FROM TOKENS TO CONTEXTS: PROGRESSIVE N-GRAM HASH ROUTING FOR SPARSE MOE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Early-layer hash routing has emerged as a practical alternative to fully learned expert selection in sparse Mixture-of-Experts (MoE) pretraining, but existing token-ID hashing answers only *where* deterministic routing can be used, not *what* routing key should be used within the shallow block. Because every occurrence of the same token is mapped to the same expert set, token-ID hashing is context-blind, can concentrate frequent-token traffic, and ignores the progressive change in representation granularity with depth. We propose PROGRAM, a layer-aware multi-prime n-gram hash routing method that treats shallow MoE routing as depth-aware routing-key design. PROGRAM replaces token-ID routing atoms with ordered local n-gram contexts, constructs K distinct experts through multi-prime hashing, stable deduplication, and fallback probing, and applies a progressive routing-order schedule $\boldsymbol{\nu} = (1, 2, 3)$ across the first three MoE layers. This yields a gradual transition from lexical identity to short-range contextual patterns: hashing determines expert identities in shallow layers, learned router scores compute adaptive mixture weights over the selected experts, and deeper layers retain standard learned top-K routing. We pretrain a 5B-parameter MoE model with 412M activated parameters per token on approximately 100B tokens and evaluate it on 12 knowledge, reasoning, and code benchmarks. In controlled comparisons, PROGRAM_{123} obtains the best result on 10 of the 12 benchmarks relative to dense-replacement baselines and achieves higher scores than a fully learned-routing MoE baseline on 11 benchmarks, with a tie on MGSM. It also yields lower language-modeling loss and lower sequence-level load-balancing loss during pretraining. These results support progressively contextualized hash routing as a useful shallow-layer routing prior. Code is available at `https://anonymous.4open.science/r/Program-FE0E/`.

1 INTRODUCTION

Sparsely gated Mixture-of-Experts (MoE) models have become a major architecture for scaling language models, increasing model capacity without proportionally increasing per-token computation (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022). This paradigm has been adopted by recent large-scale model families (Dai et al., 2024; Liu et al., 2024a; Yang et al., 2025). In a conventional MoE layer, a learned router scores all experts, selects a top-K subset for each token, and determines their mixture weights.

Despite its flexibility, learned top-K routing fails training reliably. Jointly optimizing expert selection, weighting, and specialization can produce severe load imbalance, leaving experts overloaded or under-utilized (Shazeer et al., 2017; Fedus et al., 2022). Existing solutions rely on intricate balancing losses, capacity constraints, or structured expert designs. For instance, DeepSeek combines fine-grained expert segmentation, shared-expert isolation, and auxiliary-loss-free balancing (Dai et al., 2024; Liu et al., 2024a). Meanwhile, load uniformity does not necessarily imply useful specialization, and overly uniform routing may increase expert overlap (Guo et al., 2026). This raises a broader question: *should every MoE layer rely entirely on learned representation-dependent routing?*

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.

Deterministic routing provides an alternative to fully learned expert selection. Early approaches such as *Hash Layers* (Roller et al., 2021) use fixed token-ID functions uniformly across routed layers, avoiding the need to jointly learn an additional expert selector. Large-scale studies further show that hash routing remains competitive across model scales (Clark et al., 2022). However, uniform token-ID hashing applies the same static lexical partition throughout the network. Its one-to-one token-to-expert mapping makes frequent-token traffic difficult to distribute, leading to diminishing routing gains and increasingly imbalanced expert loads as routing frequency or expert count grows.

A more recent direction introduces *layer-aware hybrid routing*. DeepSeek-V4 replaces the dense FFNs in its initial Transformer blocks with token-ID hash-routed MoE layers, while retaining learned routing in subsequent MoE layers (Xu et al., 2026). This choice suggests an important design principle: deterministic routing can serve as a useful shallow prior before representations become sufficiently contextual, whereas deeper layers benefit from learned representation-dependent routing. This principle is consistent with layer-wise analyses of generative LLMs, which suggest that lower layers prominently encode lexical information, while more complex, contextual, and abstract concepts generally require deeper representations (Liu et al., 2024b; Jin et al., 2024).

Nevertheless, existing layer-aware hash routing answers only *where* deterministic routing should be used, not *what* key it should route on or how that key should evolve across shallow layers. Token-ID hashing still assigns every occurrence of the same token to the same expert set regardless of local context. Consequently, high-frequency tokens can repeatedly concentrate traffic on fixed experts, and the same lexical routing granularity is reused across shallow layers despite their progressively changing representations.

To address this remaining limitation, we propose PROGRAM, a layer-aware multi-prime n-gram hash routing method. Rather than treating n-grams as a standalone heuristic, PROGRAM frames shallow MoE routing as *depth-aware routing-key design*: the routing key should move from lexical identity toward richer local context as the shallow representation becomes more contextual. Concretely, PROGRAM constructs ordered local n-gram contexts and maps them to candidate expert IDs using multiple prime-based hash functions. Stable deduplication and fallback probing then produce exactly K distinct experts. By refining a unigram routing key t into contextual keys (c, t), the maximum probability mass of an individual routing key cannot increase:

$$\max_{c,t} p(c, t) \leq \max_{t} p(t). \tag{1}$$

This refinement reduces key-level concentration caused by frequent tokens. It does not deterministically guarantee expert-load balance, because different contexts may still collide on the same experts; under a sufficiently dispersed hash mapping, however, lower key concentration reduces the expected load-deviation term.

Within the shallow routing block, PROGRAM adopts a progressive routing-order schedule, $\boldsymbol{\nu} = (1, 2, 3)$, forming a coarse-to-fine transition from token identity to short transitions and richer local patterns. The default configuration is denoted as PROGRAM_{123}. Hashing determines expert identities, while learned router scores compute adaptive mixture weights over the selected experts. Subsequent layers retain learned top-K routing. Thus, PROGRAM extends layer-aware hybrid routing from a placement choice into a routing-key design principle for the shallow MoE block.

Our contributions are summarized as follows:

- We identify a key limitation of existing early-layer token-ID hash routing: it decides where deterministic routing is used, but remains context-blind and reuses the same lexical routing key across shallow layers.
- We propose PROGRAM, a depth-aware routing-key design that combines local n-gram context modeling, multi-prime hashing, stable deduplication, and a progressive schedule $\boldsymbol{\nu} = (1, 2, 3)$ for deterministic expert selection, while retaining adaptive learned mixture weighting over the activated subset.
- In controlled pretraining experiments, PROGRAM_{123} improves downstream performance over dense-replacement and learned-routing controls on most benchmarks, while also reducing language-modeling and sequence-level load-balancing losses.

