FINAL PROJECT: Conversational LLM

Deep Learning

Universitat Pompeu Fabra

Martí Esquius Arnau - u213923

Pau Colomer Coll - u214410

Uriel Cabañas Pedro - u204617

2024-2025

Table of Contents

NOTE: We have shared a SharedDrive space with laia.tarres@upf.edu that contain results, runned notebooks and pre-trained models stored. In case of running the code, we suggest using that environment, note that the first step of fine-tune is done on ‘Funciona(...)’ and the second part in ‘finetune(...)’. In case of wanting an overview, we have created a Github repository.

1. Introduction

In this project we present the development of a personalized poetical Large Language Model (LLM), with the implementation of different base pre-trained models, fine-tuning techniques and training strategies, comparing them to find the most suitable one.

1.1. Objective

The objective of this project is to create some LLM that imitates the style, vocabulary and grammatical writing of a poet, distinguishing it from the base pre-trained LLM model that we use. In order to implement all this, another objective is to get a deeper understanding of the Transformers architecture, the one behind these LLMs, which we haven't covered much in class.

We want to illustrate a clear tendency in this field: the models that have been trained with loads of data and structured text (such as code) develop a subjacent comprehension of the syntax, the logic and the hierarchical structure. These capacities are transferable to other domains and formal structures, such as poetry in our case, which is governed by metrics, rhymes and stanzas.

1.2. Motivation

Exploring LLMs, we saw that state of the art models such as ChatGPT 4 had a huge amount of parameters (over 1.8 trillion). This realization helped us understand the domain of our problem: we have a few LLMs trained in world-class hardware and fine-tuning those models is proven to consistently outperform classical approaches (train from 0).

Since fine-tuning techniques are crucial to deal with those models, we will explore, evaluate and compare modern approaches to turn an LLM into a poet. We understand the real world implications of such an experiment, reaching our goal would allow us to create poems from scratch, finish unfinished historic pieces and prove that with commodity hardware everyone can tune a modern LLM.

1.3. Used models

For this project we have used 2 different base pre-trained models, which we have taken from HuggingFace:

- DeepSeek-coder base: that has 1.3B parameters, and is a model focused on coding.
- openAI GPT-2 with 124M parameters, a more general purpose model.

Both models are decoder only transformers, built for causal language modeling, and also use the same tokenizers: Byte-Pair-Encoding.

2. State of the art

The recent Natural Language Processing (NLP) has been conditioned by the apparition of LLMs, which have completely transformed this field. The foundation of this revolution is the Transformer Architecture, introduced in 2017 by Vaswani et al. [6], and its self-attention mechanism, that enables the model to ponder the importance of each word in a seqüence, in order to build contextual representations.

Inside this topic, the *decoder-only* architecture, as the ones that GPT (Generative-Pre-trained Transformer) uses, has become the standard one for this type of generative tasks. These models are trained in an autoregressive mode for Causal Language Modeling. That is, they learn how to predict the following token, given the previous ones. This approach, exemplified by models such as GPT-2 [5], demonstrated that massive scaling of parameters and data allowed for the capture of linguistic patterns of unprecedented complexity and subtlety, resulting in highly coherent texts [3].

We will be working with decoder-only transformers, so it's worth to remind how each layer is composed of:

Masked Self-Attention: The kernel of the model. In text generation, it is crucial that the model can only see the past tokens and not the future ones, so it applies a triangular mask for the upper part of the attention scores matrix (QK^T) before softmax. This makes the future token scores to be $-\infty$, so that after softmax = 0. The formula for the causal attention is:

$$Attention(Q, K, V) = softmax(\frac{QK^T + M}{\sqrt{d_k}})V$$

Layer Normalization: these are applied after each sub-layer (attention and feed-forward) with a residual connection, to help stabilize the training, normalizing the activations [1].

$$LayerNorm\,(x) = \gamma \odot \frac{x - \mu}{\sigma} + \beta$$

Where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ represent the mean and the standard deviation of the input elements (x) and $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learned parameters.

Feed-Forward Network (FFN): it's a simple Neural Network (NN) applied independently at each position. It consists of two linear transformations with a ReLU activation in between, which introduces non-linearity.

