Mainrun – Report: Recommendations for Reducing Validation Loss

This report outlines the recommendations for Mainrun assessment framework, improving the performance of a transformer-based language model trained on the Hacker News Titles dataset.

The primary objective is to reduce validation loss, thereby enhancing the model's generalization and predictive capability.

Something to note

I noticed that in the evaluate() function, it divides the total loss by *len(val_text)*, which is the number of characters in the raw validation string.

Shouldn't it be the number of tokens, *len(val_ids)*?

I guess this is the reason for having a high training loss compared to the reported validation loss. Also, it kind of makes the validation loss inconsistent (too big or too small depending on tokenization).

Anyways, continued with the current evaluate() function without any changes :)

Some Quick Wins

Recommendation 1: Switch to a More Advanced Optimizer

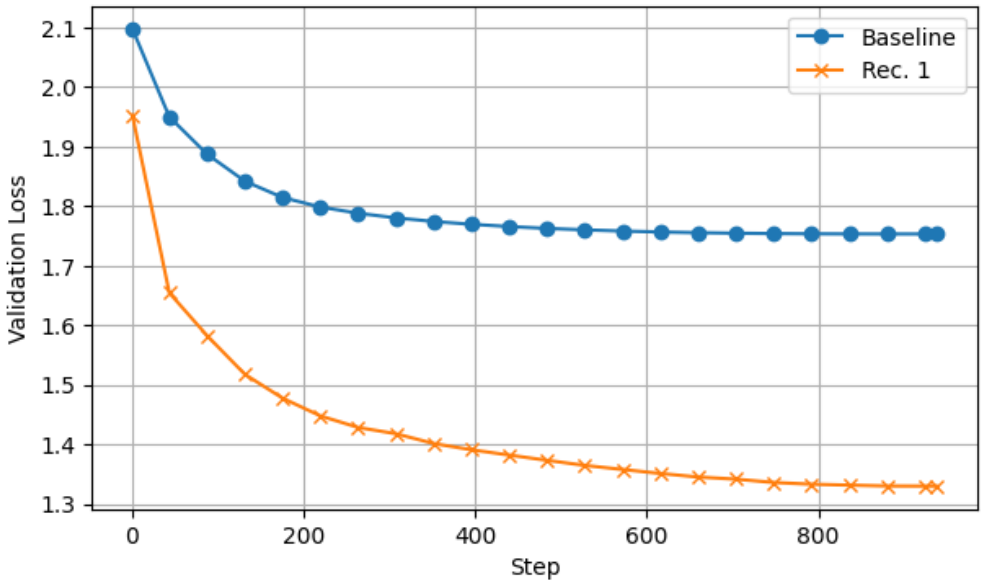
The current model uses SGD as its optimizer. For transformer architectures, AdamW is widely recognized as the preferred optimizer due to its adaptive learning rates and effective handling of sparse gradients. Adopting AdamW is expected to result in faster convergence and more stable training dynamics.

The learning rate is a critical hyperparameter. For AdamW, lower learning rates often yield better results.

Action:

- Replaced the current SGD optimizer with AdamW.
- Lowered the learning rate (lr) to 0.001.
- Used a nonzero weight decay (weight_decay), 0.01.

Validation loss comparison with Baseline.



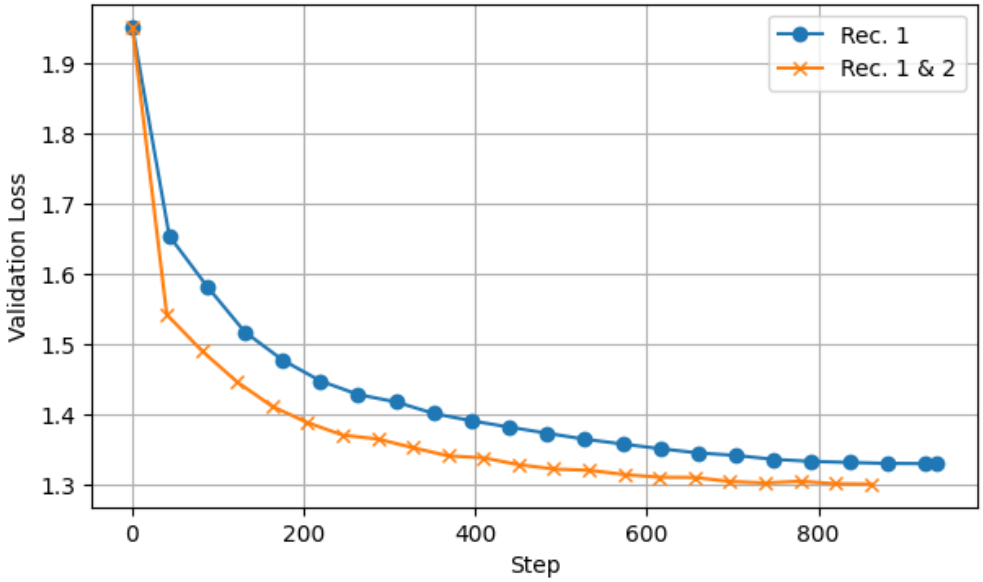
Recommendation 2: Tune the vocabulary size