2 Related Work

Sparse MoE and learned routing. Sparsely gated Mixture-of-Experts (MoE) models increase capacity by activating only a small expert subset per token (Shazeer et al., 2017). GShard (Lepikhin et al., 2020), Switch Transformer (Fedus et al., 2022), and sparse systems (Riquelme et al., 2021; Jiang et al., 2024; Muennighoff et al., 2025) scale this paradigm with learned routers that determine both expert identities and mixture weights. DeepSeekMoE further improves expert specialization with fine-grained expert segmentation and shared-expert isolation (Dai et al., 2024), which is adopted in later large-scale MoE systems (Liu et al., 2024a). However, learned routing may suffer from load imbalance and unstable expert assignment, motivating auxiliary balancing losses, routing regularization, assignment-based routing, expert-choice routing, auxiliary-loss-free biases, and routing stabilization (Fedus et al., 2022; Zoph et al., 2022; Lewis et al., 2021; Zhou et al., 2022; Wang et al., 2024; Dai et al., 2022). ProGram is complementary: it keeps learned mixture weighting and learned routing, while refining deterministic expert selection within the shallow routing block.

Hash and layer-aware hybrid routing. Deterministic routing reduces the burden on learned expert selection. Hash Layers use fixed token-ID hash functions for parameter-free expert assignment (Roller et al., 2021), and scaling analyses show that fixed-hash routing can be a competitive baseline (Clark et al., 2022). These approaches apply static token-based routing across routed layers, which cannot adapt to increasingly contextual hidden representations. DeepSeek-V4 instead applies token-ID hash routing only in initial MoE layers and learned routing thereafter (Xu et al., 2026). This separates stable shallow routing from representation-dependent deep routing, but its shallow hash key remains context-blind. ProGram extends this line by replacing fixed token-ID keys with progressively richer local contexts.

Lexical and contextual routing priors. MoE analyses show that routing decisions can correlate strongly with token identity and emerge early in pretraining (Xue et al., 2024). Layer-wise analyses further suggest a coarse-to-fine evolution from lexical information to contextual and abstract semantics (Liu et al., 2024b; Jin et al., 2024). These observations motivate token-ID hashing as a shallow lexical prior, but placement alone does not determine the right routing granularity inside the shallow block. ProGram replaces the fixed unigram key with local n-gram keys and progressively increases their order under $\boldsymbol{\nu} = (1, 2, 3)$, moving from token identity to short transitions and richer local patterns. To the best of our knowledge, prior MoE routing has not systematically combined progressive n-gram hash routing with learned mixture weighting in shallow MoE layers.

3 Method

We propose ProGram, a layer-aware multi-prime n-gram hash routing method for shallow MoE layers. As illustrated in Figure 1, ProGram replaces single token-ID routing atoms with ordered local n-gram contexts and progressively increases the routing order with layer depth. In the first three MoE layers, deterministic multi-prime hashing selects expert identities, while learned router scores retain adaptive mixture weighting over the selected experts. All subsequent MoE layers preserve standard learned top-K routing.

3.1 Early-Layer Hybrid Routing

Consider an MoE layer with E routed experts, of which K are activated per token. Let t_i be the token ID at position i, and let $\mathbf{h}_i^{(\ell)}$ be its hidden state before the MoE block at layer ℓ. A learned router R_ℓ produces routing logits $\mathbf{s}_i^{(\ell)}$

$$\mathbf{s}_i^{(\ell)} = R_\ell\left(\mathbf{h}_i^{(\ell)}\right) \in \mathbb{R}^E. \tag{2}$$

Let $\mathcal{L}_h = \{1, 2, 3\}$ denote the first three MoE layers. Their expert identities are selected by context hashing, whereas deeper layers retain learned routing:

$$\mathcal{E}_i^{(\ell)} = \begin{cases} \mathrm{HashRoute}\left(g_i^{(n_\ell)}\right), & \ell \in \mathcal{L}_h, \\ \mathrm{TopK}\left(\mathbf{s}_i^{(\ell)}, K\right), & \ell \notin \mathcal{L}_h. \end{cases} \tag{3}$$

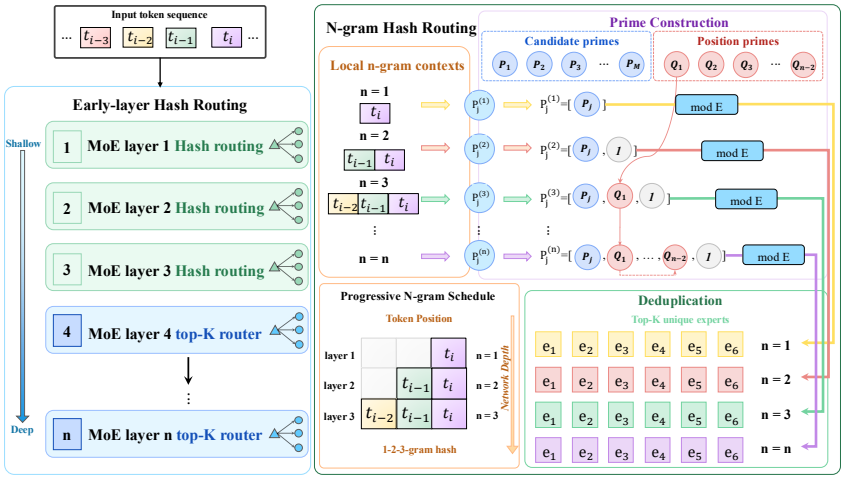


Figure 1: Overview of PROGRAM$_{123}$. The first three MoE layers adopt progressively increasing routing orders under $\boldsymbol{\nu} = (1, 2, 3)$, while subsequent MoE layers retain standard learned top-K routing. For each local n-gram context, candidate and position-specific primes define hash coefficient vectors, which map the context to candidate expert IDs through modulo-E hashing. Stable deduplication and fallback probing then select exactly K distinct experts for sparse activation.

Hashing therefore changes only expert identity selection in shallow layers. Shared experts remain unchanged, and the learned router is preserved to weight the selected experts.

3.2 LAYER-AWARE MULTI-PRIME n-GRAM HASHING

Local routing contexts. For routing order n, the ordered local context at position i is

$$g_i^{(n)} = (t_{i-n+1}, t_{i-n+2}, \ldots, t_i)\,, \tag{4}$$

where missing history positions are filled with a fixed sentinel token ID (e.g., `[BOS]`). In particular,

$$g_i^{(n)} = \begin{cases} (t_i), & n = 1, \\ (t_{i-1}, t_i), & n = 2, \\ (t_{i-2}, t_{i-1}, t_i), & n = 3. \end{cases} \tag{5}$$

Prime-based coefficient construction. For each routing order, candidate primes $\mathcal{P}_{\text{cand}} = \{P_1, \ldots, P_M\}$ $(M \geq K)$ generate M candidate expert IDs. For $n \geq 3$, fixed position primes $\mathcal{P}_{\text{pos}} = \{Q_1, \ldots, Q_{n-2}\}$ encode the intermediate positions. The j-th coefficient vector is defined as

$$\mathbf{p}_j^{(n)} = \begin{cases} [P_j], & n = 1, \\ [P_j, Q_1, \ldots, Q_{n-2}, 1], & n \geq 2. \end{cases} \tag{6}$$

All coefficients are fixed prior to training. We choose primes satisfying $\gcd(P_j, E) = \gcd(Q_k, E) = 1$ and $P_j \not\equiv P_{j'} \pmod{E}$ for all $j \neq j'$. These conditions make each multiplicative coefficient invertible modulo E and avoid identical candidate streams, but they do not by themselves imply statistical independence. The routing granularity is determined primarily by the local context and its layer-wise schedule rather than by task-specific prime tuning.