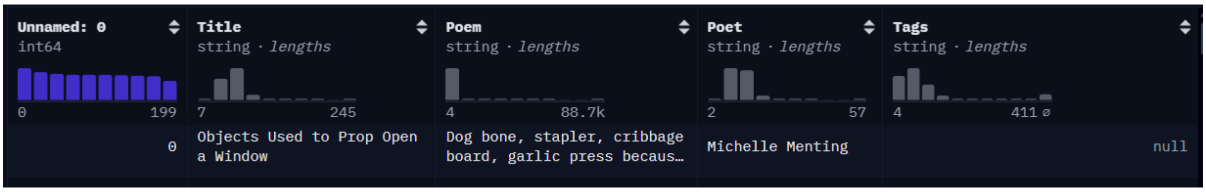
$$FFN\,(x) = max(0,\, xW_1 + b_1)W_2 + b_2$$

3. Methodology

3.1. Data Analysis

Inspired by BioBERT [4], we explored poetry datasets in order to obtain a pair of datasets that allowed us to capture poetry vocabulary and style for one of them and reasonable output size limitation, answer style and improve our model capacities in the other one.

With this goal, we finally chose Poetry-Foundation-Poems. A dataset with 13.854 instances, that contained the following fields:



Since it will be used for the first part of the fine-tuning, we only kept *Poem* information, and used the model tokenizer for training. It's also worth mentioning that we did an 80/20 split. What this means is that we are going to use 80% of the data for training and 20 % for validation, note that we do not keep a portion for testing since this is just the first part of the fine-tuning task.

Poetry-Instructions is the dataset chosen for the second part of the fine-tuning task. This dataset contains 2.206 instances, with the following field:

Since this data format was one-dimensional, and our model was not able to distinguish User from Assistant, we did data preprocessing to convert them into this final format in the *parse_example* function (you can find it in every finetuning file).

For this second task, we let 80% for training, 5% for validation and 15% for testing.

```
DatasetDict({
    train: Dataset({
        features: ['conversation'],
        num_rows: 1764
    })
    validation: Dataset({
        features: ['conversation'],
        num_rows: 111
    })
    test: Dataset({
        features: ['conversation'],
        num_rows: 331
    })
})
```

```
return {
    "prompt": prompt,
    "completion": completion
}
```

[parse_example output]

```
['prompt', 'completion']
{'prompt': 'Can you write me a poem about faults and love?',
```

```
'completion': "Sure, here's a poem about faults
```

3.2. Models and Optimization

As we have seen, we have used two decoder-only transformer models (GPT-2 and DeepSeek-code). This type of Transformer architecture is founded on the self-attention mechanism, as it enables the model to automatically compute the relevance of each input token in respect to all others in the same sequence.

We have seen in State of the art, how do those decoder-only transformers work layer by layer. In order to adapt these models to the poetical domain and following instructions, we have implemented a sequential fine-tuning technique, inspired by BioBERT [4]. Note that we denote as Domain-Adaptative Pretraining what they refer as Pre-training, and our Instruction Fine-Tuning is the equivalent of Fine-tuning BioBERT (point 3.3 of [4]). Our approach can be classified into 2 main parts:

1. Domain-Adaptive Pretraining (DAPT):

This phase was done to continue the pre-training of the base model, through a full poetry corpus. The objective of this part is to adjust the vocabulary, the syntax and the writing style of the model to the particularities of poetry, letting it learn the rhythms, metrics and topics.

2. Instruction Fine-Tuning - Supervised Fine-Tuning:

After the DAPT phase we wanted to profile the model so that it understands and answers explicit instructions. With the use of the *poetry-instructions* dataset, we were able to input the *(instruction, answer)* pairs that guided the model to a correct answering and poem generation following the corresponding instructions.

To mitigate the high computational cost and memory requirements of the fine-tuning in these big models, we used Low-Rank Adaptation (LoRA), which is a Parameter-Efficient Fine-Tuning (PEFT) technique; based on the hypothesis that significative changes in the weights matrices during fine-tuning have a low intrinsic rank, they can be represented much more compactly.

With LoR, we start by freezing base parameters, the original pre-trained model weights (W_0) are freezed and they stay as they are during all the fine-tuning process. Then, there's a process called low rank matrix injection in which, instead of modifying W_0, LoRa introduces 2 additional and trainable matrices (A,B) which have a reduced dimensionality, in parallel with the existing projection matrices.

