Towards More Efficient Models: Implementation and Optimization of GPT-2 (127M) with Tlama (124M)

M. Galindo *
EigenCore
Zacatecas, Mex.
max@eigencore.org

A. León †
EigenCore
Zacatecas, Mex.
aleon@eigencore.org

Abstract

In this work, we present an optimized implementation of the GPT-2 (127M) model, developed under the EigenCore research initiative. Our approach focuses on reducing computational time and resource requirements while maintaining competitive performance. The resulting model, named Tlama (124M), serves as the foundation for ongoing research into scalable and efficient language models. This paper details the technical improvements, including architectural adjustments and training optimizations, and discusses their impact on both inference speed and resource utilization. Our findings demonstrate the feasibility of deploying smaller, more efficient models without sacrificing performance, paving the way for future advancements in the field of natural language processing.

1 Introduction

This article marks the beginning of EigenCore's journey into the research and development of Artificial General Intelligence (AGI), with the goal of contributing positively to humanity's progress. We are committed to ensuring reliability, ethical use, and minimizing the environmental impact associated with training and deploying artificial intelligence systems. To this end, we have decided to start with a small-scale model, which we call Tlama (124M). This model represents the first step in a series of planned advancements.

In recent years, the development of artificial intelligence has become increasingly expensive and unsustainable. Companies such as OpenAI, Anthropic, and others have invested millions of dollars in training massive language models (2). These models are capable of "reasoning" and analyzing their own responses through techniques such as Supervised Fine-Tuning (SFT) and Reinforcement Learning, endowing them with remarkable understanding and capabilities.

Recently, DeepSeek introduced a novel approach to developing Large Language Models (LLMs) with a focus on reasoning and reduced computational costs. Their work emphasizes scalable model development and aligns with our motivation to build Tlama (124M) from scratch. By adhering to the scaling laws proposed in their research (5), we aim to create efficient and sustainable models that pave the way for future advancements in the field.

2 Pre-Training

In this section, we present the dataset used for pre-training Tlama (124M), highlighting its composition, and relevance to the model's performance.

*Master's in Computer Science and Engineering, National Autonomous University of Mexico.
†Master's in Control, National Autonomous University of Mexico.

2.1 Data

For the initial pre-training phase of Tlama (124M), we utilized the **edu_fineweb10B** dataset, a high-quality, education-focused subset of the FineWeb corpus. This dataset contains 10 billion tokens of carefully curated text, specifically filtered to emphasize educational content, making it particularly suitable for training models intended for reasoning and knowledge-intensive tasks. The edu_fineweb10B dataset is derived from web-crawled data but undergoes rigorous preprocessing to remove duplicates, low-quality content, and irrelevant text, ensuring a clean and efficient dataset.

The choice of edu_fineweb10B aligns with our goal of optimizing computational efficiency while maintaining competitive performance. By leveraging a curated dataset with a strong educational focus, we ensure that Tlama (124M) is exposed to diverse and meaningful linguistic patterns, as well as domain-specific knowledge, during pre-training. This approach not only reduces the risk of training on noisy or redundant data but also lays a strong foundation for downstream tasks that require reasoning and comprehension.

3 Architecture

The architecture of Tlama (124M) is based on GPT-2 (124M) by OpenAI (3), maintaining the same structure and hyperparameters to ensure a solid foundation for comparison and further development. This decision allows us to leverage well-established practices while focusing on optimizations and scalability.

Table 1: Tlama (124M) Architecture

Params	n_{layers}	d_{model}	n_{heads}	n_{k,v_heads}	**Context Length**	**Sequence Batch Size**	**Learning Rate**	**Tokens**
124M	12	768	12	12	1024	64	6e-4	10B

As this is our first model, and it is entirely based on GPT-2, we followed most of its design choices, including the learning rate and architectural hyperparameters. However, we adjusted the number of tokens used during pre-training to better align with our computational constraints and research goals. This approach provides a reliable baseline for comparison, enabling us to evaluate the effectiveness of future modifications and ensure we are on the right track.

