pre-trained models, depending on the specific use case and design. These advantages include:

*1. **Generalization**: Pre-trained models, especially large-scale ones, are trained on vast datasets and can generalize well across a variety of tasks. They often outperform traditional machine learning models that are trained from scratch on smaller, task-specific datasets.*

*2. **Time and Resource Efficiency**: Since pre-trained models already have established weights and parameters, they save significant time and computational resources. Developers only need to fine-tune them for specific tasks, rather than training from the ground up.*

*3. **Access to State-of-the-Art Performance**: Models like BERT and GPT are built using cutting-edge techniques and architectures, which allow them to achieve state-of-the-art results in various NLP tasks such as text classification, sentiment analysis, and language generation.*

*4. **Flexibility in Customization**: Pre-trained models can be fine-tuned to adapt to specific domains or tasks, ensuring that they meet particular requirements while still leveraging the general knowledge they were trained on.*

*5. **Cost-Effective Development**: Using pre-trained models eliminates the need for extensive data collection and model training, which can be expensive. Instead, developers can focus resources on fine-tuning and integration.*

*6. **Community and Ecosystem Support**: Pre-trained models like GPT and BERT have large communities and ecosystems,*

including pre-built libraries, tools, and documentation. This makes integration and troubleshooting more straightforward.

*7. **Layer-Specific Advantages**: Some pre-trained models share weights for certain layers (e.g., initial layers), which are highly generalized and transferable across tasks. This can be particularly useful for tasks requiring foundational language understanding.*

In comparison with other pre-trained models, the specific advantages depend on the architecture (e.g., transformer-based models like GPT vs. other frameworks) and the scale of training data used. Large-scale models tend to have broader applicability and better performance due to their extensive training datasets and sophisticated architectures.

Context:

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.

? 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

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

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 $_\mathrm{L oRA+PL}$.

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.

Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance. See Section F.2 for more details on the plotted data points.

 # 6 R ELATED W ORKS

We take inspiration from Li et al. (2018a); Aghajanyan et al. (2020) which show that the learned over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the change in weights during model adaptation also has a low ?intrinsic rank?, leading to our proposed Lo w- R ank A daptation (LoRA) approach. LoRA allows us to train some dense layers in a neural network indirectly by optimizing rank decomposition matrices of the dense layers? change during adaptation instead, while keeping the pre-trained weights frozen, as shown in Figure 1. Using GPT-3 175B as an example, we show that a very low rank (i.e., r in Figure 1 can be one or two) suf?ces even when the full rank (i.e., d) is as high as 12,288, making LoRA both storage- and compute-ef?cient.

Given the empirical advantage of LoRA, we hope to further explain the properties of the low-rank adaptation learned from downstream tasks. Note that the low-rank structure not only lowers the hardware barrier to entry which allows us to run multiple experiments in parallel, but also gives better interpret ability of how the update weights are correlated with the pre-trained weights. We focus our study on GPT-3 175B, where we achieved the largest reduction of trainable parameters (up to $10{,}000\times)$ without adversely affecting task performances.

Current Question:

Lora's practical benefits, advantages and limitations.

Answer:

Evaluation 5:

Evaluation of the Answer:

Relevance: Excellent

The answer directly addresses the question by specifying which weight matrices in the Transformer architecture should have LoRA applied (i.e., the query (W_q) and value (W_v) projection matrices). It clearly connects the response to the self-attention module, which was mentioned in the context.

Faithfulness: Excellent

The answer is strictly supported by the given context, referencing empirical results and practical considerations discussed in Section 4.2 and Table 5. It aligns with the context's focus on adapting weight matrices in the self-attention module under a parameter budget.

Completeness: Excellent

The answer covers all major aspects of the question, explaining the reasoning behind adapting W_q and W_v, including simplicity, efficiency, empirical performance, and resource savings. It also briefly mentions the possibility of adapting other matrices but clarifies that these are typically frozen.

Clarity: Excellent

The answer is well-structured and easy to understand. It breaks down the reasoning into clear points and uses technical terms appropriately while maintaining readability.

Conciseness: Excellent

The answer is concise and free of unnecessary repetition or verbosity. It provides sufficient detail without overloading the response with irrelevant information.

Final Evaluation:

*This answer is **Excellent** across all criteria.*

Weighted Score: 100.0

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

*Answer 5: In the Transformer architecture, LoRA can theoretically be applied to any subset of weight matrices in order to reduce the number of trainable parameters. However, based on empirical studies and practical considerations, it is recommended to apply LoRA specifically to the **query (W_q)** and **value (W_v)** projection matrices in the self-attention module. This choice is motivated by the following factors:*

*1. **Simplicity**: Adapting only the attention weights (W_q and W_v) simplifies the implementation and avoids the need to train additional components like the MLP modules, LayerNorm layers, or biases.*

*2. **Parameter Efficiency**: Focusing on these attention weight matrices significantly reduces the number of trainable parameters while still achieving competitive downstream task performance.*

*3. **Empirical Performance**: Experiments show that LoRA performs competitively even with a very small rank (r) when applied to W_q and W_v. This suggests that the update matrices (ΔW) for these specific layers have a low "intrinsic rank," making them ideal candidates for low-rank adaptations.*

*4. **Memory and Resource Savings**: Applying LoRA to only W_q and W_v reduces memory usage and computation overhead during training. For instance, on large models like GPT-3 175B, this approach reduces VRAM consumption significantly and improves training throughput.*

While adapting other weight matrices (e.g., W_k, W_o, or MLP module weights) may also be possible, these were mostly frozen in the experiments for simplicity and efficiency. Future work may explore adapting additional components, but current best

practices focus on W_q and W_v for optimal results.