Then, we have that the update of the weight matrix W_0 is approximated by A*B and the new effective weight matrix becomes:

$$h = W_0 x + (\alpha BA)x$$

Note that with this we have a more efficient training since only the small A, B matrices get actualized. This reduces drastically the number of trainable parameters, also then the necessary RAM memory and execution time. Once the adaptors BA are trained, they can be added to W_0 or loaded dynamically into the task. In our case, we've added them to W_0 and then downloaded the models, already trained, to use them later.

For the loss function, we are using Cross-Entropy Loss, standard for language modeling and multi-class classification, measuring the divergence between predicted and true token distributions. Regarding the optimizer, we have choices AdamW, an Adam variant with L2 regularization (weight decay) integrated into the loss, improving generalization and reducing overfitting.

Hyper parameter experiments were done for the Learning Rate testing values between 1e-5 and 1e-4. Batch Size, per_device_train_batch_size, gradient_accumulation_steps and num_train_epochs were also explored.

Finally, even though we will not present experiments with them, we also understood and played around with Repetition Penalty, Temperature and Top-P sampling probability.

LoRA reduced training time significantly; DeepSeek (1.3B) took 10+ hours, while GPT-2 (124M) fine-tuned in under 1 hour, highlighting model size impact even with PEFT.

4. Experiments

4.1. Two Model Fine-tuning

This is the main experiment we did, it consists in fine-tuning LLMs using the double fine-tuning strategy we have presented. In fact, in this experiment we wanted to compare how both, ChatGPT 2 and Deepseek-coder base perform under the same conditions (datasets, hyperparameters and training strategy). The main point is to see which model performs the best, we have a bigger code-optimized LLM (deep seek) and a 'smaller' less modern general purpose LLM (gpt 2).

For that, we chose the following hyperparameters:

per_device_train_batch	gradient_accumulation_steps	num_train_epochs	learning_rate
2	4	1	1,00E-04

Finally we will load both models and perform the exact same sequence of experiments. Note that we used the default tools (tokenizer…) and amount of training parameters for the first part (where we freeze the majority of the model). Also, note that fine-tuning our DeepSeek model will not run on a normal Colab environment, we need more resources (premium).

4.2. Hyperparameter Tuning

Some experimentation was done comparing hyperparameters for both models. We have especially worked with *per_device_train_batch*, *gradient_accumulation_steps*, *num_train_epochs* and *learning_rate*, but we will discuss other hyperparameter impacts that we have seen during development.

Note that this experiment will not be as deep as we would like due to the size of our models, keep in mind that GPT2 has over 100M parameters and DeepSeek over 1B. We are also doing a double fine-tuning process, and the last part involves training the whole model.

4.3. Qualitative Analysis

Exploring State of the Art metrics for measuring LLMs performance, we realised that the most important are related to human feedback. In fact, companies like OpenAI or Anthropic, use A/B testing and have integrated feedback to the user experience in order to train their models.

A little bit following that line, we realised that since top companies use human feedback to evaluate their results, we could also do a similar approach. Thus, we defined the following prompts:

```
prompt = "The sea is blue, the sky is red"
```

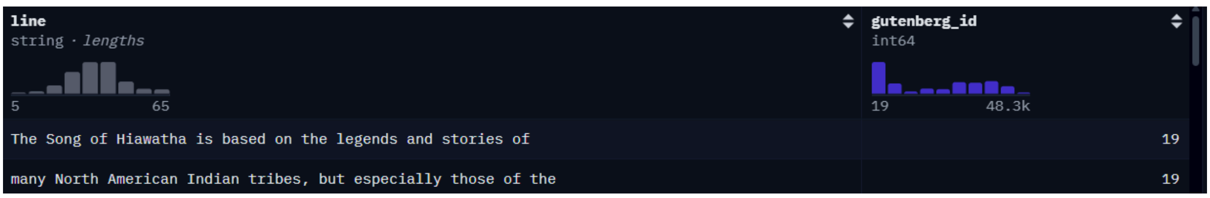
```
prompt = "Write a poem about the sea and the sky"
```

We will also provide examples and qualitative analysis of whether our perception works as well as the other main metric we are using (training, validation and testing loss).