By adhering to GPT-2's architecture, we aim to establish a strong foundation for Tlama (124M), facilitating a clear understanding of its performance and guiding our efforts toward more efficient and scalable models in subsequent iterations.

4 Hyperparameters

The weight initialization scheme for Tlama (124M) is based on the approach used in GPT-2 (3). While OpenAI was not explicitly detailed about this in their papers presenting GPT-2 and GPT-3, their source code reveals that the weights were initialized with a normal distribution using a standard deviation of **0.02**, and biases were initialized to **0**. This initialization strategy is consistent with the scale of the model and has been empirically shown to work well for transformer-based architectures.

In Tlama (124M), we adopted a similar approach:

- **Linear Layers**: Weights are initialized using a normal distribution with a mean of **0.0** and a standard deviation of **0.02**. If the layer has a `TLAMA_SCALE_INIT` attribute, the standard deviation is further scaled by $(2 \cdot \text{n\_layer})^{-0.5}$, where n_layer is the number of layers in the model. This scaling helps stabilize training in deeper networks. Biases, if present, are initialized to **0**.
- **Embedding Layers**: Weights are also initialized using a normal distribution with a mean of **0.0** and a standard deviation of **0.02**.

This initialization scheme ensures that the model starts with small, random values that are well-suited for the scale of Tlama (124M), promoting stable and efficient training.

The training process of Tlama (124M) employs a carefully designed **learning rate schedule** to ensure stable and efficient convergence. The schedule consists of three phases:

- **Warmup Phase**: During the initial steps, the learning rate increases linearly from **0** to the maximum learning rate (**6e-4**). This warmup phase helps mitigate instability at the beginning of training, allowing the model to gradually adapt to the data.
- **Cosine Decay Phase**: After the warmup phase, the learning rate follows a cosine decay schedule (6), gradually decreasing from the maximum learning rate to a minimum learning rate (**10% of the maximum**). This approach balances exploration and fine-tuning, ensuring that the model converges smoothly without overshooting optimal parameters.
- **Minimum Learning Rate Phase**: Once the maximum number of steps is reached, the learning rate stabilizes at the minimum value, allowing the model to fine-tune its parameters without significant updates.

The optimizer used for training is **AdamW** (**?**), which combines the benefits of adaptive learning rates with weight decay regularization. Weight decay is set to **0.1**, providing a balance between preventing overfitting and maintaining model performance. The learning rate is initialized to **6e-4**, a value chosen based on empirical evidence and prior work with transformer-based models like GPT-2.

This combination of a dynamic learning rate schedule and a robust optimizer ensures that Tlama (124M) can effectively learn from the training data while maintaining stability throughout the training process.

5 Infrastructures

The training of Tlama (124M) was conducted on a high-performance computing cluster equipped with **8 NVIDIA A100 GPUs**, each with **80 GB of VRAM**. This setup allowed for efficient parallel processing and handling of large batch sizes, significantly reducing training time.

However, to make the model accessible for researchers and developers with limited hardware resources, we also implemented strategies to train Tlama (124M) on a single GPU. By reducing the batch size to **8** and employing **gradient accumulation techniques**, the model can be trained on a single GPU with as little as **8 GB of VRAM**, such as an NVIDIA RTX 4060. While this approach increases the computational time, it does not compromise the quality of the model, making Tlama (124M) a versatile and scalable solution for a wide range of hardware configurations.

6 Optimization Techniques for Training Efficiency

To ensure that Tlama (124M) is both computationally efficient and scalable, we implemented a variety of optimization techniques across the model architecture and training pipeline. These optimizations not only reduce training time and resource requirements but also maintain or even improve model performance. Below, we outline the key strategies employed:

6.1 Parameter Sharing

One effective strategy to reduce the number of parameters while maintaining model expressiveness is weight sharing. We adopted this approach by sharing weights between the token embedding layer (`wte`) and the language model head (`lm_head`) (3). This reduces memory usage while maintaining the same expressive power. Additionally, it introduces an implicit regularization effect, which can improve generalization. By tying the embedding and output layers, we also ensure consistency in token representations throughout the model.

