Neuroplastic Transformers: Dynamic Architecture Adaptation During Language Model Training

Abhisar Mehta

Abstract

We present the Neuroplastic Transformer, a language model architecture that dynamically modifies its own structure during training. Rather than fixing the number of layers, attention heads, and hidden dimensions before training begins, our approach starts from a minimal 2-layer, 6.9M-parameter seed and uses gradient-derived utility signals to make function-preserving structural modifications: splitting high-utility attention heads, duplicating high-complexity layers, merging redundant heads, and pruning inactive layers. We train on a 10-billion-token subset of FineWeb-Edu for 30,000 steps on a single A100 GPU in approximately 2 hours and 16 minutes. The model grows to 19 layers and 10.4M parameters, with the controller executing 236 structural modifications. The resulting architecture is non-uniform: middle layers develop 3 attention heads while early and late layers retain 2, a pattern that emerges entirely from training dynamics without any architectural prior. We release the full implementation, training infrastructure, and visualization toolkit.

1 Introduction

Neural architecture search and related methods typically treat model structure as a hyperparameter to be optimized before training. The practitioner selects a layer count, head count, and hidden dimension, trains the model, and evaluates. If the architecture is wrong, the entire training run is wasted. This is expensive, and the search space grows combinatorially when per-layer configurations are considered.

Biological neural networks do not work this way. Synaptic connections form and dissolve throughout development and learning. Cortical regions that receive more stimulation develop denser connectivity. Unused pathways are pruned. The architecture is not separate from the learning process; it is part of it.

We take a step toward this paradigm for transformers. Our Neuroplastic Transformer begins training as a 2-layer, 2-head model with 128-dimensional embeddings. Every 500 training steps, a plasticity controller inspects gradient-derived signals and executes structural modifications. These modifications are function-preserving: the model's output does not change at the moment of modification, but new capacity is available for subsequent gradient updates to exploit.

The system consists of four components: (1) a transformer decoder with independent per-head projections and learnable residual scaling, (2) a plasticity controller that makes growth and pruning decisions, (3) a distributed training loop with DDP support and synchronized utility scores, and (4) a visualization toolkit (npviz) for recording and inspecting architectural evolution.

After 30,000 steps, the model has grown to 19 layers with 10.4 million parameters, achieving a cross-entropy loss of 4.17. The controller executed 236 structural events: 110 layer splits, 93 layer prunes, 14 head splits, and 19 head merges. The result is a non-uniform architecture that emerged entirely from the training signal.

2 Related Work

Neural Architecture Search. NAS methods [Zoph and Le, 2017, Liu et al., 2019] search over architectures by training many candidates. These require enormous compute budgets since each candidate must be trained separately. Our approach modifies a single model during a single training run.

Progressive Growing. Net2Net [Chen et al., 2016] and related methods [Wu et al., 2019] define function-preserving transformations that widen or deepen networks mid-training. Our work extends this with automated decisions driven by utility signals rather than manual scheduling.

Pruning. Structured pruning methods [Voita et al., 2019, Michel et al., 2019] remove attention heads or layers after training. We integrate pruning into the training loop itself, so freed capacity can be reallocated through subsequent growth operations.

Dynamic Networks. Mixture-of-experts models [Shazeer et al., 2017] route inputs through different subnetworks but keep the architecture fixed. Our approach changes the architecture itself.

3 Architecture

3.1 Independent-Head Attention

Standard multi-head attention uses fused $W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ matrices partitioned across heads. This

makes it impossible to add or remove a single head without reshaping shared tensors. We instead implement each attention head as an independent module:

$$\text{Head}_i(x) = W_O^{(i)} \,\text{softmax}\left(\frac{W_Q^{(i)} x \cdot (W_K^{(i)} x)^\top}{\sqrt{d_{\text{head}}}}\right) W_V^{(i)} x \tag{1}$$

where $W_Q^{(i)}, W_K^{(i)}, W_V^{(i)} \in \mathbb{R}^{d_{\text{head}} \times d_{\text{model}}}$ and $W_O^{(i)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$ are per-head. The multi-head output is the sum:

$$\text{MHA}(x) = \sum_{i=1}^{H} \text{Head}_i(x) \tag{2}$$

This additive formulation means adding or removing a head changes only the terms in the sum. It costs slightly more memory than fused projections (separate bias-free linear layers per head) but enables all plasticity operations described in Section 4.

