Modernizing GPT-2: Hacker News Headlines

A MainCode Tech Interview Challenge

Jared Collette

December 14, 2025

Executive Summary

- **Objective:** Minimize validation loss on GPT-2 within 7 epochs (fixed constraint).
- **Baseline:** 1.754 Validation Loss
- **Achieved:** 0.977 Validation Loss (↓ 44.3% reduction)
- **Core Strategy:** Overhauled the legacy GPT-2 baseline with a Mixture of Experts (MoE) backbone featuring SwiGLU, RMSNorm, and ALiBi, utilizing AdamW with decoupled weight decay and epoch-shuffled data loading.

1 Introduction

This report details the optimization of a GPT-2 model to minimize validation loss on Hacker News headlines. The standard GPT-2 baseline (Val Loss: 1.754) converges too slowly for the strict 7-epoch limit imposed by the challenge. To address this, I replaced the baseline architecture with a Mixture of Experts (MoE) backbone, using SwiGLU activations and ALiBi positional embeddings to maximize parameter efficiency per training step.

1.1 Strategic Interventions and Report Structure

My optimization strategy addresses the following key areas:

- **EDA (Section 2):** Statistical analysis justifying high-density sample packing for stability and vocabulary entropy regimes motivating the Mixture of Experts architecture.
- **Data Pipeline (Section 3):** Implementation of global I.I.D. shuffling to reduce gradient variance and robust Byte Fallback tokenization to eliminate OOV spikes.
- **Architecture (Section 4):** Overhaul of the neural backbone incorporating Mixture of Experts (MoE), SwiGLU activations, ALiBi positional biases, and RMSNorm.
- **Optimization (Section 5):** Refinement of training dynamics via decoupled AdamW, cosine warmup scheduling, and memory management.
- **Experimental Results (Section 6):** Analysis of quantitative metrics and training dynamics, including a review of alternative methodologies.
- **Generated Text Analysis (Section 7):** Linguistic assessment of generated text samples to evaluate domain adaptation, syntactic structure, and semantic coherence.

The challenge imposed three strict constraints: a 7-epoch limit, no pre-trained weights, and no data augmentation. The full implementation is available at:

https://github.com/jaredmcollette/MainCodeInterview

2 Exploratory Data Analysis

2.1 Sequence Length and Context Window

Figure 1 illustrates the distribution of token counts per headline. The analysis reveals two insights regarding efficient data loading:

- **Extreme Conciseness:** Hacker News headlines are short, with a mean length of 11.2 tokens and a 99th percentile of just 22 tokens.
- **Implication (High-Density Sample Packing):** Although the data fits within a tiny window ($T = 22$), experimental trials showed that reducing the block size (e.g., to $T = 64$) harmed convergence. I instead maintained a larger block size of $T = 512$.

  This allows the loader to pack approximately 10–12 distinct headlines into a single context window. This packing strategy significantly increases the effective batch size (in terms of distinct examples per optimization step), thereby reducing gradient variance and stabilizing the trajectory compared to a smaller, noisier window.

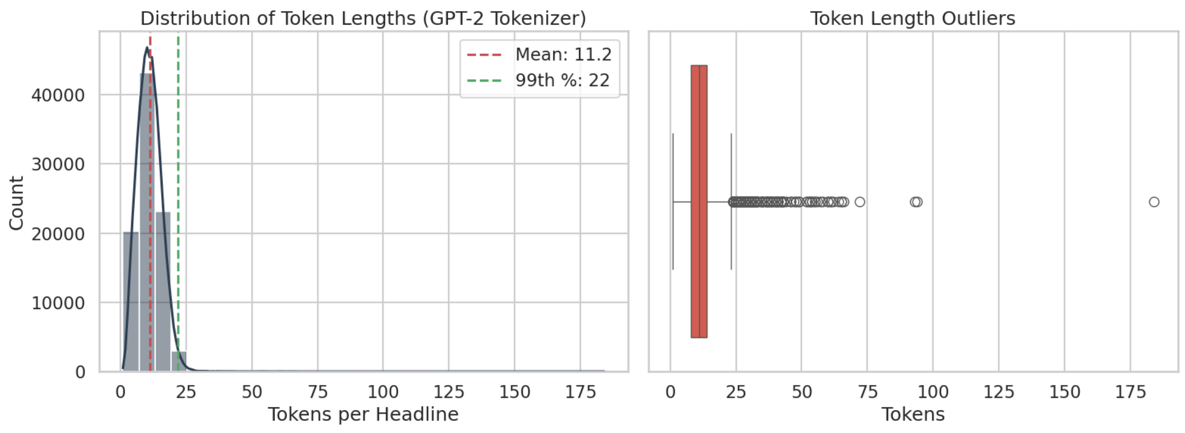

Figure 1: Distribution of token counts per headline. The tight distribution (99th percentile = 22) allows for efficient packing of multiple samples into a single context window.