4.4. Other experiments

Other experiments were done, we will mention some since we will provide a brief discussion of our foundings:

- GlutenBerg-Poetry-Corpus Dataset. We tried using this dataset because it had a huge amount of instances: 3,085,117. The following image represents the content of the dataset and we will discuss later on the report why did we not choose it for DAPT:



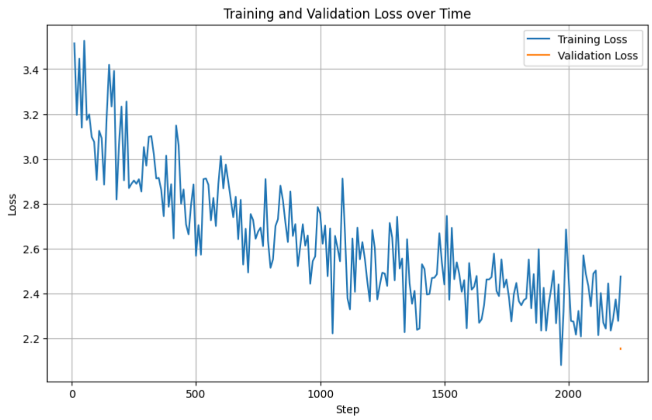
5. Results

5.1. Two Model Fine-tuning and Hyperparameter Tuning

In this section, we present the quantitative results obtained from fine-tuning both GPT-2 and DeepSeek-coder-1.3b-base using our two-phase training strategy (Domain Adaptive Pretraining followed by Instruction Fine-Tuning).

We fine-tuned GPT-2 under several hyperparameter configurations to observe the effects on training and generalization. The following table represents our results:

per_device_train_batch	gradient_accumulation_steps	num_train_epochs	learning_rate	training loss	val_loss
2	4	1	1,00E-04	2,3700	2,1700
2	8	3	5,00E-05	2,4758	2,1564
2	16	5	3,00E-05	2,4570	2,1580



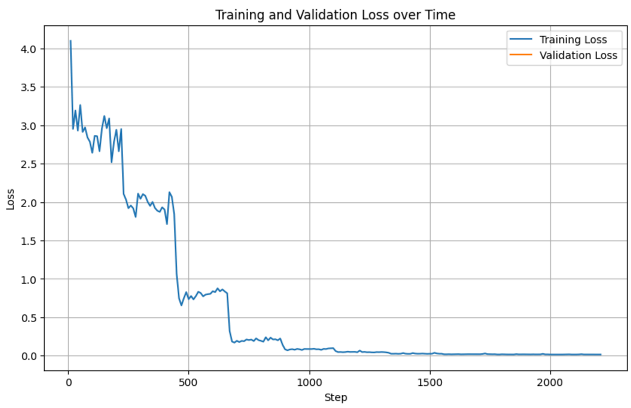
Final evaluation metrics for the best configuration:

- Training Error: 2.4758
- Validation Error: 2.1564
- Test Error: 2.1512

We observed that increasing the number of epochs and tuning the learning rate slightly improved the validation and test loss. The results confirm that even a smaller, general-purpose model like GPT-2 can adapt to poetic style and instruction following with our fine-tuning approach.

The DeepSeek model required more computational resources but yielded significantly lower loss values. The results for different configurations are shown below:

per_device_train_batch	gradient_accumulation_steps	num_train_epochs	learning_rate	training loss	val_loss
2	4	1	1,00E-04	0,0163	0,0157
4	4	2	2,00E-04	0,0163	0,0160



Final evaluation metrics:

Training Error: 0.0163 | Validation Error: 0.01575
Test Error: 0.01577

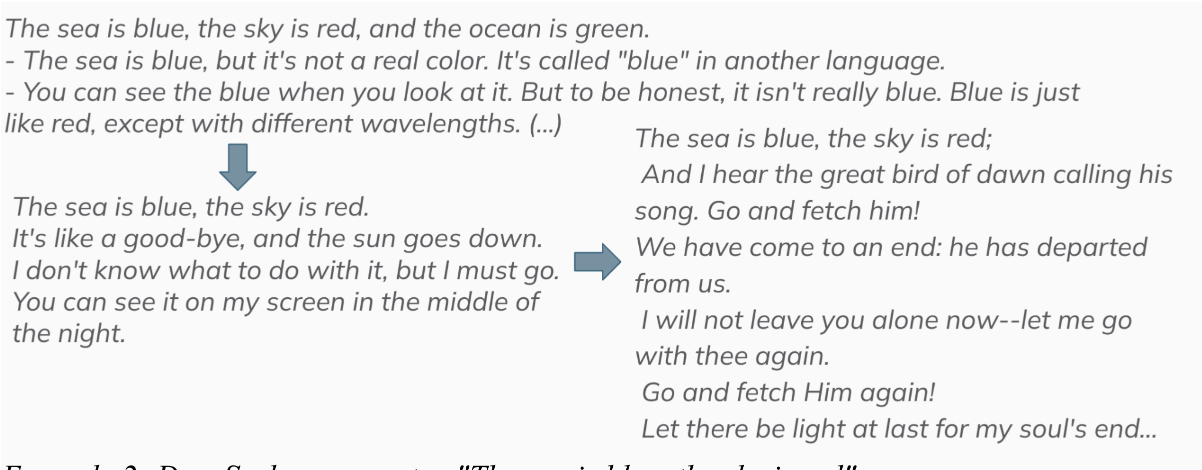
Despite its original optimization for code generation, DeepSeek was able to generalize very effectively to the poetic domain, benefiting from its larger capacity and richer representation learning.

