APPEND-ONLY MIXTURE-OF-EXPERTS: EFFICIENT TRAINING AND FLEXIBLE DEPLOYMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Append-only Mixture of Experts (MoE), which grows an MoE during training and keeps each smaller model available for deployment. We train a small base model, then add expert rings while freezing existing parameters. Each ring has its own router, so disabling later rings recovers an earlier model. We train four sizes, from 199M to 1.597B parameters, using solution traces for String Algorithms, Maze Navigation, Arithmetic Sums, and Conway's Game of Life. At 35,000 updates, the four-stage run uses 6.80 hours of recorded training duration on four H200 GPUs, compared with 21.18 hours for the tested full-size scratch baseline. Mean final-answer accuracy is 99.22% and 80.27%, respectively. All six checks of earlier prefixes produce bitwise-equal logits. Having this adjustment in size also affects decoding speed, where 199M base decodes at 209.4 tokens/s, compared with 39.8 tokens/s for the largest.

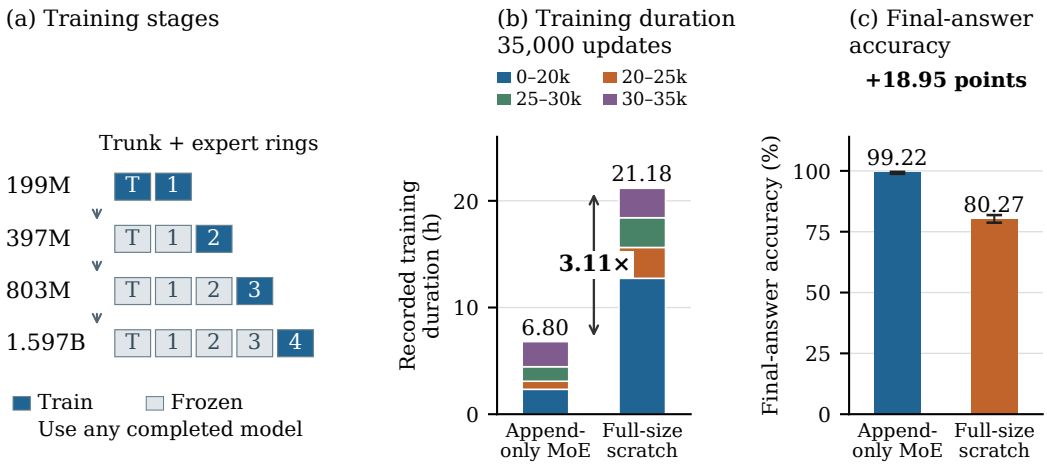

Figure 1: Append-only training and comparison with Full-size scratch. (a) We train the blue components and freeze the gray components at each stage. T denotes the shared trunk; numbered boxes are expert rings with independent routers. We recover an earlier model by disabling later rings. (b) Recorded training duration for 35,000 updates including all stages to 1.597B. Stacked colors show update ranges in thousands. (c) Final-answer accuracy on 512 examples per domain. See Section 4.3 for the scratch-run status.

1 INTRODUCTION

Mixture-of-experts (MoE) models store many more parameters than they use for any one token (Fedus et al., 2022). Qwen3.5-35B-A3B, for example, stores approximately 35 billion parameters and activates 3 billion per token (Qwen Team, 2026). This reduces computation during inference, although all weights must remain accessible. Running on hardware with less memory often requires a smaller model.

Elastic methods support different inference budgets by varying expert activation, changing expert widths, or compressing a trained model (Gu et al., 2025; Tastan et al., 2026; Taghibakhshi et al., 2026; Mo et al., 2026). We study an approach that also changes how the model is trained. Training begins at a small size and adds capacity in stages. Early updates use the smaller model, and each completed model remains available for deployment after training ends.

Append-only Mixture of Experts adds a group of experts, called a ring, and an independent router in every layer. At each expansion, we freeze the existing model and train only the new experts and routers. Each router normalizes its gates within its own ring. Disabling later rings therefore restores the earlier model's computation (Figure 1). No compression, distillation, or additional training is needed to use those smaller models.