6.2 Flash Attention

Self-attention is a major computational bottleneck in transformer models. To mitigate this, we integrated **Flash Attention**, a memory-efficient and highly optimized implementation (4). By optimizing memory access patterns and reducing redundant reads/writes, Flash Attention significantly reduces computational overhead. This method is particularly beneficial for handling longer sequences, as it improves memory efficiency and execution speed without sacrificing model quality.

6.3 Mixed Precision Training

We leveraged mixed precision training with **bfloat16** and **float16** to exploit the full computational power of modern GPUs (7). Using lower precision speeds up matrix operations while reducing memory consumption, allowing us to use larger batch sizes. Additionally, automatic loss scaling prevents numerical instability issues, ensuring robust training.

6.4 Gradient Clipping

To stabilize training and prevent exploding gradients, we applied gradient clipping with a predefined threshold (1). This technique ensures that gradient values remain within a reasonable range, preventing divergence and promoting smoother optimization. It is particularly useful in deep architectures where large gradient magnitudes can destabilize learning.

6.5 Distributed Data Parallel (DDP)

For large-scale training, we employed **Distributed Data Parallel (DDP)** (7), which allows computations to be distributed across multiple GPUs. This approach enables parallel training, significantly reducing overall training time. By overlapping computation with communication, DDP maximizes hardware utilization, ensuring efficient scalability even in multi-node environments.

6.6 Ugly Numbers and Padding

Optimizing tensor dimensions can significantly enhance computational efficiency. We adjusted the vocabulary size to hardware-friendly values, such as 50,304 instead of 50,257, to align with GPU memory architectures (6). This optimization ensures more efficient tensor operations, minimizes padding overhead, and improves overall GPU utilization.

6.7 Learning Rate Scheduling

To stabilize training and ensure better convergence, we implemented a learning rate schedule combining **warmup** and **cosine decay** (6). The warmup phase gradually increases the learning rate to prevent instability at the beginning of training. After this phase, the learning rate follows a smooth decay, refining convergence and preventing drastic parameter updates.

6.8 Gradient Accumulation

To handle hardware with limited memory, we employed gradient accumulation (7), allowing for effective batch sizes larger than what can fit in a single GPU pass. By accumulating gradients over multiple iterations before updating model parameters, we simulate the benefits of larger batch sizes without increasing memory consumption. This technique ensures stable updates and reduces the need for frequent gradient synchronization.

6.9 Torch.compile and Kernel Fusion

To further enhance execution speed, we utilized **torch.compile** (7), which optimizes computational graphs and reduces Python overhead. Kernel fusion merges multiple operations into a single step, minimizing redundant memory access and improving execution efficiency. These optimizations lead to significant speedups in both training and inference, making Tlama (124M) more efficient and accessible to a wider range of users.

These optimizations collectively enable Tlama (124M) to achieve state-of-the-art performance while remaining accessible to researchers and developers with varying levels of computational resources.

7 Preliminary Results

As a first step in developing our language model, we focused on monitoring the loss function during training and validation. Additionally, since our model has not yet been fine-tuned for chat-based interactions using Supervised Fine-Tuning (SFT), we evaluated its performance on the HellaSwag

benchmark. This benchmark assesses a model's ability to correctly complete a given sentence, making it a useful metric for evaluating natural language understanding.

Figure 1 presents both of these key metrics.