Response Latency: 8.82 seconds

Evaluation Latency: 8.72 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) that utilizes the transformer deep learning architecture. It is "pre-trained" through generative pretraining, an unsupervised learning process where the model is exposed to vast amounts of unlabeled data and learns to detect patterns, understand context, and generate coherent text. This pretraining equips the model with foundational knowledge, enabling it to handle a wide range of natural language processing tasks when fine-tuned or applied to specific applications.

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

A: Pre-trained models, such as BERT and GPT, have several advantages compared to other types of pre-trained models, depending on the specific use case and design. These advantages include:

*1. **Generalization**: Pre-trained models, especially large-scale ones, are trained on vast datasets and can generalize well across a variety of tasks. They often outperform traditional machine learning models that are trained from scratch on smaller, task-specific datasets.*

*2. **Time and Resource Efficiency**: Since pre-trained models already have established weights and parameters, they save significant time and computational resources. Developers only need to fine-tune them for specific tasks, rather than training from the ground up.*

*3. **Access to State-of-the-Art Performance**: Models like BERT and GPT are built using cutting-edge techniques and architectures, which allow them to achieve state-of-the-art results in various NLP tasks such as text classification, sentiment analysis, and language generation.*

*4. **Flexibility in Customization**: Pre-trained models can be fine-tuned to adapt to specific domains or tasks, ensuring that they meet particular requirements while still leveraging the general knowledge they were trained on.*

*5. **Cost-Effective Development**: Using pre-trained models eliminates the need for extensive data collection and model training, which can be expensive. Instead, developers can focus resources on fine-tuning and integration.*

*6. **Community and Ecosystem Support**: Pre-trained models like GPT and BERT have large communities and ecosystems, including pre-built libraries, tools, and documentation. This makes integration and troubleshooting more straightforward.*

*7. **Layer-Specific Advantages**: Some pre-trained models share weights for certain layers (e.g., initial layers), which are highly generalized and transferable across tasks. This can be particularly useful for tasks requiring foundational language understanding.*

In comparison with other pre-trained models, the specific advantages depend on the architecture (e.g., transformer-based models like GPT vs. other frameworks) and the scale of training data used. Large-scale models tend to have broader applicability and better performance due to their extensive training datasets and sophisticated architectures.

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.

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.

 # 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

We perform a sequence of empirical studies to answer the following questions: 1) Given a parameter budget constraint, which subset of weight matrices in a pre-trained Transformer should we adapt to maximize downstream performance? 2) Is the ?optimal? adaptation matrix ΔW really rank- de?cient ? If so, what is a good rank to use in practice? 3) What is the connection between ΔW and $W?$ Does ΔW highly correlate with W ? How large is ΔW comparing to $W?$ We take inspiration from Li et al. (2018a); Aghajanyan et al. (2020) which show that the learned over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the change in weights during model adaptation also has a low ?intrinsic rank?, leading to our proposed Lo w- R ank A daptation (LoRA) approach. LoRA allows us to train some dense layers in a neural network indirectly by optimizing rank decomposition matrices of the dense layers? change during adaptation instead, while keeping the pre-trained weights frozen, as shown in Figure 1. Using GPT-3 175B as an example, we show that a very low rank (i.e., r in Figure 1 can be one or two) suf?ces even when the full rank (i.e., d) is as high as 12,288, making LoRA both storage- and compute-ef?cient.

? 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.

LoRA adds trainable pairs of rank decomposition matrices in parallel to existing weight matrices. As mentioned in Section 4.2, we only apply LoRA to W_{q} and W_{v} in most experiments for simplicity. The number of trainable parameters is determined by the rank r and the shape of the original weights: $\vert\Theta\vert=2\times\hat{L}_{L o R A}\times d_{m o d e l}\times r$, where $\hat{L}_{L o R A}$ is the number of weight matrices we apply LoRA to.

guarantees that we do not introduce any additional latency during inference compared to a ?ne-tuned

model by construction.

4.2

APPLYING LORA TO TRANSFORMER

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 (Wq, Wk, Wv, Wo) and two in the MLP module. We treat Wq (or Wk, Wv) as a single matrix of dimension dmodel × dmodel, 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.

Practical Bene?ts and Limitations.

The most signi?cant bene?t comes from the reduction in memory and storage usage. For a large Transformer trained with Adam, we reduce that VRAM usage by up to 2/3 if r ?dmodel as we do not need to store the optimizer states for the frozen parameters. On GPT-3 175B, we reduce the VRAM consumption during training from 1.2TB to 350GB. With r = 4 and only the query and value projection matrices being adapted, the checkpoint size is reduced by roughly 10,000× (from 350GB to 35MB)4. This allows us to train with signi?-cantly fewer GPUs and avoid I/O bottlenecks. Another bene?t is that we can switch between tasks while deployed at a much lower cost by only swapping the LoRA weights as opposed to all the parameters. This allows for the creation of many customized models that can be swapped in and out on the ?y on machines that store the pre-trained weights in VRAM. We also observe a 25% speedup during training on GPT-3 175B compared to full ?ne-tuning5 as we do not need to calculate the gradient for the vast majority of the parameters.