Candidate expert generation. To map context into routing decisions without parameter overhead, we compute candidate expert indices via a modular inner product between the prime coefficient vector $\mathbf{p}_j^{(n_\ell)}$ and the n_ℓ-gram context vector $g_i^{(n_\ell)}$:

$$c_{i,j}^{(\ell)} = \left\langle \mathbf{p}_j^{(n_\ell)}, g_i^{(n_\ell)} \right\rangle \bmod E, \quad j \in \{1, \ldots, M\}. \tag{7}$$

For the progressive schedule ($n_\ell \in \{1, 2, 3\}$), Eq. equation 7 explicitly evaluates to:

$$c_{i,j}^{(\ell)} = \begin{cases} (P_j t_i) \bmod E, & n_\ell = 1, \\ (P_j t_{i-1} + t_i) \bmod E, & n_\ell = 2, \\ (P_j t_{i-2} + Q_1 t_{i-1} + t_i) \bmod E, & n_\ell = 3. \end{cases} \tag{8}$$

The position-specific coefficients make the hash order-sensitive and reduce permutation-induced aliasing compared with bag-of-words keys. Modulo collisions can still occur, so the candidate sequence $(c_{i,1}^{(\ell)}, \ldots, c_{i,M}^{(\ell)})$ undergoes stable deduplication, retaining the first K unique expert IDs whenever available.

3.3 Duplicate-Free Routing and Adaptive Weighting

Algorithm 1 denotes the complete routing procedure within the pool of routed experts. Shared experts, when present, operate as an independent deterministic bypass and are unaffected by the hash selection over routed experts.

If stable deduplication returns fewer than K experts, fallback probing starts from the last candidate $u_i^{(\ell)} = c_{i,M}^{(\ell)}$ and probes

$$e_m = \left(u_i^{(\ell)} + m \cdot d\right) \bmod E, \qquad \gcd(d, E) = 1. \tag{9}$$

Experts already selected are skipped. Since the stride is coprime with E, the probe visits every residue modulo E before repeating; when $K \leq E$, this deterministically completes a set of exactly K distinct routed experts. In practice, we set $d = 131$. The normalized mixture weights are computed via a restricted softmax over $\mathcal{E}_i^{(\ell)}$:

$$w_{i,e}^{(\ell)} = \frac{\exp(s_{i,e}^{(\ell)})}{\sum_{e' \in \mathcal{E}_i^{(\ell)}} \exp(s_{i,e'}^{(\ell)})}, \quad e \in \mathcal{E}_i^{(\ell)}, \tag{10}$$

and the final routed representation is given by

$$\mathbf{y}_i^{(\ell)} = \sum_{e \in \mathcal{E}_i^{(\ell)}} w_{i,e}^{(\ell)} \, f_e^{(\ell)}\left(\mathbf{h}_i^{(\ell)}\right). \tag{11}$$

Thus, hashing determines expert identities, while the learned router retains representation-dependent expert combination. During training, the hash-selected identities are discrete and deterministic, so gradients do not flow through the hash assignment itself. The router still predicts logits over the routed expert pool, and gradients from the language-modeling objective flow through the restricted softmax weights of the selected experts. Any router regularization or load-balancing term is kept the same as in the base MoE recipe; in hash-routed layers, it can affect router logits and mixture weights but not the deterministic expert identities.

3.4 Progressive Scheduling and Load-Concentration Analysis

The layer-wise routing-order schedule is formalized as

$$\boldsymbol{\nu} = (n_1, n_2, n_3), \tag{12}$$

where n_ℓ denotes the n-gram context order at the ℓ-th hash-routed layer. The default ProGram$_{123}$ configuration adopts a coarse-to-fine routing progression $\boldsymbol{\nu}_{\text{prog}} = (1, 2, 3)$:

$$\text{token identity} \longrightarrow \text{short transition} \longrightarrow \text{local contextual pattern}$$

Specifically, unigram routing provides an essential lexical anchor, whereas bigram and trigram routing progressively incorporate local transition dynamics and richer contextual patterns. Starting directly with higher-order contexts may fragment token-level regularities before stable lexical specialization is established.

This progression can be motivated through routing-key concentration. Let

$$p_b = \Pr(t_i = b), \qquad \pi_{a,b} = \Pr(t_{i-1} = a, t_i = b). \tag{13}$$

Algorithm 1 PROGRAM routing for token i at layer ℓ.

Require: $\mathbf{t}, \mathbf{h}_i^{(\ell)}, \ell, \mathcal{L}_h, \boldsymbol{\nu}, E, K, \{\mathbf{p}_j^{(n)}\}_{j=1}^{M}, d$
Ensure: $\mathcal{E}_i^{(\ell)}, \mathbf{y}_i^{(\ell)}$

$\mathbf{s}_i^{(\ell)} \leftarrow R_\ell(\mathbf{h}_i^{(\ell)})$
if $\ell \in \mathcal{L}_h$ **then**
  $n \leftarrow \nu_\ell$; $g_i^{(n)} \leftarrow (t_{i-n+1}, \ldots, t_i)$
  $\mathbf{C} \leftarrow [\langle \mathbf{p}_j^{(n)}, \mathbf{g}_i^{(n)} \rangle \bmod E]_{j=1}^{M}$
  $\mathcal{E}_i^{(\ell)} \leftarrow \mathrm{FirstKDistinct}(\mathbf{C}, K)$
  $m \leftarrow 0$
  while $|\mathcal{E}_i^{(\ell)}| < K$ **do**
    $e \leftarrow (\mathbf{C}[M] + md) \bmod E$; $m \leftarrow m + 1$
    if $e \notin \mathcal{E}_i^{(\ell)}$ **then**
      append e to $\mathcal{E}_i^{(\ell)}$
    end if
  end while
else
  $\mathcal{E}_i^{(\ell)} \leftarrow \mathrm{TopK}(\mathbf{s}_i^{(\ell)}, K)$
end if
$w_{i,e}^{(\ell)} \leftarrow \mathrm{Softmax}_{e \in \mathcal{E}_i^{(\ell)}}(s_{i,e}^{(\ell)})$
$\mathbf{y}_i^{(\ell)} \leftarrow \sum_{e \in \mathcal{E}_i^{(\ell)}} w_{i,e}^{(\ell)} f_e^{(\ell)}(\mathbf{h}_i^{(\ell)})$
return $\mathcal{E}_i^{(\ell)}, \mathbf{y}_i^{(\ell)}$

Since $p_b = \sum_a \pi_{a,b}$,

$$\max_{a,b} \pi_{a,b} \le \max_b p_b, \tag{14}$$

and

$$\sum_{a,b} \pi_{a,b}^2 \le \sum_b p_b^2. \tag{15}$$

Thus, bigram routing refines a token-level key into lower-mass contextual keys. This key-level concentration result does not by itself guarantee expert-load balance, because distinct contexts may still collide on the same experts after hashing. It instead reduces the potential source of persistent hot spots caused by frequent tokens.