Previous work has grown MoEs and trained new experts around frozen components (Chen et al., 2023; Shi et al., 2025; Wang et al., 2026). We preserve each earlier model in full, including its shared attention, so it remains usable as a prefix of every larger checkpoint. Our contributions are:

- We design an MoE that adds capacity while preserving earlier models through frozen parameters and independent ring routing.
- We compare growth from 199M to 1.597B parameters with training the full 1.597B architecture from scratch. We count the base model and every expansion, and compare both methods at 35,000 updates.
- We measure accuracy, weight storage, and decode throughput at all four sizes, run controls for growth and freezing, and check that earlier parameters and logits remain unchanged in six comparisons.

2 RELATED WORK

Growing expert models. Sparse Upcycling converts a trained dense checkpoint into an MoE (Komatsuzaki et al., 2023). Lifelong-MoE adds experts and gating dimensions while freezing earlier experts and gates (Chen et al., 2023). Shared attention remains trainable, and knowledge distillation limits forgetting. Beyond Sunk Costs grows pretrained MoEs in expert count and depth, comparing efficiency at fixed additional and fixed total compute (Wang et al., 2026). It continues to update inherited parameters. We freeze the entire inherited model and normalize each ring's routing independently, allowing earlier models to be recovered by disabling the additions.

Frozen and modular models. Progressive Neural Networks adds trainable columns that reuse frozen columns from earlier tasks (Rusu et al., 2016). FlexOlmo trains domain experts independently using a frozen public expert and shared components, then combines selected modules at inference (Shi et al., 2025). Our rings are added sequentially, with each new ring trained alongside the previously learned rings. Each prefix is a model completed earlier in that sequence.

Elastic inference. Elastic MoE varies the number of active experts (Gu et al., 2025), and MoSE supports nested feed-forward widths within experts (Tastan et al., 2026). Star Elastic extracts nested submodels from a pretrained reasoning model through architecture selection and distillation (Taghibakhshi et al., 2026). FlexMoE ranks and prunes channels within pretrained experts, then applies recovery fine-tuning to obtain several model sizes (Mo et al., 2026). Append-only MoE produces its smaller models during training and preserves their computation as it grows. We do not compare these methods experimentally. A separate systems paper also named FlexMoE improves training through dynamic expert placement and scheduling (Nie et al., 2023).

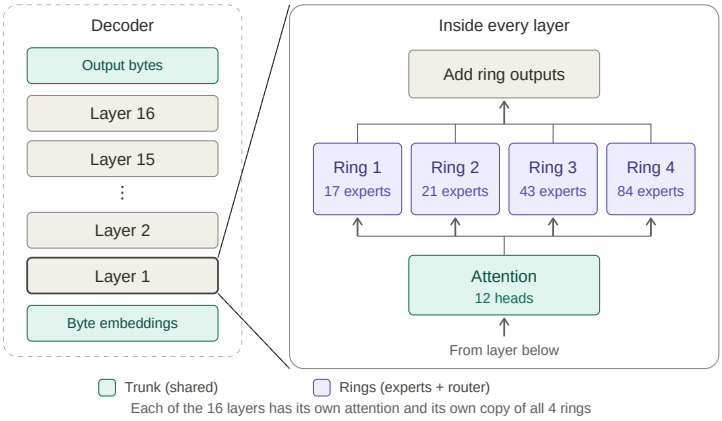

Figure 2: The network architecture showing the common trunk and the 16 layers, where each layer consists of a group of different rings. The trunk (teal) is shared by all prefixes while rings (purple) use the trunk to drive their output. We don't need to enable all the rings; we can use the model as prefix 1 (trunk and ring 1 only), prefix 2 (trunk and ring 1 and 2), etc.