A very small vocabulary (e.g., 8k) leads to words being split into multiple sub-tokens, which increases sequence length, slows training, and reduces accuracy. On the other hand, overly large vocabularies add many rare tokens, inflating the embedding matrix and softmax layer without providing meaningful gains.

Experimented with different vocabulary sizes (vocab_size) (e.g., 8k, 16k, 32k) and selected 32k. While a 64k vocabulary could potentially improve performance further, it was not pursued here due to memory constraints and the compromises it would impose on model size.

Action:

- Experimented with different vocabulary sizes (vocab_size) (e.g., 8k, 16k, 32k) and selected 32k.

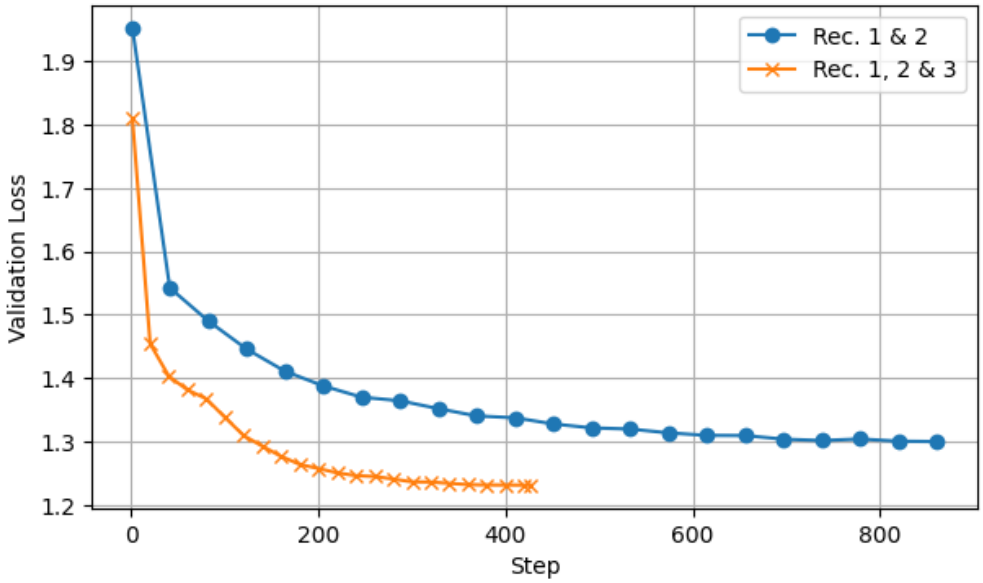


Recommendation 3: Extend Input Sequence Length

The current sequence length (block_size) is set to 128 tokens. Extending the context window enables the model to learn longer dependencies.

Action:

- Increased block_size to 256, (my computer couldn't accommodate further increase)



Changes to the Model Architecture

Recommendation 4:

4. 1 Replace LayerNorm with RMSNorm

Rationale:

- Stability with fewer parameters. RMSNorm normalizes by the root-mean-square of activations without centering, reducing parameter and compute overhead compared to LayerNorm while providing comparable or better training stability in decoder-only LMs.
- Smoother optimization at small/mid scales. Empirically, RMSNorm reduces gradient noise amplification and improves conditioning, often yielding lower perplexity for modest-sized GPTs under tight training budgets.

Action:

- All LayerNorm layers in transformer blocks were replaced with RMSNorm.

Expected impact: Small but consistent reduction in validation loss via improved gradient flow and reduced normalization overhead.