The same argument extends recursively. For an n-gram routing key $g^{(n)}$,

$$p\left(g^{(n)}\right) = \sum_x p\left(x, g^{(n)}\right), \tag{16}$$

which implies

$$\max_{g^{(n+1)}} p\left(g^{(n+1)}\right) \le \max_{g^{(n)}} p\left(g^{(n)}\right), \tag{17}$$

and

$$\sum_{g^{(n+1)}} p\left(g^{(n+1)}\right)^2 \le \sum_{g^{(n)}} p\left(g^{(n)}\right)^2. \tag{18}$$

Under an idealized dispersed-hashing assumption in which different routing keys are mapped to nearly uniform K-expert subsets, the expected squared deviation of expert loads is proportional to this squared key concentration. Therefore, increasing the routing order can reduce the expected load-deviation term, but it is not a deterministic guarantee for every corpus or hash realization. Larger n is also not always preferable because excessive context order can fragment lexical patterns. The schedule $(1, 2, 3)$ combines an initial lexical anchor with gradual contextual refinement.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

All controlled models share the same 16-layer Transformer backbone. In the sparse configuration, each MoE layer contains 192 routed experts and one shared expert, with six routed experts activated

Model	Active Params.	Knowledge					Reasoning			Code			
		MMLU-Pro	MMLU-Redux	MLQA-Zh-Zh	INCLUDE	GPQA	GSM8K	MGSM	BBH	MBPP	MBPP+	Human-Eval	Repo-Bench
Public Pretrained Models													
Pythia-410M	405M	6.92	26.49	0.76	26.21	27.23	1.74	1.60	23.04	1.20	2.91	1.83	**14.99**
Pythia-2.8B	2.8B	9.08	26.38	11.37	26.72	25.00	1.97	2.80	**28.57**	9.80	15.34	6.10	12.10
TinyLlama-1.1B	1.1B	7.94	25.97	2.47	26.98	26.56	1.36	2.00	21.47	0.40	0.53	5.49	10.12
OLMoE-1B-7B	1.3B	7.75	26.60	3.68	28.40	27.69	1.21	2.80	25.77	5.20	6.88	6.10	11.68
ProGram$_{123}$	**412M**	**9.99**	**26.94**	**15.90**	**32.38**	**29.24**	**5.16**	**8.80**	21.62	**10.60**	**16.14**	**13.41**	6.60
Our Pretrained Models													
Dense-First-1	410M	7.55	25.76	**16.55**	28.91	23.44	4.40	7.60	14.71	3.80	8.99	6.71	5.32
Dense-First-3	407M	8.00	25.25	16.12	31.29	25.45	4.70	7.60	17.85	9.80	15.08	**14.63**	5.53
Learned-MoE	412M	7.70	25.44	8.64	25.85	28.12	1.82	**8.80**	4.30	1.60	2.38	5.49	3.12
ProGram$_{123}$	**412M**	**9.99**	**26.94**	15.90	**32.38**	**29.24**	**5.16**	**8.80**	**21.62**	**10.60**	**16.14**	13.41	**6.60**

Table 1: Comparison across knowledge, reasoning, and code benchmarks. The table compares ProGram$_{123}$ with publicly available pretrained models and controlled baselines trained under our pretraining setting. Active parameters denote the number of parameters used for processing each token. Best results are in bold, and second-best results are underlined within each comparison group.

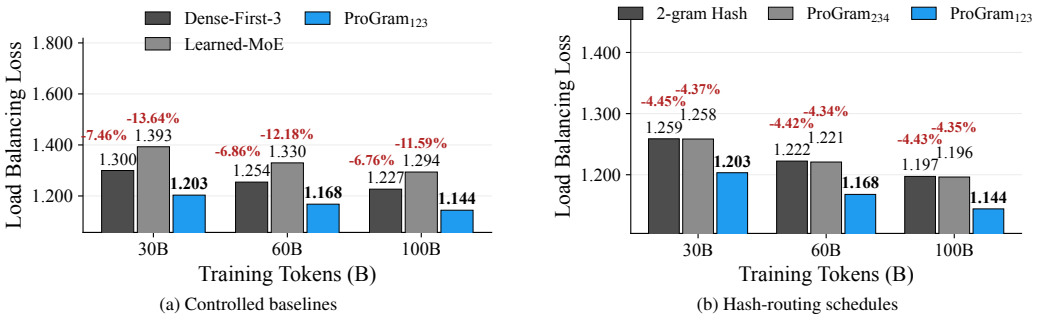


Figure 2: Sequence-level load-balancing loss during pretraining. (a) Comparison of ProGram$_{123}$ with Dense-First-3 and Learned-MoE. (b) Comparison of ProGram$_{123}$ with the fixed 2-gram hash variant and ProGram$_{234}$.

per token. Following the shallow hash-routing configuration adopted in DeepSeek-V4 (Xu et al., 2026), we apply hash-based expert selection to the first three MoE layers. In ProGram$_{123}$, these layers use unigram, bigram, and trigram routing, respectively, while all subsequent MoE layers retain standard learned top-K routing. The resulting model contains approximately 5B total parameters and activates 412M parameters per token.

All controlled variants are pretrained for approximately 100B tokens with the same data, tokenizer, optimizer, learning-rate schedule, batch configuration, and load-balancing settings; they differ only in how the first three layers are instantiated and routed. For hash-routed layers, expert identities are fixed by the hash function and receive no gradient through the discrete selection step. The learned router still produces logits over the routed expert pool, so language-modeling gradients and the load-balancing term act through the restricted softmax weights of the dispatched experts rather than through the deterministic hash identities.

We evaluate all models on 12 benchmarks spanning knowledge, reasoning, and code, as summarized in Table 1. We compare ProGram$_{123}$ with public pretrained models and controlled baselines under identical evaluation settings. Detailed benchmark configurations and baseline specifications are provided in Appendix A. We use three controlled baselines: Dense-First-1 and Dense-First-3, which replace the first one or three MoE layers with dense feed-forward layers under the same training budget, and Learned-MoE, which retains MoE layers in all 16 layers and uses learned top-K routing. These baselines test whether shallow layers should be dense and whether fully learned routing is sufficient under matched activated capacity and identical pretraining settings.