3 METHODS

As explained in the introduction, our proposed system consists of a group of layers. Each layer consists of a fixed base that we call the *trunk*, followed by what we call *rings*. A *ring* is a group of experts and its router. The *trunk* and any number of rings attached to it (either 1, 2, or more) are called a *prefix*. For example, prefix 3 is a decoder model with rings 1, 2, and 3 attached. Figure 2 explains the current network architecture. A *rung* is the saved prefix after training. Whenever we add a new ring, the earlier routers and their expert groups remain unchanged as we freeze them during training.

The network has 16 layers, width 768, 12 attention heads, RMSNorm, rotary position encoding, tied embeddings, byte tokens, and a 4,608-token context. Each layer has its own experts and routers, without sharing their weights across layers. Ring r selects k_r of n_r, where k_r is the number of active experts in that ring and n_r is the total number of experts in that ring. SwiGLU experts are used (Fedus et al., 2022) with hidden width 256 and the following expert sizes for the different rings $(n_r, k_r) = (17, 3), (21, 6), (43, 12), (84, 25)$. Within a layer, every enabled ring r receives the same normalized feed-forward input x. Its router computes $z_{r,e}(x) = w_{r,e}^{\top} x$ and $p_{r,e}(x) = \mathrm{softmax}(z_r(x))_e$ for each expert e. For top-k_r experts we calculate the gate probabilities $g_{r,e}(x) = \frac{p_{r,e}(x)}{\sum_{j \in \mathcal{K}_r(x)} p_{r,j}(x)}$ which is used to compute the output of the layer given we are using p rings:

$$f_p(x) = \sum_{r=1}^{p} \sum_{e=1}^{k_r} g_{r,e}(x) E_{r,e}(x). \tag{1}$$

where $E_{r,e}(x)$ is the expert output. Each ring selects experts independently. The layer sums the ring outputs in a fixed order, without further rescaling.

The four prefixes store 199M, 397M, 803M, and 1.597B parameters. We train the trunk and first ring together. After it is trained well, we freeze its weights, add the next ring, and continue training. In this expansion, only the new experts and routers weights are being updated. Backpropagation may still pass through frozen layers to train the new rings, but it doesn't affect any of their weights. We continue doing that until all the rings are trained.

To run a prefix p, we enable all the rings from 1 to p and disable every later ring. Disabling later rings doesn't affect the result of any part of the system; no computations depend on how many rings are enabled/disabled. Adding a new ring only changes the hidden states passed to later layers. This could affect the accuracy of the expanded model without affecting the accuracy of the previous prefix.

We train our model on solution traces of all four domains at the same time where each domain contain a group of tasks. Section 4.2 and Appendix A describe the inputs, goals, and data format of each of these domains. Each input and output are constructed as a single text generation line. We apply cross-entropy only to the output and the EOS token after it, while masking the prompt and any other padding or additional tokens. We also scale the cross-entropy loss between all tasks based on the length of the solution trace. Without that scaling, the system might favor longer tasks over shorter ones:

$$\mathcal{L}_{\mathrm{CE}} = \sum_{t=0}^{T} \frac{\sqrt{S_t}}{\sum_{i=0}^{T} \sqrt{S_i}} \frac{\mathrm{CE}_t}{S_t}. \tag{2}$$

where T is the total number of tasks across all 4 domains, CE_t is the cross entropy of task t, and S_t is the token count for task t. We use the square root to reduce the difference compared with weighting directly by token count. We add load-balancing with a coefficient of 0.01 to make sure that our routers don't select the same experts all the time, and router z-loss terms with a coefficient of 10^{-4}. Check appendix B for more details about the training process.

4 Experimental Setup

4.1 Experimental questions

The experiments address five questions:

- **Q1 - Training efficiency:** Does training the model by growing it (by adding rings) reduce training time while preserving accuracy, compared with training the full-size model with all rings attached from scratch under the same update count?
- **Q2 - Growth and freezing:** Does adding expansion rings while freezing the parent improve the system accuracy compared to continuing to train the system without freezing them?
- **Q3 - Deployment choices:** How do the four model sizes compare in accuracy, weight storage, and decode throughput?
- **Q4 - Difficulty limits:** How well do the models solve harder instances from the different domains compared to the trained model from scratch?

