NESTED SPARSITY FOR SCALABLE PRETRAINING AND CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-experts (MoE) transformers, the standard backbone for modern large-scale deep learning, employ conditional computation: only a sparse subset of model weights is used in each forward pass. In MoE transformers, this sparsity arises in the MoE operator, which generalizes the multi-layer perceptron in the classical transformer. In this work, we explore an alternative mechanism for conditional computation that analogously generalizes the attention operator. The sparsity induced by this mechanism naturally *nests* (i.e., hierarchically composes) with the sparsity induced by the MoE operator. We show that a transformer employing this *nested sparsity* mechanism, which we call NESTOR, performs better than a conventional MoE transformer at equal compute and equal sparsity in language model pretraining. We also show that NESTOR is well-suited for continual pretraining: it can be adapted to longer contexts more effectively than conventional MoEs, as well as adapt more efficiently to new domains while forgetting less on old domains. Furthermore, we demonstrate that these benefits persist as model size, dataset size, and context length grow. Our results demonstrate the potential of alternative conditional computation mechanisms for deep neural network architectures.

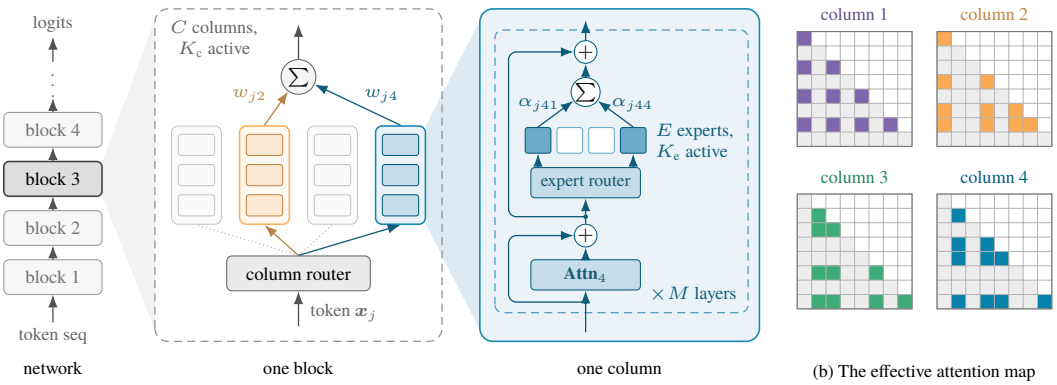
1 INTRODUCTION

Conditional computation in LLMs. Large language models (LLMs) based on variants of the transformer architecture (Vaswani et al., 2017) improve predictably as their parameters, training data, and training compute grow (Kaplan et al., 2020; Brown et al., 2020; Hoffmann et al., 2022). A central goal of architecture design is to reach a lower loss with the same training compute. A particularly successful approach is *conditional computation* (Bengio et al., 2013; Shazeer et al., 2017), in which each input uses only a small (input-dependent) "active" subset of the model weights and correspondingly less compute for training or inference. In this setting, the number of total parameters grows much faster than the active parameters and compute used on each token.

Mixtures of experts (MoEs). In modern LLMs, this form of conditional computation is implemented almost exclusively via the MoE operator (Jacobs et al., 1991; Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022), which replaces the multi-layer perceptron (MLP) in each transformer layer by many expert MLPs and routes each token to a few of them. MoE transformers need substantially less compute than dense transformers to reach the same loss (Fedus et al., 2022; Clark et al., 2022), and this advantage is retained at scale when the experts are configured appropriately (Ludziejewski et al., 2024; Abnar et al., 2025). As a result, many modern LLMs are MoE transformers (Jiang et al., 2024; DeepSeek-AI, 2024; Yang et al., 2025; Kimi Team, 2025a; OpenAI, 2025). In all of these models, however, this conditional computation is confined to the MLP weights: every token is processed by the same attention parameters in every layer. The attention operator accounts for a considerable fraction of the (active) parameters used by each token, and at long context lengths it accounts for most of the computation. One may then ask *whether conditional computation can be applied to the attention operator as well, and how it should interact with the MoE.*

Sparsity in the attention operator. Attention operators traditionally implement two forms of conditional computation. The more popular by far is the so-called "sparse attention" methods, which reduce the cost of attention by restricting the set of tokens to which each token attends, via fixed local patterns (Child et al., 2019; Beltagy et al., 2020), hashing or clustering (Kitaev et al., 2020; Roy et al., 2021), or a top-K selection for each query (Yuan et al., 2025; Lu et al., 2025; DeepSeek-AI, 2025).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) The nested sparsity mechanism in NESTOR.

(b) The effective attention map computed by each column.

Figure 1: **Nested sparsity in NESTOR.** *(a)* The NESTOR architecture. In each block, the column router sends the token $\boldsymbol{x}_j$ to $K_c = 2$ of the $C = 4$ columns. Each column has M layers, each with its own attention and its own $E = 4$ experts, of which a per-layer expert router selects $K_e = 2$. The token uses only the colored parameters. *(b)* The causal attention map computed by each column for $S = 8$ tokens, shown in terms of the whole sequence: column c computes attention only among the tokens $\mathcal{U}_c$ routed to it.

Relatedly, many LLMs interleave a few global attention layers with cheaper local or linear attention layers (Brown et al., 2020; Gemma Team, 2025; Kimi Team, 2025b). These methods sparsify the attention *map*, with the purpose of approximating full attention at a lower cost, while *all tokens still share the same attention parameters*. Other methods restrict the set of parameters used by each token, such as mixtures of attention heads (Zhang et al., 2022; Csordás et al., 2024), which route each token to a subset of the attention *parameters*, while every token still attends to the whole context. Other methods make both the map and the parameters sparse (Piekos et al., 2025), or pair mixtures of attention heads with an MoE (Shen et al., 2023; Csordás et al., 2024). In each of them, the attention is routed separately from the MLP and anew in every layer.

Continual pretraining. One of the circumstances in which such attention sparsity is most effective at reducing both training and inference costs is with long context documents. Pretrained models are routinely continually (pre-)trained in a long-context extension stage (Xiong et al., 2024; Fu et al., 2024; Grattafiori et al., 2024), and simultaneously further on new domains, such as code and mathematics (Gururangan et al., 2020; Rozière et al., 2023; Azerbayev et al., 2024; Shao et al., 2024). Since neural networks trained on a new distribution tend to perform worse on the old one (McCloskey & Cohen, 1989; French, 1999; Kirkpatrick et al., 2017), and LLMs are no exception (Luo et al., 2025), a central challenge is to maintain the performance on old data while capably handling the new data.

Our contributions. We propose NESTOR (Figure 1), a transformer architecture with *nested sparsity*: we generalize the attention operator in the same way that the MoE operator generalizes the MLP, and hierarchically composes the two resultant operators' sparsity. This yields a natural architecture that generalizes MoE transformers and has three coupled but distinct forms of sparsity: attention weight sparsity, attention map sparsity, and expert sparsity. We study it empirically and theoretically in language model pretraining and continual pretraining, showing that NESTOR architectures utilizing all sparsity axes in tandem outperform strong MoE and dense Transformer baselines at scales from 100M to 1B active parameters, at the limit of our compute resources. Namely, our contributions are:

- **Compute-efficient pretraining** (Section 3). We perform a scaling law study, finding that NESTOR is 1.13–1.29× as compute-efficient as the MoE transformer, and 3.1–4.1× as compute-efficient as a dense transformer; neither factor shrinks as the scale grows.
- **Effective continual pretraining** (Section 4). When adapted to various specialized domains, we show that NESTOR gains about as much as the MoE transformer on every new domain while forgetting less of the pretraining distribution, with fewer training FLOPs per token in the continual pretraining data. When extended from a context length of 4k to lengths up to 64k, NESTOR preserves its advantage over the MoE transformer when each model receives an equal number of continual pretraining tokens, and its advantage over the MoE transformer grows at longer context lengths when each model uses equal continual pretraining FLOPs.

- **Theoretical analysis** (Section 5). We give a theoretical account of these observations. Namely, we give a statistical model of data with hierarchical latent structure, in which a stylized version of NESTOR outperforms a stylized version of an MoE transformer. We show that this analytically tractable model reproduces the pretraining compute efficiency, its growth with the context length, and the reduced forgetting that we observe empirically.

Overall, our results suggest that conditional computation need not be confined to the MLP, and that adding and composing new sparsity mechanisms is a promising direction for architecture design. For the sake of page limits, we postpone a detailed discussion of related work to Section A.

2 NESTOR: NESTED SPARSE TRANSFORMER

This section describes the NESTOR architecture. We provide the overall structure, main intuitions, and relationships to other architectures here. A complete mathematical description is in Section B.

Conditional computation in MoE transformers. Each layer of a transformer (Vaswani et al., 2017) updates the token features $\boldsymbol{x}_1, \dots, \boldsymbol{x}_S \in \mathbb{R}^D$ using an attention operator, which mixes information across tokens, followed by an MLP, which acts on each token separately. The MoE transformer (Shazeer et al., 2017; Fedus et al., 2022; Dai et al., 2024) replaces the MLP by E *expert* MLPs, of which each token uses only $K_\mathrm{e} \ll E$. A learned router $\boldsymbol{r}$ assigns the token $\boldsymbol{x}_j$ a vector of scores $\boldsymbol{r}(\boldsymbol{x}_j) \in \mathbb{R}^E$, and the MoE operator computes (omitting normalization for simplicity, here and after):

$$\boldsymbol{x}_j + \sum_{e \in \mathcal{E}_j} \alpha_{je}\, \mathbf{MLP}_e(\boldsymbol{x}_j), \qquad \text{where} \quad \mathcal{E}_j \doteq \mathrm{TopK}(\boldsymbol{r}(\boldsymbol{x}_j), K_\mathrm{e}) \subseteq [E], \tag{1}$$

and the gating weights $(\alpha_{je})_{e \in \mathcal{E}_j}$ are the softmax of the routing scores for the selected experts (see Figure 1a). The number of parameters in the MoE operator grows with E, while the computation spent on each token grows only with K_e. This conditional computation is confined to the MLP.

Column-routed attention. We now apply the same principle to the attention operator. We replace the attention operator by C attention operators $\mathbf{Attn}_1, \dots, \mathbf{Attn}_C$ with separate parameters, which we call *columns*. A *column router* with scores $\boldsymbol{s}(\boldsymbol{x}_j) \in \mathbb{R}^C$ selects for each token a set $\mathcal{C}_j \subseteq [C]$ of K_c columns, with gating weights $(w_{jc})_{c \in \mathcal{C}_j}$, as in (1). Namely, letting $\mathcal{U}_c \doteq \{i \in [S] \colon c \in \mathcal{C}_i\}$ be the set of tokens that were routed to the same column, the routed attention computes:

$$\boldsymbol{a}_j \doteq \boldsymbol{x}_j + \sum_{c \in \mathcal{C}_j} w_{jc}\, \mathbf{Attn}_c(\boldsymbol{x}_j,\, (\boldsymbol{x}_i)_{i \in \mathcal{U}_c,\, i \le j}), \qquad \text{where} \quad \mathcal{C}_j \doteq \mathrm{TopK}(\boldsymbol{s}(\boldsymbol{x}_j), K_\mathrm{c}) \subseteq [C], \tag{2}$$

where $\mathbf{Attn}_c(\boldsymbol{x}, \cdot)$ denotes multi-head attention with the query computed from $\boldsymbol{x}$ and the keys and values computed from the given context. Each column is a sequence-to-sequence operator applied to a subsequence chosen by the router. Routing therefore sparsifies both the parameters used by a token, i.e., K_c of the C attention operators, and the attention pattern (Figure 1b).

Nested sparsity. The two routing mechanisms compose hierarchically, as in a hierarchical mixture of experts (Jordan & Jacobs, 1994; Shazeer et al., 2017), except that the first level of routing also selects the attention operator and its context. We give each column c its own E experts $\mathbf{MLP}_{c,1}, \dots, \mathbf{MLP}_{c,E}$ and its own expert router $\boldsymbol{r}_c$. In each active column $c \in \mathcal{C}_j$, this router selects the set $\mathcal{E}_{jc} \doteq \mathrm{TopK}(\boldsymbol{r}_c(\boldsymbol{a}_j), K_\mathrm{e}) \subseteq [E]$ of experts, with gating weights $(\alpha_{jce})_{e \in \mathcal{E}_{jc}}$:

$$\boldsymbol{x}_j^+ \doteq \boldsymbol{a}_j + \sum_{c \in \mathcal{C}_j} w_{jc} \sum_{e \in \mathcal{E}_{jc}} \alpha_{jce}\, \mathbf{MLP}_{c,e}(\boldsymbol{a}_j). \tag{3}$$

An expert is indexed by the pair (c, e) (see Figure 1a). One layer stores C attention operators and CE experts, while a token uses K_c and $K_\mathrm{c}K_\mathrm{e}$ of them respectively. Each layer also has a small number CE_s of experts whose weights are tied across columns (cf. the shared experts of Dai et al. (2024)).

NESTOR as a meta-architecture. We group the layers of the network into *blocks* of M layers each, and compute the column routing once per block, from the input of the block, so that a token stays in the same columns for all M layers within the block (see Figure 1a left). Within a block, each column is a small MoE transformer of depth M whose attention operates on its own subsequence;

the columns of a token share the input to each layer (Section B). Apart from the shared experts, the routers, and the residual connections, a token uses only the parameters of its K_c columns. Setting $C = 1$ in (2) and (3) recovers the regular MoE transformer exactly, and further setting $E = 1$ recovers the dense transformer, so conventional transformers are special cases of NESTOR. Actually, at the expense of a slightly more inefficient implementation, we can use any sequence-to-sequence deep architecture for each column. Thus, we may view NESTOR as a meta-architecture for sparsely and hierarchically composing parallel deep networks; the presented version is a special case.

Relation to sparse attention. Sparse attention methods restrict the set of tokens to which each token attends (the attention *pattern*) to reduce the cost of long contexts. The pattern may be fixed in advance, as in sliding window attention and its variants (Child et al., 2019; Beltagy et al., 2020); computed by clustering the tokens (Roy et al., 2021); or selected for each query by a top-K rule over blocks or tokens (Yuan et al., 2025; Lu et al., 2025; DeepSeek-AI, 2025). In all of these methods every token is processed by the same attention parameters, so only the pattern is conditional, and the purpose is to approximate full attention at a lower cost. Mixtures of attention heads (Zhang et al., 2022; Csordás et al., 2024) route each token to a subset of the attention *parameters*, and each token still attends to the full context. Column-routed attention (2) is sparse in both senses: one routing decision determines both the parameters that the token uses and the tokens to which it attends. The closest mechanism is that of Piekos et al. (2025), in which each attention head selects a fixed number of tokens and attends only among them. There, the selection is made by the heads, separately in every layer, and is independent of the MLP; in NESTOR, the selection is made by the tokens, once per block, and the selected module contains the attention operators and the experts of all M layers. This makes the sparsity nested. The two approaches can be combined, because any sparse attention pattern (or, again, any backbone network) can be applied within a column. In this work each column uses full causal attention over its tokens; the baselines in Sections 3 and 4 also use full causal attention. We discuss the relation of NESTOR to such related work further and in detail in Section A.

Further design choices. We summarize the remaining empirical design choices here, with more details in Section C. *Sparsity schedule.* The number of active columns K_c can differ between blocks. In all experiments we use $C = 8$ and increase K_c with depth, from $K_c = 3$ in the first block to $K_c = C$ in the last block, which computes full attention over the whole sequence. Hence, this architecture relates to the interleaved globalized and local (or linear) attention layers popular in modern LLMs (Brown et al., 2020; Gemma Team, 2025; Kimi Team, 2025b); here, one can interpret NESTOR as using progressively more globalized attention layers. Our experimental baselines use global attention at each layer, therefore using more attention compute than NESTOR. *Residual connections.* Equations (2) and (3) show plain residual connections. In NESTOR, each active column accumulates its outputs in its own residual stream. The streams are combined, within and across blocks, by an attention residual (Kimi Team, 2026): the input of each layer is a learned softmax-weighted mixture of the token embedding, the outputs of the earlier blocks, and the mean of the streams of the current block. The input of the output head also includes the deviation of each stream from the mean. We ablate this choice in Section C.3; note that our baselines also use and benefit from attention residuals. *Routing.* Both routers use top-K selection by each token, and are balanced without an auxiliary loss by a bias that is updated in a backpropagation-free manner (Wang et al., 2024; DeepSeek-AI, 2024). We ablate our routing and sparsity setup in Section C.3. We fully describe the final architecture in Section B.

Aspirational connection to the cerebral cortex. To some extent, one may view NESTOR as an architecture that tries to emulate the neural architecture of the cerebral cortex of a brain (Hawkins, 2021): the blocks correspond to different lobes or functional regions in the cortex, from low-level to high-level, and the many columns in each block emulate the cortical columns within each region.

3 COMPUTE-EFFICIENT PRETRAINING WITH NESTOR

In this section we show that NESTOR is more compute-efficient than strong iso-sparsity MoE transformer and dense transformer baselines in language model pretraining, and that the advantage does not shrink as the scale grows to the limits of our compute. Following prior scaling law studies of MoE transformers (see Section A), we compare architectures at the same sparsity, fit a scaling law to each, and report the projected compute that each baseline needs to reach the held-out loss of NESTOR.

Setup. We compare three architectures at four scales, which we call 100M, 200M, 500M, and 1B after the *active* parameter count. The *dense transformer* has width $D \in \{480, 640, 960, 1280\}$, depth

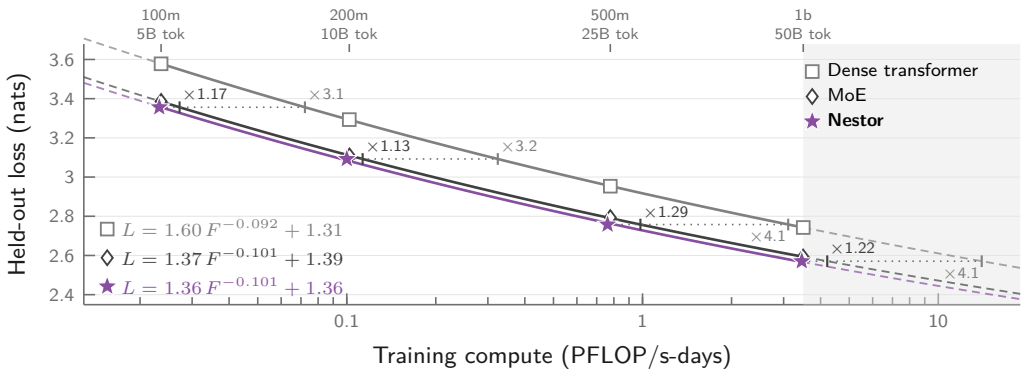


Figure 2: **Held-out loss scaling laws** for NESTOR, MoE, and dense models. Each law is fitted as $L(F) = A\,F^{-\alpha} + L_\infty$, where F is the training FLOPs. NESTOR reaches the lowest hold-out loss at every scale. The horizontal ticks mark the projected compute that the MoE and dense baselines need to match NESTOR.

$\{12, 16, 24, 32\}$, 16 heads, and MLP width $4D$. The *MoE transformer* has the same shape, with each MLP replaced by $E = 24$ experts of which $K_e = 3$ are active, so that its active MLP width is also $4D$. NESTOR uses $C = 8$ columns, increasing with depth from $K_c = 3$ in the first block to $K_c = 8$ (all columns active) in the final block. Both baselines use full causal attention with the attention residual (Kimi Team, 2026). At each scale, NESTOR uses no more training FLOPs than either baseline,[1] and stores approximately the same number of parameters as the MoE transformer. All models are trained on DCLM-baseline (Li et al., 2024) at context length $S = 4096$ for 50 tokens per active parameter (beyond the roughly 20 that are compute optimal for dense transformers; Hoffmann et al., 2022).The parameterization and optimizer settings used in these experiments are specified in Section D. We report the cross-entropy on held-out DCLM-baseline tokens at the end of training.

NESTOR is more compute-efficient at every scale. NESTOR achieves lower held-out loss than both baselines at all four model scales, as is shown in Figure 2; the losses, parameter counts, and training compute at each scale are tabulated in Table 3 of Section C.2. To quantify the compute savings at equal loss, we fit each architecture's loss-compute curve with $L(F) = AF^{-\alpha} + L_\infty$ (Kaplan et al., 2020; Hoffmann et al., 2022). For each measured NESTOR result, we trace a horizontal line to each baseline's fitted curve. The intersection gives an estimate of the baseline compute required to reach the same loss. We report this estimate, normalized by the training compute of NESTOR, as the horizontal ticks in Figure 2 and in the last column of Table 3. At the largest scale, NESTOR achieves a held-out loss of 2.571. Matching this loss is estimated to require $1.22\times$ as much compute for MoE and $4.1\times$ for the dense transformer. These largest-scale estimates extrapolate beyond the measured baseline compute range that we can afford to train. The fitted exponents of NESTOR and of the MoE transformer agree to three digits, and NESTOR has a smaller estimated coefficient and irreducible loss.

4 NESTOR ENABLES EFFICIENT CONTINUAL LEARNING

A pretrained model is often customized via continual training to learn new capabilities while retaining old ones. Concretely, pretrained models are often continually pre-trained on new domains (Gururangan et al., 2020; Rozière et al., 2023; Azerbayev et al., 2024; Shao et al., 2024) and longer contexts (Xiong et al., 2024; Fu et al., 2024; Grattafiori et al., 2024); in this setting, it is important to retain its original capabilities on old domains and shorter contexts, which continual training tends to erode (French, 1999; Luo et al., 2025). In this section we show that NESTOR is well suited to both settings: it adapts to new domains as well as the MoE transformer while forgetting less (Section 4.1), and it extends to long contexts more efficiently and effectively (Section 4.2).

[1] Our FLOP count includes attention FLOPs (Section C.1). Under the common approximation that counts only linear layers' FLOPs, NESTOR uses 4–8% more FLOPs than the baselines at each scale, at length $S = 4096$.

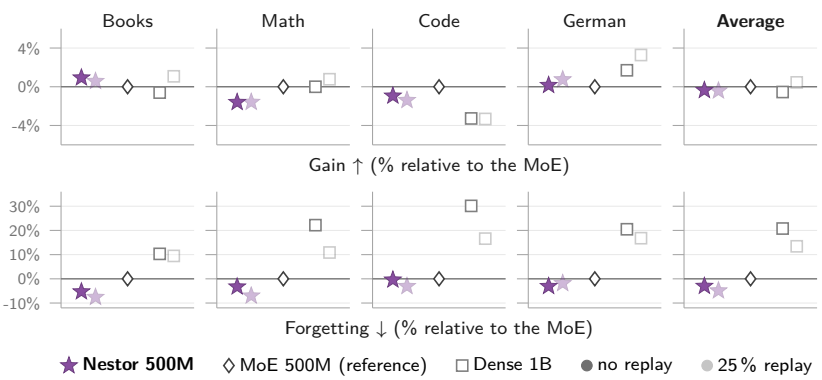


Figure 3: **Domain adaptation and forgetting relative to the MoE transformer. Top:** reduction in held-out loss on the target corpus. **Bottom:** increase in held-out loss on the pretraining corpus. Both are measured relative to each model before adaptation and shown as a percentage of the MoE transformer's value.

