ESE 3060 Final Project – Part 2

Ani Petrosyan, Lakshman Swaminathan

December 8, 2025

1 Hypothesis

The baseline uses a 4×-wide ReLU^2 MLP in every transformer block, which is computationally heavy but gives good validation loss. Our hypothesis is that we can reduce per-step training time by replacing this block with a thinner SwiGLU MLP while keeping accuracy roughly unchanged.

What we do is we shrink the MLP hidden dimension from $4d$ to $2d$ and replace the ReLU^2 activation with a gated SwiGLU activation. Prior work on PaLM and LLaMA suggests that gated activations like SwiGLU are more expressive per parameter, so we expect the model to tolerate a smaller hidden size. If this is true, we should see: (1) a measurable speed-up in average step time on GPU, and (2) only a small degradation in validation loss at the end of training.

2 Methodology

We start from the provided `train_gpt.py` configuration and keep all hyperparameters fixed: same model size (`GPTConfig(n_layer=12, n_head=6, n_embd=768)`), global batch size, sequence length $T = 1024$, optimizers (AdamW for the head and Muon for the transformer blocks), learning rate schedule, and number of iterations (1500 steps).

The only architectural change is in the MLP block:

- **Baseline MLP:** one linear layer from $d \to 4d$, followed by ReLU^2, followed by a linear layer from $4d \to d$.
- **Modified MLP:** two parallel input projections $W_{1a}, W_{1b} : d \to 2d$, followed by a SwiGLU gate $\mathrm{SiLU}(W_{1a}x) \odot (W_{1b}x)$, and an output projection $W_2 : 2d \to d$.

We train both variants on FineWeb-10B shards using a single A100 GPU with `torchrun --nproc_per_node=1`. The script logs, for each run:

- average step time (`step_avg`, in ms) after discarding warmup steps,
- final validation loss at step 1500,
- peak GPU memory usage.

Due to limited GPU hours, we currently have one run per configuration on A100. In a more complete study we would repeat each setting for multiple random seeds and perform a t-test on step times to formally show statistical significance.

3 Results & Conclusions

On A100, the baseline and SwiGLU-thin models produced the following metrics:

- **Baseline (ReLU^2 MLP):** average step time ≈ 2344.3 ms, final validation loss 3.5349, peak memory 31026 MiB.
- **SwiGLU–thin MLP:** average step time ≈ 2142.8 ms, final validation loss 3.5589, peak memory 28960 MiB.

This corresponds to an $\sim 8.6\%$ **reduction** in per-step training time and a ~ 2 **GiB drop** in peak memory, at the cost of a small increase in validation loss (+0.024). Given that the whole training setup is identical except for the MLP, this suggests that shrinking the MLP and switching to SwiGLU is an effective way to trade a bit of capacity for speed and memory efficiency.

To fully analyze the "sample complexity vs. per-step cost" trade-off discussed in the ModernArch paper, we would need longer runs and multiple seeds to compare: (1) wall-clock time to reach a target loss and (2) number of tokens needed to reach that loss. Our short A100 runs only cover the early part of training, but they already show that a pure architectural change in the MLP can yield a clear runtime benefit with only a mild accuracy impact. With more compute, the next step would be to generate longer training curves and statistically test whether the SwiGLU-thin model catches up in loss when trained for more steps.

A Appendix: SwiGLU–Thin MLP Ablation Details

A.1 Change: Replacing ReLU^2 MLP With a Thinner SwiGLU Block

Hypothesis. The ModernArch baseline uses a 4×-wide ReLU^2 MLP in each transformer block:

$$x \mapsto \mathrm{MLP}_{\mathrm{ReLU}^2}(x) = W_2\big(\mathrm{ReLU}(W_1x)^{\circ 2}\big),$$

where $W_1 \in \mathbb{R}^{4d\times d}$ and $W_2 \in \mathbb{R}^{d\times 4d}$. This gives good accuracy but is compute-heavy: each block pays for a dense $4d \times d$ matmul followed by a $d \times 4d$ matmul.