Each head retains its output tensor's gradient during the backward pass (`retain_grad()`) so we can compute per-head utility scores without hooks.

3.2 Learnable Residual Scaling

Each transformer layer has a learnable scalar α controlling its residual contribution:

$$\mathbf{h} = \mathbf{x} + \alpha \cdot \text{MHA}(\text{LN}(\mathbf{x})) \tag{3}$$

$$\mathbf{y} = \mathbf{h} + \alpha \cdot \text{FFN}(\text{LN}(\mathbf{h})) \tag{4}$$

Original layers are initialized with $\alpha = 1.0$. Newly split layers receive $\alpha = 0$, making the insertion a strict no-op. The optimizer then trains α upward if the layer is useful, or it stays near zero and gets pruned.

This is the key mechanism that makes layer insertion function-preserving. Without it, duplicating a layer would double its contribution and spike the loss.

3.3 Feed-Forward Network

Each layer's FFN is a standard two-layer network with GELU activation:

$$\text{FFN}(x) = W_2 \,\text{GELU}(W_1 x) \tag{5}$$

where $W_1 \in \mathbb{R}^{4d_{\text{model}} \times d_{\text{model}}}$ and $W_2 \in \mathbb{R}^{d_{\text{model}} \times 4d_{\text{model}}}$ (FFN multiplier of 4.0). The FFN accounts for the majority of parameters per layer.

3.4 Weight Tying

The output projection (language model head) shares weights with the token embedding: $W_{\text{lm_head}} = W_{\text{emb}}^\top$. This reduces parameter count and is standard practice for small language models. When the hidden dimension grows, both the embedding and LM head are zero-padded together.

Algorithm 1 Plasticity Controller Step

Require: model M, thresholds τ_{split}, τ_{prune}, τ_{merge}, τ_{layer}, τ_α

for each layer l in M **do**
  for each head i where $\bar{u}_i < \tau_{\text{prune}}$ and $H_l > 1$ **do**
    PruneHead(l, i)
  end for
  for each pair (i, j) where $\text{cossim}(i, j) > \tau_{\text{merge}}$ **do**
    MergeHeads(l, i, j)
  end for
end for
for each layer l where $|\alpha_l| < \tau_\alpha$ and $L > 1$ **do**
  PruneLayer(l)
end for
for each layer l, each head i where $\bar{u}_i > \tau_{\text{split}}$ **do**
  SplitHead(l, i) if under param budget
end for
for each layer l where normalized complexity $> \tau_{\text{layer}}$ **do**
  SplitLayer(l) if under layer/param cap
end for

4 Plasticity Operations

4.1 Utility and Complexity Signals

After each backward pass, we compute two signals.

Per-head utility. For each attention head i:

$$u_i = \|\nabla_{\mathbf{o}_i} \mathcal{L}\| \cdot \|\mathbf{o}_i\| \tag{6}$$

where $\mathbf{o}_i$ is the head's output tensor. The gradient norm captures how much the loss depends on the head; the output norm captures how much the head is contributing. Scores are normalized to $[0, 1]$ and smoothed with an EMA ($\beta = 0.99$).

Per-layer complexity. The mean gradient norm across all parameters in a layer, also EMA-smoothed. High complexity indicates the layer is a computational bottleneck.

4.2 Decision Logic

The plasticity controller runs every 500 steps after a 1,000-step warmup. Operations execute in a fixed order: prune heads, merge heads, prune layers, split heads, split layers, grow dimensions. Pruning runs first to free parameter budget.