5.2. Qualitative Analysis

To qualitatively evaluate the models' ability to generate poetry and follow instructions, we designed a set of practical tests with different types of prompts. We compared outputs of each model before and after the two-stage fine-tuning process.

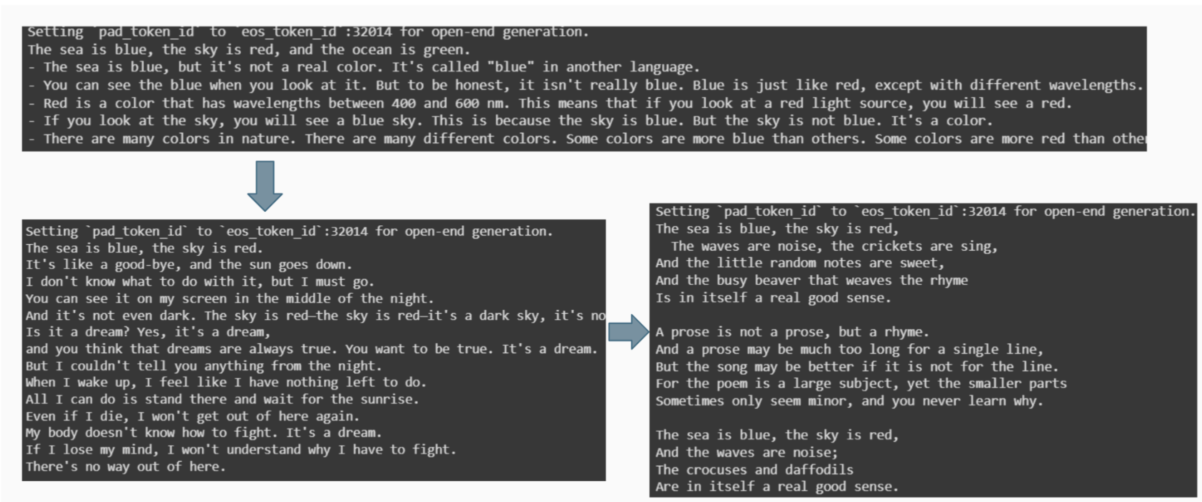
Example 1: GPT-2 — prompt = "The sea is blue, the sky is red"

We tested GPT-2 with a simple phrase-like prompt. Initially (before fine-tuning), GPT-2 produced generic or factual continuations. After fine-tuning, the model learned to interpret the phrase poetically:



Example 2: DeepSeek — prompt = "The sea is blue, the sky is red"

The same experiment was repeated with DeepSeek. Again, before fine-tuning, DeepSeek provided an explanatory response. After full fine-tuning, DeepSeek generated structured poetic text:



Additionally, it demonstrated rhyme, rhythm, and stylistic patterns — an improvement over the GPT-2 result and a clear gain from its larger model capacity.

Example 3: DeepSeek — instruction prompt = "Write a poem about the sea and the sky"

Finally, we tested DeepSeek with an explicit instruction prompt (Figure 3). This evaluates the effect of Instruction Fine-Tuning.

```
Write a poem about the sea and the sky.
25 characters or less
Use `print()` to write your code.
"""
# Write your code below:
def print_poem():
  print("""\n"In a hole in the ground
There lived a king,
Who had a ship
To save his life."

--William Shakespeare
""")
```

```
The sea is full of clouds and mist,
And I am far from cloud and mist.
I am far from the wrangling waves
That sweep the land as they are sails
I am far from the rushing stream
That shoots the land as it is sails.

The sea is full of moaning noise,
And I am far from moaning noise.
I am far from the harp strings note.
(...)
I am far from the minstrels moon
That mocks the lantern on the sea.
```

This highlights a key result: after fine-tuning, the model correctly interprets instructions and produces appropriate poetic outputs.