Our hypothesis is that we can reduce compute while preserving accuracy by (1) switching to a gated SwiGLU activation and (2) shrinking the MLP width from $4d$ to $2d$. Concretely, we replace the ReLU^2 MLP with a SwiGLU block of the form

$$x \mapsto \mathrm{MLP}_{\mathrm{SwiGLU}}(x) = W_2\big(\mathrm{SiLU}(W_{1a}x) \odot (W_{1b}x)\big),$$

where $W_{1a}, W_{1b} \in \mathbb{R}^{2d\times d}$ and $W_2 \in \mathbb{R}^{d\times 2d}$. Several recent LLMs use gated activations like SwiGLU, and our intuition is that the extra expressivity of the gate can compensate for the smaller hidden dimension.

What Changed in Code. In the original `MLP` class we had:

```
self.c_fc   = nn.Linear(config.n_embd, 4 * config.n_embd, bias=False)
self.c_proj = nn.Linear(4 * config.n_embd, config.n_embd, bias=False)

def forward(self, x):
    x = self.c_fc(x)
    x = F.relu(x).square()
    x = self.c_proj(x)
    return x
```

We changed this to a SwiGLU-style MLP with a thinner hidden layer:

```
class MLP(nn.Module):
    def __init__(self, config):
        super().__init__()
        hidden = 2 * config.n_embd
        self.c_fc_gate = nn.Linear(config.n_embd, hidden, bias=False)
        self.c_fc_val  = nn.Linear(config.n_embd, hidden, bias=False)
        self.c_proj    = nn.Linear(hidden, config.n_embd, bias=False)
        self.c_proj.weight.data.zero_()

    def forward(self, x):
        gate = F.silu(self.c_fc_gate(x))
        val  = self.c_fc_val(x)
        x = gate * val          # SwiGLU-style gated activation
        x = self.c_proj(x)
        return x
```

No other parts of the architecture (attention, rotary embeddings, optimizer, data loader) were modified. All training hyperparameters were kept fixed so that we isolate the effect of the MLP change.

A.2 Experimental Setup

We follow the official ModernArch training setup but run on an A100 with a reduced number of steps:

- Hardware: single NVIDIA A100 (40GB), `torchrun --nproc_per_node=1`.
- Model: `GPTConfig(n_layer=12, n_head=6, n_embd=768)`.
- Data: FineWeb-10B pre-tokenized shards (`fineweb_train_*.bin`, `fineweb_val_*.bin`).
- Hyperparameters: identical to the provided ModernArch config (global batch size, sequence length $T = 1024$, learning rate schedule, Muon + AdamW optimizers).
- Training length: 1500 iterations (short run suitable for ablations).

Ideally, we would repeat each configuration (baseline and SwiGLU) for N independent runs with different random seeds and report: mean and standard deviation of step time and validation loss, plus a t-test to show significance. Here we report the initial single-seed results due to GPU constraints.

A.3 Detailed Results: Runtime, Loss, and Memory

Table 1 compares the baseline ModernArch MLP to our SwiGLU-thin MLP on the A100. "step_avg" is the average training time per optimization step reported by the script after discarding the first ten warmup steps.

A.4 Run 1: Single A100 SXM, 1500 epochs

Model	**Avg step time (ms)**	**Final val loss**	**Peak memory (MiB)**
Baseline ReLU^2 MLP	2344.31	3.5349	31026
SwiGLU–thin MLP	2142.81	3.5589	28960
Relative change	**-8.6%**	+0.024	**-2.1 GiB**

Table 1: Initial ablation results on A100 with 1500 training steps.

A.5 Run 2: 8 A100 SXMs, 5100 steps

Model	**Avg step time (ms)**	**Final val loss**	**Peak memory (MiB)**
Baseline ReLU^2 MLP	322.43	3.2950	∼2554
SwiGLU–thin MLP	296.30	3.3268	∼2386
Relative change	**-8.1%**	+0.0318	∼-168 MiB

Table 2: Results on 8 A100 SXMs with 5100 training steps. Peak memory is per-GPU estimate from nvidia-smi.