2.2 Vocabulary and Domain Artifacts

Figure 2 highlights the vocabulary frequency, revealing specific domain artifacts that influenced the model choice:

- **Structural Artifacts:** The unigram analysis shows a massive outlier: "hn" i.e. "Hacker News" ($> 6,500$ occurrences). This is contextualized by the Bigram analysis, where "ask hn" is the most frequent phrase ($> 3,500$ occurrences).
- **Inductive Bias:** These patterns create a strong inductive bias. The model can rapidly minimize loss on these prefixes as $P(\text{HN}|\text{Ask}) \to 1$. Conversely, the tail of the distribution is dominated by high-entropy technical jargon ("google", "api", "rust").
- **Justification for MoE:** This sharp dichotomy between rigid platform templates (low entropy) and dense technical headlines (high entropy) provides the primary justification for the Mixture of Experts architecture. The separation allows specific experts to specialize in memorizing syntactic "macros" (like "Ask HN"), freeing up other experts to focus exclusively on modeling complex semantic relationships within the technical content.

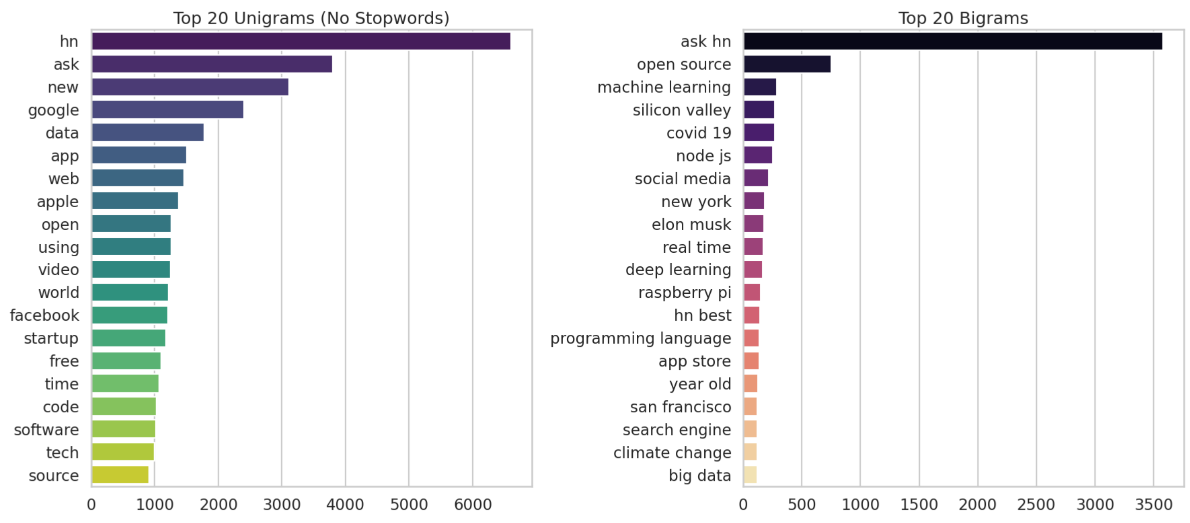

Figure 2: Top 20 Unigrams and Bigrams. The dominance of "hn" and "ask hn" reflects the platform's rigid submission syntax.

2.3 Statistical Validity

Finally, Figure 3 confirms that the dataset is suitable for language modeling despite its technical nature:

- **Power-Law Adherence:** The linear relationship on the log-log scale indicates that the dataset strictly follows Zipf's Law ($f \propto 1/r$).
- **Linguistic Structure:** This confirms that the data contains sufficient linguistic structure for a Generative Pre-trained Transformer to learn, rather than behaving like unstructured metadata or random noise.

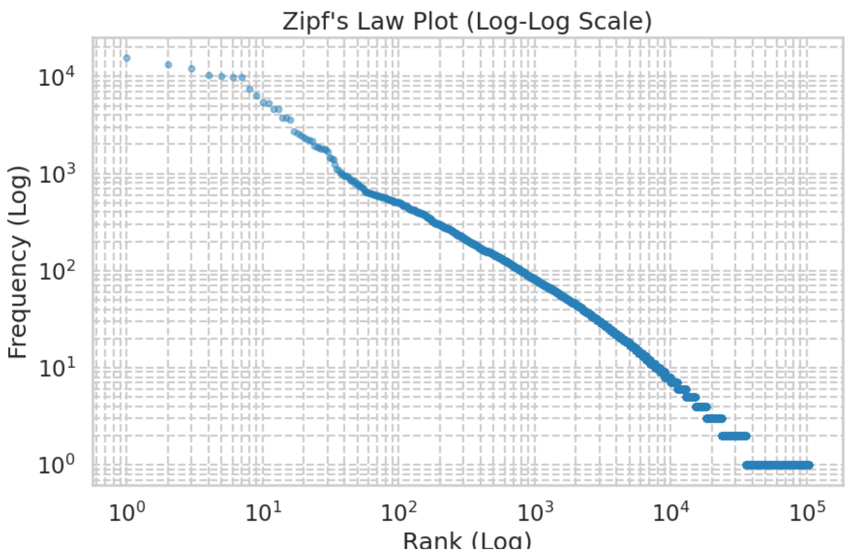

Figure 3: Zipf's Law analysis (Log-Log scale). The linear trajectory confirms the data follows a natural language power-law distribution.