4.1 Domain Adaptation

Setup. We compare the 500M Nestor, 500M MoE transformer, and 1B dense transformer, selected for their similar pretraining losses. All models first undergo context extension to 32k using equal token budgets on DCLM. Details of the starting checkpoints and their initial performance are provided in Section C.4. Each model is then trained separately on 0.5B tokens from each of four target domains, without replay of pretraining data (a common way to reduce forgetting; Ibrahim et al., 2024): books (PG-19; Rae et al., 2020), mathematics (OpenWebMath; Paster et al., 2024), code (AlgebraicStack; Azerbayev et al., 2024), and German (FineWeb-2; Penedo et al., 2025). We freeze the router parameters of both sparse models (cf. Li et al., 2025) and update all remaining trainable parameters; the dense model is fully fine-tuned. We measure adaptation by the reduction in held-out loss on the target corpus and forgetting by the increase in held-out loss on the pretraining corpus, both relative to the model before adaptation. We also repeat the experiment with 25% of the training tokens drawn from the pretraining corpus, keeping the total budget fixed at 0.5B tokens; details and results are provided in Section C.4. Table 6 reports the held-out losses before and after adaptation.

Nestor adapts as well as the baselines with less forgetting. All three models achieve similar adaptation gains: averaged across the four target domains, the gains of Nestor and the 1B dense transformer are both within 1% of the 500M MoE transformer's (Figure 3). The models differ more in how much they forget. Nestor forgets 3% less than the MoE on average, whereas the dense model forgets 21%. Thus, Nestor has a modest retention advantage over the MoE, and both sparse models forget substantially less than the dense model. Replaying 25% pretraining data reduces forgetting by about 70% across all three models at a cost of about 6% of the adaptation gain and preserves this ordering: Nestor forgets 5% less than the MoE on average and the dense model 13% more, while the two sparse models' gains remain within 2% of each other (Figure 3, faded marks).

4.2 Long Context Extension

Setup. Starting from the pretrained models of Section 3, which have context length 4k, we extend each model to a target context length $S \in \{8\text{k}, 16\text{k}, 32\text{k}, 64\text{k}\}$ by continued pretraining on long documents (Xiong et al., 2024; Fu et al., 2024; Gao et al., 2025) of at least S tokens from DCLM-baseline. Each target length is a separate run from the 4k checkpoint. The baselines are trained on 0.5B tokens. Since the FLOPs per token of Nestor decrease relative to the baselines as S grows (Figure 9), we train Nestor in two settings: at *equal tokens* (0.5B), and at *equal FLOPs*, where the saved compute is spent on more tokens of either repeated or new documents. We report the held-out loss at length S, relative to the MoE transformer of the same scale. Details of the data, the schedules, and the losses behind Figure 4 are provided in Section C.5.

The advantage of Nestor grows with context length at equal compute. At equal tokens, Nestor achieves gains from context extension comparable to those of the MoE transformer, broadly preserving its advantage before extension (Figure 4). As the context length increases, its lower FLOPs per

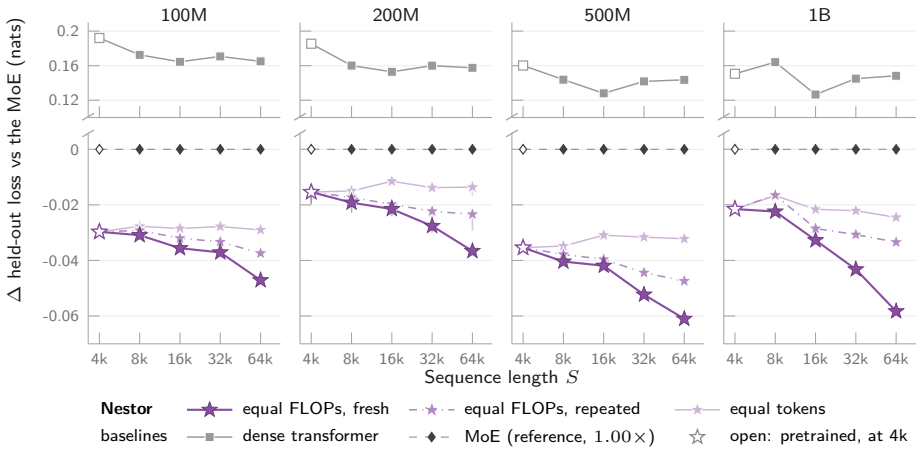


Figure 4: **Long context extension by continued pretraining.** We show the held-out loss at each context length from 4k to 64k at all scales, relative to the MoE transformer of the same scale. **Top:** the dense transformer. **Bottom:** NESTOR at equal FLOPs with new documents, at equal FLOPs with the documents of the equal-token run repeated over the run, and at equal tokens. The open markers at 4k are the pretrained checkpoints.

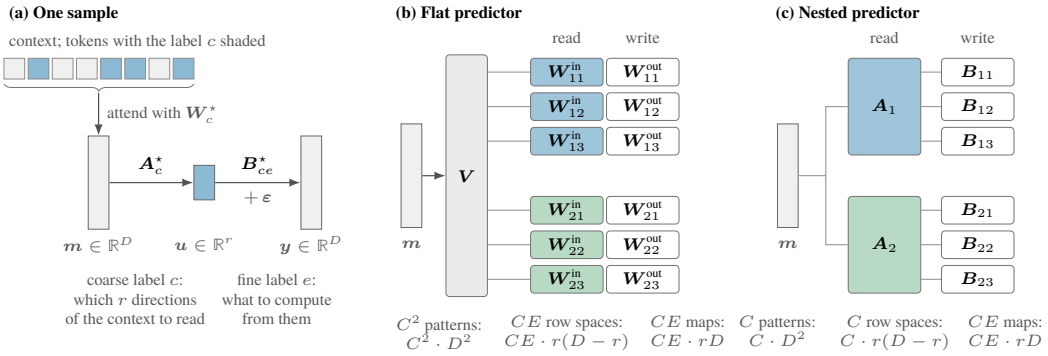


Figure 5: **The linear model and the two predictors**, for $C = 2$ coarse labels (blue, green) and $E = 3$ fine labels. *(a)* One sample: the coarse label determines the relevant tokens, the pattern $\boldsymbol{W}_c^\star$, and the row space of $\boldsymbol{A}_c^\star$; the fine label determines the map $\boldsymbol{B}_{ce}^\star$. *(b)* The flat predictor (MoE transformer): the shared attention scores every token, and each expert has its own row space, fit from its own samples. *(c)* The nested predictor (NESTOR): the attention of a column scores only the tokens with its label, and its row space is fit once from the samples of all of its experts. The free parameters are counted below each panel.

token allow more training within the same compute budget (Figure 9). Spending these savings on repeated data produces a downward trend in held-out loss relative to MoE, even without additional training data. Spending them on fresh data, as one might expect, yields the largest improvements. In Section E we quantify the saving: for equal shapes and balanced routing, at equal FLOPs NESTOR trains on $R(S) = (S_0 + S)/(S_0 + \rho S)$ times as many tokens as the MoE transformer, where ρ is the average fraction K_c/C of active columns and S_0 is the context length at which the attention FLOPs of the MoE transformer equal its other FLOPs, and $R(S)$ increases with S towards $1/\rho$.

5 THEORETICAL ANALYSIS

In this section we compare stylized versions of NESTOR and of the MoE transformer on a statistical model with hierarchical latent structure. In this analytically tractable model, we reproduce the empirical pretraining compute efficiency and continual training properties of the two architectures. We state the results informally here, and give the formal statements and the proofs in Section F.

Nested structure in the data. We assume that the next token is predicted in two stages. A coarse latent variable $c \in [C]$ determines which tokens of the context are relevant, how they are weighted,

and which statistic of them is computed. A fine latent variable $e \in [E]$ determines how this statistic is used. For example, in a corpus of code and mathematics, c may be the kind of text, the relevant tokens may be the declarations in scope or the hypotheses in effect, and e may determine whether the next token of code is a variable or an operator. In NESTOR, c corresponds to the output of the column router, the statistic to the output of the attention operator of column c, with its own pattern and value map, and e to the output of the expert router of the column. In the MoE transformer, one shared pattern weights the tokens and one shared value map computes the statistic. We study the simplest case: one layer with one head, linear attention and maps, and the squared loss (Figure 5a).

Model 1: *Let $r < D$ and $S \geq 2$. For each $c \in [C]$ and $e \in [E]$, fix an invertible pattern $\boldsymbol{W}_c^\star \in \mathbb{R}^{D\times D}$ and matrices $\boldsymbol{A}_c^\star \in \mathbb{R}^{r\times D}$ and $\boldsymbol{B}_{ce}^\star \in \mathbb{R}^{D\times r}$ of rank r. Each sample $(c, e, \boldsymbol{X}, \boldsymbol{y})$ is generated in four steps.*

1. *The labels (c, e) are drawn with $\mathbb{P}[c, e] > 0$ for all $(c, e) \in [C] \times [E]$.*
2. *The context $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_S)$ consists of tokens $\boldsymbol{x}_i = (c_i, \boldsymbol{z}_i)$ for $i < S$ and the query $\boldsymbol{x}_S = (c, \boldsymbol{z}_S)$. The labels c_i are uniform on $[C]$, and the contents $\boldsymbol{z}_i \in \mathbb{R}^D$ are drawn from a distribution with a density and finite moments, all independently of each other.*
3. *The summary is $\boldsymbol{m} \doteq \sum_{i\in\mathcal{I}} (\boldsymbol{z}_S^\top \boldsymbol{W}_c^\star \boldsymbol{z}_i)\, \boldsymbol{z}_i$ over the relevant tokens $\mathcal{I} \doteq \{i < S \colon c_i = c\}$.*
4. *The target is $\boldsymbol{y} \doteq \boldsymbol{B}_{ce}^\star \boldsymbol{A}_c^\star \boldsymbol{m} + \boldsymbol{\varepsilon}$, with independent noise $\boldsymbol{\varepsilon}$ of mean zero and covariance $\sigma^2 \boldsymbol{I}$.*

We observe T independent samples $(c_i, e_i, \boldsymbol{X}_i, \boldsymbol{y}_i)_{i\in[T]}$.

The summary $\boldsymbol{m}$ is the output of one head of linear attention at the query, over the $(S-1)/C$ tokens with the label c on average. The target $\boldsymbol{y}$ stands in for the next token, for example the update that the layer adds to the residual stream. The regression function is $\boldsymbol{f}^\star(c, e, \boldsymbol{X}) = \boldsymbol{G}_{ce}^\star \boldsymbol{m}$, where

$$\boldsymbol{G}_{ce}^\star \doteq \boldsymbol{B}_{ce}^\star \boldsymbol{A}_c^\star, \qquad \operatorname{rank} \boldsymbol{G}_{ce}^\star = r, \qquad \operatorname{row}(\boldsymbol{G}_{ce}^\star) = \operatorname{row}(\boldsymbol{A}_c^\star) \ \text{ for all } e \in [E]. \tag{4}$$

The last property is the nested structure: for a fixed coarse label c, the E maps $\boldsymbol{G}_{ce}^\star$ share the pattern $\boldsymbol{W}_c^\star$ and the row space of $\boldsymbol{A}_c^\star$, while both of these may differ from one coarse label to another.

The two predictors and their training. Both predictors take the labels as input along with the context, so no router is learned. The *nested predictor* is one layer of NESTOR with $K_\mathrm{c} = 1$: the attention of column c sums over the tokens with the label c, with its own pattern $\boldsymbol{W}_c$ and a value map $\boldsymbol{A}_c$ of width r, and expert (c, e) applies a linear map $\boldsymbol{B}_{ce}$. The *flat predictor* is one layer of an MoE transformer: a shared attention operator sums over all tokens, with a value map $\boldsymbol{V}$ of full width D, as in our baseline, and each expert is a linear MLP with r hidden units. Since a score bilinear in the raw tokens cannot depend on both labels (Theorem 1a), the shared score is bilinear in the product features $\boldsymbol{e}_{c_i} \otimes \boldsymbol{z}_i$ as in Remark 8, with one block $\boldsymbol{W}_{cc'}$ per pair of labels. The two predictors are

$$\boldsymbol{f}^\mathrm{N}(c, e, \boldsymbol{X}) \doteq \boldsymbol{B}_{ce} \boldsymbol{A}_c \sum_{i\in\mathcal{I}} (\boldsymbol{z}_S^\top \boldsymbol{W}_c \boldsymbol{z}_i)\, \boldsymbol{z}_i, \quad \boldsymbol{f}^\mathrm{F}(c, e, \boldsymbol{X}) \doteq \boldsymbol{W}_{ce}^\mathrm{out} \boldsymbol{W}_{ce}^\mathrm{in} \boldsymbol{V} \sum_{i<S} (\boldsymbol{z}_S^\top \boldsymbol{W}_{cc_i} \boldsymbol{z}_i)\, \boldsymbol{z}_i. \tag{5}$$

We fit each predictor by least squares over its function class $\mathcal{M}$, and measure its excess risk:

$$\hat{\boldsymbol{f}} \in \operatorname*{arg\,min}_{\boldsymbol{f}\in\mathcal{M}} \sum_{i\in[T]} \|\boldsymbol{y}_i - \boldsymbol{f}(c_i, e_i, \boldsymbol{X}_i)\|_2^2, \qquad \mathcal{E}(\hat{\boldsymbol{f}}) \doteq \mathbb{E}\left[\|\hat{\boldsymbol{f}}(c, e, \boldsymbol{X}) - \boldsymbol{f}^\star(c, e, \boldsymbol{X})\|_2^2 \,\middle|\, \hat{\boldsymbol{f}}\right]. \tag{6}$$

Theorem 1 (Informal version of Theorem 4): *Let $C \geq 2$.*

(a) Both predictors can represent $\boldsymbol{f}^\star$, unlike a flat predictor with a bilinear score on raw tokens.

(b) As $T \to \infty$, the excess risk of either predictor is asymptotically $\sigma^2 \chi_d^2 / T$, where d is the number of directions in weight space that change the prediction. For $r \ll D$ and $E \gg 1$,

$$d_\mathrm{N} \approx C(D^2 + ErD), \qquad d_\mathrm{F} \approx C(CD^2 + 2ErD), \qquad \frac{d_\mathrm{F}}{d_\mathrm{N}} \approx \frac{CD + 2Er}{D + Er} \in [2, C]. \tag{7}$$

(c) If a sample costs F_0 FLOPs plus F_1 FLOPs per scored token, the flat predictor requires

$$\frac{d_\mathrm{F}}{d_\mathrm{N}} \cdot \frac{S_0 + S - 1}{S_0 + (S-1)/C}, \qquad S_0 \doteq F_0/F_1, \tag{8}$$

times as many FLOPs for the same expected excess risk (in the large-data limit $T \to \infty$), a factor that grows in S from $d_\mathrm{F}/d_\mathrm{N}$ to $C d_\mathrm{F}/d_\mathrm{N}$.

The count d has three parts: the patterns, the row spaces, i.e., the r-dimensional subspaces of $\boldsymbol{m}$ on which the maps depend, and the maps from the row spaces to the output (Figure 5b,c). Both predictors have CE maps to fit. The flat predictor also has one row space per expert, fit from the samples of that expert, and C^2 pattern blocks, of which the $C(C-1)$ blocks with $c' \neq c$ are zero at the target and have to be fit as zero from data. The nested predictor has one row space and one pattern per column, fit from the samples of all experts of the column, and its routing fixes the other blocks to zero. The ratio is near C when the patterns dominate the parameter count and near 2 when the experts do. With the routing, the nested predictor scores only $1/C$ of the context, so part (c) is the multiplier $R(S)$ of Section 4.2 with $\rho = 1/C$, which grows with S as in Figure 4. As in Table 4, both parts of the nesting are needed: with $C = 1$ the predictors coincide, and with column-routed attention and experts $\boldsymbol{W}_e$ shared by all columns, the column space of $\boldsymbol{W}_e\boldsymbol{A}_c$ lies in that of $\boldsymbol{W}_e$ for every c, so this predictor cannot represent a target in which the column space of $\boldsymbol{B}^\star_{ce}$ changes with c.

Continual training and forgetting. Each predictor starts from weights that represent $\boldsymbol{f}^\star$ and is trained by gradient flow on the population squared loss of a new domain Q. Every sample of Q has the coarse label c_Q, e.g., a new corpus of code, and new maps $\boldsymbol{G}'_e \neq \boldsymbol{G}^\star_{c_Q e}$, and otherwise follows Model 1. The forgetting on the old labels (c, e) at time t is the increase of their risk since time 0.

Proposition 2 (Informal version of Proposition 5): *In the nested predictor, the columns $c \neq c_Q$ never change, so there is no forgetting on any other coarse label. In the flat predictor, $\boldsymbol{f}(c, e, \boldsymbol{X}; t) = \boldsymbol{G}^\star_{ce}\boldsymbol{V}(t)\boldsymbol{m}$ for $c \neq c_Q$, and the forgetting on (c, e) is*

$$\frac{t^2}{2}\|\boldsymbol{B}^\star_{ce}\boldsymbol{A}^\star_c\boldsymbol{A}^{\star\top}_{c_Q}\boldsymbol{\Xi}\,\boldsymbol{\Sigma}^{1/2}_{ce}\|^2_F + O(t^3), \tag{9}$$

where $\boldsymbol{\Sigma}_{ce} \doteq \mathbb{E}[\boldsymbol{m}\boldsymbol{m}^\top \mid c, e]$ and $\boldsymbol{\Xi} \neq \boldsymbol{0}$ whenever $\dot{\boldsymbol{V}}(0) \neq \boldsymbol{0}$. For orthonormal rows, the coefficient lies between multiples of $\cos^2\theta_r$ and $\cos^2\theta_1$, where $\theta_1 \leq \cdots \leq \theta_r$ are the principal angles between the row spaces of $\boldsymbol{A}^\star_c$ and $\boldsymbol{A}^\star_{c_Q}$.

In the flat predictor, the shared value map is the only source of forgetting on other labels, and the forgetting grows with the overlap between the row spaces of $\boldsymbol{A}^\star_c$ and $\boldsymbol{A}^\star_{c_Q}$. This is consistent with Figure 3.[2] Both results depend on how the relevant tokens, the patterns, and the row spaces vary across coarse labels. Nesting improves the sample and compute efficiency when these differ between labels, and, with all weights trained, it removes the forgetting through the shared value map when the row spaces overlap, which is the generic case.

6 Conclusion

In this work, we introduced nested sparsity, a conditional computation mechanism in which a router sends each token to several columns, and each column contains its own attention operators and its own experts. The resulting architecture, Nestor, generalizes both the MoE transformer and the dense transformer. In language model pretraining at scales from 100M to 1B active parameters, the MoE transformer needs 1.13–1.29× the compute of Nestor to reach the same loss at an equal number of stored parameters, and this factor shows no downward trend as the scale grows (Section 3). In continual pretraining, Nestor adapts to new domains as well as the MoE transformer while forgetting less (Section 4.1), and its advantage in long context extension grows with the context length at equal FLOPs (Section 4.2). Our theoretical analysis (Section 5) provides conceptual reasoning for why Nestor can obtain such empirical results in the context of highly structured data.

Our study has several limitations, which suggest directions for future work. First, our largest model has 1B active parameters, and we evaluate models by their held-out loss; an important next step is to test whether the compute multiplier persists at larger scales and carries over to downstream tasks, such as standard benchmarks after post-training. Second, Nestor is in some sense a meta-architecture, and we studied only the case in which each column uses full causal attention. Columns with sparse or linear attention would compound the savings at long contexts. More broadly, columns comprised of subnetworks with interesting properties could preserve them through the meta-architecture.

[2] Here, the map $\boldsymbol{V}$ is in some sense redundant: freezing it would remove this forgetting without changing the function class. Proposition 2 compares the predictors with attention and expert weights trained, as in Section 4.1.

AI Usage Statement

We used generative AI tools to assist in implementing and running experiments, developing and writing theoretical results, formatting figures, and drafting the appendices. The experiments were designed by the authors, with coding agent-style AI assistance in implementation and execution; the experimental designs, implementations, and results were verified by the authors. The direction of the theoretical results was wholly driven by the authors; the results and proofs were obtained by the authors, then both the claims and proofs were refined with AI assistance, and finally underwent substantial review, revision, and editing by the authors. AI tools also assisted with formatting figures (though all raw numbers were computed using non-AI means such as human-verified Python code) and drafting and editing appendix text, all under author supervision. No other uses of AI were made, including for ideation or hypothesis generation. The authors take full responsibility for the final content of this work, including all experimental results, theoretical results, figures, and text produced with AI assistance.

References

Samira Abnar, Harshay Shah, Dan Busbridge, Alaaeldin El-Nouby, Joshua M. Susskind, and Vimal Thilak. Parameters vs FLOPs: Scaling laws for optimal sparsity for mixture-of-experts language models. In *International Conference on Machine Learning*, 2025.

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, et al. CoLT5: Faster long-range transformers with conditional computation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. In *International Conference on Learning Representations*, 2024.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Wuyang Chen, Yanqi Zhou, Nan Du, Yanping Huang, James Laudon, Zhifeng Chen, and Claire Cui. Lifelong language pretraining with distribution-specialized experts. In *International Conference on Machine Learning*, 2023.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified scaling laws for routed language models. In *International Conference on Machine Learning*, 2022.

Cohere. Command A: An enterprise-ready large language model. *arXiv preprint arXiv:2504.00698*, 2025.

Róbert Csordás, Piotr Piekos, Kazuki Irie, and Jürgen Schmidhuber. SwitchHead: Accelerating transformers with mixture-of-experts attention. In *Advances in Neural Information Processing Systems*, 2024.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Thang Doan, Mehdi Abbana Bennani, Bogdan Mazoure, Guillaume Rabusseau, and Pierre Alquier. A theoretical analysis of catastrophic forgetting through the NTK overlap matrix. In *International Conference on Artificial Intelligence and Statistics*, pp. 1072–1080, 2021.

Mathias Drton. Likelihood ratio tests and singularities. *The Annals of Statistics*, 37(2):979–1012, 2009.

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.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. PathNet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Robert M. French. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pp. 173–178, 1991.

Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. Data engineering for scaling language models to 128K context. In *International Conference on Machine Learning*, 2024.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Charles J. Geyer. On the asymptotics of constrained M-estimation. *The Annals of Statistics*, 22(4): 1993–2010, 1994.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A. Smith, and Luke Zettlemoyer. DEMix layers: Disentangling domains for modular language modeling. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022.

Jeff Hawkins. *A Thousand Brains: A new theory of intelligence*. Basic Books, 2021.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*, 2022.

Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats L. Richter, Quentin Anthony, Timothée Lesort, Eugene Belilovsky, and Irina Rish. Simple and scalable strategies to continually pre-train large language models. *Transactions on Machine Learning Research*, 2024.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. MoH: Multi-head attention as mixture-of-head attention. In *International Conference on Machine Learning*, 2025.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. https://kellerjordan.github.io/posts/muon/, 2024.

Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, 1994.

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 preprint arXiv:2001.08361*, 2020.

Kimi Team. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025a.

Kimi Team. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025b.

Kimi Team. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Hongbo Li, Sen Lin, Lingjie Duan, Yingbin Liang, and Ness B. Shroff. Theory on mixture-of-experts in continual learning. In *International Conference on Learning Representations*, 2025.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, et al. DataComp-LM: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Jessy Lin, Luke Zettlemoyer, Gargi Ghosh, Wen-Tau Yih, Aram Markosyan, Vincent-Pierre Berges, and Barlas Oğuz. Continual learning via sparse memory finetuning. *arXiv preprint arXiv:2510.15103*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. MoBA: Mixture of block attention for long-context LLMs. *arXiv preprint arXiv:2502.13189*, 2025.

Jan Ludziejewski, Jakub Krajewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling laws for fine-grained mixture of experts. In *International Conference on Machine Learning*, 2024.

Jan Ludziejewski, Maciej Pióro, Jakub Krajewski, Maciej Stefaniak, Michał Krutul, Jan Małaśnicki, Marek Cygan, Piotr Sankowski, Kamil Adamczewski, Piotr Miłoś, et al. Joint MoE scaling laws: Mixture of experts can be memory efficient. In *International Conference on Machine Learning*, 2025.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 33:3776–3786, 2025. arXiv:2308.08747.

