Memory-Efficient Transformer Optimizations for Low-Perplexity Language Modeling

Fengting ***Yuchi***, *Matthew (Hyunjoon)* ***Jo***, *Minghao* ***Xue***, *Zhongming* ***Li***

NLP: Self-supervised Models, Spring 2025

Johns Hopkins University

{fyuchi1, hjo2, mxue7, zli369}@jh.edu

Introduction

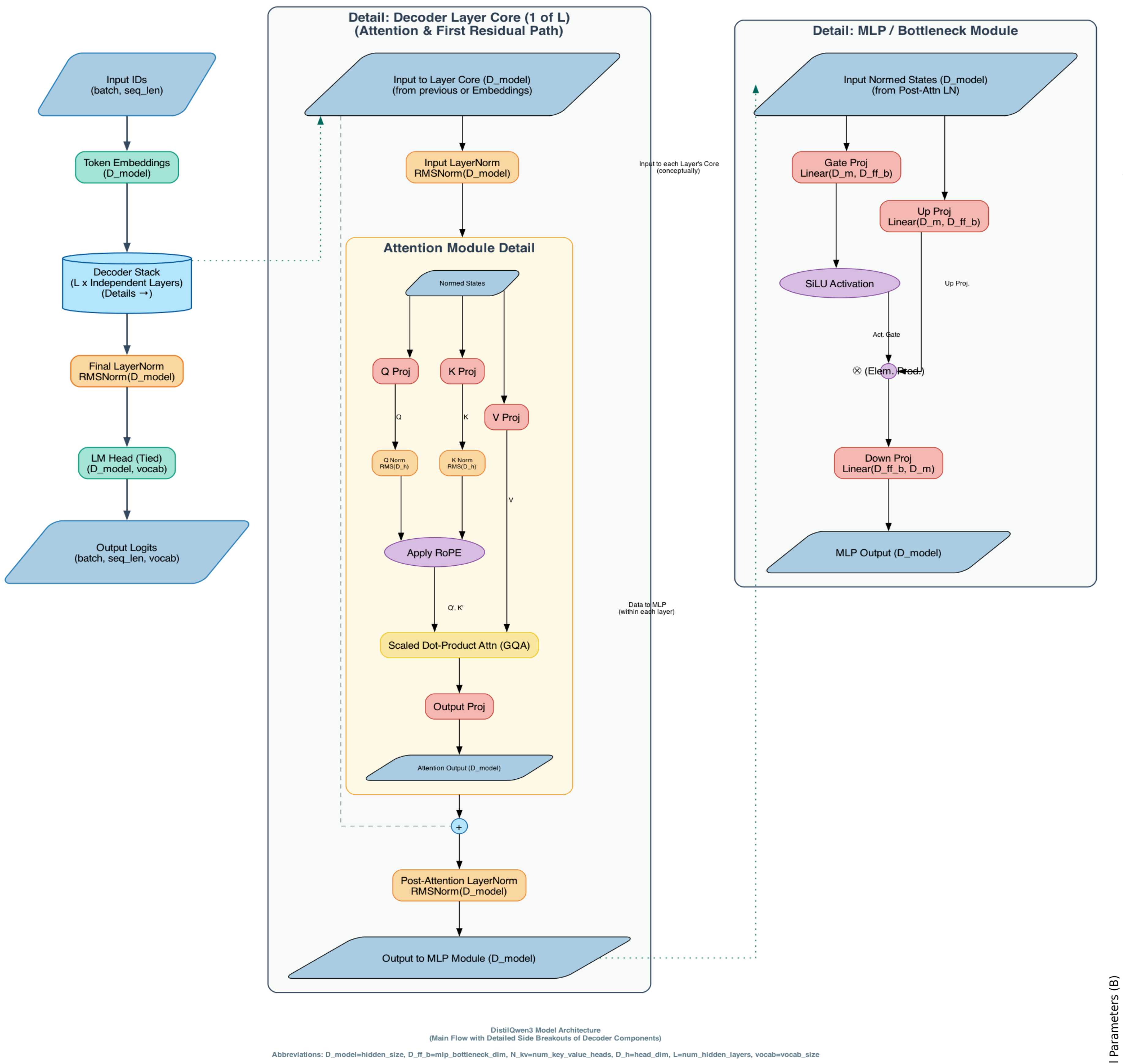
Input IDs (batch, seq_len)
Token Embeddings (D_model)
Decoder Stack (L x Independent Layers) (Details →)
Final LayerNorm RMSNorm(D_model)
LM Head (Tied) (D_model, vocab)
Output Logits (batch, seq_len, vocab)
Detail: Decoder Layer Core (1 of L) (Attention & First Residual Path)
Input to Layer Core (D_model) (from previous or Embeddings)
Input LayerNorm RMSNorm(D_model)
Attention Module Detail
Normed States
Q Proj
K Proj
V Proj
Q
K
V
Q Norm RMS(D_h)
K Norm RMS(D_h)
Apply RoPE
Q', K'
Scaled Dot-Product Attn (GQA)
Output Proj
Attention Output (D_model)
+
Post-Attention LayerNorm RMSNorm(D_model)
Output to MLP Module (D_model)
Input to each Layer's Core (conceptually)
Data to MLP (within each layer)
Detail: MLP / Bottleneck Module
Input Normed States (D_model) (from Post-Attn LN)
Gate Proj Linear(D_m, D_ff_b)
Up Proj Linear(D_m, D_ff_b)
SiLU Activation
Up Proj.
Act. Gate
⊗ (Elem. Prod.)
Down Proj Linear(D_ff_b, D_m)
MLP Output (D_model)
DistilQwen3 Model Architecture
(Main Flow with Detailed Side Breakouts of Decoder Components)
Abbreviations: D_model=hidden_size, D_ff_b=mlp_bottleneck_dim, N_kv=num_key_value_heads, D_h=head_dim, L=num_hidden_layers, vocab=vocab_size