4.2 Domains & Tasks

We generate problems in four domains using Reasoning Gym 0.1.25 (Stojanovski et al., 2025) and write a solution trace for each problem. The generators let us control difficulty and verify answers. Each domain contains 4 different tasks that has different difficulty. Appendix A.1 shows an input and target trace from each domain. The four domains are:

- **String Algorithms:** The system is provided with 64-character string and ten ordered rewrite rules. The goal is to apply these rules to that string until it can't be applied again and return the final string. The rules are checked from top to bottom and applied to the string from left to right. The first eligible match is applied. The solution trace contains the rule checks, the updated strings on every change, and an additional checksum.
- **Maze Navigation:** The system is provided with a 15×15 grid with walls, passages, and labeled start and goal cells. The goal is to find the shortest path from start to goal through that maze. The solution trace contains breadth-first search (BFS) steps to find the shortest path (neighbor cells, the successive frontier cell, the distance, etc).
- **Arithmetic Sums:** The system is provided with an arithmetic expression (add/subtract) over positive two-digit integers and asked for the answer to the expression. The trace contains a step-by-step operation output till the final value is computed.
- **Conway's Game of Life:** The system is provided with an 8×8 binary board and the goal is to iterate through Conway's Game of Life rules (Conway, 1970) for a requested number of steps. The solution trace is the board state after each step.

We use two sets of training data to train the system: small pool of 80,000 records and a large pool of 320,000 records. These records are divided equally between all the 4 domains. We use the small pool

to train prefix 1 and 2; while the large pool for training prefix 3 and 4. We also have a hard pool of records of 8,000 records. Each update during training draws 64 records with replacement divided equally between all the tasks and domains.

4.3 TRAINED MODELS

We train multiple different models under different conditions to understand how our system compares to a system from scratch or longer training time, or if we continue to train without freezing the parent. These are all the different systems being trained:

- **Append-only MoE:** Train prefix 1 (199M parameters) for 20,000 updates, then each added ring for 5,000 updates, to have prefix 2 (397M parameters), prefix 3 (803M parameters), and prefix 4 (1.597B parameters).
- **Unfrozen expansion:** Similar to the previous models but instead of freezing the parent after adding a new ring, we don't freeze the parents and let the system update both the new ring and the parents.
- **Continued parent:** At each expansion boundary instead of adding a new ring, train all the parameters for extra 5,000 updates.
- **Full-size scratch.** Initialize the complete model with all the rings (1.597B parameters) and train it fully. For comparison reasons, we train 4 models similar to all the others by starting with 20,000 updates and adding 5,000 extra updates for 3 times.

All jobs use four H200 GPUs and AdamW with $\beta = (0.9, 0.95)$, peak learning rate 6×10^{-4}, matrix weight decay 0.1, and gradient clipping at 5.0. The saved schedule warms up for 1,250 updates, then follows cosine decay to one tenth of peak at update 25,000. For every expansion and new experiment (even it is just continuation), we initialize a new optimizer. We use the same loss formula and coefficients in all the different trained models. The router losses also depend on padding and micro-batch size of 16 examples while we use only 8 for unfrozen expansion. For our biggest size model (prefix 4), we use micro-batches of size 4. We validate on 128 examples per task (2,048 total) every 5,000 updates and at job completion. We have an early stopping if the accuracy reached 99%. All the models are trained for the same number of updates. We select the best validation checkpoint. This selects the final checkpoint in all the jobs.

4.4 PERFORMANCE METRICS

We define *final-answer accuracy* as the fraction of examples whose extracted answer exactly matches the task reference. Standard test uses 512 examples per domain and hard evaluation uses 256. We score every example, including malformed outputs. This metric checks only the extracted final answer, not the intermediate trace. Appendix D.1 specifies extraction and normalization. Our confidence interval reported are not done between training seeds but more about stability of the model between different calls.