Arun Mallya and Svetlana Lazebnik. PackNet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Academic Press, 1989.

MiniMax. MiniMax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. OpenWebMath: An open dataset of high-quality mathematical web text. In *International Conference on Learning Representations*, 2024.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro von Werra, and Thomas Wolf. FineWeb2: One pipeline to scale them all – adapting pre-training data processing to every language. In *Conference on Language Modeling*, 2025.

Piotr Piekos, Róbert Csordás, and Jürgen Schmidhuber. Mixture of sparse attention: Content-based learnable sparse attention via expert-choice routing. *arXiv preprint arXiv:2505.00315*, 2025.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Yikang Shen, Zheyu Zhang, Tianyou Cao, Shawn Tan, Zhenfang Chen, and Chuang Gan. ModuleFormer: Modularity emerges from mixture-of-experts. *arXiv preprint arXiv:2306.04640*, 2023.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Daniel Li, Wen-tau Yih, Jason Weston, et al. Branch-train-MiX: Mixing expert LLMs into a mixture-of-experts LLM. In *Conference on Language Modeling*, 2024.

Benjamin Thérien, Charles-Étienne Joseph, Zain Sarwar, Ashwinee Panda, Anirban Das, Shi-Xiong Zhang, Stephen Rawls, Sambit Sahu, Eugene Belilovsky, and Irina Rish. Continual pre-training of MoEs: How robust is your router? *Transactions on Machine Learning Research*, 2025.

Aad W. van der Vaart. *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, UK, 1998.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. In *Advances in Neural Information Processing Systems*, 2020.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. Effective long-context scaling of foundation models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 2021. arXiv:2203.03466.

Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Annual Meeting of the Association for Computational Linguistics*, 2025.

Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. Mixture of attention heads: Selecting attention heads per token. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 4150–4162, 2022.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V. Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems*, 2022.

A Related Work

Scaling laws and sparsity. The loss of a language model is well described by a power law in the number of parameters, the number of training tokens, and the training compute (Kaplan et al., 2020), which determines the compute-optimal allocation between model size and data (Hoffmann et al., 2022). Several works extend these laws to MoE transformers. Clark et al. (2022) fit a law that is jointly a function of the model size and the number of experts. Ludziejewski et al. (2024) add the granularity of the experts as a variable, and find that MoE transformers with appropriately sized experts outperform dense transformers at every compute budget. Abnar et al. (2025) study the sparsity level, i.e., the fraction of inactive parameters, and find that the optimal sparsity increases with the model size, and Ludziejewski et al. (2025) fit a joint law for dense and MoE models under memory constraints. In all of these works the sparse component is solely the MLP. To avoid ambiguity when proposing and evaluating scaling laws using multiple sparsity axes, we compare architectures at the *same* sparsity: we fit a scaling law for each architecture and report the projected compute efficiency compared to a strong MoE transformer baseline to reach the same held-out loss.

Conditional computation and mixtures of experts. Mixtures of experts were introduced by Jacobs et al. (1991), and Jordan & Jacobs (1994) extended them to a tree of gating networks. Shazeer et al. (2017) revived them as a sparsely gated layer in deep networks, in line with the proposal of conditional computation (Bengio et al., 2013), and later work brought them into the transformer and refined the routing (Lepikhin et al., 2021; Fedus et al., 2022; Zhou et al., 2022). Recent MoE transformers use many small experts, some of which are shared by all tokens (Dai et al., 2024), and balance the load of the experts by adjusting a bias in the router (Wang et al., 2024; DeepSeek-AI, 2024); we adopt similar practices. Nested sparsity is closest in spirit to the hierarchical mixture of experts, since the expert router of Nestor acts within the column chosen by the column router. The difference is that in a hierarchical mixture of experts both levels of routing select among MLPs, whereas in Nestor the first level selects an attention operator together with its context, and it is held fixed over several layers. Other works route tokens around whole layers: Raposo et al. (2024) let only the top-K tokens pass through the attention operator and MLP of each layer, and Ainslie et al. (2023) apply a heavy attention operator and MLP only to the routed tokens and a light one to all tokens. In these methods there is a single routed module per layer, and tokens that are not selected skip it; in Nestor, every token is processed in every layer, by modules that depend on the token.

Sparse attention and mixtures of attention heads. A large body of work reduces the quadratic cost of attention by making the attention map sparse. The pattern can be fixed in advance, such as local or strided windows (Child et al., 2019; Beltagy et al., 2020); it can depend on the content through hashing or clustering of the queries and keys (Kitaev et al., 2020; Roy et al., 2021); or it can be selected for each query by a top-K rule over blocks or tokens (Yuan et al., 2025; Lu et al., 2025; DeepSeek-AI, 2025). A complementary line of work makes the attention parameters sparse, by routing each token to a subset of the attention heads or of their projections (Zhang et al., 2022; Csordás et al., 2024; Jin et al., 2025), possibly together with an MoE operator for the MLP (Shen et al., 2023). The mechanism of Piekos et al. (2025), in which each head selects a fixed number of tokens and attends only among them, is the closest to column-routed attention. We compare Nestor with these methods in detail in Section 2. In short, column-routed attention makes both the pattern and the parameters sparse with one routing decision, which is made by the token, held fixed over a block of layers, and shared with the experts. In particular, in our experiments Nestor improves on baselines that use full attention, which sparse attention methods aim to approximate.

Interleaved attention. Many LLMs reduce the cost of attention at the level of the layer stack, by using full attention in only some of the layers. GPT-3 alternates dense and locally banded sparse attention layers (Brown et al., 2020), and more recent models interleave sliding window and global attention layers at ratios from one to one up to five to one (Gemma Team, 2024; 2025; Cohere, 2025; OpenAI, 2025). Hybrid models replace the local layers by state space or linear attention layers (Lieber et al., 2024; MiniMax, 2025; Kimi Team, 2025b). Nestor follows the same principle with a different local operator: the number of active columns increases with depth, so the early blocks compute attention within routed subsequences and the last block computes full attention over the whole sequence (Section 2). Unlike a sliding window, the context of a column is input-dependent and can comprise tokens from anywhere in the sequence.

Continual pretraining and forgetting. Continued pretraining on text from a target domain improves performance in that domain (Gururangan et al., 2020), and it is the standard way to obtain specialized models for code and mathematics (Rozière et al., 2023; Azerbayev et al., 2024; Shao et al., 2024). Its main difficulty is catastrophic forgetting (McCloskey & Cohen, 1989; French, 1999; Kirkpatrick et al., 2017; Luo et al., 2025), which is usually mitigated by replaying pretraining data (Ibrahim et al., 2024). It has long been observed that forgetting is caused by the overlap of distributed representations, and that sparser representations reduce it (French, 1991). Accordingly, many continual learning methods give each task its own parameters, by adding, freezing, or masking parts of the network (Rusu et al., 2016; Fernando et al., 2017; Aljundi et al., 2017; Mallya & Lazebnik, 2018; Wortsman et al., 2020). For language models, modularity is typically obtained with experts: Gururangan et al. (2022) condition the experts on the domain and adapt to a new domain by training a new expert with the rest of the model frozen, Chen et al. (2023) add and freeze experts as the distribution changes, Sukhbaatar et al. (2024) train domain models separately and merge their MLPs into the experts of an MoE transformer, and Lin et al. (2025) update a sparse set of memory slots. Shen et al. (2023) route tokens among both attention and MLP modules, and find that their model forgets less than a dense model and can be extended with new modules. Thérien et al. (2025) find that the routers of standard MoE transformers are robust to the distribution shifts of continual pretraining, and Li et al. (2025) analyze MoE models in continual learning theoretically. When an MoE transformer is continually pretrained with its attention parameters trainable, these parameters are shared by all tokens, so the gradient of every new token changes them. In NESTOR, each layer applies to a token only the attention operators and experts of its columns, besides the shared experts, with no task labels and no changes to training. In the one-layer model of Section 5, where each token uses one column, this removes the forgetting on the other coarse labels. Our analysis of forgetting in Section 5 uses the overlap between the row spaces of the value maps of the old and the new data, as in Doan et al. (2021).

B MATHEMATICAL DESCRIPTION OF THE ARCHITECTURE

This section specifies the forward pass of NESTOR as trained in all experiments. It expands (2) and (3) with the normalization, the residual connections, and the routing details that Section 2 omits. The initialization and the learning rate of each parameter are in Section D, and the sizes of each model are in Section C.1.

Notation and overall structure. The input is a sequence of S tokens, and all features have width D. Let $\mathrm{rms}(\boldsymbol{v}) \doteq (\|\boldsymbol{v}\|_2^2/D + \epsilon)^{1/2}$, and let $\mathrm{Norm}(\boldsymbol{v}) \doteq \boldsymbol{\gamma} \odot \boldsymbol{v}/\mathrm{rms}(\boldsymbol{v})$ be the RMS normalization with a learned gain $\boldsymbol{\gamma} \in \mathbb{R}^D$; each use of Norm below has its own gain. No linear map has a bias. The network embeds the token $j \in [S]$ as $\boldsymbol{e}_j \in \mathbb{R}^D$, without a learned position embedding, and then applies B blocks. Block $\ell \in [B]$ has M_ℓ layers, C columns, and $K_{\mathrm{c},\ell}$ active columns per token. Below we describe one block and omit the index ℓ from its parameters and its routing variables. Each block produces one output $\bar{\boldsymbol{u}}_j^\ell \in \mathbb{R}^D$ for each token, which we define in (17). The input of block ℓ is $\boldsymbol{x}_j \doteq \bar{\boldsymbol{u}}_j^{\ell-1}$, with $\bar{\boldsymbol{u}}_j^0 \doteq \boldsymbol{e}_j$.

Column router. The router is computed once per block from the block input. Its scores are $\boldsymbol{s}_j \doteq \boldsymbol{W}_2\ \mathrm{GELU}(\boldsymbol{W}_1 \boldsymbol{x}_j) \in \mathbb{R}^C$, with a hidden layer of width 128. During training we add input-dependent noise (Shazeer et al., 2017), i.e., we replace $\boldsymbol{s}_j$ by $\boldsymbol{s}_j + \boldsymbol{\xi}_j \odot \mathrm{softplus}(\boldsymbol{W}_2'\ \mathrm{GELU}(\boldsymbol{W}_1' \boldsymbol{x}_j))$ with $\boldsymbol{\xi}_j \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$; there is no noise at evaluation. The selected columns and their gating weights are

$$\mathcal{C}_j \doteq \mathrm{TopK}(\boldsymbol{s}_j + \boldsymbol{b},\, K_\mathrm{c}) \subseteq [C], \qquad w_{jc} \doteq \frac{\exp(s_{jc})}{\sum_{c' \in \mathcal{C}_j} \exp(s_{jc'})} \quad \text{for } c \in \mathcal{C}_j, \tag{10}$$

and $\mathcal{U}_c \doteq \{i \in [S] \colon c \in \mathcal{C}_i\}$ is the set of tokens routed to column c. The bias $\boldsymbol{b} \in \mathbb{R}^C$ is used only for the selection: the gating weights do not contain it, and it receives no gradient. During training, we also include small additive Gaussian noise within the top-k operation as per Shazeer et al. (2017). After each optimizer step we update the bias by (Wang et al., 2024)

$$b_c \leftarrow b_c + \gamma\, \mathrm{sign}\Big(\frac{1}{C} \sum_{c' \in [C]} n_{c'} - n_c\Big), \tag{11}$$

where n_c is the number of tokens in the batch which selected the column c, and $\gamma = 10^{-3}$. This update raises the bias of the columns with less than the average load. We use no auxiliary balancing loss. The sets $\mathcal{C}_j$ and the weights w_{jc} are used by all M layers of the block. In the last block $K_\mathrm{c} = C$, so that $\mathcal{C}_j = [C], \mathcal{U}_c = [S]$, and the router only computes the weights w_{jc}.

Attention residual. All residual connections are replaced by the following operator, which acts on each token separately. Given a list of *sources* $\boldsymbol{z}_1, \dots, \boldsymbol{z}_N \in \mathbb{R}^D$, the operator returns the mixture

$$\textbf{AttnRes}(\boldsymbol{z}_1, \dots, \boldsymbol{z}_N) \doteq \sum_{i=1}^{N} \frac{\exp(\langle \boldsymbol{q}, \boldsymbol{z}_i \rangle / (\sqrt{D}\,\mathrm{rms}(\boldsymbol{z}_i)))}{\sum_{i'=1}^{N} \exp(\langle \boldsymbol{q}, \boldsymbol{z}_{i'} \rangle / (\sqrt{D}\,\mathrm{rms}(\boldsymbol{z}_{i'})))}\, g_i\, \frac{\boldsymbol{z}_i}{\mathrm{rms}(\boldsymbol{z}_i)}, \tag{12}$$

with a learned query $\boldsymbol{q} \in \mathbb{R}^D$ and learned gains $g_1, \dots, g_N \in \mathbb{R}$. Each use of **AttnRes** below has its own query and gains. This is the attention residual of Kimi Team (2026) with two changes, which keep the scales of the mixture and of its logits independent of the width: in the mixture, the sources are normalized to unit RMS, as in the logits, and multiplied by the gains g_i; and the logits are divided by $\sqrt{D}$. The query does not depend on the token, so the mixture weights depend on the input only through the sources. We initialize $\boldsymbol{q} = \boldsymbol{0}$, so that the mixture is initially uniform.

One layer. Within a block, each token j has one residual stream $\boldsymbol{u}_{jc}^m \in \mathbb{R}^D$ for each active column $c \in \mathcal{C}_j$, where $m = 0, \frac{1}{2}, 1, \dots, M$ counts the attention and MLP operators applied so far, and $\boldsymbol{u}_{jc}^0 \doteq \boldsymbol{0}$. Let $\bar{\boldsymbol{u}}_j^m \doteq \frac{1}{K_\mathrm{c}} \sum_{c \in \mathcal{C}_j} \boldsymbol{u}_{jc}^m$ be the mean of the streams. Layer $m \in [M]$ computes, for each $c \in \mathcal{C}_j$,

$$\boldsymbol{h}_j \doteq \mathrm{Norm}(\textbf{AttnRes}(\boldsymbol{e}_j, \bar{\boldsymbol{u}}_j^1, \dots, \bar{\boldsymbol{u}}_j^{\ell-1}, \bar{\boldsymbol{u}}_j^{m-1})), \tag{13}$$

$$\boldsymbol{u}_{jc}^{m-1/2} \doteq \boldsymbol{u}_{jc}^{m-1} + w_{jc}\, \textbf{Attn}_c(\boldsymbol{h}_j,\, (\boldsymbol{h}_i)_{i \in \mathcal{U}_c,\, i \le j}), \tag{14}$$

$$\boldsymbol{h}'_j \doteq \mathrm{Norm}(\textbf{AttnRes}(\boldsymbol{e}_j, \bar{\boldsymbol{u}}_j^1, \dots, \bar{\boldsymbol{u}}_j^{\ell-1}, \bar{\boldsymbol{u}}_j^{m-1/2})), \tag{15}$$

$$\boldsymbol{u}_{jc}^{m} \doteq \boldsymbol{u}_{jc}^{m-1/2} + \lambda_j^\mathrm{s} \sum_{e \in \mathcal{E}_j^\mathrm{s}} \alpha_{je}^\mathrm{s}\, \textbf{MLP}_e^\mathrm{s}(\boldsymbol{h}'_j) + \lambda_j^\mathrm{p}\, w_{jc} \sum_{e \in \mathcal{E}_{jc}} \alpha_{jce}\, \textbf{MLP}_{c,e}(\boldsymbol{h}'_j). \tag{16}$$

In (13) and (15), the sources are the embedding, the outputs of the earlier blocks (here the superscripts $1, \dots, \ell - 1$ are block indices), and the current mean of the streams of this block. We omit the last source in (13) when $m = 1$, since it is zero. The outputs of the attention operator of column c and of the experts $\textbf{MLP}_{c,e}$ of column c are added only to the stream of column c, and the outputs of the shared experts $\textbf{MLP}_e^\mathrm{s}$ are added to every stream. All columns take the same inputs $\boldsymbol{h}_j$ and $\boldsymbol{h}'_j$, which depend on the streams only through their mean. The output of the block is the final mean,

$$\bar{\boldsymbol{u}}_j^\ell \doteq \bar{\boldsymbol{u}}_j^{M_\ell} = \frac{1}{K_\mathrm{c}} \sum_{c \in \mathcal{C}_j} \boldsymbol{u}_{jc}^{M_\ell}. \tag{17}$$

By (14) and (16), this mean is the sum of the updates in (2) and (3) over the layers of the block, with a factor $1/K_\mathrm{c}$ on the terms of the columns and a factor λ_j^s or λ_j^p on the experts.

Attention in a column. Each column has its own projections $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{H_\mathrm{c} D_\mathrm{h} \times D}$ and $\boldsymbol{W}_O \in \mathbb{R}^{D \times H_\mathrm{c} D_\mathrm{h}}$ in each layer, where H_c is the number of heads of one column and D_h is the width of a head. In the head $\eta \in [H_\mathrm{c}]$, the token $j \in \mathcal{U}_c$ has the query, key, and value $\boldsymbol{q}_j^\eta, \boldsymbol{k}_j^\eta, \boldsymbol{v}_j^\eta \in \mathbb{R}^{D_\mathrm{h}}$, which are the corresponding slices of $\boldsymbol{W}_Q \boldsymbol{h}_j$, $\boldsymbol{W}_K \boldsymbol{h}_j$, and $\boldsymbol{W}_V \boldsymbol{h}_j$; we apply rotary position embeddings (Su et al., 2024) to the queries and the keys, using the position j of the token in the whole sequence and not its position within $\mathcal{U}_c$. The output of the head is

$$\boldsymbol{o}_j^\eta \doteq \sum_{i \in \mathcal{U}_c,\, i \le j} \frac{\exp(\langle \boldsymbol{q}_j^\eta, \boldsymbol{k}_i^\eta \rangle / D_\mathrm{h})}{\sum_{i' \in \mathcal{U}_c,\, i' \le j} \exp(\langle \boldsymbol{q}_j^\eta, \boldsymbol{k}_{i'}^\eta \rangle / D_\mathrm{h})}\, \boldsymbol{v}_i^\eta, \tag{18}$$

and $\textbf{Attn}_c(\boldsymbol{h}_j, (\boldsymbol{h}_i)_{i \in \mathcal{U}_c, i \le j}) \doteq \boldsymbol{W}_O(\boldsymbol{o}_j^1, \dots, \boldsymbol{o}_j^{H_\mathrm{c}})$. The division by D_h instead of $\sqrt{D_\mathrm{h}}$ follows μP (Yang et al., 2021). In the implementation, we gather the $|\mathcal{U}_c|$ tokens routed to column c, apply a standard causal attention kernel to this shorter sequence, and scatter the result back to the positions of these tokens. To quantify the saving, give each column H/K_c heads, so that all projections cost

the same FLOPs as in a transformer with H heads. If the routing is balanced, i.e., $|\mathcal{U}_c| \approx SK_c/C$, then the quadratic term in the attention FLOPs falls from S^2D to

$$C \cdot \Big(\frac{SK_c}{C}\Big)^2 \cdot \frac{D}{K_c} = \frac{K_c}{C} \cdot S^2 D. \tag{19}$$

The saving $(1 - K_c/C)S^2D$ grows with the context length S.

Experts. Every expert is an MLP with gated linear units (Shazeer, 2020), $\mathbf{MLP}(\boldsymbol{h}) \doteq \boldsymbol{W}_{\text{down}}(\text{SiLU}(\boldsymbol{W}_{\text{gate}}\boldsymbol{h}) \odot \boldsymbol{W}_{\text{up}}\boldsymbol{h})$. All expert routers are linear maps of $\boldsymbol{h}'_j$, and each has its own balancing bias which is updated by (11). Each layer has E_s shared experts with one router $\boldsymbol{r}^s$, and each column c has E experts with its own router $\boldsymbol{r}_c$:

$$\mathcal{E}^s_j \doteq \text{TopK}(\boldsymbol{r}^s(\boldsymbol{h}'_j) + \boldsymbol{b}^s,\, K_s) \subseteq [E_s], \qquad \alpha^s_{je} \doteq \frac{\exp(r^s_e(\boldsymbol{h}'_j))}{\sum_{e' \in \mathcal{E}^s_j} \exp(r^s_{e'}(\boldsymbol{h}'_j))}, \tag{20}$$

$$\mathcal{E}_{jc} \doteq \text{TopK}(\boldsymbol{r}_c(\boldsymbol{h}'_j) + \boldsymbol{b}_c,\, K_e) \subseteq [E], \qquad \alpha_{jce} \doteq \frac{\exp(r_{c,e}(\boldsymbol{h}'_j))}{\sum_{e' \in [E]} \exp(r_{c,e'}(\boldsymbol{h}'_j))}. \tag{21}$$

The weights of the shared experts are normalized over the selected experts, as in (10). The weights α_{jce} are normalized over all E experts of the column (Fedus et al., 2022), so that they depend on the router also when $K_e = 1$. The load n_e in the update (11) of $\boldsymbol{b}_c$ counts only the tokens in $\mathcal{U}_c$. The two families of experts are weighted by a gate which is computed from the same input,

$$(\lambda^s_j, \lambda^p_j) \doteq \text{softmax}(\boldsymbol{W}_\lambda \boldsymbol{h}'_j), \qquad \boldsymbol{W}_\lambda \in \mathbb{R}^{2\times D}. \tag{22}$$

In all experiments $E_s = 8$ and $K_s = 2$. The hidden widths of the experts are $2D/K_s$ for a shared expert and $2D/(K_cK_e)$ for an expert of a column, rounded to a multiple of 8, so that each token uses hidden units of total width $K_s \cdot 2D/K_s + K_cK_e \cdot 2D/(K_cK_e) = 4D$ in every block. We use $K_e = 2$ in the blocks where the resulting width is at least 160, and $K_e = 1$ in the other blocks.

Output. The input of the output head is an attention residual over the embedding, the output of every block, and the deviation of every stream from the mean of its block:

$$\boldsymbol{x}^{\text{out}}_j \doteq \text{Norm}(\mathbf{AttnRes}(\boldsymbol{e}_j,\, (\bar{\boldsymbol{u}}^\ell_j)_{\ell\in[B]},\, (\boldsymbol{u}^\ell_{jc} - \bar{\boldsymbol{u}}^\ell_j)_{\ell\in[B],\, c\in\mathcal{C}^\ell_j})), \tag{23}$$

where $\boldsymbol{u}^\ell_{jc}$ and $\mathcal{C}^\ell_j$ are the final stream $\boldsymbol{u}^{M_\ell}_{jc}$ and the selected columns of block ℓ. The logits of the next token are a linear map of $\boldsymbol{x}^{\text{out}}_j$, whose weights are not tied to the embedding, and the training loss is the cross-entropy of the next token, with no auxiliary term. This is the only place where the individual streams are used, since the layers in (13) and (15) use only the means. We use the deviations because the streams of a block are close to their mean after the normalization in (12), so they would add nearly the same source several times. The deviations sum to zero over $c \in \mathcal{C}^\ell_j$ and contain only what is specific to a column. Their gains in (12) are initialized at 0, and all other gains at 1, so that at initialization the output does not depend on the deviations.