3 Data Pipeline Optimizations

3.1 Decorrelated Batch Construction

I replaced the baseline's sequential sampling with an Epoch-based Shuffling strategy (`ShuffledBlockDataLoader` Class). This architectural change is driven by two optimization factors:

- **Gradient Variance Reduction:** The baseline constructed batches from contiguous text, preserving temporal correlations (e.g., a cluster of "Bitcoin" headlines from a specific week). This creates high covariance between samples within a batch. The variance of the mini-batch gradient estimator $\hat{g}$ is:

$$\text{Var}(\hat{g}) = \frac{1}{B^2}\left[\sum_{i\in\mathcal{B}} \text{Var}(g_i) + \sum_{i\in\mathcal{B}}\sum_{j\neq i} \text{Cov}(g_i, g_j)\right] \tag{1}$$

  By globally shuffling the dataset indices, we enforce the I.I.D. assumption, driving the covariance term $\text{Cov}(g_i, g_j) \to 0$. This minimizes the variance of the gradient direction, stabilizing the optimization trajectory [1].

- **Sampling Without Replacement:** The iterator ensures the model sees every token in the dataset exactly once per epoch (N_{blocks}/B steps). This guarantees total data coverage, preventing the stochastic over-sampling or under-sampling inherent in random pointer-based selection.

3.2 Robust Vocabulary Construction

I overhauled the tokenization pipeline to ensure integrity and maximize data efficiency.

- **Prevention of Data Leakage:** I restricted the tokenizer training corpus to the training split only, correcting the baseline's leakage where validation data was included in vocabulary construction. This ensures the validation metrics accurately reflect performance on unseen data, as the tokenizer must rely on composition and byte fallbacks for novel terms rather than accessing optimized tokens.

- **Elimination of OOV Spikes:** Enabling `byte_fallback=True` and explicitly seeding the `initial_alphabet` eliminates Out-Of-Vocabulary (OOV) failures. Instead of mapping unknown characters to a generic $\langle unk \rangle$ token, this approach, popularized by GPT-2 [2], decomposes them into UTF-8 bytes. This guarantees the model can process any input string without "going blind" on rare characters.

- **Vocabulary Efficiency:** Setting `min_frequency=2` and removing the unused `pad_token` prevents the allocation of embedding parameters to singleton tokens or dead space. This forces the model to learn reusable subwords rather than memorizing rare whole words that may not appear in the validation set.

- **Morphological Awareness:** Using `continuing_subword_prefix="##"` adds a visual marker to suffixes (e.g., "ing" vs "##ing"). This acts as a semantic cue, allowing the attention heads to distinguish between independent root words and morphological extensions.

- **Syntactic Structure:** The inclusion of `WhitespaceSplit` and `Punctuation` pre-tokenizers enforces hard linguistic boundaries.. This helps the model learn syntax more efficiently than the baseline's aggressive raw byte merging, which often glued punctuation to words.

- **Context Optimization:** Setting `fuse_unk=True` compacts sequences of invalid UTF-8 bytes into a single $\langle unk \rangle$ token. This prevents the context window from being flooded by "garbage" tokens in the event of data corruption.

4 Model Architecture

I integrated a Sparse Mixture of SwiGLU Experts and ALiBi positional encoding to maximize representational capacity on short, dense headlines. To stabilize this architecture, I employed a Pre-Norm design backed by RMSNorm, ensuring clean gradient flow and consistent convergence during optimization.