During development, we used standard-test errors to revise the solution trace for String Algorithms and Maze Navigation representations. These two domains were pretty complex and earlier experiments were not working as well. The final representation contains some auxiliary and helpful information such as rule checks, count updates, and explicit BFS lookups. Appendix A.2 describes the saved record of these decisions.

We compute *recorded training duration* by adding the retained training time of every job needed for a model, through each job's final log event. This includes validation and checkpoint writes during the run. It excludes startup, loading the parent and corpus, discarded attempts, external evaluation, and time spent waiting for uploads to finish. GPU-hours are four times this duration. We do not measure training FLOPs or total campaign occupancy (Appendix C.2).

5 RESULTS

5.1 TRAINING TIME AND ACCURACY (Q1)

Recorded training duration is 6.80 hours for Append-only MoE and 21.18 hours for Full-size scratch at 35,000 updates (Table 1). Growth uses 67.9% less time and GPU-hours, a $3.11\times$ ratio. Both totals include every training stage, including the append model's base. Section 4.3 describes the training protocol and the outstanding scratch-run confirmation.

Table 1: Training time and accuracy at 1.597B parameters (Q1). We train both methods for 35,000 updates and 2.24M record draws, using four H200 GPUs per job. We count training time for the base and every later stage (Section 4.4). Accuracy is the mean across four domains. The 95% intervals cover evaluation-sample variation for these trained models. See Section 4.3 for the scratch-run status.

Method	Recorded training duration (h)	GPU-hours	Accuracy (%)	95% CI
Append-only MoE	6.80	27.20	99.22	[98.79, 99.61]
Full-size scratch	21.18	84.70	80.27	[78.70, 81.84]

Final-answer accuracy is 99.22% for Append-only MoE and 80.27% for Full-size scratch, a paired difference of 18.95 percentage points (95% interval: 17.37 to 20.54). Append-only MoE leads on all four domains, with the largest gap in String Algorithms (Table 2).

Table 2: Standard-test final-answer accuracy by domain for the two final 1.597B models (Q1). Each domain has 512 examples; accuracy requires an exactly correct extracted final answer. All entries are percentages. The mean gives each domain equal weight.

Domain	Append-only MoE (1.597B)	Full-size scratch (1.597B)
String Algorithms	99.02	54.69
Maze Navigation	99.80	96.68
Arithmetic Sums	99.61	81.05
Conway's Game of Life	98.44	88.67
Mean	**99.22**	**80.27**

Figure 3 plots validation accuracy against updates and recorded training duration. At 20,000 updates, test accuracy is 99.02% for the base and 86.23% for the full-size model. Later append stages therefore start from an already accurate parent. Scratch test accuracy then changes to 75.05%, 88.23%, and 80.27% after successive blocks. The 30,000-update scratch checkpoint is better than its final checkpoint on both validation (89.65% versus 80.27%) and test (88.23% versus 80.27%). We retain the 35,000-update endpoint for the primary comparison to match the append model's total updates. Checkpoints are not selected using test accuracy.

5.2 GROWTH AND FREEZING CONTROLS (Q2)

Append-only MoE outperforms both trained controls at every expansion (Table 3). Continued parent receives more training without new parameters. Unfrozen expansion adds the same initialized ring as Append-only MoE and updates all parameters. The 803M comparison also differs in micro-batch size, as described in Section 4.3.

Append-only MoE exceeds Unfrozen expansion by 9.23, 6.64, and 6.45 percentage points. Its changes from the *unchanged parent* are much smaller: $+0.684$, $+0.049$, and -0.537 points, or 14 more, one more, and 11 fewer correct answers out of 2,048. All six trained controls finish below their starting parent. The main advantage over these controls is that append-only training keeps accuracy close to the parent's starting score. Added capacity brings little improvement on these tasks. Each control uses the shared training recipe without separate tuning.