Special cases. Let $C = K_c = 1$. Then $w_{j1} = 1$ and $\mathcal{U}_1 = [S]$, each block has one stream, the deviations in (23) are zero, and (13)–(16) are M layers of an MoE transformer with full causal attention and the block attention residual of Kimi Team (2026), with the two changes stated after (12). Our MoE baseline is this case with a single family of $E = 24$ experts, $K_e = 3$, the gating weights of (20), and no gate (22). The dense baseline further sets $E = K_e = 1$, with a single SwiGLU of hidden width $4D$ per layer and gating weight 1. Both baselines retain the same block attention residual described above.

C Further Experiments and Ablations

C.1 Architecture Configurations and Hyperparameters

This subsection gives the configuration of Nestor and of the two baselines at the four scales of Section 3, the rules that set every quantity other than the number of active columns, the experiment that selected the schedule of active columns over depth, and the FLOP accounting used throughout the paper.

Sizes. Table 1 lists the four scales. The dense transformer and the MoE transformer share the width D, the depth, 16 heads of width $D/16$, and the training budget; the MoE transformer replaces each MLP by $E = 24$ experts of which $K_e = 3$ are active. NESTOR has $C = 8$ columns in every block and the same schedule of active columns at every scale, $K_c = 3, 4, 5, 6, 6, 7$ in the first six blocks and $K_c = C$ in the last. The number of layers per block grows with the scale, and the number of experts per column E is raised at 500M and 1B so that the stored parameters match those of the MoE transformer. The batch grows with the scale so that every model trains for about the same number of steps.

Scale	D	Depth	$D/16$	Tokens per step	Steps	NESTOR: layers per block	E	FLOPs	Stored
100M	480	12	30	262,144	19,080	1, 1, 2, 2, 2, 3, 2 (13)	8	0.986×	0.38B
200M	640	16	40	524,288	19,080	2, 2, 2, 3, 3, 3, 2 (17)	8	0.981×	0.76B
500M	960	24	60	1,048,576	23,844	3, 3, 3, 4, 4, 5, 3 (25)	12	0.979×	2.44B
1B	1280	32	80	2,097,152	23,842	3, 4, 4, 6, 6, 6, 4 (33)	15	0.989×	5.53B

Table 1: The four scales. Width, depth, head width, and training budget are those of the dense transformer and of the MoE transformer; the sequence length is 4096. For NESTOR, the layers of the seven blocks (total in parentheses) at the schedule $K_c = 3, 4, 5, 6, 6, 7, 8$, the number of experts per column E, its forward FLOPs per token at $S = 4096$ relative to the dense transformer, and its stored parameters (the MoE transformer stores 0.37B, 0.79B, 2.41B, and 5.49B).

Derived quantities. Only C and the schedule of K_c are design choices; every other quantity that costs FLOPs follows a rule, so that two configurations that differ in the schedule differ in nothing else. Each column has $H_c = \text{round}(16/K_c)$ heads of the baselines' head width, that is $5, 4, 3, 3, 3, 2, 2$ heads in the seven blocks, so that the attention projections of a token cost within 10% of the baselines' and the only saving of a sparse block is in the products of queries with keys and of attention weights with values (19). The expert widths follow Section B: the shared experts and the experts of the columns of a token have a total hidden width of $4D$ in every block, the active MLP width of the baselines, and E sets the stored capacity without changing the active width. With heads and widths fixed, the cost of a layer depends only on the K_c of its block, and we choose the number of layers in each block to meet the budget: for a given schedule we enumerate the splits of the layers over the blocks and keep those whose forward FLOPs at $S = 4096$, with the accounting below, do not exceed the dense transformer's. Among the splits that meet the budget at 200M, we selected the one in Table 1, which places most of its layers in the later blocks where K_c is large. The alternatives of the search, which spend the same budget differently (a sparser or a denser first block, a lower endpoint of the schedule, more layers in the sparse blocks, wider attention or wider experts instead of layers), have losses higher by 0.004 to 0.033 nats. At the other scales we used splits of the same form, without further search. The remaining 1 to 2% of each budget is left unspent.

The schedule of active columns. Table 2 and Figure 6 compare ten schedules of K_c over depth on the 200M model, with everything else at the configuration above and every schedule priced within 0.97 to 1.02× of its FLOPs by the same rules: a constant $K_c \in \{2, 4, 6, 8\}$ with the depth set by the budget; the schedule of NESTOR reversed, which keeps the same layers, FLOPs, and stored parameters; two symmetric schedules, dense at the ends and sparse in the middle, and the reverse; and three two-level schedules on one multiset of layers, sparse then dense, dense then sparse, and the same layers interleaved. Because an expert of a column has width $2D/(K_c K_e)$, the stored parameters vary with the schedule, from 0.82× those of NESTOR at constant $K_c = 8$ to 1.63× at $K_c = 2$; storage does not explain the ordering below, since the model that stores the most is the worst.

Varying the number of active columns with depth generally improves held-out loss (Table 2; Figure 6, left). Among constant schedules, loss decreases as K_c increases, but even full attention falls behind NESTOR and the MoE transformer. Every varying schedule except the sparse-to-dense step outperforms all constant schedules. The sparsest constant schedule performs particularly poorly despite a smooth training loss curve.

The arrangement of layers also matters (Figure 6, right). Gradual schedules outperform the two-level schedules at similar mean K_c. When the layer composition, FLOPs, and stored parameters are held fixed, the dense-to-sparse step performs better than its reverse, and the symmetric schedule with a dense middle performs better than the one with dense ends. Alternating sparse and dense layers

matches the better step schedule, while placing a long sequence of sparse layers at the start gives the worst result among the varying schedules. Reversing the ramp of NESTOR has a smaller effect, close to the variation across seeds.

At 200M, the ramp of NESTOR and the symmetric schedule with a dense middle achieve the lowest losses, with no clear difference across the two seeds. Both begin with sparse layers and reach their greatest density at or after the middle. At 500M, after matching both schedules to approximately the storage of the MoE transformer, the ramp achieves lower loss in the single-seed comparison, motivating its use in NESTOR. The symmetric schedule remains a cheaper alternative in attention FLOPs at long contexts.

Schedule	Shape	K_c per block	Layers per block	Mean K_c	FLOPs	Stored	Loss
NESTOR (ramp up)		3, 4, 5, 6, 6, 7, 8	2, 2, 2, 3, 3, 3, 2	5.71	1.00×	762M	3.092 / 3.088
ramp down		8, 7, 6, 6, 5, 4, 3	2, 3, 3, 3, 2, 2, 2	5.71	1.00×	762M	3.101
constant $K_c = 2$		2	20	2.00	1.00×	1244M	3.582
constant $K_c = 4$		4	18	4.00	0.98×	719M	3.192
constant $K_c = 6$		6	16	6.00	1.00×	751M	3.157
constant $K_c = 8$		8	16	8.00	1.02×	627M	3.148
dense ends		8, 5, 3, 3, 5, 8	3 each	5.33	1.02×	822M	3.118
dense middle		3, 5, 8, 8, 5, 3	3 each	5.33	1.02×	822M	3.088 / 3.087
step up		3, 8	9, 8	5.35	0.97×	743M	3.169
step down		8, 3	8, 9	5.35	0.97×	743M	3.136
alternating		3, 8, 3, 8, …, 3	17 of 1	5.35	0.97×	745M	3.134

Table 2: Ten schedules of the active columns on the 200M model, $C = 8$, everything else at the configuration of NESTOR; constant $K_c = 8$ is full attention. The shape plots K_c from 2 to 8 against depth. Mean K_c is weighted by the layers per block; FLOPs are forward FLOPs per token at $S = 4096$ relative to NESTOR; stored parameters are totals. Held-out loss at the matched final step, seed 42 / seed 43 where two seeds were run.

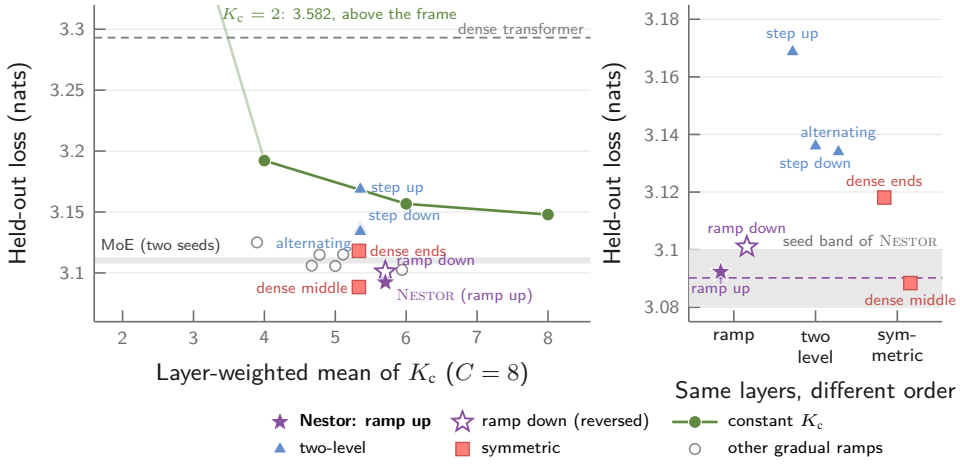


Figure 6: **The schedule of active columns.** Left: held-out loss of the ten schedules of Table 2 against the layer-weighted mean of K_c. The constant schedules (line) are the reference without a schedule; the run at $K_c = 2$ lies above the frame. The grey circles are the gradual ramps of the configuration search, the dashed line is the dense transformer, and the band spans the two seeds of the MoE transformer. Right: the order-only contrasts. Within a group, the schedules share one multiset of layers and hence identical FLOPs and stored parameters, so the spread within a group is the effect of the order alone. The shaded band is the mean of the two seeds of NESTOR plus or minus 0.010 nats, the seed band at this scale. Whiskers join two seeds.

FLOP accounting. In the forward pass of one layer of width D, a token spends $8D^2$ FLOPs on the four attention projections and $2 \cdot 3 \cdot 4D^2 = 24D^2$ FLOPs on the three matrices of a gated MLP

of width $4D$. With a causal mask, a token attends to $S/2$ tokens on average, and the products of queries with keys and of attention weights with values cost $2 \cdot 2D \cdot S/2 = 2SD$ FLOPs per token. The ratio of the two is $S/(16D)$, which is between 0.20 and 0.53 for our models at $S = 4096$. In a layer of NESTOR with C columns of width D/K_c, a token is a member of K_c columns with SK_c/C members each, so the quadratic term is $2SDK_c/C$ FLOPs per token, which is the fraction K_c/C of the $2SD$ FLOPs of full attention, as in (19). The routers, the attention residual, and the output head are not counted.

C.2 PRETRAINING RESULTS

Table 3 tabulates the pretraining runs of Section 3, which are plotted in Figure 2: the number of training tokens and the training compute at each scale, and the number of stored parameters and the held-out loss of each model.

Size	Tokens	Compute (PFLOP/s-days)	Model	Total params	Loss	Estimated compute at equal loss
100M	5B	0.02	Dense transformer	0.14B	3.578	3.1×
			MoE	0.37B	3.386	1.17×
			NESTOR	0.38B	**3.356**	1.00×
200M	10B	0.10	Dense transformer	0.23B	3.293	3.2×
			MoE	0.79B	3.108	1.13×
			NESTOR	0.76B	**3.092**	1.00×
500M	25B	0.78	Dense transformer	0.55B	2.953	4.1×
			MoE	2.41B	2.793	1.29×
			NESTOR	2.44B	**2.758**	1.00×
1B	50B	3.49	Dense transformer	1.10B	2.743	4.1×
			MoE	5.49B	2.592	1.22×
			NESTOR	5.53B	**2.571**	1.00×

Table 3: **Pretraining results at four model scales.** These are the numbers plotted in Figure 2 of Section 3. The last column is the compute that each baseline is estimated to need to match the held-out loss of NESTOR at the same scale, from the fitted law $L(F) = A\,F^{-\alpha} + L_\infty$, normalized by the training compute of NESTOR.

C.3 ARCHITECTURE ABLATIONS

Attribution of the improvement over the MoE transformer. The MLP of NESTOR differs from the MLP of the MoE transformer baseline in three ways. First, it has two groups of experts with separate routers: the E_s experts whose weights are tied across columns, and the E experts of each column. Second, the two groups are weighted by the gate (22). Third, each token uses more experts of smaller width: at the 200M active parameter scale, a token uses between 7 and 10 experts per layer in NESTOR, and 3 in the baseline. We test whether these differences, or column-routed attention alone, explain the improvement in Figure 2. In Table 4, we display the held-out loss of eight models at the 200M active parameter scale and of three models at the 500M scale. All models use the setup of Section 3, the attention residual, and an active MLP width of $4D$.

We observe the following. An MoE transformer with twice as many experts of half the width (row 2) has the same loss as the baseline (row 1), so the smaller experts of NESTOR do not explain the improvement. NESTOR with $C = 1$ has the same loss as an MoE transformer with the same number of stored parameters, both at 470M stored parameters (rows 3 and 4) and at the storage of the baseline (rows 5 and 1; at the 500M scale, rows 10 and 9). With one column, every token can use the experts of the column, so the models in rows 4, 5, and 10 have all three differences listed above and no column routing. Column-routed attention with the MLP of the baseline (rows 6 and 7) gives losses between 3.097 and 3.124 over three runs, which is the loss of the baseline within the variation between seeds. The full model (row 8), in which the experts are nested within the columns, improves on the baseline by 0.016 and 0.025 nats on the two seeds, and by 0.035 nats at the 500M scale (row 11).

	Attention	Experts	Stored	FLOPs	Held-out loss
200M active parameters, 10B tokens					
1	full	$E = 24$, $K_e = 3$ (MoE baseline)	787M	1.00×	3.108 / 3.113
2	full	$E = 48$, $K_e = 6$, half width	781M	1.00×	3.107
3	full	$E = 12$, $K_e = 3$	471M	1.00×	3.170
4	full ($C = 1$)	experts of NESTOR: $E_s = 8$, $K_s = 2$, $E = 8$, $K_e = 2$, width D	470M	1.00×	3.170
5	full ($C = 1$)	experts of NESTOR at the width of the baseline: $E_s = 16$, $K_s = 2$, $E = 8$, $K_e = 1$	787M	1.00×	3.116 / 3.113
6	column-routed	$E = 24$, $K_e = 3$, tied across columns, with an earlier schedule of active columns	935M	1.06×	3.107 / 3.124
7	column-routed	$E = 24$, $K_e = 3$, tied across columns, with the schedule of NESTOR	843M	0.98×	3.097
8	column-routed	experts of NESTOR (full model)	762M	0.98×	3.092 / 3.088
500M active parameters, 25B tokens					
9	full	$E = 24$, $K_e = 3$ (MoE baseline)	2.41B	1.00×	2.793
10	full ($C = 1$)	experts of NESTOR at the width of the baseline, as in row 5	2.41B	1.00×	2.794
11	column-routed	experts of NESTOR (full model)	2.44B	0.98×	2.758

Table 4: Held-out loss of eight models at the 200M scale and three models at the 500M scale. FLOPs are forward FLOPs per token relative to the dense transformer at $S = 4096$. Where a second seed was run, both losses are given (seed 42 / seed 43).

C.4 DOMAIN ADAPTATION SETTINGS

Starting checkpoints. We use the 500M NESTOR, 500M MoE transformer, and 1B dense transformer from Section 3. Before domain adaptation, each model undergoes context extension from 4k to 32k on 0.5B DCLM-baseline tokens, using the equal-token protocol of Section 4.2. The extension data contain documents of at least 32,768 tokens, and the rotary embedding base is 2×10^6. Table 5 reports the held-out losses of the resulting checkpoints. Each model uses the same starting checkpoint for every target domain and both replay settings. The adaptation runs are independent, and do not continue from a model adapted to another domain.

Table 5: Held-out loss before domain adaptation, after context extension to 32k. All columns use the 32k evaluation described below. These are the starting losses used to compute adaptation and forgetting in Figure 3.

Model	DCLM	PG-19	OpenWebMath	AlgebraicStack	German
500M NESTOR	2.779	2.732	2.417	1.789	4.171
500M MoE	2.810	2.765	2.449	1.813	4.224
1B dense	2.779	2.719	2.412	1.720	4.164

Data. We use the published training and test splits of PG-19, OpenWebMath, AlgebraicStack, and the German subset of FineWeb-2, as cited in Section 4.1. OpenWebMath and AlgebraicStack are taken from Proof-Pile-2. The available training data comprise the PG-19 training split, the first 12 OpenWebMath training files, all 79 AlgebraicStack training files, and the first two German training files of FineWeb-2. All corpora use the `cl100k_base` tokenizer. We separate documents with an end-of-text token, concatenate them, and train on shuffled 32,768-token sequences. Sequences may cross document boundaries; causal attention is not reset at those boundaries. The same data and sampling order are used for all three models within each setting.

Adaptation protocol. Each run trains for 477 optimizer steps with 32 sequences per step, or 1,048,576 tokens per step. This gives 500,170,752 tokens, rounded to 0.5B in the text. For both sparse models, we freeze the router parameters and load-balancing biases and disable routing noise. All remaining trainable parameters, including attention projections, experts, normalization parameters, embeddings, the output head, and attention residuals, are updated. Routing decisions are re-

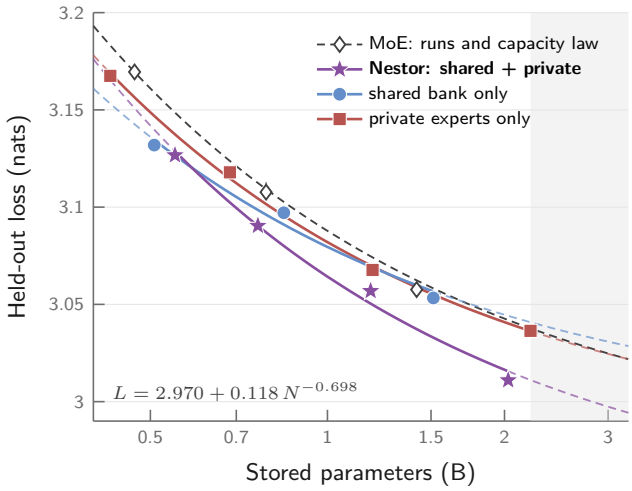


Figure 7: Ablation of the placement of the experts of NESTOR, at the 200M scale. We vary the stored parameters of three designs, with the columns, the schedule of active columns, the attention residual, the training recipe and the 10B-token budget of Section 3 held fixed, so that every model has the same active FLOPs: the design of NESTOR, $E_s = 8$ shared experts with $K_s = 2$ and $E \in \{4, 8, 16, 32\}$ experts per column with $K_e = 2$; a shared bank only, $E_s \in \{12, 24, 48\}$ with $K_s = 3$ and no experts in the columns, which is the MLP of the MoE baseline with column-routed attention; and experts in the columns only, $E \in \{2, 4, 8, 16\}$ with $K_e = 1$ and no shared experts. The dashed line is the capacity law $L(N) = c + aN^{-0.698}$ of the MoE baseline, fitted to its runs with 16, 24 and 48 experts (diamonds), and each design is fitted with the same exponent, solid over its measured span and dashed beyond it. Whiskers are seed twins. NESTOR is 0.018–0.031 nats below the law at every size. With experts in the columns only, the loss is below the law at small sizes and approaches the law as the size grows. With a shared bank only, the loss is on the law from 24 to 48 experts. The shared bank with 96 experts is omitted: its router collapsed, and 54 of its 96 experts per layer received under 1% of the tokens.

computed from the current hidden states, so freezing the routers does not fix individual token assignments. The dense model is fully fine-tuned. Each model, domain, and replay setting has one run, using random seed 42. The data sampler uses seed 0 to shuffle the training sequences.

Optimization. All three models use the same optimizer settings in both replay conditions. We initialize fresh optimizer states and restart the learning-rate schedule for each run. Muon updates attention and expert matrices with base learning rate 0.0025, retaining the matrix-shape scaling of Section D. AdamW updates the remaining trainable parameters with base learning rate 7.5×10^{-5}, including token embeddings, the output head, normalization parameters, attention residual queries and gains, and the path gates of NESTOR. Muon uses momentum 0.95, Nesterov updates, and five Newton–Schulz iterations; AdamW uses $\beta = (0.9, 0.95)$ and $\epsilon = 10^{-8}$. The decoupled weight-decay coefficient is 0.1 for attention and expert matrices, token embeddings, the output head, and path gates. Normalization parameters, biases, and attention residual queries and gains receive no weight decay. We clip the global gradient norm to 1.0. The schedule starts with 47 steps of linear warmup, followed by cosine annealing for the remaining 430 steps. The cosine scheduler uses $T_{\max} = 477$ and $\eta_{\min} = 10^{-5}$, so the run ends before the scheduled minimum is reached.

Compute reference. At the 32k context length used for adaptation, the estimated forward FLOPs per token for NESTOR are approximately $0.84\times$ those of the 500M MoE transformer and $0.43\times$ those of the 1B dense transformer. These estimates count attention projections, causal attention, and MLPs, assume balanced token assignments across columns, and exclude routers, attention residuals, normalization, and the output head. Since all models receive the same token budget, the same ratios apply to these forward FLOP totals over each adaptation run. We report them as a reference for computational cost; the adaptation and forgetting results compare models trained on equal token budgets.

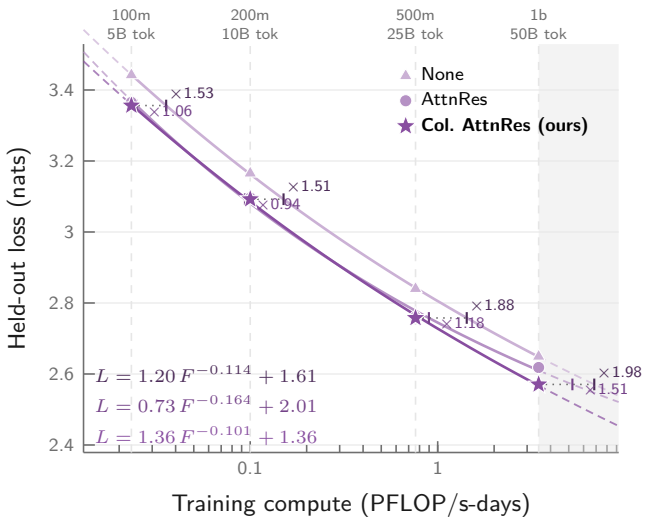


Figure 8: Ablation of the attention residual of NESTOR at 100M, 200M, 500M, and 1B, on the axes of Figure 2. We compare the column attention residual (Col. AttnRes), the merged attention residual (AttnRes), which is that of Kimi Team (2026) with the changes of Section B, and no attention residual. The rest of the model is unchanged. Each series is fitted by $L(F) = A\,F^{-\alpha} + L_\infty$, and the horizontal ticks mark the projected compute that each variant needs to match Col. AttnRes. Removing the attention residual increases the loss by 0.073–0.086 nats at every scale, which corresponds to 1.5–2× the compute. The two attention residuals have the same loss up to 500M and differ at 1B.

Evaluation. We evaluate the starting checkpoint and the final checkpoint at step 477 on identical held-out sequences. For PG-19, we use the first 32,768 tokens of each of the 77 test books that reach this length. For OpenWebMath, AlgebraicStack, and German, we use the first 200 non-overlapping 32,768-token sequences of each test stream, allowing sequences to contain multiple documents. To measure forgetting, we use the first 32,768 tokens of each of 200 DCLM-baseline validation documents that reach this length. This DCLM evaluation set is separate from the extension and replay data. We average next-token cross-entropy over all prediction targets, excluding the first token of each sequence. All models use the same evaluation sequences, without parameter updates during evaluation. Adaptation is the initial loss minus the final loss on the target corpus; forgetting is the final loss minus the initial loss on DCLM. The averages in Figure 3 and Table 6 give equal weight to each of the four target domains.