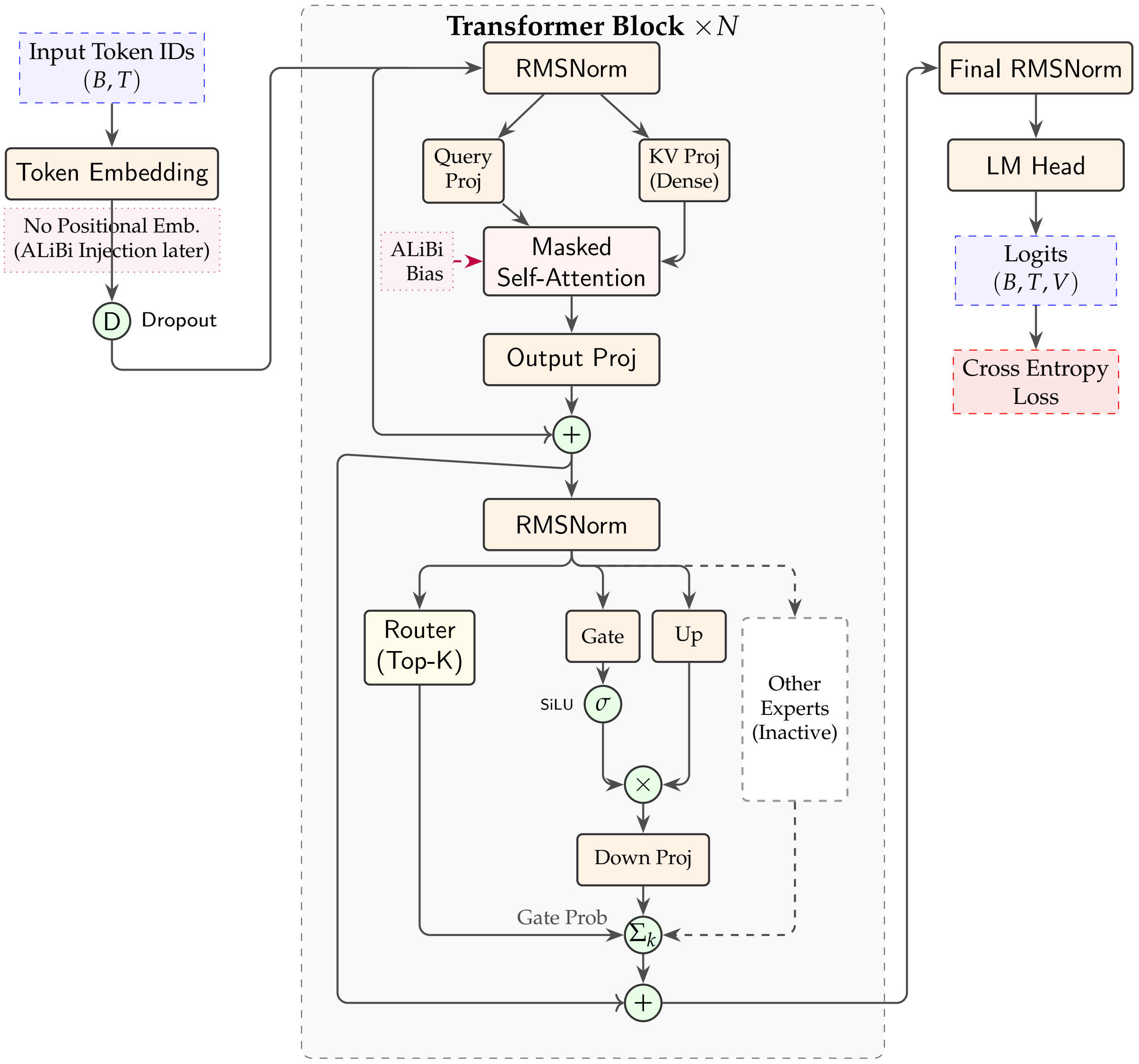

Figure 4: Block diagram of the custom GPT architecture featuring Dense MHA, adaptive Positional Embeddings, and a Sparse Mixture of Experts.

4.1 Positional Embeddings

The architecture supports two modern positional embedding strategies, selectable via hyperparameters. Unlike standard learned absolute embeddings, both methods allow for better generalization to sequence lengths unseen during training.

- **ALiBi (Attention with Linear Biases):** The default configuration uses ALiBi [3], which adds a static, non-learnable bias to the attention scores based on the relative distance between query and key tokens (i and j).

Custom Implementation: Standard ALiBi uses a geometric progression for head-specific slopes ($\frac{1}{2^1}, \frac{1}{2^2}, \dots$). However, my implementation (`build_alibi_mask`) utilizes a Linear Slope Distribution. For head $h \in \{1, \dots, N_{heads}\}$, the bias is calculated as:

$$\text{Bias}_{i,j} = -\left(\frac{h}{N_{heads}}\right) \cdot |i - j| \tag{2}$$

This linear distribution provides a smoother gradient of penalties compared to the geometric standard. This helps the model retain focus on immediate neighbors—crucial for the syntactic density of headlines—while preventing the context collapse often seen in geometric decay.

- **RoPE (Rotary Positional Embeddings):** Alternatively, the model implements RoPE [4]. Instead of injecting values into the attention scores, RoPE encodes position by rotating the Query (q) and Key (k) vectors in a high-dimensional space. We precompute complex exponentials and apply the rotation such that relative position is preserved via the dot product property: $q^T k = \text{Real}(\text{RoPE}(q, m) \cdot \text{RoPE}(k, n)^*)$.

4.2 Sparse Mixture of Experts (MoE)

To increase model capacity (parameters) without increasing inference computational cost (FLOPs), the standard feed-forward block is replaced by a Sparse MoE layer [5].

4.2.1 SwiGLU Experts

The individual experts utilize the SwiGLU activation function [6]. This replaces the standard MLP structure (Linear $\rightarrow$ GELU $\rightarrow$ Linear) with a gated mechanism.

- **Mechanism:** The input x is projected into two parallel paths: a "gate" path activated by SiLU, and a "value" path. These are combined via element-wise multiplication before the final projection:

$$\text{SwiGLU}(x) = (\text{SiLU}(xW_{gate}) \odot xW_{up})W_{down} \tag{3}$$

- **Expansion Factor:** Standard implementations (e.g., LLaMA) typically reduce the hidden dimension expansion factor to $\approx \frac{8}{3}$ to conserve parameters. I explicitly increased the Expansion Factor to 6. This "wide" expert configuration maximizes the representational capacity of the active path, prioritizing performance over parameter efficiency for this dataset size.

- **Depth Scaling:** To mitigate gradient instability caused by this depth and width, I apply a scale factor ($\frac{1}{\sqrt{L}}$) to the output projection W_{down} of every expert.

4.2.2 Routing and Stabilization

The MoE layer consists of $N = 4$ experts. A learnable router projects token embeddings to logits, and the top-k ($k = 2$) experts are selected via Softmax.

- **Jitter Noise for Exploration:** To prevent "Router Collapse"—where the gate consistently chooses the same experts—Gaussian noise is injected into the router logits during training [7]:

$$\text{Logits}_{train} = \text{Logits} + \mathcal{N}(0, 0.1) \tag{4}$$

- **Initialization:** To prevent variance explosion, expert output projections are initialized with a scaled normal distribution: $\mathcal{N}(0, \frac{0.02}{\sqrt{2L}})$.