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

EMPIRICAL EXPERIMENTS

We evaluate the downstream task performance of LoRA on RoBERTa (Liu et al., 2019), De-BERTa (He et al., 2021), and GPT-2 (Radford et al., b), before scaling up to GPT-3 175B (Brown et al., 2020). Our experiments cover a wide range of tasks, from natural language understanding (NLU) to generation (NLG). Speci?cally, we evaluate on the GLUE (Wang et al., 2019) benchmark

for RoBERTa and DeBERTa. We follow the setup of Li & Liang (2021) on GPT-2 for a direct comparison and add WikiSQL (Zhong et al., 2017) (NL to SQL queries) and SAMSum (Gliwa et al., 2019) (conversation summarization) for large-scale experiments on GPT-3. See Appendix C for more details on the datasets we use. We use NVIDIA Tesla V100 for all experiments.

5.1

BASELINES

To compare with other baselines broadly, we replicate the setups used by prior work and reuse their reported numbers whenever possible. This, however, means that some baselines might only appear in certain experiments.

Fine-Tuning (FT) is a common approach for adaptation. During ?ne-tuning, the model is initialized to the pre-trained weights and biases, and all model parameters undergo gradient updates.A simple variant is to update only some layers while freezing others. We include one such baseline reported in prior work (Li & Liang, 2021) on GPT-2, which adapts just the last two layers (FTTop2).

*4We still need the 350GB model during deployment; however, storing 100 adapted models only requires 350GB + 35MB * 100 ?354GB as opposed to 100 * 350GB ?35TB.*

5For GPT-3 175B, the training throughput for full ?ne-tuning is 32.5 tokens/s per V100 GPU; with the same number of weight shards for model parallelism, the throughput is 43.1 tokens/s per V100 GPU for LoRA.

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

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?" and provides a detailed explanation of its advantages, motivations, and applications, aligning with the context provided.

***Faithfulness**: Excellent*

The answer is strictly supported by the given context. It accurately reflects the advantages of self-attention mentioned in the text, such as efficiency, modeling dependencies regardless of distance, flexibility, interpretability, and simplicity.

***Completeness**: Excellent*

The answer covers all major aspects of the question, including the motivations for using self-attention, its benefits over recurrent and convolutional layers, and its impact on model interpretability and efficiency.

***Clarity**: Excellent*

The answer is well-structured, logically organized, and easy to understand. Each point is clearly explained without ambiguity, making it accessible to readers.

***Conciseness**: Excellent*

The answer is concise and avoids unnecessary repetition or verbosity. It delivers comprehensive information without overloading the response with extraneous details.

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, also known as intra-attention, is a mechanism that allows a model to relate different positions within a single sequence to compute a representation of that sequence. It has several advantages and motivations for its use, especially when compared to recurrent and convolutional layers in sequence modeling and transduction tasks. Here's why self-attention is particularly useful:

*1. **Modeling Dependencies Regardless of Distance**: Self-attention mechanisms can model dependencies between all positions in a sequence simultaneously, without regard to their distance in the input or output sequences. This is a significant advantage over recurrent networks, which process input sequentially and require many steps to capture long-range dependencies.*

*2. **Efficiency**: Unlike recurrent layers, which process sequences step-by-step, self-attention can compute representations in parallel for all positions in the sequence. This parallelization leads to faster computation and makes self-attention models more efficient, especially for long sequences.*

*3. **Flexibility in Sequence Lengths**: By using positional embeddings (such as sinusoidal embeddings), self-attention models can potentially extrapolate to sequence lengths longer than those encountered during training. This makes them more adaptable to varying input lengths compared to recurrent models.*

*4. **Interpretable Models**: Self-attention mechanisms can yield more interpretable models by allowing inspection of attention distributions. Individual attention heads often learn to perform specific tasks and exhibit behavior related to the syntactic and semantic structure of sentences, making it easier to understand how the model processes information.*

*5. **Simplicity and Effectiveness**: Self-attention eliminates the need for sequence-aligned recurrent or convolutional layers, simplifying the architecture while maintaining or improving performance. The Transformer model, for example, relies entirely on self-attention for computing input and output representations and achieves state-of-the-art results across various tasks.*

In summary, self-attention is a powerful mechanism that addresses some of the limitations of recurrent and convolutional layers, making it an integral part of modern sequence transduction models. Its ability to model long-range dependencies efficiently, handle variable-length sequences, and provide interpretability makes it a compelling choice for tasks such as reading comprehension, summarization, and more.

Response Latency: 7.89 seconds

Evaluation Latency: 5.65 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].

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

Attention mechanisms have become an integral part of compelling sequence modeling and transduc- tion models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences [2 , 16]. In all but a few cases [22], however, such attention mechanisms are used in conjunction with a recurrent network.

We are excited about the future of attention-based models and plan to apply them to other tasks. We plan to extend the Transformer to problems involving input and output modalities other than text and to investigate local, restricted attention mechanisms to ef?ciently handle large inputs and outputs such as images, audio and video. Making generation less sequential is another research goals of ours.

In this section we compare various aspects of self-attention layers to the recurrent and convolu- tional layers commonly used for mapping one variable-length sequence of symbol representations $(x_{1},...,x_{n})$ to another sequence of equal length $\left(z_{1},...,z_{n}\right)$, with $x_{i},z_{i}\in\dot{\mathbb{R}^{d}}$, such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention we consider three desiderata.

To the best of our knowledge, however, the Transformer is the ?rst transduction model relying entirely on self-attention to compute representations of its input and output without using sequence- aligned RNNs or convolution. In the following sections, we will describe the Transformer, motivate self-attention and discuss its advantages over models such as [14, 15] and [8].