4.3 Function-Preserving Modifications

Head Split. Clone the parent head's W_Q, W_K, W_V with small Gaussian noise ($\sigma = 0.01$) to break symmetry. Both the parent's and child's W_O are multiplied by 0.5, so the sum of their outputs equals the original head's output.

Head Merge. Average the W_Q, W_K, W_V of two heads; sum their W_O. The merged head approximates the combined output of both originals.

Head Prune. Delete the head module entirely. This is not strictly function-preserving (the head's contribution is lost), but is only triggered when utility is below 0.05, meaning the contribution is negligible.

Layer Split. Deep-copy the layer and insert the copy immediately after. Set the copy's $\alpha = 0$. Reset the copy's utility and complexity scores to zero. The copy contributes nothing until α is trained upward.

Layer Prune. Delete the layer if $|\alpha| < 0.01$. Like head pruning, this removes a negligible contribution.

Dimension Growth. Zero-pad all weight matrices to increase d_{model} by 64. Embeddings, LayerNorms ($w_{\text{new}} = 1, b_{\text{new}} = 0$), attention projections, and FFN layers are all padded so that new dimensions have no effect until trained. Triggered only when 80% of heads exceed the split threshold.

4.4 Post-Modification Recovery

After any architectural change, the system: (1) rebuilds the AdamW optimizer, migrating momentum/variance state for surviving parameters by matching on tensor identity, (2) re-wraps the model in DDP if running distributed (the old DDP wrapper holds stale parameter references), and (3) logs the event to npviz for later visualization.

5 Training Infrastructure

5.1 Distributed Training

The training loop supports single-GPU, multi-GPU (`torchrun`), and multi-node configurations through PyTorch DDP. A critical requirement for distributed plasticity is that all ranks must make identical structural decisions. We achieve this by all-reducing utility scores and complexity scores before each plasticity step:

```
# Allreduce so all ranks decide identically
dist.all_reduce(utility_scores, op=ReduceOp.AVG)
dist.all_reduce(complexity_scores, op=ReduceOp.AVG)
dist.broadcast(alpha, src=0)
```

After an architecture change, the DDP wrapper is destroyed and rebuilt since it caches parameter references that become stale when layers are added or removed.

5.2 Data Pipeline

We support two data paths. The fast path uses a pre-tokenized binary file (`.bin`, uint16, memory-mapped with NumPy), sharded across DDP ranks by contiguous chunks. The fallback path streams from HuggingFace Datasets, tokenizing on the fly with the GPT-2 BPE tokenizer. Both paths pack tokens into fixed-length sequences of 512 tokens with a stride of 512 (no overlap).

For our experiments, the model streamed directly from the FineWeb-Edu `sample-10BT` split [Penedo et al., 2024].

5.3 Mixed Precision and Gradient Handling

Training uses PyTorch's `autocast` with float16 on CUDA and a `GradScaler` for loss scaling. Gradients are accumulated over 2 micro-steps (effective batch size $64 \times 2 = 128$ sequences, or 65,536 tokens per step). Gradient norms are clipped to 1.0 after unscaling.

5.4 Learning Rate Schedule

We use cosine decay with linear warmup:

$$\text{lr}(t) = \begin{cases} \eta_{\max} \cdot t / T_{\text{warm}} & t < T_{\text{warm}} \\ \frac{\eta_{\max}}{2}\left(1 + \cos\left(\frac{\pi(t - T_{\text{warm}})}{T_{\max} - T_{\text{warm}}}\right)\right) & \text{otherwise} \end{cases} \tag{7}$$

with $\eta_{\max} = 6 \times 10^{-4}$, $T_{\text{warm}} = 300$ steps, $T_{\max} = 30{,}000$ steps. AdamW weight decay is 0.1 for all parameters except biases, LayerNorm parameters, and α values.