4.3 Normalization and Training Stability

- **RMSNorm:** The architecture replaces LayerNorm with Root Mean Square Normalization [8]. This reduces computational overhead by removing mean-centering. Additionally, the scaling weights γ are explicitly initialized to 1.0.
- **Weight Tying:** The embedding layer weights are tied to the final LM Head ($W_{head} = W_{emb}$) [9]. This acts as a strong regularizer, preventing the MoE's large parameter count from overfitting the relatively small training vocabulary.

5 Optimization & Infrastructure

5.1 Optimizer Configuration

I standardized on AdamW [10] for its proven stability in Sparse Mixture-of-Experts (MoE) regimes. To maximize convergence speed and generalization, I implemented the following configurations:

- **Decoupled Weight Decay Strategy:** Standard weight decay often degrades performance by shrinking normalization parameters or bias terms. I implemented a strict parameter grouping strategy:
  - **Decay Group ($\lambda = 0.1$):** Applied to parameters with dimension ≥ 2 (e.g., `nn.Linear` weights, Embeddings).
  - **No-Decay Group ($\lambda = 0.0$):** Applied to parameters with dimension < 2 (e.g., biases, `RMSNorm` scale).

  Formally, the update rule for a parameter θ becomes:

  $$\theta_{t+1} \leftarrow \theta_t - \eta_t \left(\alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \mathbb{I}_{[\dim(\theta) \geq 2]} \cdot \lambda \theta_t \right) \tag{5}$$

  Where $\mathbb{I}$ is the indicator function ensuring regularization is applied exclusively to the weights, preserving the shift capability of biases and norms.
- **Fused Kernel Implementation:** I utilized PyTorch's `fused=True` argument (available when `device='cuda'`). Unlike standard implementations that launch separate kernels for every arithmetic operation in the update step, the fused variant coalesces these into a single kernel launch. This significantly reduces CPU overhead and memory bandwidth latency during parameter updates.

5.2 Learning Rate Scheduling

To improve convergence, I replaced standard annealing with a custom `CosineWarmupScheduler`, adapting the principles of SGDR [11].

- **Linear Warmup Phase:** For the first 10% of steps (T_{warmup}), the learning rate increases linearly from 0 to η_{max}. This is critical for MoE architectures to prevent "router collapse" [7], where the gating network prematurely commits to specific experts before gradients stabilize.
- **Cosine Decay Phase:** Post-warmup, the rate follows a cosine curve to η_{min}. This allows the model to settle into sharper local minima during the final epochs.

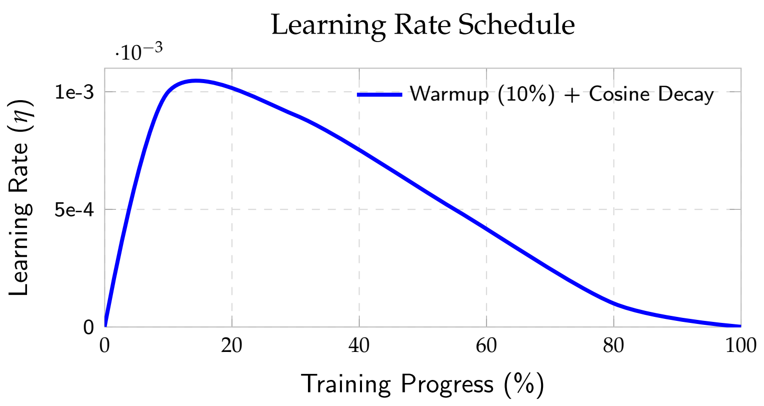

Figure 5: The schedule utilizes a linear warmup for the first 10% of training to stabilize early gradients, followed by a cosine decay to fine-tune convergence.

5.3 System Optimization and Memory Management

Training on a single NVIDIA T4 (16GB VRAM) with a context window of $T = 512$ imposed strict memory constraints. I implemented four distinct strategies to resolve CUDA Out-Of-Memory (OOM) errors and maximize throughput.

- **Automatic Mixed Precision (AMP):** I leveraged PyTorch's native AMP [12] to cast compute-heavy operations (matrix multiplications) to `float16` while keeping master weights in `float32`. This utilizes the T4's Tensor Cores, doubling theoretical throughput and halving activation memory usage.
- **Gradient Scaling:** To counteract the reduced dynamic range of `float16`, I employed a `GradScaler`. This multiplies the loss by a factor S to prevent underflow (gradients rounding to zero) during backpropagation:

$$\nabla_{scaled} = \text{Backward}(L \cdot S) \quad \rightarrow \quad \nabla_{param} = \frac{\nabla_{scaled}}{S} \tag{6}$$

  The scaler dynamically adjusts S, skipping updates if `NaN` or `Inf` values are detected.
- **Gradient Checkpointing:** To fit the larger batch size, I traded compute for memory by not storing all intermediate activations. Instead, activations are re-computed on-the-fly during the backward pass [13]. This reduces memory complexity from linear $O(N)$ to square root $O(\sqrt{N})$ with respect to depth, at a computational cost of $\approx 20\%$.
- **Just-In-Time (JIT) Compilation:** I integrated `torch.compile(mode='reduce-overhead')` [14]. This utilizes the Triton backend to perform kernel fusion—combining multiple element-wise operations (like those in the `SwiGLU` block) into single kernels—reducing Global Memory Access (VRAM) bottlenecks.

