Assignment 2: Building a Small-Scale Foundation Model from Scratch

Student: Aman Khan (002050777)
Date: October 23, 2025

1. Background and Motivation

Training a small-scale transformer from scratch provides an excellent opportunity to understand the core architecture and learning dynamics of foundation models. By implementing a mini-GPT and performing next-token prediction, I learned how tokenization, model design, and hyperparameters influence the model's ability to learn and generalize language patterns.

2. Learning Objectives

This assignment aimed to enable me to:

- Implement a transformer-based language model using PyTorch
- Train the model from scratch on preprocessed text data for next-token prediction
- Track important training metrics such as loss and perplexity
- Experiment with hyperparameters like learning rate, batch size, sequence length, and number of layers
- Save and reload model checkpoints
- Visualize and interpret training progress

3. Model Architecture

The mini-GPT model I built follows the assignment specifications for a basic transformer language model:

- Transformer layers: 2 layers of transformer encoder blocks
- Embedding dimension: 128 (within the 64–256 range specified)
- Multi-head attention: 4 heads
- Positional encoding: Learnable positional embeddings
- Normalization and activation: Layer normalization applied in each transformer layer

- Output: Linear layer outputs logits for next-token prediction

The resulting model has approximately 6.7 million trainable parameters. While small relative to production models, it was sufficient for this experiment.

4. Dataset Details

The dataset was the preprocessed and tokenized text from Assignment 1. Key properties:

- Sequence length: 512 tokens (extended slightly beyond the suggested 32–128 for richer context)
- Vocabulary size: 50,257 tokens (using GPT-2 tokenizer)
- Train/Validation split: 90/10
- Data batching: Implemented shuffling and batching for efficient training

Each training example consisted of a token sequence, from which the model predicted the next token at each step.

5. Training Setup

Training was conducted using the following configuration:

- Platform: Google Colab
- Hardware: Tesla T4 GPU with 15 GB VRAM
- Framework: PyTorch 2.6.0
- Batch size: 8 (reduced from larger sizes due to memory constraints)
- Learning rate: 5e-4 (Adam optimizer, default betas)
- Loss function: Cross-entropy loss for token prediction
- Epochs: 10 full passes over the dataset
- Gradient clipping: Applied to avoid unstable training

During each training iteration, I performed: forward pass, loss calculation, backpropagation, and optimizer step. After each epoch, validation loss and perplexity were computed to monitor generalization.

6. Results and Analysis

Epoch	Training Loss	Validation Loss	Training Perplexity	Validation Perplexity
1	10.30	9.78	29,964	17,674
5	6.93	7.02	1,024	1,121
10	6.10	6.54	446	693

The steady reduction in loss and perplexity throughout training confirmed effective learning. The small gap between training and validation losses (0.44) implies good generalization. Starting with a perplexity near 30,000, the model ultimately reached around 700, showing it learned meaningful token prediction despite the limited data and model size.

7. Training Visualizations

The training curves (see attached `training_metrics.png`) highlight:

- Loss curves: Smooth, closely tracking training and validation loss, demonstrating stable training without overfitting
- Perplexity curves: Mirroring the loss curves, confirming that the model's predictive performance improved progressively

8. Challenges and Solutions

- Memory constraints: Initial batch sizes caused CUDA out-of-memory errors, resolved by reducing to batch size 8, balancing speed and resource limits.
- Data loading: Path issues in Google Colab were fixed by verifying file locations and adjusting dataset loading paths.
- GPU utilization: Added explicit device checks to ensure training utilized the GPU, improving efficiency.

These challenges provided insight into practical training setup and hardware considerations.

9. Learnings

The assignment helped consolidate both theoretical and practical knowledge:

- Transformer mechanisms: Gained clearer understanding of self-attention and layer stacking
- Training dynamics: Observed how loss and perplexity evolve with training
- Resource management: Learned to navigate GPU memory limitations and configuration
- Debugging and visualization: Developed skills to monitor metrics and troubleshoot issues

10. Conclusion

This project successfully implemented and trained a mini-GPT from scratch, illustrating the core steps of building a transformer-based language model. Despite its small scale, the model demonstrated strong learning with reasonable perplexity, highlighting the value of the exercise for understanding foundational NLP models.

The experience forms a solid base for exploring larger, more complex models and for further experimentation with hyperparameter tuning and advanced training techniques.

Appendix: Submitted Files

- `mini_gpt_checkpoint.pt` — model weights and metadata
- `training_metrics.png` — plots of loss and perplexity
- `assignment2_notebook.ipynb` — full code with training loop
- This report (PDF)