Method	Knowledge					Reasoning			Code			
	MMLU-Pro	MMLU-Redux	MLQA-Zh-Zh	INCLUDE	GPQA	GSM8K	MGSM	BBH	MBPP	MBPP+	Human-Eval	Repo-Bench
PROGRAM *w/o Progression*												
Token-ID Hash (1-gram)	4.58	25.01	9.13	28.89	25.00	2.42	2.00	10.98	4.00	8.47	1.83	5.20
2-gram Hash	8.18	24.86	9.87	32.16	23.66	**5.69**	5.20	12.29	2.20	5.03	9.15	3.33
3-gram Hash	6.40	25.98	13.43	28.02	<u>26.79</u>	<u>5.61</u>	7.20	<u>17.52</u>	7.60	11.64	<u>14.02</u>	<u>6.18</u>
4-gram Hash	<u>9.02</u>	25.80	<u>13.61</u>	27.81	24.33	4.70	<u>8.40</u>	12.10	<u>9.00</u>	<u>15.34</u>	10.37	4.83
PROGRAM_{234}	8.36	<u>26.19</u>	12.83	**33.04**	25.00	3.30	<u>8.40</u>	8.69	5.00	9.00	**14.63**	4.42
PROGRAM_{123}	**9.99**	**26.94**	**15.90**	<u>32.38</u>	**29.24**	5.16	**8.80**	**21.62**	**10.60**	**16.14**	13.41	**6.60**

Table 2: Ablation of different n-gram hash-routing schedules across knowledge, reasoning, and code benchmarks. The fixed-order variants use the same n-gram order in all three shallow MoE layers, whereas PROGRAM_{234} and PROGRAM_{123} use progressive routing schedules. Best results are in bold, and second-best results are underlined.

4.2 COMPARISON WITH PUBLIC & CONTROLLED BASELINES

PROGRAM_{123} with public pretrained models and controlled baselines. Among the public models, PROGRAM_{123} achieves the best result on 10 of 12 benchmarks while activating only 412M parameters per token, ranking first on all five knowledge benchmarks, two of three reasoning benchmarks, and three of four code benchmarks. It also outperforms the larger Pythia-2.8B model on most benchmarks despite substantially fewer active parameters, suggesting that the gains are not limited to individual tasks or explained solely by larger active computation.

The same pattern holds under the controlled pretraining setting. PROGRAM_{123} achieves the best result on 10 of 12 benchmarks, with only MLQA-Zh-Zh and HumanEval favoring the dense baselines. This shows that simply replacing shallow MoE layers with dense computation does not match the overall performance of structured shallow routing. Against LEARNED-MOE, which has the same total and activated parameter counts, PROGRAM_{123} performs better on 11 benchmarks and ties on MGSM, isolating the benefit of the routing policy itself. This suggests that deterministic expert selection in shallow layers provides a useful structural prior, while learned mixture weighting and deeper learned routing preserve representation-dependent adaptation.

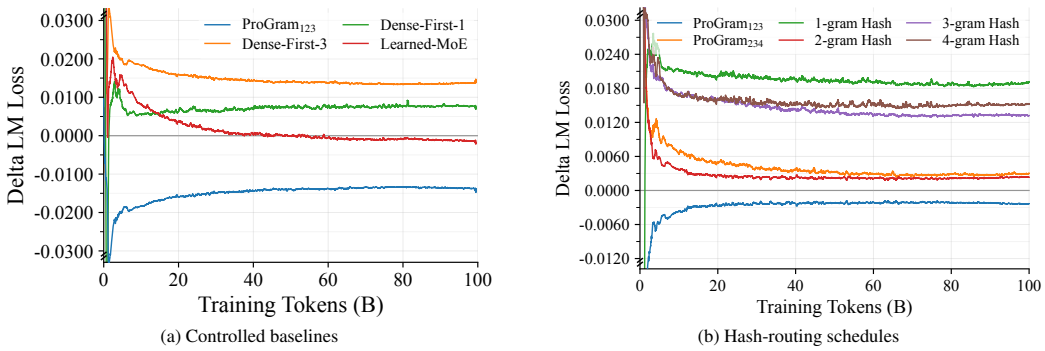
Figure 2(a) provides training-time evidence. At 30B, 60B, and 100B tokens, PROGRAM_{123} reduces load-balancing loss by 7.46%, 6.86%, and 6.76% relative to DENSE-FIRST-3, and by 13.64%, 12.18%, and 11.59% relative to LEARNED-MOE, respectively. Figure 3(a) further shows lower language-modeling loss than the controlled baselines over most of pretraining. Together, these results indicate that the shallow routing prior improves expert traffic distribution and optimization quality while leaving deeper routing fully adaptive.

4.3 ABLATION: N-GRAM ORDER AND PROGRESSIVE SCHEDULE

We compare PROGRAM_{123} with four fixed-order variants using 1-gram, 2-gram, 3-gram, and 4-gram hash routing across the first three MoE layers. The 1-gram variant is denoted Token-ID Hash (1-gram), as it corresponds to early-layer token-ID hash routing under our architecture and training recipe. We additionally compare with the progressive PROGRAM_{234} schedule. All variants share the same architecture, data, optimization settings, and training budget, differing only in the routing orders of the first three MoE layers. This isolates the effect of fixed routing order versus progressively changing routing granularity with depth.

Table 2 shows that PROGRAM_{123} achieves the best result on 9 of 12 benchmarks and the second-best result on INCLUDE, ranking first on four of five knowledge benchmarks, two of three reasoning benchmarks, and three of four code benchmarks. No alternative schedule achieves comparable consistency across all three capability categories.

The fixed-order variants obtain isolated gains on individual tasks, but none performs strongly across the full evaluation suite. Token-ID Hash (1-gram) preserves a stable token-level partition but cannot distinguish different local usages of the same token. Higher-order fixed variants introduce richer



(a) Controlled baselines (b) Hash-routing schedules

Figure 3: Language-modeling loss dynamics throughout 100B-token pretraining. (a) Comparison of PROGRAM$_{123}$ with the controlled baselines. (b) Comparison of PROGRAM$_{123}$ with alternative hash-routing schedules.

context, yet their performance is not monotonic with n, indicating that simply increasing context order is insufficient and that its placement across depth also matters.

Comparison with PROGRAM$_{234}$ isolates the role of the unigram starting point. Although PROGRAM$_{234}$ achieves the best result on INCLUDE and HumanEval, PROGRAM$_{123}$ outperforms it on 10 of 12 benchmarks. Both schedules increase context order with depth, but only PROGRAM$_{123}$ begins with a lexical anchor before moving to richer local patterns. This suggests that progressive routing benefits from both increasing context order and beginning with a stable token-level partition.

These results are consistent with the intended coarse-to-fine organization of the first three MoE layers. Unlike fixed-order variants that use the same contextual granularity throughout, PROGRAM$_{123}$ changes the routing key with depth, preserving broad lexical regularities early while progressively separating context-dependent usages without imposing high-order fragmentation from the outset.