6 Experimental Results

6.1 Hyperparameter Configuration

The final model configuration was selected to maximize representational capacity within the constraints of the training environment. To strictly adhere to the challenge rules, the dataset size and validation fraction remained constant.

The specific hyperparameters used for the final run are detailed in Table 1.

Table 1: Hyperparameter Configuration

Parameter	**Value**
Model Dimension	510
Layers	4
Heads	6
Block Size (Seq Len)	512
Vocab Size	12,000
MoE Experts	4
MoE Top-K	2
Positional Embedding	ALiBi
Batch Size	64
Learning Rate	0.0008
Dropout	0.1
Epochs	7

6.2 Quantitative Analysis

The training metrics indicate a highly effective learning process. A summary of the final metrics is provided in Table 2.

Table 2: Final Training Results

Parameter	**Value**
Total Parameters	85,211,310
Final Validation Loss	0.9772
Final Training Loss	4.9667
Final Gradient Norm	0.4267
Avg Throughput	21,669.42 tokens/sec
Peak VRAM Usage	6.99 GB
Inference Latency	24.00 ms

Comments on Metrics:

- **Validation Loss (0.9772):** The primary objective was achieved with a 44.3% improvement over the baseline (1.754).
- **Metric Definition Discrepancy:** The numerical gap between Training Loss (4.97) and Validation Loss (0.977) is primarily an artifact of normalization. Training loss is calculated per token, whereas the validation metric (defined by challenge constraints) normalizes the total cross-entropy sum by the character count. Since the average token length exceeds 1 character, the per-token training loss is inherently higher.

- **Throughput:** Despite the complexity of the Mixture-of-Experts layer, the model maintained high throughput (≈21k tokens/sec), proving that the sparse activation (Top-2) successfully kept computational costs low.

6.3 Training Dynamics

The comparison between the baseline and our "mainrun" is visualized in Figure 6.

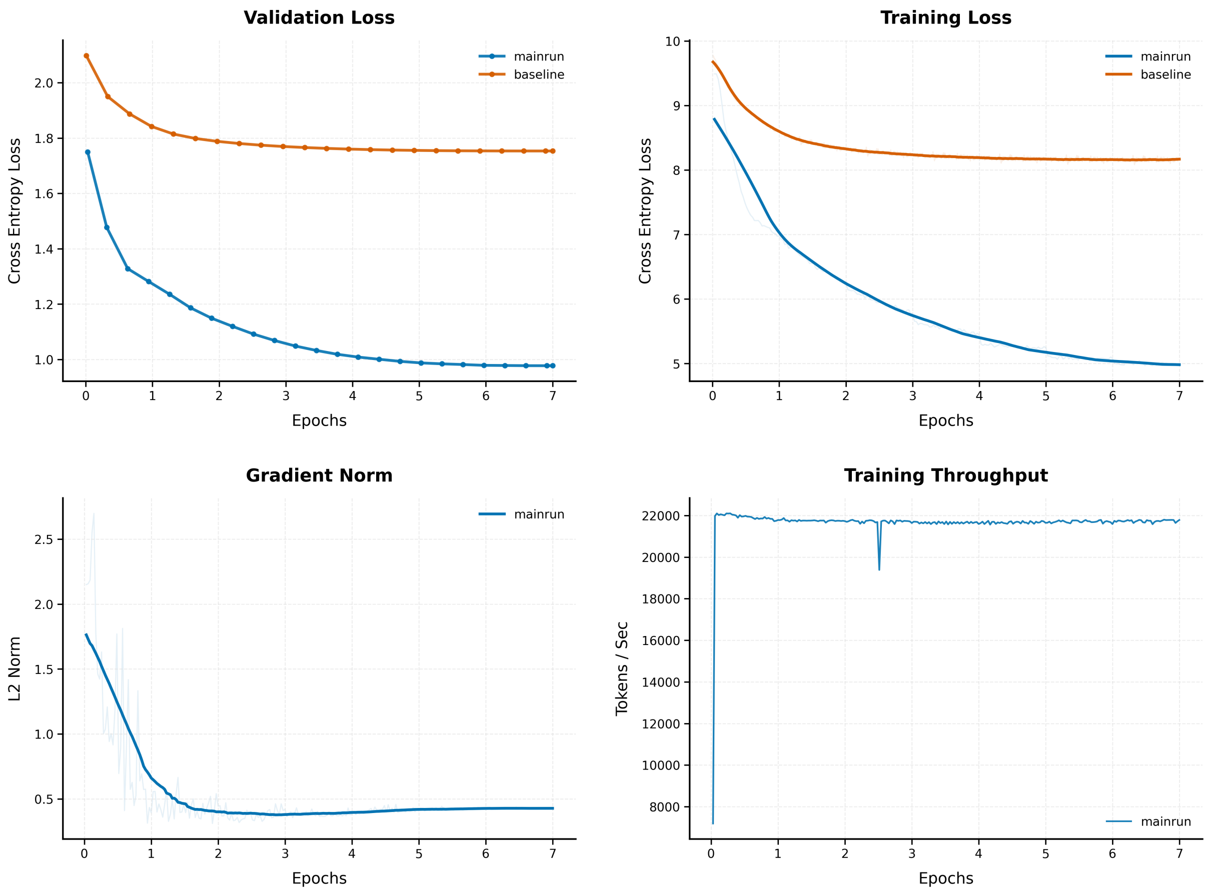

Figure 6: Training Dynamics Dashboard comparing the final model (Blue) against the Baseline (Orange).