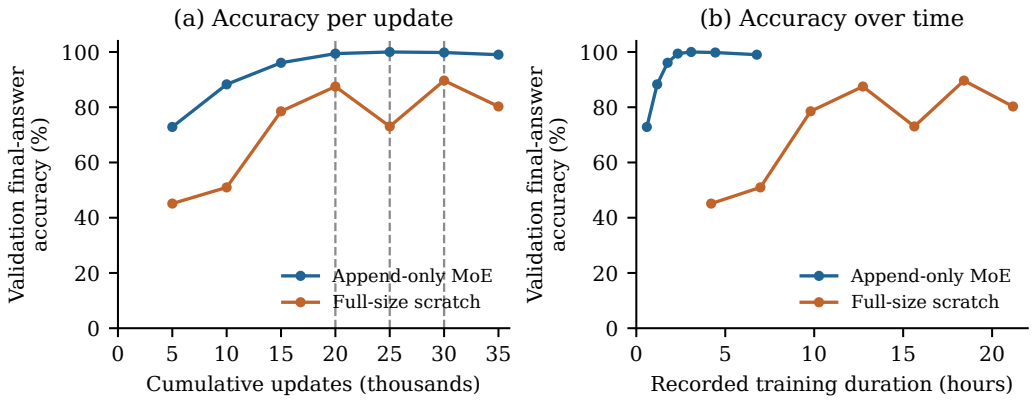

Figure 3: Validation accuracy during training (Q1). Final-answer accuracy is plotted against cumulative updates (left) and recorded training duration (right). Append-only MoE grows from 199M to 1.597B; Full-size scratch has 1.597B parameters throughout. Both use four H200 GPUs. Markers show validation checks, joined without smoothing. Dashed lines mark the block boundaries at 20,000, 25,000, and 30,000 updates, when the append model expands. Both runs use one seed and the same validation examples. See Section 4.3 for the scratch-run status.

Table 3: Growth and freezing controls (Q2). We start all three training conditions from the same parent and run 5,000 additional updates. Parent accuracy is measured before those updates. Entries are mean standard-test final-answer accuracy (%). Continued parent adds no parameters; Append-only MoE and Unfrozen expansion add identically initialized rings. Gain is Append-only MoE minus Unfrozen expansion in percentage points, with a paired 95% interval for these trained models. The 803M pair also differs in micro-batch size. Model sizes count stored parameters (M: millions; B: billions).

Expansion	Parent accuracy	Append-only MoE	Continued parent	Unfrozen expansion	Gain vs. unfrozen [95% CI]
199M → 397M	99.02	99.71	90.38	90.48	9.23 [8.02, 10.45]
397M → 803M	99.71	99.76	92.43	93.12	6.64 [5.52, 7.81]
803M → 1.597B	99.76	99.22	82.96	92.77	6.45 [5.31, 7.62]

5.3 PRESERVATION AND DEPLOYMENT (Q3)

All six preservation checks pass: inherited parameters are identical, and logit differences are exactly zero. Each comparison uses 64 validation examples and eight random sequences capped at 2,048 tokens, in the deterministic runtime specified in Appendix D.2.

The 199M model scores 99.02% and has 0.740 GiB of serialized FP32 weights (Table 4). With the full checkpoint loaded, selecting its 199M prefix gives an aggregate decode throughput of 209.4 tokens/s at batch 8, compared with 39.8 for the largest prefix, a $5.26\times$ ratio. Earlier checkpoints can also be loaded as complete models. We did not measure their separately loaded peak memory or end-to-end task latency.

Table 4: Accuracy, storage, and decode throughput at each model size (Q4). We measure accuracy at the end of each append stage. For throughput, we load the full 1.597B model on one H200 and select each prefix: batch 8, 256 prompt tokens, and 64 decode steps regardless of EOS. Throughput counts tokens across all eight sequences. Weight size counts serialized FP32 weights for that prefix, not loaded GPU memory. M and B denote millions and billions of parameters.