Figure 2(b) provides corresponding training-time evidence. At 30B, 60B, and 100B tokens, PROGRAM$_{123}$ reduces load-balancing loss relative to the fixed 2-gram variant by 4.45%, 4.42%, and 4.43%, and relative to PROGRAM$_{234}$ by 4.37%, 4.34%, and 4.35%, respectively. The consistent margins indicate that the advantage persists throughout pretraining. Figure 3(b) shows the same overall ranking, with PROGRAM$_{123}$ maintaining the most favorable trajectory over most of pretraining. Together, the downstream and training results support the schedule $\boldsymbol{\nu} = (1, 2, 3)$ over both fixed-order and more context-heavy alternatives.

5 Conclusion and Limitations

We presented PROGRAM, a context-aware prior for shallow MoE routing that extends token-ID hashing to local n-gram contexts and progressively increases routing order across the first three layers. The default PROGRAM$_{123}$ schedule, $\boldsymbol{\nu} = (1, 2, 3)$, uses multi-prime hashing and fallback probing to construct duplicate-free expert sets while preserving learned mixture weighting. This decouples deterministic expert selection from adaptive weighting while retaining learned top-K routing in deeper layers, gradually refining routing keys from lexical identity to short-range contextual patterns. In controlled pretraining experiments, PROGRAM$_{123}$ outperforms dense-replacement and fully learned-routing MoE baselines on most benchmarks and achieves the best result on 9 of 12 tasks among the evaluated hash-routing schedules. Lower language-modeling and sequence-level load-balancing losses further support this progressive routing strategy. These results suggest that shallow MoE routing benefits from depth-aware routing granularity rather than a uniform hash order or an abrupt transition to richer contexts.

Our study is limited to a single model scale, a 100B-token pretraining budget, and hash routing in the first three MoE layers. The theoretical analysis characterizes routing-key concentration and expected load behavior under dispersed hashing, but cannot deterministically guarantee expert-load balance for every corpus or hash realization. Moreover, the schedule and hash coefficients are fixed before training rather than adapted to model depth or data characteristics. Future work will more systematically study adaptive routing-order schedules and larger, longer-context MoE models.

AI USE STATEMENT

In this work, we used generative AI tools to assist with literature discovery and summarization, the drafting of basic code during research and experimentation, and language polishing of parts of the manuscript. All AI-assisted literature findings were checked against the original sources, and AI-assisted code was reviewed and tested by the authors before use. AI-assisted text was manually reviewed and revised to ensure that it accurately reflects the methods, experimental results, and claims of this work. The authors made all final decisions regarding the research methodology, experiments, analysis, and conclusions. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information for reproducing our experiments in the main paper and appendix. The main paper specifies the PROGRAM routing algorithm, model architecture, training setup, and controlled baseline design. The appendix provides the complete controlled model configurations, evaluation environment, benchmark-specific protocols, few-shot settings, random seeds, checkpoint selection, and execution details. It also includes computational complexity analysis and theoretical results supporting the proposed routing mechanism. Code, evaluation scripts, and configuration files will be released upon publication.

REFERENCES

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International conference on machine learning*, pp. 2397–2430. PMLR, 2023.

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*, pp. 4057–4086. PMLR, 2022.

Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. Stablemoe: Stable routing strategy for mixture of experts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7085–7095, 2022.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu 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 (Volume 1: Long Papers)*, pp. 1280–1297, 2024.

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.

Hongcan Guo, Haolang Lu, Guoshun Nan, Bolun Chu, Jialin Zhuang, Yuan Yang, Wenhao Che, Xinye Cao, Sicong Leng, Qimei Cui, et al. Advancing expert specialization for better moe. *Advances in Neural Information Processing Systems*, 38:48767–48809, 2026.

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.

Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, et al. Exploring concept depth: How large language models acquire knowledge and concept at different layers? *arXiv preprint arXiv:2404.07066*, 2024.

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. *arXiv preprint arXiv:2006.16668*, 2020.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pp. 6265–6274. PMLR, 2021.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Zhu Liu, Cunliang Kong, Ying Liu, and Maosong Sun. Fantastic semantics and where to find them: Investigating which layers of generative llms reflect lexical semantics. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 14551–14558, 2024b.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. Olmoe: Open mixture-of-experts language models. In *International Conference on Learning Representations*, volume 2025, pp. 62061–62121, 2025.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34:8583–8595, 2021.

Stephen Roller, Sainbayar Sukhbaatar, Jason Weston, et al. Hash layers for large sparse models. *advances in neural information processing systems*, 34:17555–17566, 2021.

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. *arXiv preprint arXiv:1701.06538*, 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.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. Openmoe: An early effort on open mixture-of-experts language models. *arXiv preprint arXiv:2402.01739*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

A APPENDIX

A.1 PUBLIC PRETRAINED BASELINES.

We compare against public pretrained models Pythia-410M, Pythia-2.8B (Biderman et al., 2023), TinyLlama-1.1B (Zhang et al., 2024), and OLMoE-1B-7B (Muennighoff et al., 2025). Their total/active parameter counts are 405M/405M, 2.8B/2.8B, 1.1B/1.1B, and 6.9B/1.3B, respectively.

A.2 CONTROLLED MODEL CONFIGURATIONS

All controlled models use the same 16-layer Transformer backbone and the same pretraining data, tokenizer, optimization recipe, batch configuration, learning-rate schedule, load-balancing settings, and approximately 100B-token training budget. Each MoE layer contains 192 routed experts and one shared expert, with $K = 6$ routed experts activated per token. The controlled variants differ only in the treatment of the first three layers and in whether expert identities are selected by hashing or by a learned top-K router.

Table 3: Controlled model configurations. "Learned" denotes standard learned top-K expert selection. Hash schedules specify the n-gram order used in the first three MoE layers.

Model	Layers 1–3	Layers 4–16	Total Params	Active Params	Main purpose
Dense-First-1	Dense, Learned, Learned	Learned MoE	4.8B	410M	One-layer dense replacement
Dense-First-3	Dense, Dense, Dense	Learned MoE	4.2B	407M	Three-layer dense replacement
Learned-MoE	Learned MoE	Learned MoE	5B	412M	Fully learned-routing MoE
PROGRAM$_{123}$	1-, 2-, 3-gram hash	Learned MoE	5B	412M	Default progressive schedule
PROGRAM$_{234}$	2-, 3-, 4-gram hash	Learned MoE	5B	412M	Context-heavier progression
Fixed n-gram	n, n, n, $n \in \{1, 2, 3, 4\}$	Learned MoE	5B	412M	No progression

A.3 EVALUATION ENVIRONMENT

We evaluate Hugging Face-format checkpoints with `lm-evaluation-harness` and the vLLM backend (vLLM 0.13). Unless otherwise stated, evaluation uses tensor parallelism of 8, `dtype=auto`, `gpu_memory_utilization=0.8`, `enforce_eager=true`, a maximum context length of 4096, and automatic batch-size selection. The default random seed is 1234. MGSM-Zh is additionally evaluated with seeds 42, 2025, and 1024, and its reported score is averaged over the four seeds. We evaluate the final approximately 100B-token checkpoint after conversion to Hugging Face format.