Adaptation with replay of the pretraining corpus. In the runs without replay, all training tokens come from the target corpus. We repeat these runs with a mixture containing approximately 75% target-domain tokens and 25% DCLM-baseline tokens, keeping the total budget and all other training settings unchanged. The DCLM sequences are taken from those already used during context extension. The target sequences are a subset of those used in the corresponding run without replay. We shuffle the combined sequences before training; the 25% share is approximate because the data are sampled in whole sequences. Figure 3 (faded marks) and Table 6 show the results. Averaged over the four domains, replay reduces forgetting by about 70% for each model, while reducing the adaptation gain by about 6%. With replay, NESTOR and the MoE transformer have mean gains of 0.687 and 0.687, respectively, and mean forgetting of 0.073 and 0.076. NESTOR forgets less on each of the four corpora. The 1B dense transformer has a mean gain of 0.696 and mean forgetting of 0.087. Table 6 lists the per-domain results and averages for both replay settings, including the values summarized in Section 4.1.

C.5 LONG CONTEXT EXTENSION SETTINGS

This subsection supplements the long context extension experiments in Section 4.2. We report results through 64k at the 100M, 200M, 500M, and 1B scales.

Table 6: **Loss before and after domain adaptation.** Target-corpus and DCLM held-out losses at 32k, before adaptation and after 0.5B training tokens. Gain is target loss before minus after; forgetting is DCLM loss after minus before, corresponding to Figure 3. Average gives equal weight to the four domains. Differences and averages are computed before rounding to three decimal places.

		Target-corpus loss			DCLM loss		
Domain	Model	Before	After	Gain ↑	Before	After	Forgetting ↓
No replay							
Books	500M NESTOR	2.732	2.548	0.184	2.779	2.996	0.217
	500M MoE	2.765	2.582	0.182	2.810	3.039	0.229
	1B dense	2.719	2.538	0.181	2.779	3.032	0.252
Math	500M NESTOR	2.417	2.084	0.332	2.779	2.885	0.106
	500M MoE	2.449	2.111	0.338	2.810	2.920	0.110
	1B dense	2.412	2.075	0.338	2.779	2.914	0.134
Code	500M NESTOR	1.789	1.084	0.705	2.779	3.073	0.294
	500M MoE	1.813	1.102	0.712	2.810	3.106	0.296
	1B dense	1.720	1.031	0.688	2.779	3.164	0.385
German	500M NESTOR	4.171	2.465	1.706	2.779	3.176	0.398
	500M MoE	4.224	2.521	1.704	2.810	3.221	0.410
	1B dense	4.164	2.431	1.733	2.779	3.273	0.494
Average	500M NESTOR	2.777	2.045	0.732	2.779	3.033	0.254
	500M MoE	2.813	2.079	0.734	2.810	3.072	0.261
	1B dense	2.754	2.019	0.735	2.779	3.096	0.316
25% *replay*							
Books	500M NESTOR	2.732	2.573	0.158	2.779	2.828	0.049
	500M MoE	2.765	2.607	0.158	2.810	2.863	0.053
	1B dense	2.719	2.560	0.159	2.779	2.837	0.058
Math	500M NESTOR	2.417	2.113	0.304	2.779	2.832	0.053
	500M MoE	2.449	2.140	0.309	2.810	2.867	0.057
	1B dense	2.412	2.101	0.311	2.779	2.842	0.063
Code	500M NESTOR	1.789	1.131	0.658	2.779	2.867	0.088
	500M MoE	1.813	1.146	0.668	2.810	2.901	0.091
	1B dense	1.720	1.074	0.646	2.779	2.885	0.106
German	500M NESTOR	4.171	2.546	1.626	2.779	2.879	0.101
	500M MoE	4.224	2.611	1.614	2.810	2.913	0.102
	1B dense	4.164	2.497	1.667	2.779	2.899	0.120
Average	500M NESTOR	2.777	2.091	0.687	2.779	2.851	0.073
	500M MoE	2.813	2.126	0.687	2.810	2.886	0.076
	1B dense	2.754	2.058	0.696	2.779	2.866	0.087

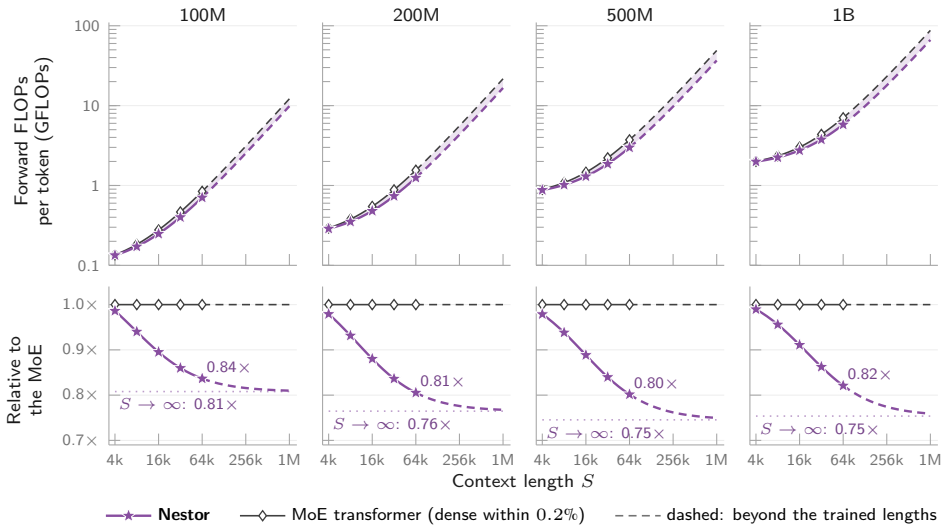


Figure 9: **Forward FLOPs per token against the context length.** Top: the forward FLOPs per token of NESTOR and of the baselines at all scales, with the accounting of Section C.1. The baseline curve uses the MoE transformer. The dense transformer has the same FLOPs at 100M and 500M; rounding the MoE expert widths introduces a difference of less than 0.2% at 200M and 1B over the plotted lengths. Bottom: the FLOPs per token of NESTOR relative to the MoE transformer. The markers are the context lengths of Figure 4, and the dashed curves continue the same formula to 1M tokens. The projections and MLPs of NESTOR cost 4–8% more than those of the baselines, and its attention costs 0.75–0.81× as much. The share of attention in the total grows with S, so the relative cost of NESTOR falls from 0.98–0.99× at 4k to 0.80–0.84× at 64k, and tends to the attention ratio as $S \to \infty$ (dotted).

Starting checkpoints. We extend the pretrained dense transformer, MoE transformer, and NESTOR of Section 3 at all four scales. These models were pretrained at context length 4k with rotary base 10^4; their 4k held-out losses are the first column of Table 8. Each target length is a separate run from the 4k checkpoint, and no run continues from a model extended to a shorter length. At 200M, NESTOR was pretrained with two seeds, and every setting is run from both checkpoints, with the data order of the extension run following the seed; the baselines, and all models at the other scales, have one run per length.

Data. For a target length S, the extension corpus consists of DCLM-baseline documents of at least S tokens, taken from a part of the corpus that was not used in pretraining. Documents are separated by an end-of-text token and concatenated, and the training sequences are consecutive S-token windows of this stream, so a sequence may cross a document boundary; causal attention is not reset at those boundaries. At a given length, all models train on the same sequences in the same order (data sampler seed 0). The held-out set at length S consists of the first S tokens of each of 200 documents of at least S tokens (100 at 64k), taken from a third part of the corpus that is disjoint from both the pretraining and the extension data. The losses at 4k are those of Section 3, on its held-out set.

Extension at equal tokens. Each run trains for 477 optimizer steps with 1,048,576 tokens per step (128, 64, 32, and 16 sequences per step at 8k, 16k, 32k, and 64k), which gives 500,170,752 tokens, rounded to 0.5B in the text. The rotary base is raised from 10^4 to 5×10^5, 10^6, 2×10^6, and 5×10^6 at the four lengths, in training and in evaluation. All parameters are trained, including the routers. The optimizer, the weight decay, and the gradient clipping are those of Section C.4: Muon with peak learning rate 0.0025 and AdamW with peak learning rate 7.5×10^{-5}, a quarter of the pretraining peaks, with fresh optimizer states, 47 steps of linear warmup (10%), and cosine annealing over the remaining steps.

Extension at equal FLOPs. Let $r(S)$ be the forward FLOPs per token of NESTOR relative to the MoE transformer at length S, with the accounting of Section C.1; its values are printed under the panels of Figure 4 and listed in Table 7, and $1/r(S)$ is the measured counterpart of the multiplier $R(S)$ of Section 4.2. The equal-FLOP runs of NESTOR train on $0.5\text{B}/r(S)$ tokens at the same tokens per step, that is for $T = \lceil 477/r(S) \rceil$ steps, with a warmup of 10% of T and the cosine schedule spread over the T steps, and with both peak learning rates multiplied by $r(S)$, so that the product of the number of steps and the peak learning rate is that of the equal-token run. At the unscaled peak learning rate, the same runs were worse by 0.002 to 0.014 nats at 500M and 1B, with the difference growing with S. In the setting with new documents, the run draws its sequences from the full extension corpus, so the additional $T - 477$ steps use documents that the equal-token run did not. In the setting with repeated documents, the corpus is restricted to the sequences consumed by the equal-token run, extended by $(T - 477)$ steps' worth of sequences drawn from that set without replacement, so that no sequence appears more than twice, and the run is one shuffled pass over this multiset; the two copies of a repeated sequence therefore fall at random positions of the run, at a mean distance of $T/3$ steps. The two settings have the same number of steps, tokens, and FLOPs, and differ only in the data.

Table 7: FLOP ratio $r(S)$ of NESTOR to the MoE transformer at each context length, and, in parentheses, the number of optimizer steps of the equal-FLOP runs, $\lceil 477/r(S) \rceil$. The equal-token runs take 477 steps.

Scale	8k	16k	32k	64k
100M	0.940 (508)	0.895 (533)	0.860 (555)	0.837 (570)
200M	0.933 (512)	0.881 (542)	0.837 (570)	0.806 (592)
500M	0.938 (509)	0.888 (537)	0.840 (568)	0.802 (595)
1B	0.956 (499)	0.911 (524)	0.863 (553)	0.821 (581)

Evaluation. Each run is evaluated at its final step on the held-out sequences of its length, with the rotary base of the run and in full precision. We average the next-token cross-entropy over all prediction targets of all sequences, excluding the first token of each sequence. The same sequences are used for every model and setting at a length. Figure 4 plots these losses minus the loss of the MoE transformer of the same scale at the same length; at 200M, the whiskers join the two seeds. Table 8 lists the losses.

Results. The loss of the dense transformer is higher than that of the MoE transformer by 0.13 to 0.19 nats at every length and scale. At equal tokens, NESTOR keeps its pretraining margin over the MoE transformer at every length: -0.028 to -0.030 nats at 100M, -0.012 to -0.015 at 200M, -0.031 to -0.035 at 500M, and -0.017 to -0.025 at 1B. At equal FLOPs the margin grows with the context length. With new documents it reaches -0.047, -0.037, -0.061, and -0.058 nats at 64k at the four scales; with repeated documents, -0.037, -0.023, -0.047, and -0.033. The repeated documents give losses within 0.006 nats of the new documents at 8k and 16k at every scale; at 32k and 64k, the new documents are better by 0.004 to 0.025 nats.

C.6 ROUTING STATISTICS AND ROUTING COUNTERFACTUALS

This section describes how the trained routers of NESTOR and of the MoE transformer use their choices, what each level of the routing encodes, and how much of the held-out loss rests on each level. The first two questions are answered by a census of the routing of the 500M models of Section 4.1, the third by counterfactual evaluations of the pretrained models of Section 3.

Routing census. For the 500M NESTOR and the 500M MoE transformer after context extension to 32k (the starting checkpoints of Section C.4), we record the selections of every router on the first 64 held-out sequences of 32,768 tokens of each of the five corpora of Section 4.1 (DCLM-baseline, books, mathematics, code, and German; 2.1M tokens per corpus), in evaluation mode. We repeat the census on the models adapted to each corpus, with and without frozen routers, on the same sequences, to count the assignments that the adaptation changes. Figure 10 shows, per router family and layer, the effective number of choices $\exp H(p)/E$ of the distribution p of the assignments over

Table 8: Held-out loss at each context length, before extension (4k) and after extension to 8k, 16k, 32k, and 64k, for the runs of Figure 4. The three settings of NESTOR start from the same 4k checkpoint; repeated and new refer to the documents of the equal-FLOP runs. At 200M, NESTOR is given for both seeds (seed 42 / seed 43); the baselines and the other scales have one seed.

Model	4k	8k	16k	32k	64k
100M					
Dense transformer	3.578	3.511	3.495	3.501	3.490
MoE transformer	3.386	3.338	3.330	3.330	3.325
NESTOR, equal tokens	3.356	3.311	3.302	3.302	3.296
NESTOR, equal FLOPs, repeated	–	3.309	3.298	3.297	3.287
NESTOR, equal FLOPs, new	–	3.307	3.295	3.293	3.277
200M					
Dense transformer	3.293	3.256	3.248	3.244	3.237
MoE transformer	3.108	3.096	3.095	3.084	3.080
NESTOR, equal tokens	3.092/3.088	3.081/3.078	3.084/3.083	3.070/3.068	3.066/3.063
NESTOR, equal FLOPs, repeated	–	3.079/3.076	3.075/3.073	3.061/3.058	3.056/3.051
NESTOR, equal FLOPs, new	–	3.077/3.073	3.074/3.071	3.056/3.054	3.043/3.042
500M					
Dense transformer	2.953	2.971	2.970	2.952	2.948
MoE transformer	2.793	2.827	2.842	2.810	2.805
NESTOR, equal tokens	2.758	2.792	2.811	2.779	2.773
NESTOR, equal FLOPs, repeated	–	2.789	2.802	2.766	2.757
NESTOR, equal FLOPs, new	–	2.787	2.800	2.758	2.744
1B					
Dense transformer	2.743	2.797	2.810	2.779	2.780
MoE transformer	2.592	2.633	2.683	2.634	2.632
NESTOR, equal tokens	2.571	2.616	2.661	2.612	2.607
NESTOR, equal FLOPs, repeated	–	2.616	2.655	2.603	2.599
NESTOR, equal FLOPs, new	–	2.610	2.650	2.591	2.574

the E choices, the share of choices that are rarely or never selected, and the mean per-token entropy of the router's softmax as a fraction of $\log E$.

Balance and decisiveness. On the pretraining corpus both models use their choices almost uniformly (effective choices of at least $0.97E$ in every router family and layer), with one exception: the first-layer router of the MoE transformer has collapsed, with two experts receiving about 70% of the tokens each and one expert never selected ($0.41E$). The shifted corpora concentrate the load in both models, most strongly on code and German, where a third to a half of the experts of some layers receive less than half of the uniform load; the columns concentrate less (15–21%). The column router is nearly indifferent per token: in the six routed blocks its softmax has entropy 0.95–0.997 of $\log C$, so the gating weights of the selected columns are within a few percent of uniform, and its scores have a root mean square of 0.15–0.41, below the mean magnitude 0.44 of the balancing bias. The selection is nevertheless input dependent and reproducible: routing with the training-time noise agrees with the noise-free routing on 85–98% of the tokens per block, and only 1–16% of the tokens select what the bias alone would select. The expert routers are decisive, with per-token entropies of 0.4–0.9 of $\log E$.

What each level encodes. Table 9 gives two views of the same picture. The column assignment carries almost no information about the corpus (0.05 bits of a possible 2.32), while the choice of expert inside a column carries as much as the choice of expert in the MoE transformer (0.28 and 0.25 bits). The preference of an expert of a column for a corpus factorizes into the preference of the column and the preference within the column, and the second factor is three times the first in mean absolute $\log_2$ enrichment (0.84 against 0.29). The column assignment is also the most stable decision under domain adaptation: with frozen routers, 4.7% of the column assignments on the target corpus change, against 26% of the assignments to the experts of the columns and 22% of the assignments in the MoE transformer. The coarse level of the routing is thus a partition that the corpora share, and the fine level inside each column is where the corpus-specific choices are made.

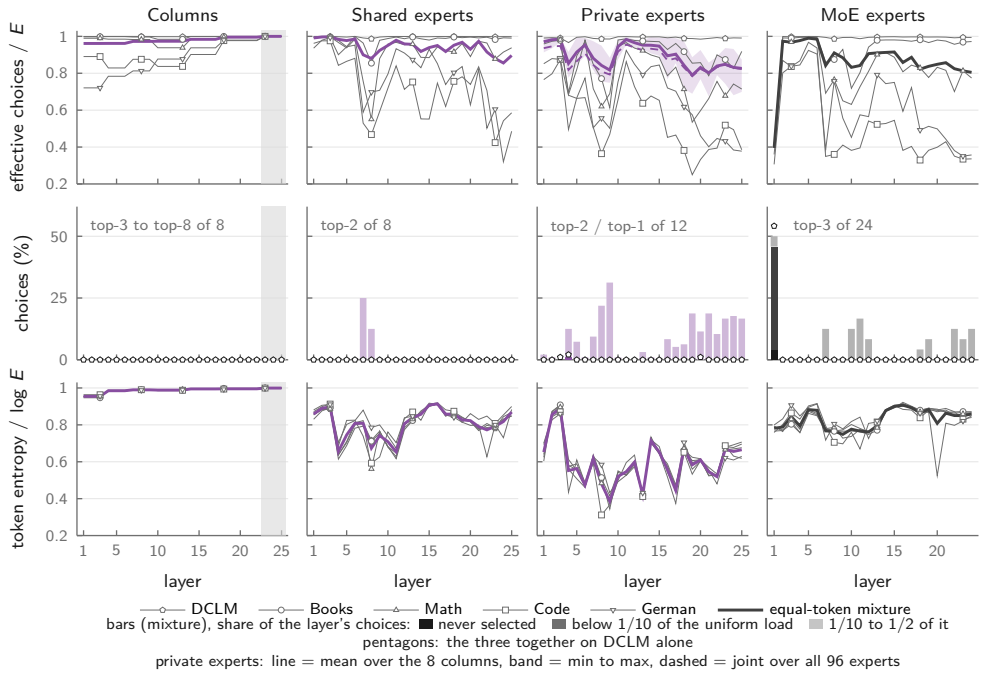


Figure 10: **Routing statistics of the trained 500M models** across depth, one column of panels per router family. **Top:** effective number of choices of the aggregate assignment distribution on each held-out corpus (thin lines) and on their equal-token mixture (heavy line). **Middle:** share of the layer's choices that are never selected or selected well below the uniform load. **Bottom:** mean per-token entropy of the router's softmax as a fraction of $\log E$, which measures how decisive each routing decision is rather than how balanced the load is. The last block of NESTOR selects every column (shaded).

Routing decision	Corpus information (bits; range over layers)	Assignments changed by adaptation routers frozen	all parameters trained
NESTOR: which column	0.05 (0.01–0.08)	4.7%	13.7%
NESTOR: which shared expert	0.13 (0.01–0.36)	15.6%	24.4%
NESTOR: which expert of the column	0.28 (0.04–0.54)	25.8%	32.6%
MoE transformer: which expert	0.25 (0.02–0.48)	21.7%	30.7%

Table 9: What each routing level encodes, for the 500M models of Figure 10. *Corpus information:* mutual information, in bits, between the corpus (five corpora with equal token counts, entropy 2.32 bits) and the selected choice, with each selection counted as one draw; mean over the routed layers, range in parentheses. For the experts of a column, the information is conditional on the column, so that the column and the expert rows add up to the information of the pair. *Assignments changed:* share of the tokens of the adaptation corpus whose selected set differs from the set before adaptation, after 0.5B tokens of adaptation without replay, averaged over the four corpora of Section 4.1; with frozen routers, every change comes from the routers' inputs. Models trained on the pretraining corpus alone already change 2 / 8 / 15 / 11% of the assignments in the four rows.

Routing counterfactuals. To measure how much of the held-out loss rests on each routing level, we re-evaluate the pretrained models of Section 3 on the held-out set of Figure 2, at context length 4k, with the decisions of one router family replaced at evaluation time and without any training. In the *shuffled* condition, every token takes the complete output (selected set and gating weights) of a uniformly random other token of the same sequence, so that the number of tokens in every column and expert, and hence the attention cost, are unchanged, and only the pairing of token and decision is destroyed. In the *random* condition, every token selects a uniformly random subset of K choices, weighted by the router's own scores at those choices. A replaced column routing changes the tokens that the expert routers of each column see; these routers then route the new tokens with their learned parameters, so the column condition isolates the column decision. With the learned routing, the

evaluation reproduces the losses of Table 3 to four decimals. Table 10 reports the increase in held-out loss for NESTOR and the MoE transformer at 200M, 500M, and 1B, and for the column-routed model with the shared expert bank of the MoE transformer (row 7 of Table 4).

	NESTOR			MoE transformer		Column-routed,
Replaced routing	200M	500M	1B	200M	500M / 1B	shared bank
columns, shuffled	0.40 / 0.48	0.74	0.62	–	–	0.50
columns, random	0.39 / 0.46	0.69	0.79	–	–	0.53
shared experts, shuffled	2.59 / 2.53	2.11	2.13	–	–	–
experts of the columns, shuffled	0.99 / 1.08	1.36	1.31	–	–	–
both expert levels, shuffled	3.96 / 3.77	3.82	3.61	–	–	–
experts, shuffled	–	–	–	4.01 / 4.23	4.47 / 4.51	4.08
experts, random	–	–	–	3.88 / 3.99	4.16 / 4.57	4.16
all routing, shuffled	4.14 / 3.94	4.09	4.23	–	–	4.39

Table 10: Increase in held-out loss (nats) of the pretrained models when one routing level is replaced at evaluation time. The columns of the table are the models of Table 3 and the column-routed model with the shared bank of the MoE transformer (row 7 of Table 4, at 200M); the 200M models are given for both seeds, as seed 42 / seed 43. For reference, the loss of the MoE transformer exceeds the loss of NESTOR by 0.016 / 0.025 at 200M, 0.035 at 500M, and 0.022 at 1B.

	Block (active columns of 8)								
Model	1 (3)	2 (4)	3 (5)	4 (6)	5 (6)	6 (7)	7 (8)	Sum	All blocks
NESTOR, seed 42	0.003	0.124	0.049	0.045	0.064	0.027	0.001	0.312	0.398
NESTOR, seed 43	0.002	0.124	0.057	0.060	0.070	0.027	0.002	0.341	0.479
Column-routed, shared bank	0.054	0.046	0.058	0.078	0.039	0.030	0.009	0.314	0.500

Table 11: Increase in held-out loss (nats) of the 200M column-routed models when the column routing of one block is shuffled and the other blocks keep their learned routing, next to the sum over blocks and the cost of shuffling all blocks at once (the first row of Table 10). The last block selects every column, so shuffling changes only its gating weights.

Reading. Three observations follow. First, the column assignment is load bearing at every scale: replacing it costs 0.40–0.48 nats at 200M, 0.74 at 500M, and 0.62 at 1B, which is 18–28 times the difference between the losses of the MoE transformer and of NESTOR at the same scale, and the shuffled and random conditions agree. The trained model relies on which columns a token visits, although its column gates are nearly uniform, so the columns are not a random partition of the tokens. Second, the column assignment is equally load bearing in the column-routed model whose experts are shared by all columns (0.50 nats). The value of the column decision is therefore a property of column-routed attention rather than of the nesting, and the attribution of the improvement of NESTOR to the nesting rests on the trained ablations of Section C.3, not on these counterfactuals. Third, the two expert levels of NESTOR together are about as load bearing as the single expert level of the MoE transformer at 200M (3.8–4.0 against 4.0–4.2 nats), and the two drift apart with scale (3.6 against 4.5 at 1B) while the column level gains, so a growing share of the value of the routing sits in the columns; inside NESTOR, the shared experts carry most of the value of the expert level. Table 11 resolves the column decision by block. In NESTOR, the assignment of the first block, which is computed before any attention layer, carries nothing, the second block carries the most, and the value decays towards the last block; the sum over blocks is below the cost of replacing all blocks at once, so the routers of later blocks depend on features computed under the learned routing of the earlier blocks. In the column-routed model with the shared bank, the assignment of the first block does matter (0.054 nats).