4.2 Replace Absolute Positional Embeddings with RoPE (Rotary Position Embeddings)

Rationale:

- Better inductive bias for relative order. RoPE encodes positions through rotations in Q/K space, naturally modeling relative distances—beneficial for next-token prediction and generalization beyond seen lengths.
- Parameter efficiency. Eliminates a full learned position table over block_size, slightly reducing parameters and removing a potential source of overfitting.
- Improved extrapolation and attention sharpness. RoPE maintains dot-product geometry across positions, which can produce crisper attention patterns, improving likelihood under strict epoch budgets.

Action:

- Removed learned absolute pos_emb and applied rotary phase to the Q and K projections in attention.

Additional:

- Embed Scaling and Attention Harmonization: Scaled token embeddings by $d\_model^{-0.5}$ before entering the model to match the scale used in attention layers. This prevents early activation blow-up, keeps training dynamics stable, and helps the model optimize more smoothly, especially with AdamW and tied output weights.

- Torch 2 Scaled Dot-Product Attention for Stability: Replaced manual attention computation with PyTorch's fused scaled_dot_product_attention for improved numerical stability and more reliable gradients, resulting in better generalization and slightly lower validation loss, particularly on longer sequences.

Expected impact: Improved sample-efficiency and generalization; commonly lowers validation loss vs. learned absolute embeddings at this scale.

4.3 Upgrade MLP from GELU to SwiGLU (Gated MLP)

Rationale:

- Gating improves expressivity per parameter. SwiGLU multiplies a SiLU-activated gate with a value pathway, enabling more selective feature amplification/suppression than vanilla GELU.
- Across many decoder stacks, SwiGLU produces modest but robust perplexity reductions at comparable compute.
- Stability under dropout. SwiGLU often tolerates moderate dropout with less degradation, helpful when regularization is needed within fixed training steps.

Action:

- The 2-layer GELU MLP (with ~4× hidden size) was replaced by a SwiGLU feed-forward (gated) block with typical ~8/3× expansion in the gated branch.

Expected impact: Consistent small-to-moderate validation-loss improvement via better feature selection and smoother gradients.

4.4 Use Parallel Residual Block Topology (NeoX-style)

Rationale:

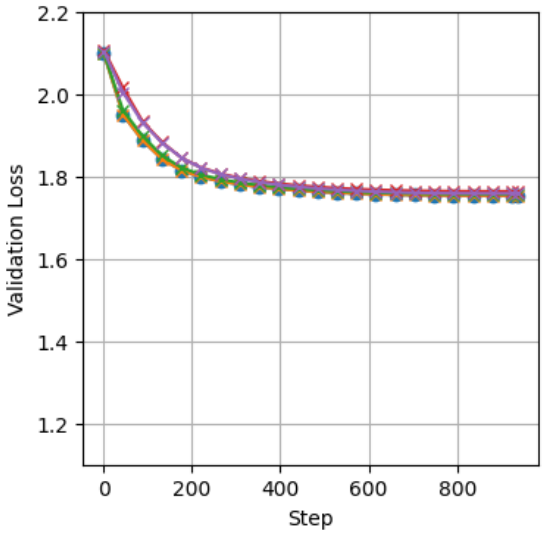
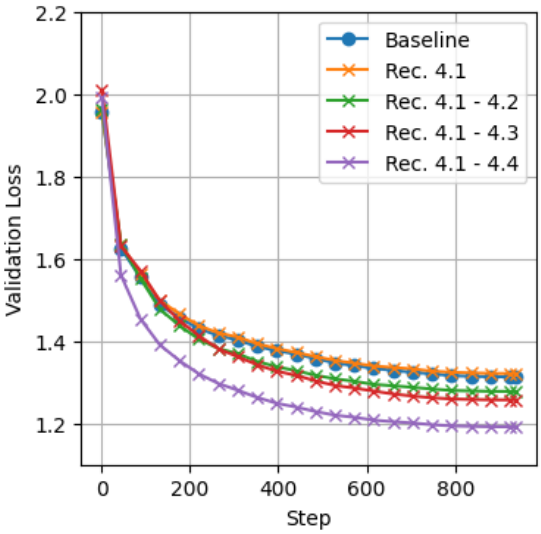
- Easier optimization. Parallel residual pathways decouple the two sub-layers' gradients and reduce depth-wise gradient attenuation.
- Faster representational mixing per layer. With the same number of layers, you effectively get two transformations in parallel, improving capacity per step, useful when epochs are capped.

Action:

- Within each transformer block, attention and MLP are computed from (normalized) inputs in parallel and their outputs are summed with the residual, rather than strictly sequential attn→MLP.