 # 3 Model Architecture

 # 7 Conclusion

In this work, we presented the Transformer, the ?rst sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

 # Attention Is All You Need

Illia Polosukhin ?? illia.polosukhin@gmail.com

 # Abstract

Instead of performing a single attention function with d_{model} -dimensional keys, values and queries, we found it bene?cial to linearly project the queries, keys and values h times with different, learned linear projections to d_{k} , d_{k} and d_{v} dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_{v} -dimensional output values. These are concatenated and once again projected, resulting in the ?nal values, as depicted in Figure 2.

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 providing the score itself.

***Faithfulness:** Excellent*

The answer is strictly supported by the given context, specifically referencing Table 1 where the average score of BERT LARGE is listed as 82.1.

***Completeness:** Excellent*

The answer covers all major aspects of the question by naming the model and providing the score without omitting any relevant details.

***Clarity:** Excellent*

The answer is well-structured and easy to understand, clearly stating both the model and its score.

***Conciseness:** Excellent*

The answer is concise and avoids unnecessary repetition or verbosity.

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: Among the five models in Table 1, **BERT LARGE** has the highest average score, which is **82.1**.*

Response Latency: 2.99 seconds

Evaluation Latency: 7.05 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 82.1

NaN 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

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

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 82.1

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 NaN

System NaN NaN Dev NaN Test NaN 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 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] Row 40: Top Leaderboard Systems (Dec 10th, 2018)

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 22: #2 Ensemble - QANet

NaN

NaN

-

-

84.5

90.5

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 33: Table 2: SQuAD 1.1 results. The BERT ensemble i...

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 1: System

NaN

NaN

MNLI-(m/mm)

QQP

QNLI

SST-2

CoLA

STS-B

MRPC

RTE

Average

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 about LoRA fine-tuning by explaining its mechanism, advantages, and limitations.

***Faithfulness:** Excellent*

The answer is strictly supported by the provided context, accurately describing LoRA's functionality, benefits, and limitations as outlined in the text.

***Completeness:** Excellent*

The answer comprehensively covers all major aspects of LoRA fine-tuning, including its workings, advantages, limitations, and implications for deployment and efficiency.

***Clarity:** Excellent*

The answer is well-structured, logically organized, and easy to understand, with clear explanations of technical concepts like low-rank matrices and their role in fine-tuning.

***Conciseness:** Excellent*

The answer avoids unnecessary repetition or verbosity, delivering all relevant information in a concise manner.

Final Rating:

*The answer is **Excellent** across all criteria.*

Weighted Score: 100.0

Question 8: what's LoRA fine-tuning?

Answer 8: LoRA (Low-Rank Adaptation) fine-tuning is a method for adapting pre-trained models to downstream tasks efficiently by injecting low-rank matrices into the existing weight matrices of the model, rather than fine-tuning all parameters. This approach significantly reduces the number of trainable parameters required for adaptation, making it more computationally efficient and lowering storage requirements.

How LoRA Works:

*1. **Injecting Low-Rank Matrices**: LoRA introduces two small, trainable matrices, \(A\) and \(B\), into the weight matrices of the model during fine-tuning. These matrices are designed to capture task-specific adaptations without modifying the original pre-trained weights.*

*2. **Optimization**: Instead of updating all the parameters of the model during training, LoRA only updates the injected low-rank matrices. This drastically reduces the number of parameters that need gradient updates, making training faster and requiring less memory.*

*3. **Expressiveness**: By adjusting the rank \(r\) of the injected matrices, LoRA can approximate the expressive power of full fine-tuning. In the extreme case, when \(r\) equals the rank of the original weight matrices, LoRA converges to the behavior of full fine-tuning.*

*4. **Deployment**: After training, the low-rank matrices \(A\) and \(B\) can be merged with the original weight matrices for deployment. This means LoRA does not introduce any additional inference latency compared to a fully fine-tuned model.*

Advantages of LoRA Fine-Tuning:

*- **Efficiency**: LoRA significantly reduces the hardware requirements for fine-tuning, making it suitable for resource-constrained environments.*

*- **Modularity**: A pre-trained model can be shared and reused for multiple tasks by swapping out small LoRA modules, reducing storage and overhead for task switching.*

*- **Orthogonality**: LoRA is compatible with and can be combined with other adaptation methods, such as prefix-tuning, to*

enhance performance further.

*- **Low-Rank Sufficiency**: Empirical results suggest that a low-rank adaptation matrix is often sufficient to achieve competitive performance, indicating that the task-specific updates have a low "intrinsic rank."*

Limitations:

*- **Batching Complexity**: If weights are merged for deployment to eliminate inference latency, it becomes difficult to batch inputs for different tasks in a single forward pass. However, this can be addressed by keeping the weights unmerged for scenarios where latency is not critical.*

*- **Sensitivity to Hyperparameters**: Certain combinations with other methods, such as prefix-layer tuning, can make optimization more challenging due to sensitivity to hyperparameters like learning rates.*

In summary, LoRA fine-tuning is an efficient and modular approach to adapting pre-trained models for downstream tasks, leveraging low-rank matrices to reduce computational complexity while maintaining competitive performance.

Response Latency: 13.79 seconds

Evaluation Latency: 4.64 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 $_\mathrm{L oRA+PL}$.

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 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.

LoRA possesses several key advantages.