No chat template is applied. Sampling hyperparameters such as temperature and top-p are not overridden by the launch script; decoding therefore follows the corresponding task configuration in the evaluation harness. For the knowledge and code tasks, the generation budget is explicitly capped at 1024 new tokens. The harness logs per-example outputs for later inspection. Dataset access is configured in offline mode, and executable code evaluation is enabled for the coding benchmarks.

A.4 BENCHMARK-SPECIFIC PROTOCOLS

Table 4 summarizes the task identifiers and few-shot settings used by the evaluation script. Metrics follow the definitions implemented by the corresponding task configurations in `lm-evaluation-harness`.

For MLQA-Zh-Zh, the evaluator produces both exact match and token-level F1. We report exact match multiplied by 100 in the main comparison tables and retain F1 as an auxiliary metric. For RepoBench, the evaluator produces both exact match and edit similarity. We report exact match multiplied by 100 in the main tables and retain edit similarity as an auxiliary metric.

A.5 EXECUTION AND REPRODUCIBILITY NOTES

The evaluation script sets `HF_DATASETS_OFFLINE=1`, enables executable-code evaluation, and stores both aggregate results and logged samples under a model-specific output directory. The same checkpoint path and evaluation command structure are used for all controlled models. To avoid

Table 4: Evaluation protocols. "Default" means that the generation limit is inherited from the task configuration because the launch script does not override it. MLQA-Zh-Zh and RepoBench report exact match in the main tables, with scores multiplied by 100.

Benchmark	Harness task	Shots	Reported metric	Max new tokens	Seeds
MMLU-Pro	`mmlu_pro`	5	Accuracy	1024	1234
MMLU-Redux	`mmlu_redux_generative`	5	Generative exact-match accuracy	1024	1234
MLQA-Zh-Zh	`mlqa_zh_zh`	5	Exact match (EM, $\times 100$)	1024	1234
INCLUDE	`include_few_shot_og`	5	Accuracy	1024	1234
GPQA (main)	`leaderboard_gpqa_main`	5	Accuracy	1024	1234
GSM8K	`gsm8k`	8	Exact match	Default	1234
MGSM-Zh	`mgsm_native_cot_zh`	8	Exact match	Default	1234, 42, 2025, 1024
BBH	`bbh_cot_fewshot`	3	Exact match	Default	1234
MBPP	`mbpp`	3	Pass@1 by execution	1024	1234
MBPP+	`mbpp_plus`	3	Pass@1 by execution	1024	1234
HumanEval	`humaneval`	0	Pass@1 by execution	1024	1234
RepoBench	`repobench`	3	Exact match (EM, $\times 100$)	1024	1234

configuration drift, the final release will provide the exact harness revision, task YAML files, model-conversion script, and raw result JSON files.

All evaluation loops use tensor parallelism of 8 and the same model-loading arguments. The released script will define all path and parallelism variables explicitly so that the commands can be executed without relying on shell-state variables.

B Computational Complexity

Notation. Let $N = B \cdot S$ be the number of tokens processed by a MoE layer in one forward pass (B batch size, S sequence length), E the number of routed experts, and K the number of routed experts selected per token. ProGram generates M hash candidates per token, where

$$M = \min\big(\max(2K,\, K{+}6),\, |\mathcal{P}_{\text{cand}}|\big);$$

with $K{=}6$ and $|\mathcal{P}_{\text{cand}}|{=}16$ this gives $M{=}12$. The n-gram order is bounded by $n \le n_{\max}{=}4$.

Per-token routing cost. For each token the router performs: (i) building the n-gram context by $n{-}1$ tensor shifts, $O(n)$; (ii) generating M candidate expert ids, each a single modular multiply–add over integer token ids, $O(M{+}n)$; (iii) a stable, order-preserving deduplication implemented as a pairwise comparison over the M candidates, $O(M^2)$; and (iv) a fallback linear probe with stride $d{=}131$ (chosen coprime to E, so it enumerates distinct experts) that is triggered only when fewer than K distinct candidates remain, $O(K)$ and empirically almost never invoked. The router scores used to weight the selected experts are the same softmax logits as in a learned router. Aggregating over all tokens, the routing overhead is

$$O\big(N\,(M^2 + n)\big) \;=\; O(N),$$

since M and n are small constants. This matches the asymptotic class of a naive single-/multi-hash router ($O(NK)$) and is no larger than the $O(NE)$ score-and-top-K selection of a learned router.

Negligible relative to expert compute. The extra work is a few hundred integer operations per token ($M^2{+}n \approx 150$), whereas the expert FFNs alone cost $6\,d\,H_e\,K \approx 1.9 \times 10^7$ floating-point operations per token (forward pass; $d{=}1024$, $H_e{=}512$, $K{=}6$), not counting attention. The routing overhead is therefore on the order of 10^{-6} of the layer's compute and is not measurable in end-to-end wall-clock time.

No extra parameters, communication, or storage. Hash routing is parameter-free: the router weights are identical to a learned MoE, so ProGram adds no trainable parameters. Because exactly K real experts are selected per token, the all-to-all dispatch/combine volume is identical to a naive hash router and to a learned top-K router; ProGram introduces no additional communication. Candidate ids are computed on the fly, so no token$\to$expert lookup table is materialized (avoiding the $O(V)$ or $O(VK)$ storage used by some hash-layer implementations, where V is the vocabulary size). The only auxiliary memory is the transient $[N, M, M]$ boolean tensor used for deduplication, i.e. $O(NM^2)$ bytes that are freed immediately; for $M{=}12$ this is a few MB per layer and can be reduced to $O(NM \log M)$ with a sort-based deduplication if desired.

C WHY CONTEXTUAL HASHING IMPROVES LOAD BALANCE

This section explains why replacing token-ID routing atoms with contextual n-grams can improve expert-load balance. The argument has two parts. First, contextual refinement deterministically reduces the concentration of the routing-key distribution. Second, under a sufficiently dispersed hash mapping, the expected deviation of expert loads from the uniform target is directly controlled by this concentration.

C.1 THE UNIGRAM LOAD BOTTLENECK

Let $\mathcal{G}_n$ denote the set of observed n-gram routing atoms, with probability mass function

$$p_n(g) = \Pr(G_n = g), \qquad g \in \mathcal{G}_n. \tag{19}$$

Suppose each routing atom activates a set $\mathcal{H}(g)$ of K distinct experts among E experts. Define the normalized load of expert e as

$$L_e^{(n)} = \frac{1}{K} \sum_{g \in \mathcal{G}_n} p_n(g)\, \mathbf{1}\{e \in \mathcal{H}(g)\}\,. \tag{20}$$

Since $\sum_e L_e^{(n)} = 1$, perfectly balanced routing corresponds to $L_e^{(n)} = 1/E$ for every expert.

Proposition 1. For deterministic token-ID routing,

$$\max_e L_e^{(1)} \geq \frac{p_{\max}^{(1)}}{K}, \qquad p_{\max}^{(1)} = \max_t p_1(t). \tag{21}$$

Hence, if $p_{\max}^{(1)} > K/E$, perfect balance is impossible for any deterministic token-to-expert assignment.