Expected impact: Slightly improved convergence speed and stability; typically a small validation-loss drop at fixed compute.

The plots below show how the validation loss curve changes with each added recommendation compared to the baseline. All the hyperparameters were default except with AdamW (learning rate = 0.001 and weight decay = 0.01).

With SDG	With AdamW (Rec. 1)
	

Notably, these recommendations improved validation loss only when using the AdamW optimizer. With SGD, no improvements were observed.

AdamW adaptively rescales updates and decouples weight decay, which:

- Stabilizes layers with different gradient scales,
- Preserves small but useful embedding gradients,
- Keeps updates stable with RMSNorm, SwiGLU, RoPE.

SGD, without per-parameter adaptation, applies the same LR to all parameters. Without careful tuning of LR, momentum, warmup, and weight decay:

- Some layers under-train,
- Others destabilize,
- Validation gains from the architecture don't appear.

According to what I could find, it's not that the architecture "only works with AdamW", but that SGD needs transformer-specific fine-tuning to match its performance.

Side-Effects of Recommendation 4 on Model Size & Compute

Parameters:

- Removed absolute positional table (minor reduction).
- RMSNorm similar/better efficiency than LayerNorm (slightly fewer parameters).

- SwiGLU may have comparable or slightly higher MACs than 4× GELU depending on chosen hidden width; expected compute change is small relative to total attention cost.

Throughput:

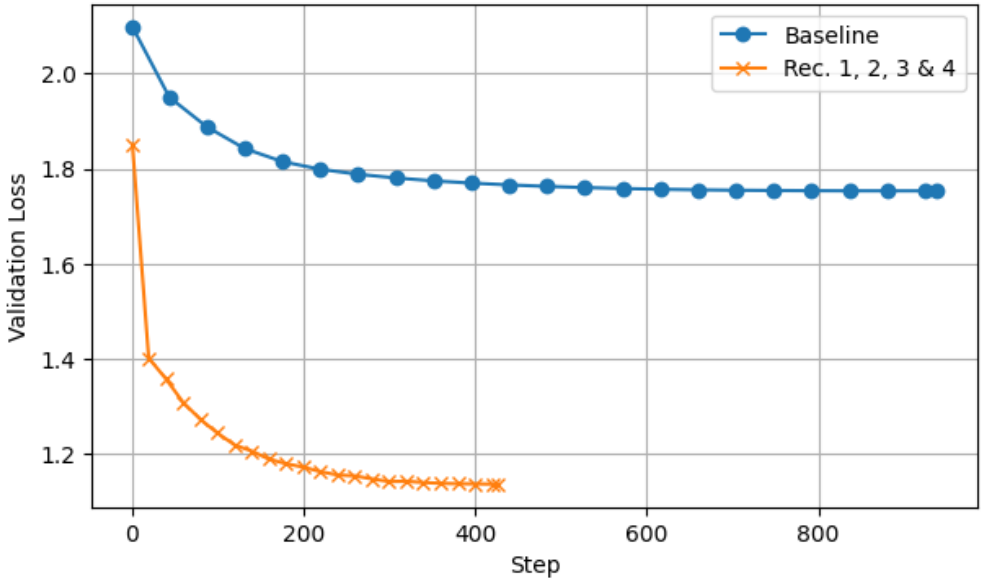
- SDPA often improves runtime and reduces memory; useful but not required for this task.

Memory:

- Comparable to baseline; minor improvements from fused attention kernels.

Overall, compute footprint remains close to baseline while improving inductive biases and optimization stability.

The final validation loss, combining all 4 recommendations was **1.137.**



Attempted changes with no success :(

- Switch the scheduler from CosineAnnealingLR to CosineAnnealingWarmRestarts. No noticeable improvement, requires further analysis.
- Add domain user symbols (URLs, tech tokens):
  Things like http, ://, .com, C++, API, GPU, GPT-4 were whitelisted so they become single pieces when present. Rationale: HN titles are full of these; merging them reduces fragmentation and makes patterns easier to learn.
- Tune n_layer, n_head and d_model. Slight improvement by increasing the values but considering the trade-off of compromising the model size further, did not pursue.
- Swap BPE → SentencePiece Unigram (with byte-fallback + NFKC), thinking Unigram would give more stable, often shorter segmentations on noisy, mixed tech text like HN titles.

Prepared by:

Dilantha Haputhanthrige
Date: 4 October 2025