As shown in Table 4, LoRA matches or exceeds the ?ne-tuning baseline on all three datasets. Note that not all methods bene?t monotonically from having more trainable parameters, as shown in Fig- ure 2. We observe a signi?cant performance drop when we use more than 256 special tokens for pre?x-embedding tuning or more than 32 special tokens for pre?x-layer tuning. This corroborates similar observations in Li & Liang (2021). While a thorough investigation into this phenomenon is out-of-scope for this work, we suspect that having more special tokens causes the input distri- bution to shift further away from the pre-training data distribution. Separately, we investigate the performance of different adaptation approaches in the low-data regime in Section F.3.

 # E C OMBINING L O RA WITH P REFIX T UNING

LoRA can be naturally combined with existing pre?x-based approaches. In this section, we evaluate two combinations of LoRA and variants of pre?x-tuning on WikiSQL and MNLI.

LoRA+Pre?xEmbed $(\mathbf{LO}\mathbf{AA}+\mathbf{PE})$ combines LoRA with pre?x-embedding tuning, where we insert $l_{p}+l_{i}$ special tokens whose embeddings are treated as trainable parameters. For more on pre?x- embedding tuning, see Section 5.1.

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 11: The hyperparameters for GPT-2 LoRA on E2E, WebNLG and DART.

](images/519dcc915a5487a9e8068229b3dcacd8a62b30fde06ed03524298006e58c7cb1.jpg)

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

Current Question:

what's LoRA fine-tuning?

Answer:

Evaluation 9:

Evaluation of the Answer

Relevance: Excellent

The answer directly compares LoRA and full fine-tuning, addressing key aspects such as parameter efficiency, computational efficiency, inference latency, flexibility, and performance?all of which are relevant to the question.

Faithfulness: Excellent

The answer is strictly supported by the provided context, referencing specific details like the storage requirements for $|\Delta\Phi|$, the computational efficiency of LoRA's low-rank updates, and the optimization challenges associated with full fine-tuning.

Completeness: Excellent

The answer covers all major aspects of the question comprehensively, including comparisons of efficiency, latency, flexibility, and performance. It provides a holistic analysis without missing any critical points.

Clarity: Excellent

The explanation is well-structured, using clear headings to organize the comparison. Complex concepts are explained in an accessible manner, making the content easy to understand.

Conciseness: Excellent

The answer avoids unnecessary repetition or verbosity. It provides concise descriptions and focuses on the essential points of comparison without digressing.

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: The comparison between **LoRA** (Low-Rank Adaptation) and **full fine-tuning** involves several key aspects:*

*### 1. **Parameter Efficiency**:*

*- **Full Fine-Tuning**: Updates all parameters of the model (Φ_{0}), meaning $|\Delta\Phi| = |\Phi_{0}|$. For large models like GPT-3 (175 billion parameters), this requires storing a separate set of updated parameters for each task, making it storage-intensive and computationally expensive.*

*- **LoRA**: Introduces task-specific parameter increments $\Delta\Phi$ encoded by a much smaller set of parameters Θ, where $|\Theta| \ll |\Phi_{0}|$. This significantly reduces the storage and deployment costs, as only the low-rank updates Θ need to be stored.*

*### 2. **Computational Efficiency**:*

*- **Full Fine-Tuning**: Requires backpropagation through the entire model, which involves updating all layers during training. This can be computationally expensive, especially for large-scale models.*

*- **LoRA**: Focuses on specific components of the model, such as the weights (W_q, W_k, W_v, W_c) within certain layers, reducing the computational overhead while still achieving effective adaptation.*

*### 3. **Inference Latency**:*

*- **Full Fine-Tuning**: Since all parameters are updated, inference latency is equivalent to the pre-trained model, but deploying multiple fine-tuned models for different tasks increases storage and memory requirements.*

*- **LoRA**: Introduces minimal inference latency because the low-rank updates are lightweight. Adapter layers integrate seamlessly into the model architecture without significant performance drops.*

*### 4. **Flexibility**:*

*- **Full Fine-Tuning**: Highly flexible, as it allows complete modification of the model for each task. However, this flexibility comes at the cost of increased storage and computational demands.*

*- **LoRA**: Provides a balance between flexibility and efficiency by focusing on specific layers or components of the model. It is particularly useful for scenarios where storage and computational resources are limited.*

*### 5. **Performance**:*

*- **Full Fine-Tuning**: Typically achieves the best performance since all parameters are optimized for the downstream task, but it may not always justify the cost for marginal improvements.*

*- **LoRA**: While it may not always match the performance of full fine-tuning, it achieves competitive results with significantly fewer parameters and lower resource requirements.*

Summary:

*LoRA is a **parameter-efficient alternative** to full fine-tuning, particularly suited for large language models like GPT-3. It addresses the challenges of storage, computational overhead, and deployment scalability while maintaining strong performance on downstream tasks. Full fine-tuning, on the other hand, remains the gold standard for maximizing task-specific performance but is far less practical for large-scale models in resource-constrained environments.*

Response Latency: 11.51 seconds

Evaluation Latency: 6.85 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

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

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

Fine-Tuning (FT) is a common approach for adaptation. During ?ne-tuning, the model is initialized to the pre-trained weights and biases, and all model parameters undergo gradient updates.A simple variant is to update only some layers while freezing others. We include one such baseline reported in prior work (Li & Liang, 2021) on GPT-2, which adapts just the last two layers $(\mathbf{FT}^{\mathbf{Top2}})$).