5.5 Visualization: npviz

We developed npviz, a companion toolkit for recording and visualizing architectural evolution. It consists of:

- **Recorder**: attaches to the model and logs architecture snapshots, rewiring events, per-head importance scores, and training metrics to JSONL files at each logging interval.
- **Timeline**: data model for replaying architectural history.
- **Plots**: generates loss curves, architecture evolution diagrams, layer detail views, gradient norm plots, and learning rate schedules.
- **Web Viewer**: interactive browser-based visualization.
- **Video**: renders architecture evolution as an animation.

6 Experimental Setup

6.1 Hardware

Training was performed on a single NVIDIA A100 40GB GPU (`a2-highgpu-1g` instance on Google Cloud Platform, `europe-west4-a` zone). Total training wall time was approximately 2 hours and 16 minutes (20:00 to 22:16 UTC on April 4, 2026).

6.2 Configuration

Table 1 lists all hyperparameters.

Table 1: Training and plasticity hyperparameters.

Parameter	Value
Initial architecture	
d_{model}	128
d_{head}	64
Initial layers	2
Initial heads/layer	2
FFN multiplier	4.0
Vocabulary size	50,257 (GPT-2 BPE)
Max sequence length	512
Training	
Optimizer	AdamW
Peak learning rate	6×10^{-4}
Weight decay	0.1
Schedule	Cosine with 300-step warmup
Batch size	64
Gradient accumulation	2 steps
Effective batch (tokens)	65,536
Max gradient norm	1.0
Total steps	30,000
Precision	float16 (autocast)
Plasticity	
Warmup	1,000 steps
Interval	500 steps
Head split threshold	$\bar{u}_i > 0.8$
Head prune threshold	$\bar{u}_i < 0.05$
Head merge threshold	cosine sim > 0.95
Layer split threshold	normalized complexity > 0.85
Layer prune threshold	$\lvert\alpha\rvert < 0.01$
Dim growth saturation	80% heads above split threshold
Dim growth increment	64
Utility EMA decay	0.99
Hard caps	
Max layers	24
Max heads/layer	16
Max d_{model}	1,024
Max parameters	100M

7 Results

7.1 Training Loss

Figure 1 shows the training loss curve annotated with plasticity events. Loss drops rapidly from 10.69 to 5.32 during the first 2,000 steps as the model and its architecture co-adapt. The curve continues downward through the stable phase (steps 5,000–20,000), reaching 4.32 at step 10,000. The second growth phase around step 25,000 does not produce a visible loss spike, confirming that function-preserving modifications work as intended. Final loss is 4.17 at step 30,000.

7.2 Architecture Evolution

Figure 2 shows the evolution of layer count, parameter count, and total head count over training. Three distinct phases are visible.

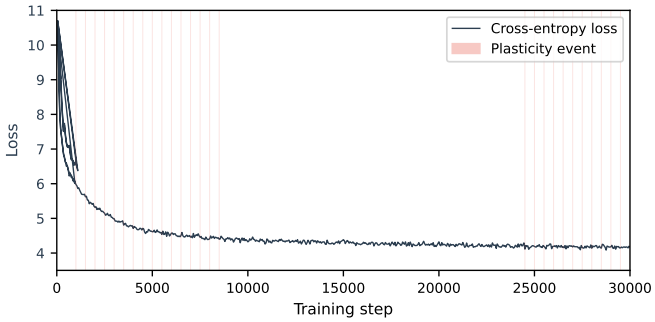

Figure 1: Training loss over 30,000 steps. Red vertical lines mark steps where plasticity operations occurred. No loss spikes are visible at modification points.

Table 2: Architecture at key training milestones.