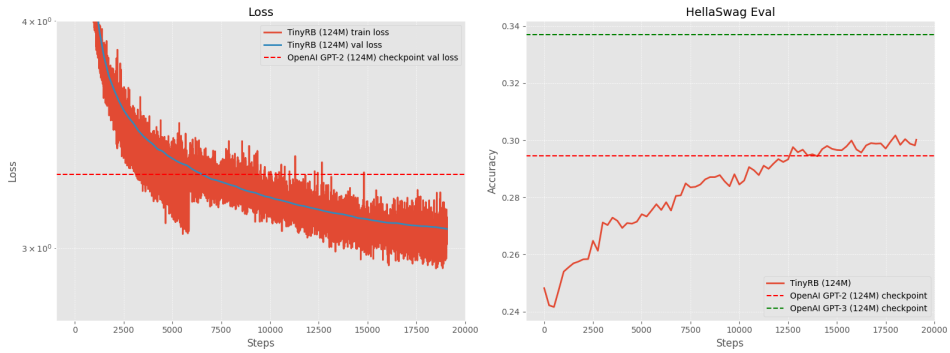

Figure 1: Training loss and validation performance on the HellaSwag benchmark. The left plot shows the training and validation loss over training steps, while the right plot compares accuracy to existing models.

8 Analysis

From the left plot in Figure 1, we observe that the loss function is steadily decreasing at a promising rate, suggesting that our implementation is functioning correctly and that the model is improving as expected. Interestingly, we surpass the validation loss of the GPT-2 (124M) baseline, which is a strong indication of the effectiveness of our training approach. One notable observation is the periodic spike in training loss, which we suspect is due to insufficient data shuffling, causing periodic variations in the optimization process.

On the right plot, we see that Tlama (124M) (124M) exceeds GPT-2 (124M) on the HellaSwag benchmark, demonstrating stronger sentence completion capabilities. This result aligns with findings from the DeepSeek team (5), who noted that GPT-2 was trained on the WebText dataset, a highly diverse corpus. We hypothesize that this diversity may contribute to GPT-2's relatively lower performance on this specific task, emphasizing the importance of dataset selection in model training.

Additionally, although Tlama (124M) has not yet reached GPT-3 (124M) performance, the trend in our results suggests that further training steps could narrow this gap. With additional fine-tuning and training, we anticipate that Tlama (124M) will continue improving and potentially reach GPT-3 performance levels in specific benchmarks.

9 Future Work

This research represents just the beginning of our exploration into language model development. The experience gained in implementing this model has been invaluable, equipping us with the necessary insights to build scalable, cost-efficient, and highly performant models.

Moving forward, we plan to develop an enhanced version of Tlama (124M) that strictly adheres to the Scaling Laws proposed in (6). Additionally, we aim to conduct a more extensive evaluation of the model, analyzing its capabilities across multiple benchmarks to understand the full potential of compact language models. Further optimizations in data preprocessing, architectural refinements, and training strategies will be explored to push Tlama's performance closer to larger-scale models.

10 Conclusion

In this work, we introduced Tlama (124M), a compact yet efficient language model. Through rigorous experimentation, we demonstrated its ability to outperform GPT-2 (124M) on the HellaSwag benchmark while maintaining a stable and effective training process. Our findings highlight the

importance of dataset selection, model architecture, and training techniques in achieving competitive performance with smaller models.

Despite its promising results, Tlama (124M) represents only the beginning of our exploration into the potential of smaller, highly optimized models. Future efforts will focus on **scaling strategies**, **improved data augmentation techniques**, and **advanced fine-tuning methodologies** to push the boundaries of what these models can achieve. By continuing to refine our approach, we aim to uncover the full potential of models like Tlama (124M), demonstrating that they can not only match but also exceed expectations in terms of efficiency and performance. Our goal is to explore how far these models can go, challenging the notion that larger models are always superior.

References

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*, 5998–6008.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 1877–1901.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*.

Dao, T., Fu, D., Ermon, S., Rudra, A., & Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*.

DeepSeek. (2023). DeepSeek LLM: Scaling open-source language models with long-termism. *DeepSeek Blog*.

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*.

PyTorch. (2023). PyTorch documentation. Retrieved from `https://pytorch.org/docs/stable/`