Figure 1. Model Architecture

This default project tackles the challenge of training low-perplexity language models on a constrained 20GB MIG GPU—reflecting real-world deployment limits. Rather than scaling up, we explore architectural and training innovations to boost efficiency.

We combine memory-efficient attention, lightweight MLPs, and cross-layer parameter sharing with distillation and quantization. Our hypothesis: with smart compression and optimized configurations, small models can rival larger ones—delivering strong performance under tight compute budgets.

Methods

Following previous works, we implemented **DistilQwen3** (Figure 1), a lightweight Transformer language model designed for memory efficiency.

Key architectural innovations include:

- **Cross-layer parameter sharing**: One decoder layer is reused N times, reducing memory and parameter count significantly.
- **Grouped Query Attention (GQA)** with **Rotary Positional Embeddings (RoPE)** for scalable and position-aware attention.
- **MLP bottleneck layers**: SwiGLU-activated low-rank projections reduce intermediate size and accelerate convergence.
- **RMSNorm** instead of LayerNorm for lighter normalization.
- **Weight tying** between the embedding and output projection layers.
- **Shared layer with per-layer indexing**, enabling depth-wise operations without increasing parameters.

These design choices allow us to train deep decoders with limited GPU memory while maintaining competitive performance.

For data selection, we pretrained on the **RedPajama-1T CommonCrawl subset**, a high-quality, web-scale corpus aligned with our evaluation domain. Streaming from Hugging Face enabled training under memory constraints. For fine-tuning, we used task-specific datasets: **DailyDialog** (dialogue), **SQuAD** (Q&A), and **CNN/DailyMail** (summarization). All data was processed using Hugging Face tokenizers and PyTorch loaders in a unified pretrain–finetune–evaluate pipeline.