Step	Layers	Heads	Params	Loss
0	2	4	6.9M	10.69
1,000	2	4	6.9M	6.61
2,000	6	15	7.8M	5.32
5,000	10	21	8.5M	4.61
10,000	9	19	8.3M	4.32
20,000	9	19	8.3M	4.32
25,000	19	46	10.6M	4.19
30,000	19	42	10.4M	4.17

Phase 1: Rapid growth (steps 1,000–5,000). The first plasticity event at step 1,000 splits layer 0 (the only layer with normalized complexity 1.0), creating a 3-layer model. By step 2,000, head splitting and layer splitting produce a 6-layer model with non-uniform heads [2,4,2,2,2,3]. Growth continues to 10 layers and 8.5M parameters by step 5,000.

Phase 2: Stable plateau (steps 5,000–20,000). The architecture stabilizes at 9 layers and 8.3M parameters. The controller continues running but growth and pruning roughly cancel out. Zero events occur between steps 10,000 and 20,000, indicating the architecture found a locally stable configuration for the available hidden dimension.

Phase 3: Late expansion (steps 20,000–30,000). Beginning at step 20,000, layer splits resume. A burst of 192 events between steps 25,000 and 30,000 (84 layer prunes, 88 layer splits, 13 head merges, 7 head splits) rapidly restructures the model to 19 layers. Many newly created layers are immediately pruned in the next cycle, but the net effect is a doubling of depth.

Table 2 provides exact values at key milestones.

7.3 Structural Events

Figure 3 shows the timeline of all 236 structural events colored by type. Table 3 breaks down event counts by training phase.

The late expansion phase (steps 25,000–30,000) ac-

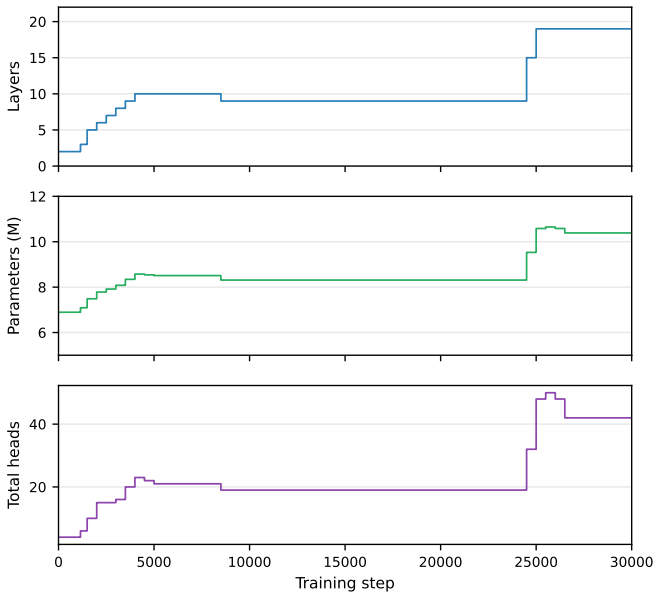

Figure 2: Architecture evolution: layer count, parameter count, and total attention heads over training. Three phases are visible: rapid growth, stable plateau, and late expansion.

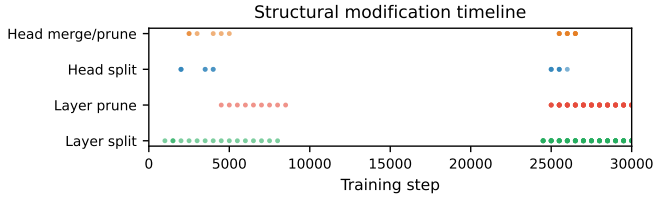

Figure 3: All 236 structural modifications over training. Layer splits and prunes dominate, with a dense cluster in the late expansion phase (steps 25,000–30,000).

counts for 81% of all events. At each plasticity step during this phase, the controller prunes 8 near-zero-alpha layers and simultaneously splits 8 high-complexity layers, producing a rapid churn. This is visible in the training log: at step 30,000 alone, 8 layers are pruned and 8 are split in a single controller invocation.