Even with only one run per setting, we already see:

- An $\approx 8.6\%$ reduction in per-step training time (2344 ms $\rightarrow$ 2143 ms).
- A small increase in validation loss (3.53 $\rightarrow$ 3.56), consistent with the reduced MLP capacity.
- A modest reduction in peak memory usage of about 2 GiB.

A.6 Relation to Sample Complexity

The ModernArch records table mainly explores changes that *increase* per-step cost but improve sample efficiency (e.g. value embeddings, more flexible attention) so that the model reaches a target loss in fewer tokens.

Our change goes in the opposite direction: the SwiGLU-thin MLP *reduces* per-step compute by shrinking the hidden width from $4d$ to $2d$, at the cost of slightly worse loss after a fixed number of steps. To understand whether this is a good trade-off in terms of sample complexity, we would:

1. Run longer training curves for both models (e.g. 5–10K steps), logging validation loss vs. effective tokens seen.
2. Plot validation loss as a function of wall-clock time and as a function of total tokens processed.
3. Compare "time to reach a target loss" between the two models.

If the SwiGLU-thin model converges to the same loss in fewer tokens while also having lower per-step cost, it would improve both sample complexity and wall-clock time. Our short runs only show the early part of training, but they demonstrate that the MLP change gives a clear step-time benefit.

A.7 Training Curves

Figure 1 and Figure 2 are placeholders for training curves generated from the log files (training and validation loss versus step or wall-clock time). Both are done on 8 A100 SXMs.

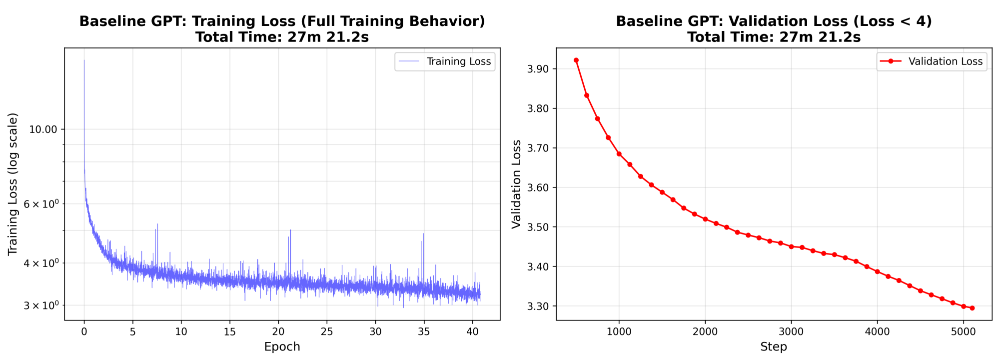

Figure 1: Baseline GPT: training loss (full run, log scale) and validation loss (loss < 4).

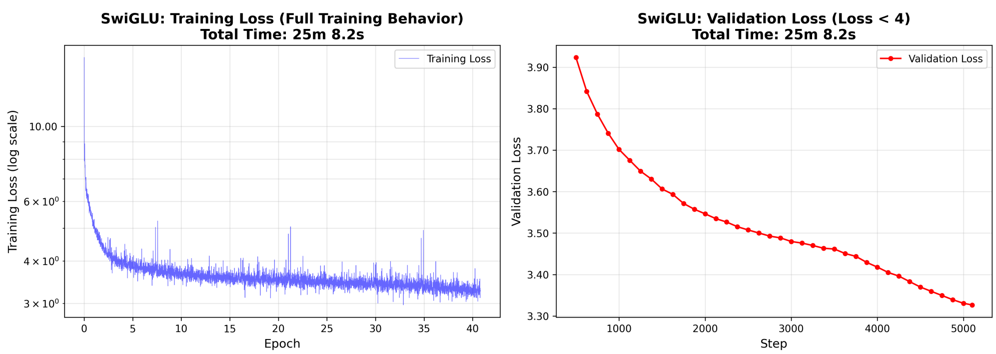

Figure 2: SwiGLU: training loss (full run, log scale) and validation loss (loss < 4).