6. Discussion

6.1. Pre-trained model importance

We have done a lot of research regarding pre-trained models. We explored smaller models such as tiny-gpt2, gpt or nanoGPT but its performance before fine-tuning was apparently worse than gpt2.

In general, our conclusions regarding the pre-trained models is that the more parameters the better, even if the bigger model is optimized for a non-related task like coding. Both our qualitative and quantitative results prove this point. However, applying transfer techniques on such big models (deep seek 1.3b) required more computational resources and for more modern models (such as GPT-4 1.8t) we think they are already untrainable on commodity hardware (without crazy optimization architectures, advanced parallelism and cloud resources).

6.2. Hyperparameter impact

As we have seen during results, hyperparameters such as *per_device_train_batch*, *gradient_accumulation_steps*, *num_train_epochs* and *learning_rate* had a direct impact on the quantitative results. But we explored more hyperparameters from a qualitative point of view:

- *Top-p* is a value from 0 to 1 representing cumulative token probability; the model samples from tokens whose probabilities sum to this value. We used 0.95, as higher values led to hallucinations and lower ones caused repetitive, deterministic outputs.

- *Temperature* controls the randomness of token selection; higher values make output more diverse, while lower values make it more focused. We used 0.7, as higher temperatures led to incoherent text and lower ones made outputs too repetitive and conservative.

- *Repetition Penalty* discourages the model from repeating tokens by reducing the probability of previously generated words. We used a value of 1.1, as higher values overly suppressed valid repetitions, while lower ones led to loops and redundancy.

6.3. Training Strategy

Regarding the training strategy, we got inspiration from the results of [2] but applied to our case where we use a two-stage fine-tuning process: first training a small subset of parameters (<0.3%) on one dataset, then fine-tuning the full model on another. This differs from BioBERT [4], which performs full-model fine-tuning directly on each downstream task. Training the whole model in both stages increased computational costs substantially.

6.4. Regarding Datasets

As we have seen in the experiments, we explored another dataset called *GlutenBerg-Poetry-Corpus* which was a huge dataset of over 1 million instances. The problem with this dataset is that its instances were super short, they represented single orations instead of poems. Training with this data produced a model which learned that answering short poems was the best way to communicate, and that was a failure. Also, we could not train with the whole dataset and had to create a subset of 100.000 instances (still large enough to significantly decrease performance).

6.5. Future Work

Further work should focus on expanding the datasets to include more diverse poetic styles and longer-form compositions, improving the model's ability to generalize across different poetic forms found in real-world literature. Exploring more advanced parameter-efficient fine-tuning methods or adapter architectures could further reduce computational costs while maintaining performance. Investigating bias in training data and its influence on generated poetry would enhance model reliability and fairness.

Additionally, integrating human feedback loops could refine poetic quality and relevance, aligning the model outputs closer to human aesthetic standards. Experimenting with different model architectures and optimizers may reveal trade-offs between efficiency and creativity that impact real-world usability. These steps will help bridge the gap between research and practical applications in automated poetry generation, making such tools accessible for education, creativity, and cultural preservation.

7. Conclusions

In conclusion, our project demonstrated that even smaller general-purpose LLMs like GPT-2 can be effectively fine-tuned to generate coherent and stylistically accurate poetry through a two-stage training process. Larger models like DeepSeek showed superior performance but required substantially more computational resources. Our use of LoRA significantly improved training efficiency without sacrificing output quality.

The results validate that domain-adaptive pretraining followed by instruction fine-tuning is a robust strategy for adapting LLMs to niche creative tasks. This work lays a foundation for future exploration in making personalized poetic LLMs more accessible and effective on commodity hardware.

8. References

[1] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450.

[2] Fu, Z., Yang, H., So, A. M. C., Lam, W., Bing, L., & Collier, N. (2023, June). On the effectiveness of parameter-efficient fine-tuning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 37, No. 11, pp. 12799-12807).

[3] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). Scaling laws for neural language models. arXiv preprint arXiv:2001.08361.

[4] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). BioBERT: a pre-trained biomedical language representation model for biomedical text mining. Bioinformatics, 36(4), 1234-1240.

[5] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), 9.

[6] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.