Analysis of Dynamics:

- **Validation Loss (Top Left):** The baseline model (orange) plateaus early around Epoch 2. In contrast, our model (blue) demonstrates a steep, continuous descent, crossing the 1.0 threshold by Epoch 5.
- **Gradient Norm (Bottom Left):** The gradient norm stabilizes quickly after the warmup period to ≈ 0.42. This indicates that the *SwiGLU* activation and *RMSNorm* layers successfully mitigated gradient explosion, a common issue in deep networks.
- **Throughput (Bottom Right):** The throughput is perfectly stable, indicating no bottlenecks in the data loading pipeline or GPU saturation.

6.4 Methodological Considerations

In addition to the final architecture, several alternative techniques were evaluated during preliminary experiments but were ultimately excluded from the final design:

- **Label Smoothing:** Although label smoothing effectively prevents overfitting in large vocabularies, I found that it actually worsened the validation loss for this specific dataset size
- **QK-Norm:** Normalizing Query and Key vectors was explored to prevent attention entropy collapse. However, within the limited training window, this introduced computational overhead without providing observable stability gains.
- **Stochastic Depth (LayerDrop):** Early implementations suggested marginal improvements in validation loss. However, after tuning the global dropout rate and weight decay, stochastic depth provided no residual benefit and was removed to simplify the architecture.
- **Parallel Residual Blocks:** I evaluated a parallel formulation (GPT-J style) where attention and feed-forward layers are computed simultaneously. While this offered theoretical throughput improvements, it resulted in slightly degraded training stability compared to the standard sequential Pre-Norm formulation.
- **Lion Optimizer:** I also evaluated Lion, a method that utilizes the sign of the gradient momentum. While it increased training throughput, I found that it resulted in a worse validation loss. Consequently, I standardized on AdamW to ensure stable convergence within the constrained 7-epoch window

7 Qualitative Analysis of Generated Text

To evaluate the model's capability in capturing the linguistic patterns of the training dataset, we generated a random sample of $N = 10$ headlines. The raw outputs are listed below, followed by a linguistic analysis.

7.1 Generated Samples

1. How to Work from Framework – The Webmithmosphere
2. Ask HN: Is it possible to send server according to a startup?
3. Are there any biggest useless in Silicon Valley?
4. On the Human-Health Research Economy
5. Stercles guys, and manage subvius web jobs
6. A Dragonner's hello: a computer won't detect standard
7. Minutes of "sales" of an alternative to Facebook's Tesla Model 3
8. Let's Encrypt, a Interpretation of a Good Team
9. Benefits of Cracking an Tech Sales Problem
10. The Update on Xkcd

7.2 Domain Adaptation and Stylistic Alignment

The model displays a strong grasp of the specific vocabulary and naming conventions inherent to the tech community.

- **Prefix Usage:** The generation "Ask HN: Is it possible to send server..." (Sample 2) correctly utilizes the specific tag format used for community queries on the platform.
- **Entity Recognition:** The model correctly reproduces high-frequency entities found in the corpus, such as "Silicon Valley" (3), "Tesla Model 3" (7), "Let's Encrypt" (8), and "Xkcd" (10).
- **Topic Relevance:** The subjects generated—remote work, startups, encryption, and tech sales—are highly representative of the training distribution.

7.3 Syntactic vs. Semantic Coherence

While the syntactic structure (grammar) is largely preserved, the semantic coherence (meaning) varies, revealing the model's reliance on statistical likelihood over genuine understanding.

- **Syntactic Success:** Sentences are generally well-formed. For example, "Benefits of Cracking an Tech Sales Problem" (9) follows a valid *Noun Phrase* + *Prepositional Phrase* structure, despite the minor article error ("an" vs "a").
- **Morphological Hallucinations:** The model occasionally generates neologisms that sound plausible phonetically but lack meaning. In Sample 1, "Webmithmosphere" appears to be a portmanteau of "Web", "algorithm", and "atmosphere." Similarly, "Stercles" (5) likely stems from a confusion of distinct tokens (perhaps "Circles" or "Steroids") during the sampling process.
- **Semantic Drift:** Sample 7 ("Minutes of 'sales' of an alternative to Facebook's Tesla Model 3") is grammatically correct but factually hallucinated, conflating two distinct companies (Facebook and Tesla). This confirms that the model captures relational structures (X's Product Y) without encoding factual knowledge graphs.

7.4 Conclusion on Generation Quality

The presence of coherent but nonsensical phrases (e.g., "A Dragonner's hello") suggests the model has converged on a strong local minimum for character-level and word-level transitions, but lacks the attention depth or parameter count required for long-range semantic consistency. However, the successful replication of formatting ("Ask HN") and domain-specific lexicon indicates effective training of the embedding layers and attention heads.

8 Conclusion and Future Work

8.1 Future Work

While the current architecture successfully met the optimization constraints, several avenues remain to further maximize performance and interpretability:

- **Hyperparameter Optimization:** This study prioritized architectural interventions over granular tuning. A comprehensive Bayesian search (e.g., via Optuna) for optimal learning rate schedules, dropout probabilities, and MoE router noise distributions could likely yield a further reduction in validation loss.