Model size (stored)	Active parameters (M)	Accuracy (%)	Weights (GiB)	Aggregate decode throughput, batch 8 (tokens/s)
199M	66.5	99.02	0.740	209.4
397M	95.1	99.71	1.479	115.3
803M	152.2	99.76	2.993	69.4
1.597B	275.9	99.22	5.950	39.8

The 803M model has the highest observed mean accuracy, 99.76%, and is smaller and faster than the 1.597B model. It answers only one more question correctly than the 397M model, so the difference is too small to rank them reliably. The final rung scores 99.22%, a net gain of four answers over the base and a loss of 11 relative to 803M, concentrated in Conway's Game of Life. Increasing model size does not improve accuracy at every stage.

5.4 PERFORMANCE ON HARDER PROBLEMS (Q4)

Mean accuracy on the hard strata is 1.27% for the final Append-only MoE and 0.59% for Full-size scratch. The append model answers none of the hard Maze Navigation, Arithmetic Sums, or Conway's Game of Life questions correctly.

For String Algorithms, 188 of 256 reference traces exceed the 2,752-token decode budget, and 186 append predictions are truncated. All references in the other three domains fit their task budgets. The append model has only one truncated maze output and none in Arithmetic Sums or Conway's Game of Life, so the string length limit cannot explain these other failures. Their causes remain unresolved. Appendix E reports every score and truncation count; no examples are excluded.

6 DISCUSSION AND LIMITATIONS

Starting with a small model may make early optimization easier. Once the base has learned useful representations, new experts can use them without changing the earlier weights. This is one possible explanation for the strong base and the advantage over the trained controls. We have not separated the effects of initial model size, freezing, and reusing earlier computation. Fewer parameters alone do not guarantee that less training data is needed; larger language models can be more sample-efficient (Kaplan et al., 2020).

The near-perfect base accuracy leaves little room for additional capacity to help. The 803M model performs better than the final rung while using fewer weights and decoding faster. We train to 1.597B at lower measured cost, although these tasks do not appear to need that many parameters. The control results also apply to a shared training recipe, without separate tuning, and the 803M comparison changes the micro-batch size as well as which parameters train.

We use one training seed and four procedural domains. Standard-test errors informed the representations during development, and the bootstrap intervals do not account for that exposure or variation across training seeds. Performance on harder instances is poor. Recorded training duration excludes startup, discarded work, and external evaluation. The inference benchmark keeps the full model loaded, so it does not measure the GPU memory needed to load smaller models separately. These experiments also do not measure general language-model performance. The scratch-run records require the confirmation described in Section 4.3 before submission.

7 Conclusion

Append-only Mixture of Experts trains a sequence of larger models while preserving the earlier ones. Each expansion adds experts and routers and leaves existing parameters fixed. Our four-stage run uses 67.9% less recorded training duration than the tested full-size scratch baseline and finishes with higher final-answer accuracy at the same update count. The smaller models remain available without retraining, with fewer stored weights and faster prefix decoding. We find little accuracy gain from added capacity on this suite, which the base already nearly solves.

Reproducibility Statement

We include the archived training code, task generators, renderers, scorer, and inference benchmark, together with resolved configurations, corpus manifests, verification reports, and run logs. Full test and hard-set predictions and the analysis scripts reproduce the tables, intervals, and figures. Four reconstructed examples are checked against saved reference traces. Trained weights and the full training corpus are not included. The README gives the code commit, dependencies, regeneration commands, and the unresolved scratch-run confirmation from Section 4.3.

AI Use Statement

Generative AI tools assisted with manuscript drafting, language editing, literature lookup, evidence review, numerical analysis, and LaTeX preparation. The authors are responsible for the final text, citations, and experimental claims.

References

Wuyang Chen, Yanqi Zhou, Nan Du, Yanping Huang, James Laudon, Zhifeng Chen, and Claire Cui. Lifelong language pretraining with distribution-specialized experts. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 5383–5395. PMLR, 2023. URL https://proceedings.mlr.press/v202/chen23aq.html.