We illustrate our re para me tri z ation in Figure 1. We use a random Gaussian initialization for A and zero for B , so $\Delta W=B A$ is zero at the beginning of training. We then scale $\Delta W x$ by $\textstyle{\frac{\alpha}{r}}$, where α is a constant in r . When optimizing with Adam, tuning α is roughly the same as tuning the learning rate if we scale the initialization appropriately. As a result, we simply set α to the ?rst r we try and do not tune it. This scaling helps to reduce the need to retune hyperparameters when we vary r (Yang & Hu, 2021).

In this paper, we adopt a more parameter-ef?cient approach, where the task-speci?c parameter increment $\Delta\Phi\,=\,\Delta\Phi(\Theta)$ is further encoded by a much smaller-sized set of parameters Θ with $|\Theta|\ll|\Phi_{0}|$. The task of ?nding $\Delta\Phi$ thus becomes optimizing over Θ :

We used the Adam optimizer [17] with $\beta_{1}=0.9$, $\beta_{2}=0.98$ and $\epsilon=10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$

l r a t e=d_{\mathrm{model}}^{-0.5}\cdot\mathrm{min}(s t e p_n u m^{-0.5},s t e p_n u m\cdot w a r m u p_s t e p s^{-1.5})

$$

![We turn our attention to the effect of rank r on model performance. We adapt $\{W_{q},W_{v}\}$, $\{W_{q},W_{k},W_{v},W_{c}\}$, and just W_{q} for a comparison.

](images/7fc87f8ebf73859ce1dc408949cc12426ff1745ee73f125e7202b7cd582e6f37.jpg)

During full ?ne-tuning, the model is initialized to pre-trained weights Φ_{0} and updated to $\Phi_{0}+\Delta\Phi$ by repeatedly following the gradient to maximize the conditional language modeling objective:

$$

\operatorname{max}_{\Phi}\sum_{(x,y)\in\mathcal{Z}}\sum_{t=1}^{|y|}\log\left(P_{\Phi}(y_{t}|x,y_{<t})\right)*

$$

Current Question:

compare it to full fine-tuning

Answer:

Evaluation 10:

Evaluation of the Answer:

Relevance:** **Fair

The question specifically asks about Table 4 and the highest Dev and Test scores. The provided answer mentions various scores and models but does not definitively identify if Table 4 directly lists these scores or models. The connection to Table 4 is unclear, making the response partially relevant.

Faithfulness:** **Fair

The answer extrapolates information from other tables without clear evidence that Table 4 contains the mentioned data. It is partially supported by the context but lacks direct confirmation from Table 4.

Completeness:** **Fair

The answer does not fully address the question because it fails to provide a definitive identification of the highest Dev and Test scores from Table 4 due to the lack of clarity in the context. It partially discusses related scores and models, but the information is incomplete.

Clarity:** **Good

The answer is structured well and easy to understand. It clearly explains the possible alignment between the scores and Table 4, even though the connection is ambiguous.

Conciseness:** **Excellent

The answer avoids unnecessary repetition and verbosity, conveying its points succinctly.

Final Ratings:

*- **Relevance:** Fair*

*- **Faithfulness:** Fair*

*- **Completeness:** Fair*

*- **Clarity:** Good*

*- **Conciseness:** Excellent*

Weighted Score: 66.0

Question 10: According to Table 4, which model has the highest Dev and Test scores? What are the highest scores of Dev and Test respectively?

Answer 10: Based on the provided context, Table 4 pertains to SWAG Dev and Test accuracies. However, the context does not provide the specific scores or models listed in Table 4 directly. Instead, it references various models and their scores across different tasks and datasets.

*From the information available, the highest scores for Dev and Test in other tables are associated with **BERT LARGE (Ens.+TriviaQA)**:*

*- **Dev (EM/F1):** 86.2 / 92.2*

*- **Test (EM/F1):** 87.4 / 93.2*

*If these scores align with Table 4, then **BERT LARGE (Ens.+TriviaQA)** would be the model with the highest Dev and Test scores. However, if Table 4 contains other data not explicitly provided in this context, the answer might differ.*

Response Latency: 11.69 seconds

Evaluation Latency: 7.13 seconds

[Prompt used for Q10]
You are a helpful assistant. Use the following history and context to answer.

History:

Context:

[Sheet: Sheet2] Row 10: Table 4: SWAG Dev and Test accuracies. Human pe...

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 40: 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 82.1

NaN 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 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 NaN

System NaN NaN Dev NaN Test NaN 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 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 82.1

NaN NaN 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 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 NaN NaN NaN

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

NaN NaN NaN EM F1 EM F1 NaN NaN NaN NaN NaN NaN

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

#1 Single-MIR-MRC(F-Net) NaN NaN - - 74.8 78 NaN 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] Row 25: R.M.Reader(Ensemble)

NaN

NaN

81.2

87.9

82.3

88.5

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] Row 22: #2 Ensemble - QANet

NaN

NaN

-

-

84.5

90.5

NaN

NaN

NaN

NaN

NaN

[Sheet: Sheet1] Row 31: BERT LARGE(Ens.+TriviaQA)

NaN

NaN

86.2

92.2

87.4

93.2

NaN

NaN

NaN

NaN

NaN

Current Question:

According to Table 4, which model has the highest Dev and Test scores? What are the highest scores of Dev and Test respectively?

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:

Relevance: Worse
The answer does not correctly define a GPT pre-trained model. GPT models are generative pre-trained transformers used for various natural language processing tasks, and the description provided is unrelated to GPT pretraining.