D OPTIMIZATION AND INITIALIZATION

Optimization. The matrices of the attention operators and of the experts are trained by Muon (Jordan et al., 2024) with the base learning rate $\eta_{\mathrm{M}} = 0.01$. The update of a matrix in $\mathbb{R}^{D_{\mathrm{out}} \times D_{\mathrm{in}}}$

is multiplied by $\sqrt{D_{\text{out}}/D_{\text{in}}}$, so that its spectral norm scales as in the spectral condition of Yang et al. (2023). All other parameters are trained by AdamW (Loshchilov & Hutter, 2019) with the base learning rate $\eta_{\text{A}} = 3 \times 10^{-4}$. These are the token embeddings, the output head, the expert routers of the MoE baseline and of NESTOR (including the routers of the shared experts and of the experts of each column, and the gate (22)), the queries and gains of the attention residual, and the normalization gains. The only exception is the last layer of the column router, which has the learning rate $\eta_{\text{A}}/(D_{\text{in}}\sqrt{M})$, where $D_{\text{in}} = 128$ is the hidden width of the router and M is the number of layers in its block: this router is evaluated once and used by all M layers, so its gradient is a sum of M terms. The Muon base learning rate was selected at 200M and reused at the other scales. The AdamW base learning rate was fixed, without a separate sweep.

Setting	Value
Learning rate, Muon	0.01
Learning rate, AdamW	3×10^{-4}
Muon momentum	0.95, Nesterov
Newton–Schulz iterations	5
AdamW (β_1, β_2), ϵ	$(0.9, 0.95)$, 10^{-8}
Warmup	linear, 8% of the steps
Decay	cosine, minimum 10^{-5}
Weight decay	0.1 on matrices, 0 on vectors
Gradient clipping	1.0
Dropout	0
Sequence length	4,096
Precision	bfloat16 autocast
Seed	42 (43 for the second seed)

Table 12: Pretraining hyperparameters, shared by all architectures and scales.

Initialization. The three architectures use the same initialization. The query, key, and value projections, the input and gate matrices of the experts, the expert routers, and the gate (22) are initialized with standard deviation $0.1/\sqrt{8}$. The output projections of the attention operators and the output matrices of the experts use this standard deviation divided by $\sqrt{2L}$, where L is the number of layers of the model. The token embeddings and the output head use standard deviation 0.1; the output of the embedding is multiplied by 12, and the output logits are divided by 8. The last layer of the column router is initialized with standard deviation $1/D_{\text{in}}$, so that the column routing starts near uniform, and the queries of the attention residual are initialized at zero.

E COMPUTE MULTIPLIERS UNDER SCALING LAWS

This section complements Sections 3 and 4.2 with an accounting argument for the full architectures, which Theorem 4(c) carries out within Model 1 for a fixed model size. We take as a hypothesis that the losses of the full architectures follow a scaling law with common exponents and coefficients that depend on the architecture, and show that the compute multiplier between the two architectures is then the same at every loss, and that the attention FLOPs that NESTOR saves make it grow with the context length.

E.1 THE COMPUTE MULTIPLIER

Persistence with scale. Theorem 4 holds for fixed C, E, r, D, and S. There, the excess risks of the two architectures have the same exponent in T, the different coefficients $\sigma^2 d_s$, and the different costs F_s per sample, and (33) is the compute multiplier. We take as a hypothesis that the losses of the full architectures have the same form when the model size also grows, which Model 1 does not capture: a scaling law of the form of Hoffmann et al. (2022), with common exponents and coefficients that depend on the architecture. Under this hypothesis, the compute multiplier is the same at every scale. Let N be the number of active parameters, T the number of training tokens, and F the training FLOPs. Our compute accounting includes the attention FLOPs (Section C.1). At a fixed context

length,

$$F = k_0(1+\nu_s)NT, \tag{24}$$

where $k_0 NT$ is the FLOPs of the projections and the experts, and ν_s is the ratio of the attention FLOPs of architecture s to these FLOPs. For equal shapes and balanced routing, (19) gives $\nu_{\mathrm{N}} = \rho\,\nu_{\mathrm{F}}$ for NESTOR (N) and the MoE transformer (F), where $\rho \in (0,1)$ is the average of K_c/C over the layers of NESTOR.

Proposition 3: *For each architecture s, let $L_s(N,T) = L_\infty + a_s N^{-\alpha} + b_s T^{-\beta}$ and $F = k_0(1+\nu_s)NT$, where $\alpha, \beta, a_s, b_s, k_0 > 0$ and $\nu_s \geq 0$ are constants. Let $p \doteq \alpha\beta/(\alpha+\beta)$ and $q_s \doteq (1+\nu_s)\, a_s^{1/\alpha} b_s^{1/\beta}$. Then*

$$\min_{N,T:\ k_0(1+\nu_s)NT=F} L_s(N,T) = L_\infty + H\,(k_0 q_s/F)^p, \qquad H \doteq (\beta/\alpha)^{\alpha/(\alpha+\beta)} + (\alpha/\beta)^{\beta/(\alpha+\beta)}. \tag{25}$$

For every loss $L > L_\infty$, the compute that the MoE transformer needs to reach L, divided by the compute that NESTOR needs, is

$$\frac{q_{\mathrm{F}}}{q_{\mathrm{N}}} = \frac{1+\nu_{\mathrm{F}}}{1+\nu_{\mathrm{N}}} \cdot \left(\frac{a_{\mathrm{F}}}{a_{\mathrm{N}}}\right)^{1/\alpha} \left(\frac{b_{\mathrm{F}}}{b_{\mathrm{N}}}\right)^{1/\beta}. \tag{26}$$

The multiplier (26) has two factors. The first is the saving in attention FLOPs, and the second is the ratio of the coefficients of the law. The ratio $d_{\mathrm{F}}/d_{\mathrm{N}}$ of Theorem 4 is an example of the latter at a fixed size, and the second factor of (33) is the former. Neither factor depends on L, so under the hypotheses of the proposition the multiplier stays constant while the difference of the losses goes to zero. In Figure 2, the fitted exponents of NESTOR and of the MoE transformer are equal to three digits, and the measured multiplier 1.13–$1.29\times$ has no trend over our four scales. The fits have separate offsets, which the proposition does not allow; we discuss their effect at the end of Section E.3. Section E.3 also extends the proposition to a ratio ν_s that changes with the model size, and to the training path with a fixed ratio T/N that our experiments use.

E.2 LONG CONTEXTS

The attention FLOPs of one token are proportional to the context length S, and the other FLOPs do not depend on S. Let S_0 be the context length at which the two are equal for the MoE transformer, so that $\nu_{\mathrm{F}} = S/S_0$ and $\nu_{\mathrm{N}} = \rho S/S_0$. The first factor of (26) is then

$$R(S) \doteq \frac{1+\nu_{\mathrm{F}}}{1+\nu_{\mathrm{N}}} = \frac{S_0+S}{S_0+\rho S}. \tag{27}$$

In the context extension experiments the models are fixed and S grows. For equal shapes and balanced routing, at equal FLOPs NESTOR trains on $R(S)$ times as many tokens as the MoE transformer. For our models the measured counterpart is $1/r(S)$ (Table 7), which also accounts for the slightly different shapes of the two architectures (Figure 9). The multiplier $R(S)$ increases with S and converges to $1/\rho$, as we show below. This explains why the margin at equal FLOPs grows with the context length in Figure 4. At equal tokens the margin is preserved, which agrees with the assumption of Section 5 that the restricted context is sufficient for the prediction. In pretraining, S is fixed and the width D grows. The length S_0 is proportional to D, because the attention FLOPs grow as SD and the other FLOPs as D^2. Then $R(S) \to 1$, and the pretraining advantage at large width must come from the second factor of (26).

We now derive S_0 and the properties of $R(S)$. In one layer, let a token cost ϕ FLOPs in the projections and the experts of either architecture and ωS FLOPs in the attention of the MoE transformer. By (19), it costs $\omega(K_c/C)S$ FLOPs in the attention of NESTOR. We sum over the layers and obtain $\nu_{\mathrm{F}} = \omega S/\phi = S/S_0$ and $\nu_{\mathrm{N}} = \rho S/S_0$ with $S_0 \doteq \phi/\omega$. The backward pass multiplies all costs by the same factor. For a layer of width D, ϕ is proportional to D^2 and ω to D, so S_0 is proportional to D. The accounting of Section C.1 gives $S_0 = 16D$ for our models. The ratio $R(S)$ of (27) has the derivative $R'(S) = S_0(1-\rho)/(S_0+\rho S)^2 > 0$ and the limit $1/\rho$ as $S \to \infty$.

To convert the saving to a loss, we assume a learning curve for the continued training of NESTOR at the context length S,

$$L_{\mathrm{N}}(T;S) = \ell(S) + b(S)(T+\tau_S)^{-\beta}, \qquad \tau_S \geq 0, \tag{28}$$

where τ_S accounts for the pretrained initialization. Let $\Delta_T(S)$ be the loss of the MoE transformer minus the loss of NESTOR when both train on T tokens. When NESTOR trains on $R(S)T$ tokens at equal FLOPs, the difference is

$$\Delta_F(S) = \Delta_T(S) + b(S)\big[(T+\tau_S)^{-\beta} - (R(S)T+\tau_S)^{-\beta}\big]. \tag{29}$$

The second term is positive and increasing in $R(S)$. Therefore $\Delta_F(S) - \Delta_T(S)$ grows with S when $b(S)$ and τ_S change little with S, which agrees with Figure 4.

E.3 PROOF OF PROPOSITION 3 AND EXTENSIONS

The law has the form of Hoffmann et al. (2022). Clark et al. (2022) study routed models, where the exponents can depend on the number of experts. At the fixed sparsity of our experiments, the relation between the stored and the active parameters is absorbed in a_s.

Proof of Proposition 3. Let $\kappa_s \doteq k_0(1+\nu_s)$, and substitute $T = F/(\kappa_s N)$. As a function of $\log N$, the excess loss $a_s N^{-\alpha} + b_s(\kappa_s N/F)^{\beta}$ is a sum of a decreasing and an increasing exponential. It is strictly convex and diverges at both ends, so it has a unique minimizer, at which $\alpha a_s N^{-\alpha} = \beta b_s T^{-\beta}$. Let $u \doteq a_s N^{-\alpha}$, so that $b_s T^{-\beta} = \alpha u/\beta$. Then

$$\frac{F}{\kappa_s} = NT = \Big(\frac{a_s}{u}\Big)^{1/\alpha}\Big(\frac{\beta b_s}{\alpha u}\Big)^{1/\beta} = a_s^{1/\alpha} b_s^{1/\beta}\Big(\frac{\beta}{\alpha}\Big)^{1/\beta} u^{-1/p}. \tag{30}$$

Since $\kappa_s a_s^{1/\alpha} b_s^{1/\beta} = k_0 q_s$ and $p/\beta = \alpha/(\alpha+\beta)$, we obtain $u = (\beta/\alpha)^{\alpha/(\alpha+\beta)}(k_0 q_s/F)^p$ and $\alpha u/\beta = (\alpha/\beta)^{\beta/(\alpha+\beta)}(k_0 q_s/F)^p$. Their sum is (25). Solving (25) for F at a loss $L > L_\infty$ gives $F = k_0 q_s (H/(L-L_\infty))^{1/p}$, which is proportional to q_s. This gives (26). □

Attention ratio that depends on the model size. At a fixed context length, the ratio ν_s decreases as the width grows (Section E.2), so it is a function $\nu_s(N)$. The frontier then has no closed form, but the comparison of the architectures remains valid. Suppose that $a_{\mathrm{N}} \le a_{\mathrm{F}}$, $b_{\mathrm{N}} \le b_{\mathrm{F}}$, and $\nu_{\mathrm{N}}(N) \le \nu_{\mathrm{F}}(N)$ for all N. Let (N,T) be any allocation of the MoE transformer with compute F. With the same compute, NESTOR can train a model of size N on $T' = T(1+\nu_{\mathrm{F}}(N))/(1+\nu_{\mathrm{N}}(N)) \ge T$ tokens, and $L_{\mathrm{N}}(N,T') \le L_{\mathrm{F}}(N,T)$. Therefore NESTOR has the lower frontier at every F.

Fixed ratio of tokens to parameters. Our models are trained with $T = \varrho N$ for a constant ϱ, which is not the optimal allocation. On this path, $L_s - L_\infty = a_s N^{-\alpha} + b_s \varrho^{-\beta} N^{-\beta}$ and $F = k_0(1+\nu_s(N))\varrho N^2$. Let $\alpha = \beta$ and $k_s \doteq a_s + b_s \varrho^{-\alpha}$. Architecture s reaches the loss L at the size $N_s = (k_s/(L-L_\infty))^{1/\alpha}$, and the ratio of the compute of the two architectures at this loss is

$$\frac{F_{\mathrm{F}}}{F_{\mathrm{N}}} = \frac{1+\nu_{\mathrm{F}}(N_{\mathrm{F}})}{1+\nu_{\mathrm{N}}(N_{\mathrm{N}})} \cdot \Big(\frac{k_{\mathrm{F}}}{k_{\mathrm{N}}}\Big)^{2/\alpha}. \tag{31}$$

This has the same two factors as (26). The second factor is constant. The first factor is constant when ν_s is constant, and then $L_s - L_\infty = k_s(k_0(1+\nu_s)\varrho/F)^{\alpha/2}$ is one power law in F, which is the form fitted in Figure 2. When the width grows at a fixed context length, the first factor converges to 1, and the multiplier converges to $(k_{\mathrm{F}}/k_{\mathrm{N}})^{2/\alpha}$. If $\alpha \ne \beta$, the two terms of $L_s - L_\infty$ change their relative size with N, and the multiplier depends on L. In this case $a_{\mathrm{N}} \le a_{\mathrm{F}}$, $b_{\mathrm{N}} \le b_{\mathrm{F}}$, and $\nu_{\mathrm{N}} \le \nu_{\mathrm{F}}$ are still sufficient for NESTOR to have the lower loss at every F when the compute $(1+\nu_s(N))N^2$ increases with N: with the same compute, NESTOR then trains a model that is at least as large on at least as many tokens. The fits in Figure 2 have a separate offset for each architecture, and the offset of NESTOR is lower. A lower limit L_∞ for NESTOR makes the multiplier grow with the scale.

F THEORETICAL ANALYSIS

In this section we give the formal versions of the results stated informally in Section 5, together with further discussion. The setup is that of Section 5: the data are generated by Model 1, the nested and flat predictors (5) are fit by least squares, and a fit is measured by its excess risk, both as in (6). We state the results on sample and compute efficiency in Section F.1 and on forgetting in Section F.2, prove them in Sections F.3 and F.4, and compare the nested predictor with other flat predictors in Section F.5.

F.1 Sample and Compute Efficiency

We write $\boldsymbol{e}_c \in \mathbb{R}^C$ for the c-th standard basis vector. The *narrow flat predictor* is the flat predictor in (5) whose attention score is computed from the raw tokens $\boldsymbol{x}_i = (\boldsymbol{e}_{c_i}, \boldsymbol{z}_i) \in \mathbb{R}^{C+D}$ rather than from the product features, so that its score is $\boldsymbol{x}_S^\top \boldsymbol{M} \boldsymbol{x}_i$ for a matrix $\boldsymbol{M} \in \mathbb{R}^{(C+D)\times(C+D)}$ that is shared by all pairs of labels, and its values are $\boldsymbol{z}_i$ as in (5).

Theorem 4: *In Model 1, let $C \geq 2$, and let $\hat{\boldsymbol{f}}^{\mathrm{N}}$ and $\hat{\boldsymbol{f}}^{\mathrm{F}}$ be least squares fits of the nested and flat predictors (5) on T samples.*

(a) Both predictors can represent the regression function $\boldsymbol{f}^\star$: the nested predictor with $\boldsymbol{W}_c = \boldsymbol{W}_c^\star$, $\boldsymbol{A}_c = \boldsymbol{A}_c^\star$, and $\boldsymbol{B}_{ce} = \boldsymbol{B}_{ce}^\star$, and the flat predictor with $\boldsymbol{W}_{cc} = \boldsymbol{W}_c^\star$, $\boldsymbol{W}_{cc'} = \boldsymbol{0}$ for $c' \neq c$, $\boldsymbol{V} = \boldsymbol{I}$, $\boldsymbol{W}_{ce}^{\mathrm{in}} = \boldsymbol{A}_c^\star$, and $\boldsymbol{W}_{ce}^{\mathrm{out}} = \boldsymbol{B}_{ce}^\star$. The narrow flat predictor cannot represent $\boldsymbol{f}^\star$ for any value of its weights.

(b) As $T \to \infty$, we have $T\,\mathcal{E}(\hat{\boldsymbol{f}}^{\mathrm{N}}) \rightsquigarrow \sigma^2 \chi^2_{d_{\mathrm{N}}}$ and $T\,\mathcal{E}(\hat{\boldsymbol{f}}^{\mathrm{F}}) \rightsquigarrow \sigma^2 \chi^2_{d_{\mathrm{F}}}$ in distribution, where

$$d_{\mathrm{N}} = \underbrace{C \cdot D^2}_{\text{patterns}} + \underbrace{C \cdot r(D-r)}_{\text{row spaces}} + \underbrace{CE \cdot rD}_{\text{maps}} - C, \quad d_{\mathrm{F}} = \underbrace{C^2 \cdot D^2}_{\text{patterns}} + \underbrace{CE \cdot r(D-r)}_{\text{row spaces}} + \underbrace{CE \cdot rD}_{\text{maps}} - C. \tag{32}$$

The limits have the means $\sigma^2 d_{\mathrm{N}}$ and $\sigma^2 d_{\mathrm{F}}$, and $1 < d_{\mathrm{F}}/d_{\mathrm{N}} \leq C$.

(c) Let $F_0, F_1 > 0$, and charge a sample $F_0 + F_1 k$ FLOPs when the predictor scores k context tokens, so that the expected FLOPs per sample are $F_{\mathrm{N}} \doteq F_0 + F_1(S-1)/C$ and $F_{\mathrm{F}} \doteq F_0 + F_1(S-1)$. If predictor $s \in \{\mathrm{N}, \mathrm{F}\}$ is trained on $T = F/F_s$ samples, then $F\,\mathcal{E}(\hat{\boldsymbol{f}}^s) \rightsquigarrow \sigma^2 F_s \chi^2_{d_s}$, whose mean is $\sigma^2 d_s F_s$. The two means are equal when the flat predictor uses

$$\kappa(S) \doteq \frac{d_{\mathrm{F}}}{d_{\mathrm{N}}} \cdot \frac{S_0 + S - 1}{S_0 + (S-1)/C}, \qquad S_0 \doteq F_0/F_1, \tag{33}$$

times the FLOPs of the nested predictor, and κ increases from $d_{\mathrm{F}}/d_{\mathrm{N}}$ at $S = 1$ to $C \cdot d_{\mathrm{F}}/d_{\mathrm{N}}$ as $S \to \infty$.

Interpretation of part (a). The two predictors can represent the same target, because the experts of the flat predictor can include the value maps of the columns, and the blocks of its pattern can implement the routing. The second requires that the score of the shared attention depend on the labels of both tokens, which is impossible for a bilinear score of the raw tokens: in the narrow flat predictor, the term of the score that is bilinear in $(\boldsymbol{z}_S, \boldsymbol{z}_i)$ does not depend on the labels, so it cannot vanish on the tokens of the other labels and equal $\boldsymbol{z}_S^\top \boldsymbol{W}_c^\star \boldsymbol{z}_i$ on the relevant tokens. The product features give the shared attention a residual stream that is C times wider, and Remark 8 gives the same construction for the full architectures. Part (b) quantifies the statistical cost of this width.

Interpretation of part (b). The predictors differ in what they must estimate from the data. The number d is the dimension of the set of functions that a predictor can represent near the target, which is the number of directions in weight space that change the prediction. It is smaller than the number of stored weights, because $\boldsymbol{V}$ is redundant, an invertible change of the r hidden units does not change the prediction, and multiplying the pattern of a label by a constant and its maps by the inverse does not change the prediction either, which accounts for the term $-C$ in both counts. Each dimension adds σ^2 to the mean of the limit of $T\,\mathcal{E}$, as in the classical asymptotic theory of least squares over a smooth model (van der Vaart, 1998; Geyer, 1994).

The three terms of (32) are the patterns, the row spaces, and the maps; the last two are shown in Figure 5b,c. The nested predictor estimates one pattern $\boldsymbol{W}_c \in \mathbb{R}^{D\times D}$ per column from the tokens routed to the column. The flat predictor estimates one block $\boldsymbol{W}_{cc'}$ per pair of labels, i.e., C^2 blocks, of which the $C(C-1)$ blocks with $c' \neq c$ are zero at the target: the flat predictor must estimate from data that these blocks are zero, whereas the routing of Nestor removes the tokens of the other labels before the pattern is applied. A matrix $\boldsymbol{G}$ of rank r is determined by its row space and by a linear map from the row space to $\mathbb{R}^D$. The row space is an r-dimensional subspace of $\mathbb{R}^D$ and has $r(D-r)$ free parameters, and the map has rD. Both predictors estimate a separate map for each of the CE experts. In the flat predictor each expert has its own $\boldsymbol{W}_{ce}^{\mathrm{in}}$, so the row space of each expert is estimated from the samples of that expert only. In the nested predictor all experts of a column have

the row space of $\boldsymbol{A}_c$, which is estimated from the samples of all E experts of the column. Altogether,

$$d_{\mathrm{F}} - d_{\mathrm{N}} = C(C-1) \cdot D^2 + C(E-1) \cdot r(D-r), \tag{34}$$

where the first term is due to the routed attention and the second to the nesting of the experts in the columns. If the flat predictor is fitted on kT samples, then $T\,\mathcal{E}(\hat{\boldsymbol{f}}^{\mathrm{F}}) \rightsquigarrow (\sigma^2/k)\,\chi^2_{d_{\mathrm{F}}}$. This limit has the same mean as the limit $\sigma^2\chi^2_{d_{\mathrm{N}}}$ of the nested predictor on T samples when $k = d_{\mathrm{F}}/d_{\mathrm{N}}$. We use this ratio of the limiting means to summarize the number of samples that the two predictors need. It equals $(C(C+E)D^2-C)/(C(1+E)D^2-C)$ at $r = D$, where a row space has no free parameters, and it tends to C as $D \to \infty$ for fixed r and E. Dropping the terms of lower order in D/r and in E gives the forms (7): $d_{\mathrm{N}} \approx CD(D+Er)$, $d_{\mathrm{F}} \approx CD(CD+2Er)$, and the ratio $(C+2x)/(1+x)$ with $x \doteq Er/D$, which decreases in x from C to 2 when $C \geq 2$.

The dimensions in (32) do not depend on the probabilities of the labels or on the context length S. An expert with probability π_{ce} receives about $\pi_{ce}T$ samples, so the error of its weights is larger by the factor $1/\pi_{ce}$. It is also used with probability π_{ce}, and the two factors cancel in the excess risk (6). Every module therefore adds σ^2 to the mean of the limit for each of its dimensions, however few samples it receives. The count d_{N} includes all C patterns and value maps and all CE experts, so it accounts for the cost of specialization.