7.4 Gradient Dynamics

Figure 4 shows gradient norms and learning rate over training. Gradient norms rise from 0.93 to approximately 1.0 during warmup (reflecting the gradient clip), then stabilize between 0.6 and 0.8 for the remainder of training. Architectural modifications do not produce gradient norm spikes, further confirming the stability of function-preserving operations.

7.5 Final Architecture

Figure 5 shows the per-layer head count and alpha values of the final 19-layer model. Several patterns stand out.

Table 3: Structural events by training phase.

Phase	L-split	L-prune	H-split	H-merge
1k–2k	3	0	0	0
2k–5k	6	1	7	5
5k–10k	7	8	0	1
10k–20k	0	0	0	0
20k–25k	6	0	0	0
25k–30k	88	84	7	13
Total	**110**	**93**	**14**	**19**

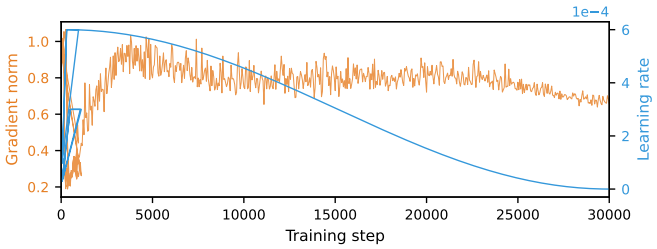

Figure 4: Gradient norm (orange) and learning rate (blue) over training. Gradient norms remain stable through all architectural changes.

Non-uniform head distribution. Layers 9–12 have 3 heads; all others have 2. These middle layers developed extra heads through splitting during the early growth phase and retained them through 30,000 steps, suggesting they benefit from more parallel attention patterns.

Negative alphas. Layers 2, 9, 11, and 15 have negative α values (-0.16, -0.49, -0.04, -0.53). The model learned to *subtract* these layers' contributions from the residual stream. This is a form of learned inhibition that would not emerge in standard transformers where residual connections are unscaled.

Dominant layer. Layer 17 has $\alpha = 1.50$, the largest magnitude in the model, suggesting it plays a disproportionately important role in the final computation.

Near-zero alternation. Every other layer in the late-expansion region (4, 6, 8, 10, 12, 14, 16, 18) has $\alpha \approx 0$. These are newly split layers that have not yet developed beyond their initial zero contribution. With more training, they would either grow or be pruned in subsequent plasticity cycles.

7.6 Head Distribution Over Time

Figure 6 shows a heatmap of heads per layer over training, with time on the horizontal axis and layer index on the vertical. The model starts as a 2-layer band at the bottom, grows vertically as layers are added, and develops warm spots (higher head counts) in the middle layers.

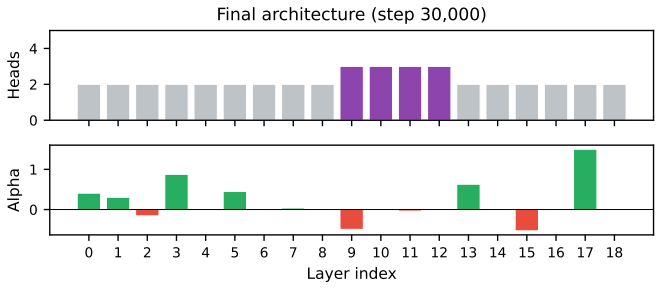

Figure 5: Final architecture at step 30,000. Top: heads per layer (purple = 3 heads). Bottom: learned residual alpha per layer (green = positive, red = negative).