Faithfulness: Worse
The answer is not supported by the provided context. The context describes pretraining as initializing the model with weights and biases learned from a large dataset, which is not reflected in the answer.

Completeness: Worse
The answer omits key aspects of what a GPT pre-trained model is, such as its reliance on large-scale text data, use of transformer architecture, and its generative capabilities. It instead introduces irrelevant and incorrect information.

Clarity: Bad
The structure and phrasing are misleading because the definition provided does not align with the question or the context, causing confusion.

Conciseness: Good

The answer is concise in its phrasing, but this is undermined by the irrelevance and lack of accuracy in the content.

Weighted Score: 28.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.35 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 question about advantages compared with other pre-trained models. Instead, it provides incorrect and irrelevant information about GPT's design and capabilities, such as claiming it is bidirectional and designed for image classification.

Faithfulness: **Worse**

The context provided does not support the claims made in the answer. The answer contains false statements about GPT, such as its inability to process sequential data and lack of an attention mechanism, which contradict its actual design and purpose as a text-based model with self-attention mechanisms.

Completeness: **Worse**

The answer fails to cover any advantages of GPT compared to other pre-trained models, such as its scalability, adaptability, or the ability to perform well with efficient fine-tuning methods like LoRA discussed in the context.

Clarity: **Bad**

The answer is misleading and poorly structured, making it difficult to understand the intended argument. Additionally, the inclusion of false information detracts from clarity.

Conciseness: **Bad**

While the answer is short, it contains unnecessary and incorrect information, making the response verbose in terms of presenting irrelevant content.

Final Ratings:

- Relevance: Worse
- Faithfulness: Worse
- Completeness: Worse
- Clarity: Bad
- Conciseness: Bad

Weighted Score: 24.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.62 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, which asks for the key advantages of LoRA as described in the context. It incorrectly discusses satellite technology unrelated to LoRA.

Faithfulness: **Worse**

The answer is not supported by the provided context, which describes LoRA's advantages in machine learning, such as efficient task-switching and reduced hardware requirements. The discussion of satellite relays has no basis in the context provided.

Completeness: **Worse**

The answer fails to cover any of the major aspects of LoRA's advantages mentioned in the context, such as task-switching efficiency, reduced storage requirements, lower hardware barriers, and lack of inference latency.

Clarity: **Bad**

While the sentence itself is grammatically correct and understandable, its content is completely irrelevant to the question, which makes it misleading and confusing.

Conciseness: **Good**

The answer is concise and free of unnecessary repetition, but this does not improve its overall quality due to irrelevance and lack of faithfulness.

Weighted Score: 28.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.63 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 about LoRA's practical benefits, advantages, and limitations. Instead, it introduces unrelated concepts like aviation control systems and electromagnetic feedback loops, which are not mentioned in the context.

Faithfulness: **Worse**
The answer is entirely unsupported by the provided context. The context discusses VRAM consumption, GPU usage, task switching, checkpoint size, and training speed, none of which are reflected in the answer.

Completeness: **Worse**
The answer fails to cover any of the major aspects outlined in the context, such as memory reduction, GPU efficiency, checkpoint size reduction, task switching, or speed improvements, nor does it mention the limitations described.

Clarity: **Fair**
The answer is somewhat structured and easy to understand, but it is irrelevant and disconnected from the question, which undermines its usefulness.

Conciseness: **Good**
The answer is concise, but this is primarily due to its lack of relevant content rather than proper

summarization.

Weighted Score: 30.0

Response Latency: 0.00 seconds

Evaluation Latency: 6.95 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 about which specific weight matrices in the transformer should be adapted using LoRA. The mention of "Fourier-transformed attention gradient maps" and "non-linear thermostat modules" is irrelevant to the question.

Faithfulness: **Worse**

The answer is not supported by the given context, which specifies weight matrices in the self-attention module and provides details about parameter budgets for adapting those matrices using LoRA.

Completeness: **Worse**

The answer fails to cover any aspect of the question. The context specifies adapting attention weights with LoRA, but the answer does not address this or mention any relevant weight matrices.

Clarity: **Worse**

The answer is unclear and includes terms like "Fourier-transformed attention gradient maps" and "non-linear

thermostat modules," which are not explained and seem unrelated to the topic.

Conciseness: **Bad**

While the answer is not overly verbose, it introduces unnecessary jargon that detracts from conciseness without adding value or relevance.

Weighted Score: 22.0

Response Latency: 0.00 seconds

Evaluation Latency: 7.07 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: ### Evaluation of Answer

Relevance: Worse

The answer does not directly address the question, "Why Self-Attention?" Instead, it provides unrelated and nonsensical information about neuron temperatures, dolphin vocalizations, holographic inputs, and other irrelevant topics. The response fails to explain the benefits or purpose of self-attention as described in the context.

Faithfulness: Worse

The answer is not supported by the provided context at all. The context explains self-attention as an attention mechanism for computing representations of sequences and its applications in tasks like reading comprehension and textual entailment. None of the claims in the answer, such as "rotational dropout" or "holographic inputs," are present or supported by the context.

Completeness: Worse

The answer does not cover any major aspects of the question. It fails to address the purpose, mechanism, or benefits of self-attention, which are clearly described in the context.

Clarity: Worse

The answer is convoluted, nonsensical, and filled with jargon that is irrelevant and confusing. It is not structured in a way that makes sense or is easy to understand.