John Conway. Conway's game of life. *Scientific American*, 223:120–123, 1970.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://arxiv.org/abs/2101.03961.

Naibin Gu, Zhenyu Zhang, Yuchen Feng, Yilong Chen, Peng Fu, Zheng Lin, Shuohuan Wang, Yu Sun, Hua Wu, Weiping Wang, and Haifeng Wang. Elastic MoE: Unlocking the inference-time scalability of mixture-of-experts. arXiv:2509.21892, 2025. URL https://arxiv.org/abs/2509.21892. Version 2, revised 2026.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. arXiv:2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. arXiv:2212.05055, 2023. URL https://arxiv.org/abs/2212.05055. Version 2; first submitted 2022.

Fan Mo, Yuxuan Han, Geng Zhang, Wangbo Zhao, and Yang You. FlexMoE: One-for-all nested intra-expert pruning for MoE language models. arXiv:2606.27866, 2026. URL https://arxiv.org/abs/2606.27866.

Xiaonan Nie, Xupeng Miao, Zilong Wang, Zichao Yang, Jilong Xue, Lingxiao Ma, Gang Cao, and Bin Cui. FlexMoE: Scaling large-scale sparse pre-trained model training via dynamic device placement. arXiv:2304.03946, 2023. URL https://arxiv.org/abs/2304.03946.

Qwen Team. Qwen3.5-35B-A3B model card. Hugging Face model repository, 2026. URL https://huggingface.co/Qwen/Qwen3.5-35B-A3B. Accessed 23 September 2026.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. arXiv:1606.04671, 2016. URL https://arxiv.org/abs/1606.04671.

Weijia Shi, Akshita Bhagia, Kevin Farhat, Niklas Muennighoff, Pete Walsh, Jacob Morrison, Dustin Schwenk, Shayne Longpre, Jake Poznanski, Allyson Ettinger, Daogao Liu, Margaret Li, Dirk Groeneveld, Mike Lewis, Wen-tau Yih, Luca Soldaini, Kyle Lo, Noah A. Smith, Luke Zettlemoyer, Pang Wei Koh, Hannaneh Hajishirzi, Ali Farhadi, and Sewon Min. FlexOlmo: Open language models for flexible data use. arXiv:2507.07024, 2025. URL https://arxiv.org/abs/2507.07024.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning Gym: Reasoning environments for reinforcement learning with verifiable rewards. arXiv:2505.24760, 2025. URL https://arxiv.org/abs/2505.24760. Version 2.

Ali Taghibakhshi, Ruisi Cai, Saurav Muralidharan, Sharath Turuvekere Sreenivas, Aditya Vavre, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Sheldon Liang, Marcin Chochowski, Zijia Chen, Akhiad Bercovich, Ran Zilberstein, Ran El-Yaniv, Yonatan Geifman, Daniel Korzekwa, Yoshi Suhara, Oluwatobi Olabiyi, Ashwath Aithal, Nima Tajbakhsh, and Pavlo Molchanov. Star Elastic: Many-in-one reasoning LLMs with efficient budget control. arXiv:2605.07182, 2026. URL https://arxiv.org/abs/2605.07182.

Nurbek Tastan, Stefanos Laskaridis, Karthik Nandakumar, and Samuel Horvath. MoSE: Mixture of slimmable experts for efficient and adaptive language models. arXiv:2602.06154, 2026. URL https://arxiv.org/abs/2602.06154. Version 2.

Ruizhe Wang, Yucheng Ding, Xiao Liu, Yaoxiang Wang, Peng Cheng, Baining Guo, Zhengjun Zha, and Yeyun Gong. Beyond sunk costs: Boosting LLM pre-training efficiency via orthogonal growth of mixture-of-experts. arXiv:2510.08008, 2026. URL https://arxiv.org/abs/2510.08008. Version 2; first submitted 2025.