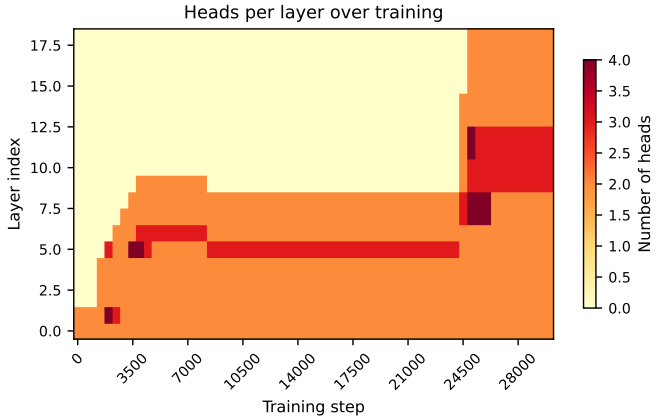

Figure 6: Heads per layer over training (color intensity = head count). The model grows vertically as layers are added and develops higher head counts in middle layers.

7.7 Loss vs. Model Size

Figure 7 plots loss against parameter count, colored by training step. The trajectory shows that loss decreases both from continued training (moving along the color gradient) and from architectural growth (moving rightward). The two growth phases produce distinct rightward jumps at approximately 6.9M→8.5M and 8.3M→10.6M parameters.

7.8 Qualitative Outputs

Table 4 shows sample completions from the final model (temperature 0.7, top-k 40, max 60 tokens). The model produces grammatical, topically relevant text but exhibits repetition, which is typical for small language models. It was trained on educational web text and performs best on expository prompts.

8 Discussion

8.1 The Split-Prune Cycle

The most striking result is the high ratio of structural events to net architectural change. The controller executed 236 events to produce a model with 17 more layers than it started with. This is not inefficient; it is the mechanism by which the model explores architectural possibilities. Each layer split creates a hypothesis ("this region needs more capacity"), and subsequent training either confirms it (α grows) or rejects it (α stays near zero and the layer is pruned). This is a form of architecture search conducted within a single training run, analogous to biological synaptic overproduction followed by experience-dependent pruning.

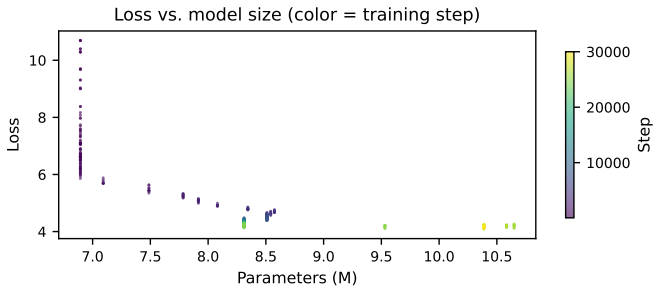

Figure 7: Loss vs. parameter count over training (color = training step). Rightward jumps correspond to growth phases.

Table 4: Sample completions from the 10.4M-parameter model.

Prompt: *The history of mathematics begins* **Output:** with the work of a group of scientists who studied mathematics at the University of Maryland. The book was published in the year 2011. The book was published in the journal The Science and Technology Center at the University of Maryland, New York.
Prompt: *Photosynthesis is the process by which* **Output:** it is carried out in the atmosphere. This can be done by: The first step is to get a clear picture of the sun, but there is no light in the sun. The first step is to get a clear picture of the sun.
Prompt: *In a democracy, citizens* **Output:** are elected in the United States. The UN is responsible for the promotion of the nation's political and political institutions. The UN's election, which will be held by the UN's UN and the UN government, will be held on a national basis.

8.2 The Quiet Middle

Between steps 10,000 and 20,000, zero architectural events occurred. The model's loss continued to decrease (4.32 to 4.32, approximately flat) while the learning rate decayed from 4.55×10^{-4} to 1.53×10^{-4}. During this phase, the model was refining its weights within a fixed 9-layer structure. The late expansion at step 20,000+ suggests that as the learning rate dropped, the utility and complexity distributions shifted enough to trigger new growth, or that the model needed additional depth to

continue improving at the reduced learning rate.