Nesting needs both of its parts, which agrees with Table 4, where neither part helps alone. With $C = 1$, the nested predictor is a flat predictor, and the narrow flat predictor represents the target. With column-routed attention and experts $\boldsymbol{W}_e$ shared by all columns, the predictor is $\boldsymbol{W}_e\boldsymbol{A}_c \sum_{i\in\mathcal{I}}(\boldsymbol{z}_S^\top \boldsymbol{W}_c \boldsymbol{z}_i)\boldsymbol{z}_i$. The column space of $\boldsymbol{W}_e\boldsymbol{A}_c$ is contained in that of $\boldsymbol{W}_e$, which has dimension at most r, so this predictor represents the target only if, for each e, the maps $\boldsymbol{B}^\star_{ce}$ have the same column space for all c. Nesting also helps only when the coarse variable changes the computation. If every token were relevant to every query and neither the patterns nor the row spaces of the $\boldsymbol{A}^\star_c$ depended on c, then a shared attention with a label-free pattern and a value map of width r, followed by the experts, would represent the target with one pattern and one row space, whereas the nested predictor would estimate a separate copy of both for each column.

Interpretation of part (c). The excess risks of the two predictors decay at the same rate in the number of samples, and they differ in the coefficient $\sigma^2 d_s$ and in the cost F_s of a sample. The multiplier (33) is the product of the two ratios. The first factor, $d_{\mathrm{F}}/d_{\mathrm{N}}$, is the cost to the flat predictor of estimating its C^2 pattern blocks and CE row spaces. The second factor, $(S_0+S-1)/(S_0+(S-1)/C)$, is the cost of scoring all $S-1$ context tokens when only $(S-1)/C$ of them are relevant; it is the multiplier $R(S-1)$ of (27) with the fraction $\rho = 1/C$ of active columns, and $S_0 = F_0/F_1$ is the crossover length of Section 4.2, at which the cost of scoring the context equals the other cost of a sample. Section E derives the same multiplier for the full architectures from a scaling-law hypothesis. This is the mechanism behind Figure 4: at equal tokens the two predictors differ by the first factor alone, and at equal FLOPs the second factor grows with the context length. At a fixed context length, S_0 grows with the cost F_0 of the maps, so the second factor tends to one as the width grows, and the advantage at large widths in Figure 2 is the first factor.

F.2 Forgetting

We formalize the continual training of Section 5. The new domain Q is a distribution over $(e, \boldsymbol{X}, \boldsymbol{y})$: the fine label $e \in [E]$ is drawn with probability $\pi^Q_e > 0$, the context $\boldsymbol{X}$ is drawn as in Model 1 with the query label c_Q, the summary $\boldsymbol{m} = \sum_{i\in\mathcal{I}}(\boldsymbol{z}_S^\top \boldsymbol{W}^\star_{c_Q} \boldsymbol{z}_i)\boldsymbol{z}_i$ is computed with the pattern of the label c_Q, and $\boldsymbol{y} = \boldsymbol{G}'_e\boldsymbol{m} + \boldsymbol{\varepsilon}$ for fixed matrices $\boldsymbol{G}'_e \in \mathbb{R}^{D\times D}$ and noise $\boldsymbol{\varepsilon}$ as in Model 1. Let $\boldsymbol{\Sigma}^Q_e \doteq \mathbb{E}_Q[\boldsymbol{m}\boldsymbol{m}^\top \mid e]$, and let $\boldsymbol{\Sigma}_{ce} \doteq \mathbb{E}[\boldsymbol{m}\boldsymbol{m}^\top \mid c, e]$ under Model 1; both are positive definite. The nested predictor starts at the weights of Theorem 4(a), and so does the flat predictor. Each predictor follows the gradient flow $\dot{\boldsymbol{\theta}} = -\boldsymbol{\nabla}_{\boldsymbol{\theta}} L_Q(\boldsymbol{\theta})$ of the population loss

$$L_Q(\boldsymbol{\theta}) \doteq \frac{1}{2} \sum_{e\in[E]} \pi^Q_e \mathop{\mathbb{E}}_{Q} \left[\|\boldsymbol{y} - \boldsymbol{f}(c_Q, e, \boldsymbol{X}; \boldsymbol{\theta})\|_2^2 \,\middle|\, e \right], \tag{35}$$

where $\boldsymbol{f}(\cdot\,;\boldsymbol{\theta})$ is either predictor in (5). The forgetting at time t on the old data with the labels (c, e) is

$$\Phi_{ce}(t) \doteq \frac{1}{2}\,\mathbb{E}\left[\|\boldsymbol{f}(c, e, \boldsymbol{X}; \boldsymbol{\theta}(t)) - \boldsymbol{f}^\star(c, e, \boldsymbol{X})\|_2^2 \,\middle|\, c, e \right], \tag{36}$$

which is the increase of the risk of Model 1 on these labels.

Proposition 5: *In the setting above, let $c \neq c_Q$ and $e \in [E]$.*

(a) For the nested predictor, $\boldsymbol{W}_c$, $\boldsymbol{A}_c$, and $\boldsymbol{B}_{ce}$ stay at their initial values, so $\boldsymbol{f}(c, e, \boldsymbol{X}; \boldsymbol{\theta}(t)) = \boldsymbol{f}^\star(c, e, \boldsymbol{X})$ and $\Phi_{ce}(t) = 0$ for all $t \geq 0$.

(b) For the flat predictor, the blocks $\boldsymbol{W}_{cc'}$ and the experts $\boldsymbol{W}_{ce}^{\text{in}}, \boldsymbol{W}_{ce}^{\text{out}}$ stay at their initial values, so $\boldsymbol{f}(c, e, \boldsymbol{X}; \boldsymbol{\theta}(t)) = \boldsymbol{G}_{ce}^\star \boldsymbol{V}(t)\boldsymbol{m}$ for all $t \geq 0$, and as $t \to 0$,

$$\Phi_{ce}(t) = \frac{t^2}{2} \|\boldsymbol{B}_{ce}^\star \boldsymbol{A}_c^\star \boldsymbol{A}_{c_Q}^{\star\top} \boldsymbol{\Xi}\, \boldsymbol{\Sigma}_{ce}^{1/2}\|_F^2 + O(t^3), \qquad \boldsymbol{\Xi} \doteq \sum_{e' \in [E]} \pi_{e'}^Q \boldsymbol{B}_{c_Q e'}^{\star\top} (\boldsymbol{G}_{c_Q e'}^\star - \boldsymbol{G}'_{e'}) \boldsymbol{\Sigma}_{e'}^Q. \tag{37}$$

Moreover, $\dot{\boldsymbol{V}}(0) = -\boldsymbol{\nabla}_{\boldsymbol{V}} L_Q(\boldsymbol{\theta}(0)) = -\boldsymbol{A}_{c_Q}^{\star\top} \boldsymbol{\Xi}$, so $\boldsymbol{\Xi} \neq \boldsymbol{0}$ if and only if the new domain moves $\boldsymbol{V}$ at initialization. The leading coefficient in (37) is zero if $\boldsymbol{A}_c^\star \boldsymbol{A}_{c_Q}^{\star\top} = \boldsymbol{0}$, and it is positive if $\boldsymbol{A}_c^\star \boldsymbol{A}_{c_Q}^{\star\top}$ is invertible and $\boldsymbol{\Xi} \neq \boldsymbol{0}$.

When the rows of $\boldsymbol{A}_c^\star$ and of $\boldsymbol{A}_{c_Q}^\star$ are orthonormal, the singular values of $\boldsymbol{A}_c^\star \boldsymbol{A}_{c_Q}^{\star\top}$ are the cosines of the principal angles between the two row spaces, so this matrix is the overlap matrix by which Doan et al. (2021) characterize forgetting in the linearized regime. In this case, since $\boldsymbol{B}_{ce}^\star$ is injective and $\boldsymbol{\Sigma}_{ce}^{1/2}$ is invertible, the leading coefficient of (37) lies between $\frac{1}{2}\cos^2\theta_r\, \sigma_{\min}^2(\boldsymbol{B}_{ce}^\star)\, \|\boldsymbol{\Xi}\boldsymbol{\Sigma}_{ce}^{1/2}\|_F^2$ and $\frac{1}{2}\cos^2\theta_1\, \|\boldsymbol{B}_{ce}^\star\|_{\text{op}}^2\, \|\boldsymbol{\Xi}\boldsymbol{\Sigma}_{ce}^{1/2}\|_F^2$, where $\theta_1 \leq \cdots \leq \theta_r$ are the principal angles. The proposition compares the two predictors at equal time under gradient flow. When trained on Q, the flat predictor changes the pattern blocks and the experts of the label c_Q, as the nested predictor changes its column c_Q, and in addition it changes $\boldsymbol{V}$. The change of $\boldsymbol{V}$ is the only source of forgetting on the other coarse labels. The nested predictor has no such shared weight, and only the weights of its column c_Q change. Since $\boldsymbol{V}$ is redundant, the flat predictor with $\boldsymbol{V} = \boldsymbol{I}$ held fixed has the same function class, and it does not forget on the other coarse labels either. The forgetting in part (b) is therefore due to training a value map that is shared by all labels, as the MoE transformer does in Section 4.1.

F.3 Proof of Theorem 4

Both predictors are linear in a fixed set of polynomial features of the context, so least squares over a predictor is least squares over a class of functions inside one finite-dimensional linear space. Least squares over a class is a projection of the unconstrained least squares estimator onto the class. The error of the unconstrained estimator is asymptotically an isotropic Gaussian vector in the norm that defines the excess risk, and the projection keeps only its component in the tangent space of the class. The limit of $T\,\mathcal{E}$ is therefore σ^2 times a χ^2 variable whose number of degrees of freedom is the dimension of the class at the target. The dimension is the number of weights of the predictor minus the dimension of the set of reparametrizations that leave the prediction fixed, which we compute with the following lemma.

Lemma 6: *For $\boldsymbol{G}, \boldsymbol{W} \in \mathbb{R}^{D \times D}$, let $P(\boldsymbol{G}, \boldsymbol{W})$ be the map $(\boldsymbol{q}, \boldsymbol{z}) \mapsto \boldsymbol{G}(\boldsymbol{q}^\top \boldsymbol{W}\boldsymbol{z})\,\boldsymbol{z}$ from $\mathbb{R}^D \times \mathbb{R}^D$ to $\mathbb{R}^D$, an element of the finite-dimensional space $\mathcal{P}$ of polynomial maps that are linear in $\boldsymbol{q}$ and homogeneous quadratic in $\boldsymbol{z}$, and fix a norm on $\mathcal{P}$.*

(i) $P(\boldsymbol{G}, \boldsymbol{W}) = 0$ if and only if $\boldsymbol{G} = \boldsymbol{0}$ or $\boldsymbol{W} = \boldsymbol{0}$, and there is $\kappa_0 > 0$ with $\|P(\boldsymbol{G}, \boldsymbol{W})\| \geq \kappa_0 \|\boldsymbol{G}\|_F \|\boldsymbol{W}\|_F$ for all $\boldsymbol{G}, \boldsymbol{W}$.

(ii) Let $\boldsymbol{W}$ be invertible and $\boldsymbol{G} \neq \boldsymbol{0}$. Then $P(\boldsymbol{G}', \boldsymbol{W}') = P(\boldsymbol{G}, \boldsymbol{W})$ if and only if $(\boldsymbol{G}', \boldsymbol{W}') = (\mu^{-1}\boldsymbol{G}, \mu\boldsymbol{W})$ for some $\mu \neq 0$, and the differential of P at $(\boldsymbol{G}, \boldsymbol{W})$ vanishes on $(\dot{\boldsymbol{G}}, \dot{\boldsymbol{W}})$ if and only if $(\dot{\boldsymbol{G}}, \dot{\boldsymbol{W}}) = (\mu\boldsymbol{G}, -\mu\boldsymbol{W})$ for some $\mu \in \mathbb{R}$.

Proof. Since $\boldsymbol{q}^\top \boldsymbol{W}\boldsymbol{z} = (\boldsymbol{W}\boldsymbol{z})^\top \boldsymbol{q}$, the map $P(\boldsymbol{G}, \boldsymbol{W})$ sends $(\boldsymbol{q}, \boldsymbol{z})$ to $(\boldsymbol{G}\boldsymbol{z})(\boldsymbol{W}\boldsymbol{z})^\top \boldsymbol{q}$, so $P(\boldsymbol{G}, \boldsymbol{W}) = P(\boldsymbol{G}', \boldsymbol{W}')$ if and only if $(\boldsymbol{G}\boldsymbol{z})(\boldsymbol{W}\boldsymbol{z})^\top = (\boldsymbol{G}'\boldsymbol{z})(\boldsymbol{W}'\boldsymbol{z})^\top$ for all $\boldsymbol{z} \in \mathbb{R}^D$.

Part (i). If $P(\boldsymbol{G}, \boldsymbol{W}) = 0$, then every $\boldsymbol{z}$ lies in $\ker \boldsymbol{G}$ or in $\ker \boldsymbol{W}$, and a vector space is not the union of two proper subspaces, so $\boldsymbol{G} = \boldsymbol{0}$ or $\boldsymbol{W} = \boldsymbol{0}$. The map P is bilinear and continuous, and it does not vanish on the compact set $\{\|\boldsymbol{G}\|_F = \|\boldsymbol{W}\|_F = 1\}$, so its minimum κ_0 there is positive, and the bound follows by homogeneity.

Part (ii). Let $U \doteq \mathbb{R}^D \setminus \ker \boldsymbol{G}$, which is open and dense. For $\boldsymbol{z} \in U$, the matrix $(\boldsymbol{G}\boldsymbol{z})(\boldsymbol{W}\boldsymbol{z})^\top$ is nonzero of rank one, so $\boldsymbol{G}'\boldsymbol{z} = a(\boldsymbol{z})\boldsymbol{G}\boldsymbol{z}$ and $\boldsymbol{W}'\boldsymbol{z} = a(\boldsymbol{z})^{-1}\boldsymbol{W}\boldsymbol{z}$ for a scalar $a(\boldsymbol{z}) \neq 0$. Hence every $\boldsymbol{z} \in U$ is an eigenvector of $\boldsymbol{W}^{-1}\boldsymbol{W}'$. A linear map for which every vector of a nonempty open set is an eigenvector is a multiple of the identity: for linearly independent eigenvectors $\boldsymbol{z}_1, \boldsymbol{z}_2 \in U$ and small t, the vector $\boldsymbol{z}_1 + t\boldsymbol{z}_2 \in U$ is an eigenvector only if $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ have the same eigenvalue, so the eigenvalue is constant on U, which spans $\mathbb{R}^D$. Therefore $\boldsymbol{W}' = \mu\boldsymbol{W}$ for some $\mu \neq 0$, and then $\boldsymbol{G}'\boldsymbol{z} = \mu^{-1}\boldsymbol{G}\boldsymbol{z}$ on U, hence everywhere. The converse is clear. The differential of P at $(\boldsymbol{G}, \boldsymbol{W})$ sends $(\dot{\boldsymbol{G}}, \dot{\boldsymbol{W}})$ to $\boldsymbol{z} \mapsto (\dot{\boldsymbol{G}}\boldsymbol{z})(\boldsymbol{W}\boldsymbol{z})^\top + (\boldsymbol{G}\boldsymbol{z})(\dot{\boldsymbol{W}}\boldsymbol{z})^\top$. If this vanishes, then for $\boldsymbol{z} \in U$ the two rank-one terms cancel, so $\dot{\boldsymbol{G}}\boldsymbol{z} = b(\boldsymbol{z})\boldsymbol{G}\boldsymbol{z}$ and $\dot{\boldsymbol{W}}\boldsymbol{z} = -b(\boldsymbol{z})\boldsymbol{W}\boldsymbol{z}$ for a scalar $b(\boldsymbol{z})$, and the argument above gives $\dot{\boldsymbol{W}} = -\mu\boldsymbol{W}$ and $\dot{\boldsymbol{G}} = \mu\boldsymbol{G}$. The converse is again clear. □

Proof of Theorem 4. Part (a). The two constructions represent $\boldsymbol{f}^\star$ by substitution: in the flat predictor, the terms with $c_i \neq c$ have the score zero, and the remaining sum is $\boldsymbol{m}$. For the narrow flat predictor, write $\boldsymbol{G}_{ce} \doteq \boldsymbol{W}_{ce}^{\text{out}}\boldsymbol{W}_{ce}^{\text{in}}\boldsymbol{V}$ and suppose that it represents $\boldsymbol{f}^\star$. Then, for every (c, e) and every configuration of the labels of the context, the two predictions agree for almost every $(\boldsymbol{z}_1, \dots, \boldsymbol{z}_S)$, hence as polynomials. Setting $\boldsymbol{z}_i = \boldsymbol{0}$ for $2 \leq i \leq S-1$ leaves the identity

$$\boldsymbol{G}_{ce}\,(\boldsymbol{x}_S^\top \boldsymbol{M}\boldsymbol{x}_1)\,\boldsymbol{z}_1 = \boldsymbol{1}\{c_1 = c\}\,\boldsymbol{G}_{ce}^\star(\boldsymbol{z}_S^\top \boldsymbol{W}_c^\star \boldsymbol{z}_1)\,\boldsymbol{z}_1 \qquad \text{for all } \boldsymbol{z}_S, \boldsymbol{z}_1 \in \mathbb{R}^D,\ c_1 \in [C], \tag{38}$$

where $\boldsymbol{x}_S = (\boldsymbol{e}_c, \boldsymbol{z}_S)$ and $\boldsymbol{x}_1 = (\boldsymbol{e}_{c_1}, \boldsymbol{z}_1)$. Write $\boldsymbol{x}_S^\top \boldsymbol{M}\boldsymbol{x}_1 = \alpha + \boldsymbol{\beta}^\top \boldsymbol{z}_1 + \boldsymbol{\gamma}^\top \boldsymbol{z}_S + \boldsymbol{z}_S^\top \boldsymbol{M}_{22}\boldsymbol{z}_1$, where α, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$ depend on (c, c_1) and $\boldsymbol{M}_{22}$ does not. The right side of (38) with $c_1 = c$ is nonzero, so $\boldsymbol{G}_{ce} \neq \boldsymbol{0}$. Since $C \geq 2$, there is $c_1 \neq c$, for which the right side is zero; then the score vanishes for all $\boldsymbol{z}_S$ and all $\boldsymbol{z}_1 \notin \ker \boldsymbol{G}_{ce}$, hence identically, so $\boldsymbol{M}_{22} = \boldsymbol{0}$. For $c_1 = c$, the left side then has no term of degree one in $\boldsymbol{z}_S$ and two in $\boldsymbol{z}_1$, whereas the right side is $P(\boldsymbol{G}_{ce}^\star, \boldsymbol{W}_c^\star)(\boldsymbol{z}_S, \boldsymbol{z}_1)$ in the notation of Lemma 6, which is of exactly these degrees and nonzero by Lemma 6(i). This contradiction proves the claim.

Part (b), the function space. Let $\mathcal{H}$ be the space of functions of $(c, e, \boldsymbol{X})$ of the form

$$\boldsymbol{h}(c, e, \boldsymbol{X}) = \sum_{i \in [S-1]} \boldsymbol{p}_{cec_i}(\boldsymbol{z}_S, \boldsymbol{z}_i), \qquad \boldsymbol{p}_{cec'} \in \mathcal{P} \ \text{ for all } (c, e, c') \in [C] \times [E] \times [C], \tag{39}$$

with $\mathcal{P}$ as in Lemma 6. Both classes lie in $\mathcal{H}$: the nested predictor has $\boldsymbol{p}_{cec} = P(\boldsymbol{B}_{ce}\boldsymbol{A}_c, \boldsymbol{W}_c)$ and $\boldsymbol{p}_{cec'} = 0$ for $c' \neq c$, and the flat predictor has $\boldsymbol{p}_{cec'} = P(\boldsymbol{W}_{ce}^{\text{out}}\boldsymbol{W}_{ce}^{\text{in}}\boldsymbol{V}, \boldsymbol{W}_{cc'})$. Define $\|\boldsymbol{h}\|_\pi^2 \doteq \mathbb{E}\,\|\boldsymbol{h}(c, e, \boldsymbol{X})\|_2^2$, so that $\mathcal{E}(\boldsymbol{h}) = \|\boldsymbol{h} - \boldsymbol{f}^\star\|_\pi^2$. This is a norm on $\mathcal{H}$: if $\|\boldsymbol{h}\|_\pi = 0$, then for every (c, e) and every configuration of the labels, which has positive probability, the sum in (39) vanishes for almost every $(\boldsymbol{z}_1, \dots, \boldsymbol{z}_S)$, hence as a polynomial, and setting all but one of $\boldsymbol{z}_1, \dots, \boldsymbol{z}_{S-1}$ to zero shows that every $\boldsymbol{p}_{cec'} = 0$. In particular a function in $\mathcal{H}$ determines its components $\boldsymbol{p}_{cec'}$, and the finite moments of the contents make $\|\cdot\|_\pi$ finite. Let $\|\boldsymbol{h}\|_T^2 \doteq T^{-1}\sum_{i \in [T]} \|\boldsymbol{h}(c_i, e_i, \boldsymbol{X}_i)\|_2^2$ be the empirical norm. By the law of large numbers applied to a basis of $\mathcal{H}$, $\|\cdot\|_T \to \|\cdot\|_\pi$ uniformly on the unit sphere of $\mathcal{H}$, almost surely, so $\|\cdot\|_T$ is eventually a norm and the unconstrained least squares estimator $\hat{\boldsymbol{h}}^{\text{ols}} \in \arg\min_{\boldsymbol{h} \in \mathcal{H}} \sum_i \|\boldsymbol{y}_i - \boldsymbol{h}(c_i, e_i, \boldsymbol{X}_i)\|_2^2$ is eventually unique. By the normal equations, for every $\boldsymbol{h} \in \mathcal{H}$,

$$\frac{1}{T}\sum_{i \in [T]} \|\boldsymbol{y}_i - \boldsymbol{h}(c_i, e_i, \boldsymbol{X}_i)\|_2^2 = \frac{1}{T}\sum_{i \in [T]} \|\boldsymbol{y}_i - \hat{\boldsymbol{h}}^{\text{ols}}(c_i, e_i, \boldsymbol{X}_i)\|_2^2 + \|\boldsymbol{h} - \hat{\boldsymbol{h}}^{\text{ols}}\|_T^2. \tag{40}$$

Therefore a least squares estimator over a closed class $\mathcal{M} \subseteq \mathcal{H}$ is a projection of $\hat{\boldsymbol{h}}^{\text{ols}}$ onto $\mathcal{M}$ in the norm $\|\cdot\|_T$.

Since $\boldsymbol{f}^\star \in \mathcal{H}$ and $\boldsymbol{\varepsilon}$ is independent of $(c, e, \boldsymbol{X})$ with mean zero and covariance $\sigma^2\boldsymbol{I}$, the unconstrained problem is a linear regression with the feature map given by a basis $\boldsymbol{h}_1, \dots, \boldsymbol{h}_p$ of $\mathcal{H}$, and the central limit theorem gives $\sqrt{T}(\hat{\boldsymbol{h}}^{\text{ols}} - \boldsymbol{f}^\star) \rightsquigarrow \boldsymbol{\xi} \doteq \sum_j \xi_j \boldsymbol{h}_j$, where $(\xi_j)_j$ is a centered Gaussian vector with covariance $\sigma^2\boldsymbol{\Gamma}^{-1}$ and $\boldsymbol{\Gamma}_{jk} \doteq \mathbb{E}[\boldsymbol{h}_j(c, e, \boldsymbol{X})^\top \boldsymbol{h}_k(c, e, \boldsymbol{X})]$ is the Gram matrix of $\|\cdot\|_\pi$. That is, $\boldsymbol{\xi}$ is an isotropic Gaussian vector with variance σ^2 in the norm $\|\cdot\|_\pi$.