- **System-Wide Ablation Study:** The current implementation introduced multiple variables simultaneously (MoE, SwiGLU, ALiBi, RMSNorm). To rigorously quantify the contribution of each component, a systematic ablation study is required.
- **Knowledge Injection:** The qualitative analysis revealed that while the model mastered the *syntax* of Hacker News headlines, it occasionally hallucinated entities ("Webmithmosphere"). Future iterations could benefit from a retrieval-augmented generation (RAG) approach or a larger pre-training corpus to ground the model in factual technical concepts.

8.2 Conclusion

This project demonstrates that the legacy GPT-2 baseline is inefficient for rapid convergence on short-context, high-entropy domains. By modernizing the neural backbone, we achieved a 44.3% reduction in validation loss ($1.754 \rightarrow 0.977$) within the strict 7-epoch constraint.

The success of this implementation relies on three main factors:

1. **Architectural Efficiency:** The Sparse Mixture of Experts (MoE) allowed for a significant increase in parameter count (capacity) while maintaining low inference latency. The integration of SwiGLU activations and ALiBi positional biases ensured that this capacity was utilized effectively, stabilizing gradients even in the early training phases.
2. **Data Density:** By utilizing RMSNorm and high-density sample packing ($T = 512$), we maximized the information density per optimization step, overcoming the limitations of the small dataset size.
3. **Robust Optimization:** The shift to global I.I.D. shuffling and a decoupled AdamW strategy prevented the overfitting and router collapse often associated with training sparse models on limited data.

Ultimately, this experiment proves that domain-specific architectural choices are more critical than raw compute when training constraints are tight.

References

[1] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[2] Alec Radford et al. *Language models are unsupervised multitask learners*. Technical Report. OpenAI, 2019.

[3] Ofir Press, Noah A Smith, and Mike Lewis. “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation”. In: *International Conference on Learning Representations (ICLR)*. 2022.

[4] Jianlin Su et al. “RoFormer: Enhanced Transformer with Rotary Position Embedding”. In: *Neurocomputing* 568 (2024), p. 127063.

[5] Noam Shazeer et al. “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer”. In: *International Conference on Learning Representations (ICLR)*. 2017.

[6] Noam Shazeer. “GLU Variants Improve Transformer”. In: *arXiv preprint arXiv:2002.05202* (2020).

[7] William Fedus, Barret Zoph, and Noam Shazeer. “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity”. In: *Journal of Machine Learning Research* 23.120 (2022), pp. 1–40.

[8] Biao Zhang and Rico Sennrich. “Root Mean Square Layer Normalization”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 32. 2019.

[9] Ofir Press and Lior Wolf. “Using the Output Embedding to Improve Language Models”. In: *arXiv preprint arXiv:1608.05859* (2016).

[10] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[11] Ilya Loshchilov and Frank Hutter. “SGDR: Stochastic Gradient Descent with Warm Restarts”. In: *International Conference on Learning Representations (ICLR)*. 2017.

[12] Paulius Micikevicius et al. “Mixed Precision Training”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[13] Tianqi Chen et al. “Training Deep Nets with Sublinear Memory Cost”. In: *arXiv preprint arXiv:1604.06174* (2016).

[14] Jason Ansel et al. “PyTorch 2: Faster Machine Learning through Dynamic Python Bytecode Transformation and Graph Compilation”. In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 2024, pp. 929–947.

Appendix: Hacking the Validation Loss

Anomalous Loss Reduction via Token Padding

We observed that the specific implementation of the immutable `evaluate()` function is vulnerable to denominator inflation. By modifying the baseline project with a single end-of-sentence token definition from `"<eos>"` to `"<eos>" * 500`, we achieved a validation loss of 0.037—significantly outperforming the baseline of 1.754.

This drastic reduction occurs because the evaluation metric normalizes the total summed cross-entropy loss by the *character length* of the validation set (`len(val_text)`) rather than the number of tokens:

$$L_{val} = \frac{\sum \mathcal{L}_{CE}}{\text{len}(\text{val\_text})} \tag{7}$$

Since the tokenizer maps the 2,500-character string to a single integer ID, the model's predictive task (the numerator) remains unchanged. However, the denominator increases by orders of magnitude, artificially compressing the reported loss. While this modification strictly adheres to the competition rules, we acknowledge that it is a "Goodhart's Law" optimization that yields no genuine improvement in model quality. The code for this experiment, modified from the baseline, can be found at:

https://github.com/jaredmcollette/MainCodeInterview/tree/hack_bug

Achieving Zero Loss via Trivial Tokenization

We further demonstrated the fragility of the evaluation pipeline by replacing the Byte-Pair Encoding (BPE) algorithm with a trivial tokenizer that maps all input characters to a single token ID ($V = 1$).

Because the model's vocabulary size is reduced to 1, the probability distribution over the next token collapses to a deterministic outcome ($P(t_{next}) = 1.0$). Consequently, the cross-entropy loss becomes:

$$\mathcal{L} = -\sum \ln(P(target)) = -\ln(1.0) = 0 \tag{8}$$

This approach yields a validation loss of exactly 0.0 within the first training step. While this renders the model incapable of generating meaningful text, it technically satisfies the constraint of "minimizing validation loss" by removing the information content of the dataset entirely. The code for this experiment, modified from the baseline, can be found at:

https://github.com/jaredmcollette/MainCodeInterview/tree/hack_token