8.3 Why Middle Layers Get More Heads

Layers 9–12 in the final model (corresponding to the central third of the network) developed 3 heads each. This is consistent with findings from attention analysis in fixed-architecture transformers [Voita et al., 2019], where middle layers tend to exhibit more diverse attention patterns (syntactic, positional, and semantic) while early layers focus on local patterns and late layers focus on next-token prediction. Our result suggests this is not just an artifact of fixed architectures but reflects a genuine computational need that dynamic growth can discover.

8.4 Limitations

We trained only a single model at a single scale. We did not compare against a fixed-architecture baseline of equivalent compute or parameter budget, which would be necessary to establish whether dynamic growth provides a genuine advantage. The plasticity thresholds were hand-selected and not tuned. The utility score conflates multiple signals. The dimension growth operation was never triggered, leaving that codepath untested in practice.

The model is small by modern standards. It remains to be seen whether the split-prune cycle, non-uniform head distribution, and negative alphas persist at scales of hundreds of millions or billions of parameters.

The late expansion phase, where 81% of events occurred in the final 5,000 steps, may be partly an artifact of the cosine learning rate schedule: as the learning rate approaches zero, alpha values for newly split layers cannot be trained upward quickly, leading to immediate pruning and re-splitting. A learning rate restart or warmup after architectural changes might produce more stable growth.

8.5 Future Work

Comparison against fixed-architecture baselines of equivalent FLOP budget would be the most important next step. Other directions: learning the plasticity thresholds themselves, applying the approach to fine-tuning rather than pre-training, testing dimension growth with lower saturation thresholds, and scaling to larger models where the computational benefits of adaptive architecture may be more pronounced.

9 Implementation

The full implementation is organized as follows:

- `model/transformer.py`: `NeuroplasticTransformer` and `DynamicTransformerLayer` with growth/prune methods.
- `model/attention.py`: `DynamicMultiHeadAttention` with independent `AttentionHead` modules and split/merge/prune operations.
- `model/plasticity.py`: `PlasticityController` implementing the decision logic from Algorithm 1.
- `model/utils.py`: Function-preserving operations (`zero_pad_linear`, `zero_pad_embedding`, `zero_pad_layer_norm`, `clone_with_noise`).
- `training/trainer.py`: DDP-aware training loop with plasticity integration, optimizer rebuild, and npviz recording.
- `training/data.py`: Memory-mapped binary dataloader and streaming HuggingFace fallback.
- `training/config.py`: All hyperparameters as a dataclass.
- `train.py`: Entry point supporting single-GPU, `torchrun`, and multi-node.
- `chat.py`: Interactive inference script for the trained model.

Dependencies: PyTorch $\geq$ 2.1, Transformers $\geq$ 4.36, Datasets $\geq$ 2.16, matplotlib, tqdm. Optional: wandb for experiment tracking.

10 Conclusion

We described and trained a Neuroplastic Transformer that modifies its own architecture during training using gradient-derived utility and complexity signals. Starting from 2 layers and 6.9M parameters, the model grew to 19 layers and 10.4M parameters over 30,000 steps on a single A100 GPU, executing 236 function-preserving structural modifications in approximately 2 hours and 16 minutes.

The resulting architecture is non-uniform, with more attention heads in middle layers, negative residual scaling in several layers, and a dominant late layer with $\alpha = 1.5$. These patterns emerged entirely from the training signal without any explicit architectural prior. The split-prune cycle, in which layers are speculatively added and removed if they do not develop useful representations, provides a mechanism for in-training architecture search that requires no additional compute beyond the single training run.

References

T. Chen, I. Goodfellow, and J. Shlens. Net2Net: Accelerating learning via knowledge transfer. In *ICLR*, 2016.

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019.

P. Michel, O. Levy, and G. Neubauer. Are sixteen heads really better than one? In *NeurIPS*, 2019.

G. Penedo, H. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay. The FineWeb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024.

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *ACL*, 2019.

B. Wu, Z. Liu, Z. Yuan, G. Sun, and H. Wu. Splitting steepest descent for growing neural architectures. In *NeurIPS*, 2019.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.