Part (b), the limit. Let $\mathcal{M}$ be a closed class that contains $\boldsymbol{f}^\star$ and is, near $\boldsymbol{f}^\star$, a smooth manifold of dimension d with tangent space $\mathcal{T}$ at $\boldsymbol{f}^\star$. Its least squares estimator $\hat{\boldsymbol{f}}$ satisfies $\|\hat{\boldsymbol{f}} - \hat{\boldsymbol{h}}^{\text{ols}}\|_T \leq \|\boldsymbol{f}^\star - \hat{\boldsymbol{h}}^{\text{ols}}\|_T = O_{\mathbb{P}}(T^{-1/2})$, so $\hat{\boldsymbol{f}} - \boldsymbol{f}^\star = O_{\mathbb{P}}(T^{-1/2})$. The objective in (40) is quadratic, its Hessian converges to that of $\|\cdot\|_\pi^2$, and a smooth manifold is Chernoff regular with tangent cone $\mathcal{T}$.

The theorem on constrained estimators of Geyer (1994, Section 4) then gives $\sqrt{T}(\hat{\boldsymbol{f}} - \boldsymbol{f}^\star) \rightsquigarrow \boldsymbol{P}_{\mathcal{T}}\boldsymbol{\xi}$, where $\boldsymbol{P}_{\mathcal{T}}$ is the projection onto $\mathcal{T}$ that is orthogonal in $\|\cdot\|_\pi$. By the continuous mapping theorem, $T\mathcal{E}(\hat{\boldsymbol{f}}) = \|\sqrt{T}(\hat{\boldsymbol{f}} - \boldsymbol{f}^\star)\|_\pi^2 \rightsquigarrow \|\boldsymbol{P}_{\mathcal{T}}\boldsymbol{\xi}\|_\pi^2$, which has the distribution $\sigma^2\chi^2_d$.

Part (b), the classes are manifolds. It remains to show that the two classes are smooth manifolds near $\boldsymbol{f}^\star$ and to compute their dimensions. We parametrize each class by the matrices that enter Lemma 6. The matrices of rank r in $\mathbb{R}^{p\times q}$ form a smooth manifold of dimension $r(p+q-r)$, and since the rank is lower semicontinuous, the closed set of matrices of rank at most r coincides with this manifold near each of its points. For the nested predictor, let $\bar{\boldsymbol{G}}_c \in \mathbb{R}^{ED\times D}$ stack $\boldsymbol{G}_{c1}, \dots, \boldsymbol{G}_{cE}$. A rank factorization shows that $(\boldsymbol{G}_{ce})_e = (\boldsymbol{B}_{ce}\boldsymbol{A}_c)_e$ for some $\boldsymbol{A}_c$ and $\boldsymbol{B}_{ce}$ if and only if $\operatorname{rank}\bar{\boldsymbol{G}}_c \le r$. The nested class is therefore the image $\mathcal{M}_{\mathrm{N}} = P_{\mathrm{N}}(\mathcal{D}_{\mathrm{N}})$ of the polynomial map $P_{\mathrm{N}}\colon (\boldsymbol{G},\boldsymbol{W}) \mapsto (P(\boldsymbol{G}_{ce},\boldsymbol{W}_c)\mathbf{1}\{c'=c\})_{c,e,c'}$ on the domain $\mathcal{D}_{\mathrm{N}} \doteq \{(\boldsymbol{G},\boldsymbol{W})\colon \operatorname{rank}\bar{\boldsymbol{G}}_c \le r \text{ for all } c\}$, with $\boldsymbol{W} = (\boldsymbol{W}_c)_c$. For the flat predictor, $\boldsymbol{W}^{\mathrm{out}}_{ce}\boldsymbol{W}^{\mathrm{in}}_{ce}\boldsymbol{V}$ with $\boldsymbol{V} = \boldsymbol{I}$ attains every matrix of rank at most r, so the flat class is $\mathcal{M}_{\mathrm{F}} = P_{\mathrm{F}}(\mathcal{D}_{\mathrm{F}})$ with $P_{\mathrm{F}}\colon (\boldsymbol{G},\boldsymbol{W}) \mapsto (P(\boldsymbol{G}_{ce},\boldsymbol{W}_{cc'}))_{c,e,c'}$ on $\mathcal{D}_{\mathrm{F}} \doteq \{(\boldsymbol{G},\boldsymbol{W})\colon \operatorname{rank}\boldsymbol{G}_{ce} \le r \text{ for all } (c,e)\}$, with $\boldsymbol{W} = (\boldsymbol{W}_{cc'})_{c,c'}$. The targets are $\boldsymbol{\theta}^\star_{\mathrm{N}} \doteq (\boldsymbol{G}^\star, (\boldsymbol{W}^\star_c)_c)$ and $\boldsymbol{\theta}^\star_{\mathrm{F}} \doteq (\boldsymbol{G}^\star, \boldsymbol{W}^\star)$ with $\boldsymbol{W}^\star_{cc} = \boldsymbol{W}^\star_c$ and $\boldsymbol{W}^\star_{cc'} = \mathbf{0}$ for $c' \ne c$. Since $\boldsymbol{B}^\star_{ce}$ is injective and $\boldsymbol{A}^\star_c$ is surjective, $\operatorname{rank}\boldsymbol{G}^\star_{ce} = \operatorname{rank}\bar{\boldsymbol{G}}^\star_c = r$, so each domain is a smooth manifold near its target, of dimension

$$\dim\mathcal{D}_{\mathrm{N}} = Cr(ED + D - r) + CD^2, \qquad \dim\mathcal{D}_{\mathrm{F}} = CEr(2D - r) + C^2D^2. \tag{41}$$

Both classes are closed, so the least squares estimators exist. Let $P_s(\boldsymbol{\theta}_k) \to \boldsymbol{h}$ with $\boldsymbol{\theta}_k \in \mathcal{D}_s$, and call $\boldsymbol{W}_c$, respectively $(\boldsymbol{W}_{cc'})_{c'}$, the pattern blocks of the label c. If they are all zero, then so are the components of $P_s(\boldsymbol{\theta}_k)$ with the query label c, and we may replace $\boldsymbol{G}_{ce}$ by $\mathbf{0}$ and the pattern blocks by any blocks of unit norm without changing $P_s(\boldsymbol{\theta}_k)$ or leaving $\mathcal{D}_s$. Otherwise, the action of $(\mathbb{R}\setminus\{0\})^C$ defined below rescales the pattern blocks of the label c to unit total Frobenius norm without changing $P_s(\boldsymbol{\theta}_k)$. After this normalization, for each (c,e) some pattern block $\boldsymbol{W}$ of the label c has $\|\boldsymbol{W}\|_F \ge C^{-1/2}$, and Lemma 6(i) gives $\|\boldsymbol{G}_{ce}\|_F \le C^{1/2}\kappa_0^{-1}\|P(\boldsymbol{G}_{ce},\boldsymbol{W})\|$, which is bounded because the components of $P_s(\boldsymbol{\theta}_k)$ converge. Hence $(\boldsymbol{\theta}_k)_k$ is bounded, a subsequence converges to some $\tilde{\boldsymbol{\theta}}$ in the closed set $\mathcal{D}_s$, and $\boldsymbol{h} = P_s(\tilde{\boldsymbol{\theta}}) \in \mathcal{M}_s$ by continuity.

Let $s \in \{\mathrm{N},\mathrm{F}\}$. The group $(\mathbb{R}\setminus\{0\})^C$ acts on $\mathcal{D}_s$ by $\boldsymbol{G}_{ce} \mapsto \lambda_c^{-1}\boldsymbol{G}_{ce}$ and $\boldsymbol{W}_c \mapsto \lambda_c\boldsymbol{W}_c$, respectively $\boldsymbol{W}_{cc'} \mapsto \lambda_c\boldsymbol{W}_{cc'}$ for all c', and P_s is invariant under this action. We claim that near $\boldsymbol{\theta}^\star_s$ the differential of P_s on the tangent space of $\mathcal{D}_s$ has the kernel of dimension exactly C that is tangent to the orbits, and that the fiber $P_s^{-1}(\boldsymbol{f}^\star)\cap\mathcal{D}_s$ is the orbit of $\boldsymbol{\theta}^\star_s$. For the fiber, Lemma 6(ii) applied to the components (c,e,c), where $\boldsymbol{W}^\star_c$ is invertible and $\boldsymbol{G}^\star_{ce} \ne \mathbf{0}$, gives $\boldsymbol{G}_{ce} = \mu_{ce}^{-1}\boldsymbol{G}^\star_{ce}$ and $\boldsymbol{W}_{cc} = \mu_{ce}\boldsymbol{W}^\star_c$, where μ_{ce} does not depend on e because the pattern is shared by the experts of the label c; and for the flat predictor, Lemma 6(i) applied to the components (c,e,c') with $c' \ne c$, where $\boldsymbol{f}^\star$ vanishes and $\boldsymbol{G}_{ce} \ne \mathbf{0}$, gives $\boldsymbol{W}_{cc'} = \mathbf{0}$. For the kernel, the tangent space of the orbit through a point of $\mathcal{D}_s$ consists of the directions $\dot{\boldsymbol{G}}_{ce} = \mu_c\boldsymbol{G}_{ce}$, $\dot{\boldsymbol{W}}_{cc'} = -\mu_c\boldsymbol{W}_{cc'}$, and it has dimension C wherever the blocks of each label are not all zero, which is an open condition that holds at the target; the kernel contains it by invariance. At the target, the kernel is no larger: Lemma 6(ii) applied to the components (c,e,c) gives $\dot{\boldsymbol{W}}_{cc} = -\mu_c\boldsymbol{W}^\star_c$ and $\dot{\boldsymbol{G}}_{ce} = \mu_c\boldsymbol{G}^\star_{ce}$ with μ_c common to the experts of the label, and, for the flat predictor, the components (c,e,c') with $c' \ne c$ reduce to $P(\boldsymbol{G}^\star_{ce},\dot{\boldsymbol{W}}_{cc'}) = 0$, so $\dot{\boldsymbol{W}}_{cc'} = \mathbf{0}$ by Lemma 6(i). Since the rank of the differential is lower semicontinuous, it is at least $\dim\mathcal{D}_s - C$ near the target, and it is at most this wherever the orbit has dimension C. Hence P_s has the constant rank $\dim\mathcal{D}_s - C$ near $\boldsymbol{\theta}^\star_s$, and by the constant rank theorem there is a neighborhood U of $\boldsymbol{\theta}^\star_s$ in $\mathcal{D}_s$ such that $P_s(U)$ is an embedded submanifold of $\mathcal{H}$ of dimension $d_s = \dim\mathcal{D}_s - C$.

We show that $\mathcal{M}_s$ coincides with $P_s(U)$ near $\boldsymbol{f}^\star$. Otherwise there are $\boldsymbol{\theta}_k \in \mathcal{D}_s$ with $P_s(\boldsymbol{\theta}_k) \to \boldsymbol{f}^\star$ and $P_s(\boldsymbol{\theta}_k) \notin P_s(U)$. The components (c,e,c) of $P_s(\boldsymbol{\theta}_k)$ converge to the nonzero components $P(\boldsymbol{G}^\star_{ce},\boldsymbol{W}^\star_c)$ of $\boldsymbol{f}^\star$, so by Lemma 6(i) the diagonal blocks of $\boldsymbol{\theta}_k$ are eventually nonzero, and by the invariance of P_s we may rescale so that they have the Frobenius norms of the $\boldsymbol{W}^\star_c$. The bound in Lemma 6(i) then shows that the $\boldsymbol{G}_{ce}$ of $\boldsymbol{\theta}_k$ are bounded and, since the components (c,e,c) are bounded away from zero, also bounded away from zero, so that all blocks of $\boldsymbol{\theta}_k$ are bounded. A subsequence converges to some $\tilde{\boldsymbol{\theta}}$ in the closed set $\mathcal{D}_s$ with $P_s(\tilde{\boldsymbol{\theta}}) = \boldsymbol{f}^\star$, which lies in the orbit of $\boldsymbol{\theta}^\star_s$. Applying the inverse group element to the subsequence gives points of U with the same images, a contradiction. Therefore $\mathcal{M}_s$ is a smooth manifold of dimension d_s near $\boldsymbol{f}^\star$, the limit above applies, and (41) gives (32). The difference (34) is positive for $C \ge 2$, and

$$Cd_{\mathrm{N}} - d_{\mathrm{F}} = C\big[Cr(D-r) + Er((C-2)D + r) - (C-1)\big] \ge C\big[C - (C-1)\big] > 0, \tag{42}$$

since $1 \le r < D$ gives $r(D-r) \ge 1$.

Part (c). With $T = F/F_s$ samples, $F\,\mathcal{E}(\hat{\boldsymbol{f}}^s) = F_s \cdot T\,\mathcal{E}(\hat{\boldsymbol{f}}^s) \rightsquigarrow \sigma^2 F_s \chi^2_{d_s}$ by part (b). The means $\sigma^2 d_s F_s/F'_s$ at the budgets F'_s are equal when $F'_{\mathrm{F}}/F'_{\mathrm{N}} = d_{\mathrm{F}}F_{\mathrm{F}}/(d_{\mathrm{N}}F_{\mathrm{N}}) = \kappa(S)$, and κ is increasing in S with the stated limits because $(S_0+S-1)/(S_0+(S-1)/C)$ is. □

F.4 Proof of Proposition 5

Proof of Proposition 5. The summary $\boldsymbol{m}$ of Model 1 has $\boldsymbol{\Sigma}_{ce} \succ \mathbf{0}$: for $\boldsymbol{v} \neq \mathbf{0}$, on the event $\mathcal{I} = \{1\}$ the scalar $\boldsymbol{v}^\top \boldsymbol{m} = (\boldsymbol{z}_S^\top \boldsymbol{W}_c^\star \boldsymbol{z}_1)(\boldsymbol{v}^\top \boldsymbol{z}_1)$ is a nonzero polynomial, because $\boldsymbol{W}_c^\star$ is invertible, so it is nonzero almost surely. The same argument gives $\boldsymbol{\Sigma}_e^Q \succ \mathbf{0}$.

The loss (35) depends on the weights only through the predictions $\boldsymbol{f}(c_Q, e, \boldsymbol{X}; \boldsymbol{\theta})$, $e \in [E]$. For the nested predictor these depend only on $\boldsymbol{W}_{c_Q}$, $\boldsymbol{A}_{c_Q}$, and $\boldsymbol{B}_{c_Q e}$, so the gradient with respect to $\boldsymbol{W}_c$, $\boldsymbol{A}_c$, and $\boldsymbol{B}_{ce}$ is zero for $c \neq c_Q$, these weights stay at their initial values, and $\boldsymbol{f}(c, e, \boldsymbol{X}; \boldsymbol{\theta}(t)) = \boldsymbol{f}^\star(c, e, \boldsymbol{X})$. This is part (a).

For the flat predictor, $\boldsymbol{f}(c_Q, e, \boldsymbol{X}; \boldsymbol{\theta}) = \boldsymbol{W}^{\text{out}}_{c_Q e} \boldsymbol{W}^{\text{in}}_{c_Q e} \boldsymbol{V} \sum_{i \in [S-1]} (\boldsymbol{z}_S^\top \boldsymbol{W}_{c_Q c_i} \boldsymbol{z}_i) \boldsymbol{z}_i$ depends only on the blocks $\boldsymbol{W}_{c_Q c'}$, the experts of the label c_Q, and $\boldsymbol{V}$. Hence the blocks $\boldsymbol{W}_{cc'}$ and the experts of a label $c \neq c_Q$ stay at their initial values $\boldsymbol{W}_{cc} = \boldsymbol{W}_c^\star$, $\boldsymbol{W}_{cc'} = \mathbf{0}$, $\boldsymbol{W}^{\text{in}}_{ce} = \boldsymbol{A}_c^\star$, and $\boldsymbol{W}^{\text{out}}_{ce} = \boldsymbol{B}^\star_{ce}$, so that $\boldsymbol{f}(c, e, \boldsymbol{X}; \boldsymbol{\theta}(t)) = \boldsymbol{B}^\star_{ce} \boldsymbol{A}_c^\star \boldsymbol{V}(t) \boldsymbol{m} = \boldsymbol{G}^\star_{ce} \boldsymbol{V}(t) \boldsymbol{m}$, and

$$\Phi_{ce}(t) = \frac{1}{2}\,\mathbb{E}\left[\|\boldsymbol{G}^\star_{ce}(\boldsymbol{V}(t) - \boldsymbol{I})\boldsymbol{m}\|_2^2 \,\middle|\, c, e\right] = \frac{1}{2}\|\boldsymbol{G}^\star_{ce}(\boldsymbol{V}(t) - \boldsymbol{I})\boldsymbol{\Sigma}^{1/2}_{ce}\|_F^2. \tag{43}$$

On the new domain the pattern is that of the label c_Q, so at $t = 0$ the predictor's summary $\sum_i (\boldsymbol{z}_S^\top \boldsymbol{W}_{c_Q c_i} \boldsymbol{z}_i) \boldsymbol{z}_i$ equals the summary $\boldsymbol{m}$ of Q. Since $\boldsymbol{\varepsilon}$ has mean zero and is independent of $\boldsymbol{X}$, for any matrix $\boldsymbol{G}$ we have $\mathbb{E}_Q[(\boldsymbol{G}\boldsymbol{m} - \boldsymbol{y})\boldsymbol{m}^\top \mid e] = (\boldsymbol{G} - \boldsymbol{G}'_e)\boldsymbol{\Sigma}_e^Q$, and therefore

$$\boldsymbol{\nabla}_{\boldsymbol{V}} L_Q(\boldsymbol{\theta}(0)) = \sum_{e \in [E]} \pi_e^Q \left(\boldsymbol{W}^{\text{out}}_{c_Q e} \boldsymbol{W}^{\text{in}}_{c_Q e}\right)^\top (\boldsymbol{G}^\star_{c_Q e} - \boldsymbol{G}'_e)\boldsymbol{\Sigma}_e^Q = \boldsymbol{A}^{\star\top}_{c_Q} \boldsymbol{\Xi}, \tag{44}$$

using $\boldsymbol{W}^{\text{out}}_{c_Q e} \boldsymbol{W}^{\text{in}}_{c_Q e} = \boldsymbol{B}^\star_{c_Q e} \boldsymbol{A}^\star_{c_Q}$ at $t = 0$. Since $\boldsymbol{A}^{\star\top}_{c_Q}$ is injective, this gradient is zero if and only if $\boldsymbol{\Xi} = \mathbf{0}$. The vector field of the gradient flow is polynomial in $\boldsymbol{\theta}$, so $\boldsymbol{\theta}(t)$ is smooth and $\boldsymbol{V}(t) = \boldsymbol{I} - t\boldsymbol{A}^{\star\top}_{c_Q}\boldsymbol{\Xi} + O(t^2)$. Substituting into (43) gives (37). Finally, $\boldsymbol{B}^\star_{ce}$ is injective and $\boldsymbol{\Sigma}^{1/2}_{ce}$ is invertible, so the leading coefficient is zero if and only if $\boldsymbol{A}^\star_c \boldsymbol{A}^{\star\top}_{c_Q} \boldsymbol{\Xi} = \mathbf{0}$; this holds when $\boldsymbol{A}^\star_c \boldsymbol{A}^{\star\top}_{c_Q} = \mathbf{0}$, and when $\boldsymbol{A}^\star_c \boldsymbol{A}^{\star\top}_{c_Q}$ is invertible it holds if and only if $\boldsymbol{\Xi} = \mathbf{0}$. □

F.5 Further Comparisons

We close with two remarks on the scope of Theorem 4: the first considers other shapes of the flat predictor, and the second relates the product features of the flat predictor to the full architectures.

Remark 7: *The comparison in Theorem 4 does not depend on the shape of the experts of the flat model. If its experts have rank $k > r$, its class contains $\mathcal{M}_{\mathrm{F}}$, so its tangent cone $\mathcal{K}$ at $\boldsymbol{f}^\star$ contains the tangent space $\mathcal{T}$ of $\mathcal{M}_{\mathrm{F}}$. The class is the image of a polynomial map and therefore Chernoff regular (Drton, 2009, Lemma 3.3), the limit of $T\mathcal{E}$ is the squared norm of a projection of $\boldsymbol{\xi}$ onto $\mathcal{K}$, and for a closed cone this is $\|\boldsymbol{\xi}\|_\pi^2 - \operatorname{dist}(\boldsymbol{\xi}, \mathcal{K})^2 \ge \|\boldsymbol{P}_{\mathcal{T}}\boldsymbol{\xi}\|_\pi^2$. Wider experts therefore do not reduce the limit. If $Cr < D$ and the row spaces of the matrices $\boldsymbol{A}_c^\star$ are linearly independent, the flat model can also represent the target with a shared value map $\boldsymbol{V} \in \mathbb{R}^{Cr \times D}$ that stacks the matrices $\boldsymbol{A}_c^\star$ and experts $\boldsymbol{W}_{ce} \in \mathbb{R}^{D \times Cr}$ of rank r, i.e., linear MLPs with r hidden units. The map $(\boldsymbol{V}, (\boldsymbol{W}_{ce})_{c,e}) \mapsto (\boldsymbol{W}_{ce}\boldsymbol{V})_{c,e}$ is invariant to $\boldsymbol{V} \mapsto \boldsymbol{X}\boldsymbol{V}$, $\boldsymbol{W}_{ce} \mapsto \boldsymbol{W}_{ce}\boldsymbol{X}^{-1}$ for invertible $\boldsymbol{X}$, and the kernel of its differential at the target consists of these directions only, because $\bigcap_{c,e} \ker \boldsymbol{W}_{ce} = \{\mathbf{0}\}$ there. The map terms of the dimension of this class at the target are therefore $CrD + CEr(Cr + D - r) - (Cr)^2 = Cr(D - Cr) + CEr(Cr + D - r)$, which exceed the map terms $Cr(ED + D - r)$ of d_{N} by $C(C-1)(E-1)r^2$, while the pattern terms are those of d_{F}. The extra terms count the parameters that each expert needs to select the r coordinates of its column among the Cr outputs of the shared value map.*

Remark 8: *An MoE transformer can simulate a column of* NESTOR *with a residual stream that is wider by a constant factor. Consider one head of column c, with logits $\langle \boldsymbol{Q}_c \boldsymbol{x}_j, \boldsymbol{K}_c \boldsymbol{x}_i \rangle$ and values $\boldsymbol{V}_c \boldsymbol{x}_i$ for $i \in \mathcal{U}_c$. In an MoE transformer, let the experts of the preceding layer be routed by the column labels, and let the expert for the label c output $\boldsymbol{Q}_c \boldsymbol{x}_i$, $\boldsymbol{K}_c \boldsymbol{x}_i$, and $\boldsymbol{V}_c \boldsymbol{x}_i$ in separate coordinates of the residual stream. A shared head whose queries, keys, and values are these coordinates has the same logits and values for all pairs of tokens in the same column, and the following experts can apply the output map of the column. The pairs in different columns are removed by adding $\lambda(\mathbf{1}\{c_i = c_j\} - 1)$ to the logits, which needs C more coordinates and is exact as $\lambda \to \infty$. This simulation stores the same number of parameters as the columns. It needs a residual stream that is wider by the total width of the queries, keys, and values of the active columns. This width is a constant multiple of D, and it increases the size of every other matrix. We therefore expect the two architectures to differ by constants at equal resources. The product features of the flat predictor in (5) are this construction for one layer, and Theorem 4 gives the resulting constant; Proposition 3 in Section E shows that a constant in the coefficients is a constant compute multiplier.*