Experiments and analysis

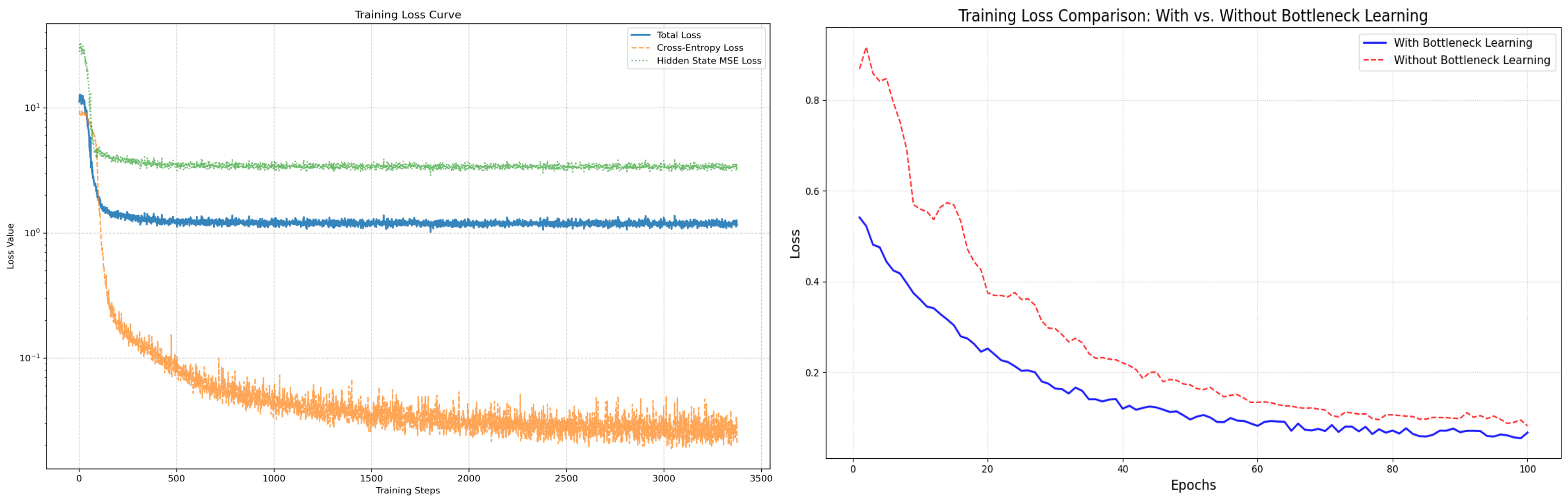

Figure 2. Training Loss

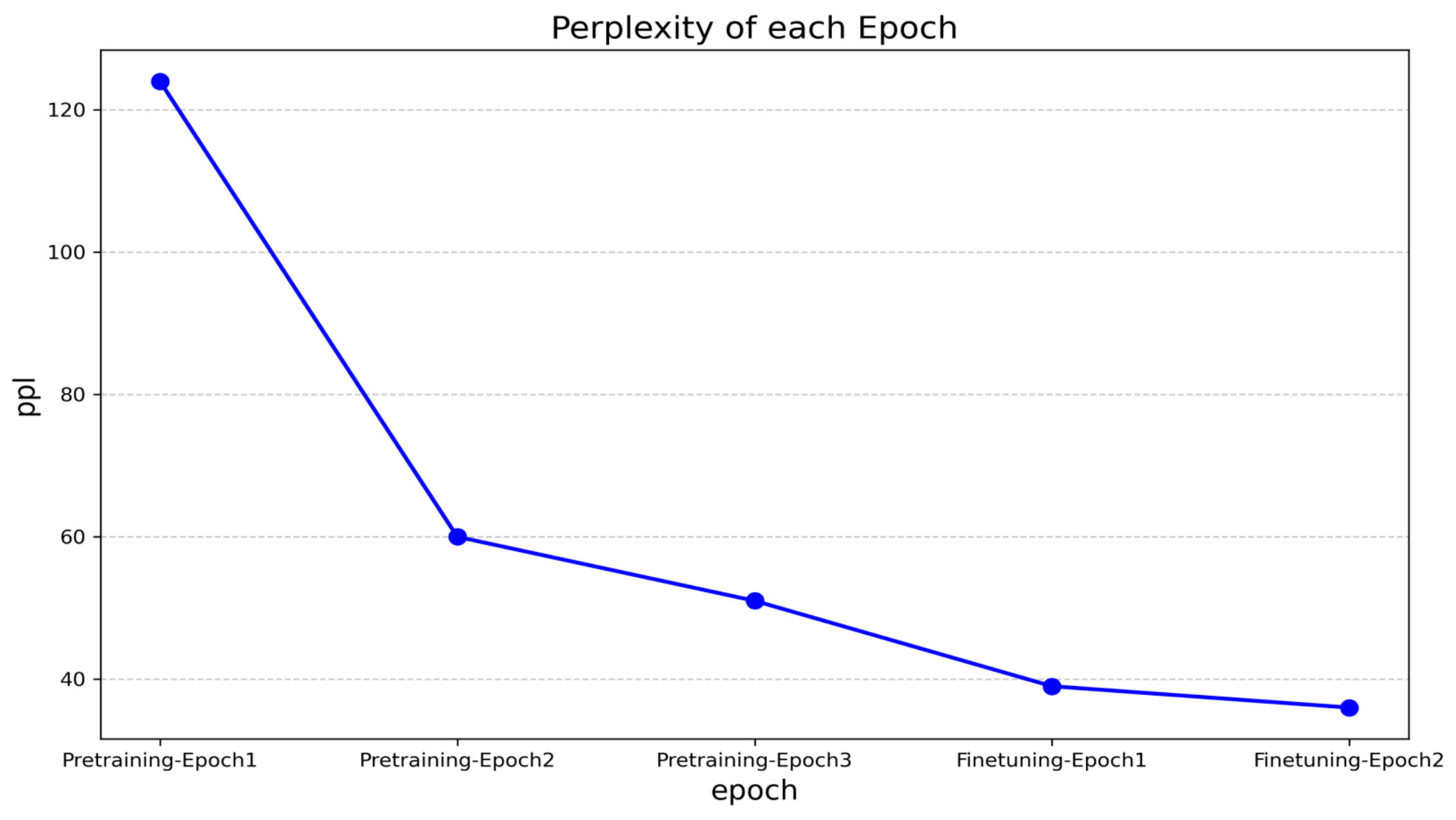

Figure 3. Perplexity

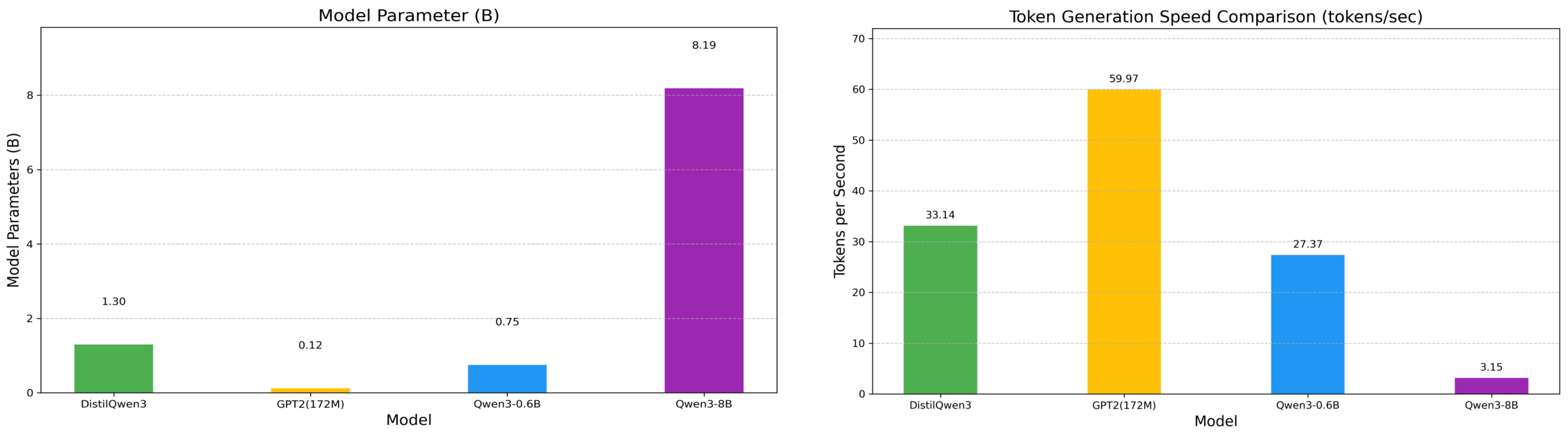

Figure 4 Model Comparisons

For distillation, we use Qwen3-8B as teacher model and DistilQwen3 as student model. We train the model on a shard of C4 dataset, using KL divergence as our loss.

The training loss of the distillation is shown in Figure 2. We discover that the model with bottleneck learning achieves a lower initial loss, converges faster, and reaches a lower final training loss.

Our model shows steady perplexity reduction across training (Figure 3). During pretraining, perplexity dropped from >120 to ~50 over 3 epochs. Fine-tuning brought further gains, reaching <40 by the second epoch. This consistent downward trend indicates effective learning and adaptation.

We also conduct comparisons among different models (Figure 4). **DistilQwen3** strikes a strong balance—its parameter size is mid-range, larger than GPT2 and Qwen3-0.6B, but much smaller than Qwen3-8B. Despite its size, it outperforms smaller models in generation speed, hinting at architectural efficiencies from weight sharing and bottlenecked MLPs. This validates our design as both compact and performant under resource constraints.

Future work

We will choose several pre-trained models as base models and fine-tune them with task-specific datasets. We will compare the training loss and the development loss of each base model and fine-tuning method. We will experiment on the following fine-tuning methods: Full Fine-Tuning, Feature-Based Tuning, Adapter Tuning, LoRA, and Prefix Tuning.

References

- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 2020.
- Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in neural information processing sysems, 35:16344–16359, 2022.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. ICLR, 1(2):3, 2022.
- Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451, 2020.
- Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. On the effect of dropping layers of pre-trained transformer models. Computer Speech & Language, 77:101429, 2023.