Conciseness: Worse

The response is overly verbose and filled with unnecessary information that detracts from the answer. It introduces unrelated and fictional details that have no bearing on the question or context.

Final Assessment

The answer fails to satisfy all evaluation criteria and provides irrelevant, unfaithful, incomplete, unclear, and verbose information.

Weighted Score: 20.0

Response Latency: 0.00 seconds

Evaluation Latency: 7.48 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 provided answer:

Relevance: Worse

The answer does not correctly identify the model with the highest average score from Table 1. It incorrectly states that BiLSTM+ELMo+Attn has an average of 95.3, which is not supported by the context provided. Furthermore, it misrepresents the average scores for BERT(LARGE) and BiLSTM+ELMo+Attn, making the answer irrelevant to the question.

Faithfulness: Worse

The answer provides information not supported by the given context. Specifically, it fabricates scores and claims about model performance that contradict the data in Table 1. This violates faithfulness to the provided

context.

Completeness: Bad

The answer misses key aspects of the question by failing to correctly identify the highest average score and the corresponding model. While it attempts to provide a comparison, this comparison is based on fabricated or incorrect data, which negatively impacts completeness.

Clarity: Fair

The structure of the answer is clear, but the incorrect information and fabricated claims undermine its clarity. A reader would be misled due to the inaccuracies in the data presented.

Conciseness: Good

The answer is concise and avoids unnecessary verbosity. However, the brevity does not compensate for the fundamental inaccuracies in the provided information.

Weighted Score: 32.0

Response Latency: 0.00 seconds

Evaluation Latency: 8.12 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:

Relevance: Bad

The answer does not accurately describe LoRA fine-tuning as outlined in the context. LoRA fine-tuning, according to the context, involves injecting low-rank matrices into pre-trained weight matrices to reduce computational overhead while leveraging pre-trained models, not replacing the entire neural network or discarding pre-trained models.

Faithfulness: Worse

The answer is completely inconsistent with the given context. The context explicitly emphasizes that LoRA fine-tuning uses pre-trained models and optimizes smaller injected matrices while retaining the original weights, contrary to the claim that the pre-trained model is discarded.

Completeness: Bad

The answer omits key aspects of LoRA fine-tuning, such as its efficiency, orthogonality to other methods, and advantages like reduced hardware requirements and task-switching overhead. It fails to address the mechanisms and benefits described in the context.

Clarity: Good

The answer is relatively easy to understand in terms of structure, but the clarity does not compensate for its inaccuracies.

Conciseness: Excellent

The answer is succinct and avoids unnecessary repetition or verbosity.

Final Ratings:

- **Relevance:** Bad

- **Faithfulness:** Worse

- **Completeness:** Bad

- **Clarity:** Good

- **Conciseness:** Excellent

Weighted Score: 46.0

Response Latency: 0.00 seconds

Evaluation Latency: 8.40 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: Relevance: **Fair**
The answer attempts to distinguish between full fine-tuning and LoRA fine-tuning, but key aspects from the context provided are not directly addressed, and some claims seem off-topic or incorrect in relation to the question.

Faithfulness: **Bad**
The claims made about LoRA fine-tuning, such as updating every parameter aggressively and requiring retraining for each new task, are not supported by the context provided. Additionally, the description of full fine-tuning freezing all parameters contradicts standard understanding of fine-tuning techniques.

Completeness: **Fair**

The answer touches on differences between full fine-tuning and LoRA but oversimplifies or misrepresents key aspects. Major points, such as technical specifics or advantages/disadvantages directly supported by the context, are missing.

Clarity: **Fair**

While the answer is grammatically correct, it introduces contradictions and unclear claims, making it difficult for readers to fully understand the comparison.

Conciseness: **Good**

The response avoids unnecessary verbosity and sticks to a brief format, but the content itself includes errors that detract from overall quality.

Weighted Score: 58.0

Response Latency: 0.00 seconds

Evaluation Latency: 9.32 seconds

[Fake Test 10] Question: According to Table 4, which model has the highest Dev and Test scores? What are the highest scores of Dev and Test respectively?

Fake Answer: ESIM+ELMo scored the highest on Test, while ESIM+GloVe scored the highest on DEV. The highest score for DEV is 51.9, while that for Test is 52.7.

LLM Evaluation: ### Evaluation:

Relevance: Worse

The answer does not directly address the question, as the models mentioned (ESIM+ELMo and ESIM+GloVe)

are not present in the provided context, and the scores provided (51.9 and 52.7) are not supported by the table data. The question specifically asks about Table 4, but the answer deviates entirely from the context.

Faithfulness: Worse

The answer is not supported by the given context. The context provides Dev and Test scores for BERT LARGE(Ensemble) and unet(Ensemble), but the answer mentions unrelated models and incorrect scores not found in the provided data.

Completeness: Worse

The answer fails to cover all major aspects of the question. It does not identify the correct model with the highest Dev and Test scores or provide the accurate scores from the context.

Clarity: Fair

While the structure of the answer is clear, the incorrect information makes it difficult to understand and trust the response. The clarity is undermined by its lack of alignment with the context.

Conciseness: Good

The answer is concise and free from unnecessary repetition. However, conciseness alone does not compensate for the inaccuracies present.

Final Ratings:

- **Relevance:** Worse
- **Faithfulness:** Worse
- **Completeness:** Worse
- **Clarity:** Fair
- **Conciseness:** Good

Weighted Score: 30.0

Response Latency: 0.00 seconds

Evaluation Latency: 25.83 seconds