Proof. Let $t^\star$ satisfy $p_1(t^\star) = p_{\max}^{(1)}$. Every occurrence of $t^\star$ activates the same K experts, so each expert in $\mathcal{H}(t^\star)$ receives at least $p_{\max}^{(1)}/K$ normalized traffic. If this quantity exceeds $1/E$, at least one expert must exceed the uniform target. □

This formalizes the heavy-frequency bottleneck of token-ID hashing: no fixed token-to-expert table can divide the traffic of one token among more than its predetermined K experts.

C.2 CONTEXT REFINEMENT REDUCES KEY CONCENTRATION

Define

$$p_{\max}^{(n)} = \max_{g \in \mathcal{G}_n} p_n(g), \qquad S_2^{(n)} = \sum_{g \in \mathcal{G}_n} p_n(g)^2. \tag{22}$$

An n-gram atom g is refined into $(n+1)$-gram atoms (x, g):

$$p_n(g) = \sum_x p_{n+1}(x, g). \tag{23}$$

Proposition 2. Increasing the routing order cannot increase either concentration measure:

$$p_{\max}^{(n+1)} \leq p_{\max}^{(n)}, \tag{24}$$

$$S_2^{(n+1)} \leq S_2^{(n)}. \tag{25}$$

Proof. For every refinement (x, g),

$$p_{n+1}(x, g) \leq \sum_{x'} p_{n+1}(x', g) = p_n(g), \tag{26}$$

which proves Eq. 24. Moreover, for each fixed g,

$$\sum_x p_{n+1}(x, g)^2 \leq \left(\sum_x p_{n+1}(x, g) \right)^2 = p_n(g)^2. \tag{27}$$

Summing over all g proves Eq. 25. □

The second inequality is strict whenever at least one routing atom appears under multiple positive-probability contexts. Thus, frequent tokens occurring in diverse contexts are split into several lower-mass routing atoms instead of remaining one indivisible traffic source.

C.3 Expected Expert-Load Deviation

Consider an idealized dispersed-hashing model in which each routing atom is assigned independently to a uniformly sampled K-element subset of the E experts. This model isolates the effect of routing-key concentration.

Proposition 3. For every expert e,

$$\mathrm{E}\left[L_e^{(n)}\right] = \frac{1}{E}, \tag{28}$$

and

$$\mathrm{E}\left[\sum_{e=1}^{E}\left(L_e^{(n)} - \frac{1}{E}\right)^2\right] = \frac{1 - K/E}{K}\, S_2^{(n)}. \tag{29}$$

Proof. Let

$$I_{g,e} = \mathbf{1}\{e \in \mathcal{H}(g)\}\,. \tag{30}$$

For a uniformly sampled K-element subset,

$$\mathrm{E}[I_{g,e}] = \frac{K}{E}, \qquad \mathrm{Var}(I_{g,e}) = \frac{K}{E}\left(1 - \frac{K}{E}\right). \tag{31}$$

Substituting into Eq. 20 gives $\mathrm{E}[L_e^{(n)}] = 1/E$. Assuming independent assignments for distinct routing atoms,

$$\mathrm{Var}(L_e^{(n)}) = \frac{1 - K/E}{KE}\, S_2^{(n)}. \tag{32}$$

Summing over all E experts yields Eq. 29. □

Combining Propositions 2 and 3 gives

$$\mathrm{E}\left[\sum_{e}\left(L_e^{(n+1)} - \frac{1}{E}\right)^2\right] \le \mathrm{E}\left[\sum_{e}\left(L_e^{(n)} - \frac{1}{E}\right)^2\right]. \tag{33}$$

Thus, under dispersed hashing, increasing the routing order cannot increase expected squared load imbalance. Chebyshev's inequality and a union bound further give

$$\Pr\left(\max_e \left|L_e^{(n)} - \frac{1}{E}\right| \ge \epsilon\right) \le \frac{(1 - K/E)S_2^{(n)}}{K\epsilon^2}. \tag{34}$$

Contextual refinement lowers $S_2^{(n)}$ and therefore tightens this upper bound on large expert-load deviations.

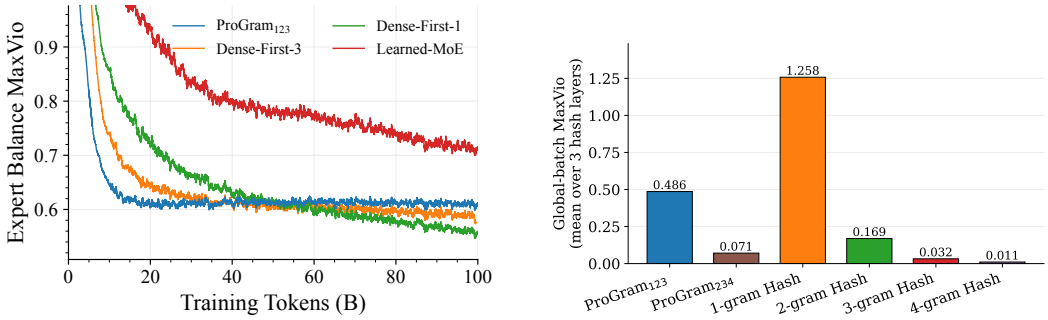
C.4 Candidate Collisions and Multi-Prime Hashing

The preceding analysis concerns the distribution of routing atoms across experts. A separate issue is whether two candidate hash streams produce the same expert for one atom. Under the idealized assumption that M candidate IDs are independent and uniform over E experts,

$$\Pr(c_j = c_{j'}) = \frac{1}{E}, \qquad j \ne j'. \tag{35}$$

If

$$Z = \sum_{j<j'} \mathbf{1}\{c_j = c_{j'}\} \tag{36}$$



(a) Training dynamics of controlled baselines

(b) Offline evaluation of hash schedules

Figure 4: Expert-balance MaxVio. (a) Per-micro-batch MaxVio throughout 100B-token pretraining, averaged over MoE layers and logging intervals. (b) Global-batch MaxVio averaged over the first three hash-routing layers. Lower values indicate more uniform expert utilization.

is the number of colliding candidate pairs, then

$$\mathrm{E}[Z] = \frac{\binom{M}{2}}{E}, \tag{37}$$

and

$$\Pr(Z \geq 1) \leq \frac{\binom{M}{2}}{E}. \tag{38}$$

Hence, raw pairwise collision probability is small when the number of candidate streams is modest relative to the expert count. Moreover, PROGRAM does not require collision-free raw candidates: stable deduplication removes repeated IDs, and fallback probing guarantees exactly K distinct experts.

For the fixed multi-prime construction, the conditions

$$\gcd(P_j, E) = \gcd(Q_k, E) = 1 \tag{39}$$

make multiplication by each coefficient a permutation of $\mathbb{Z}_E$, so each individual stream can reach every expert ID as the corresponding residue varies. Requiring $P_j \not\equiv P_{j'} \pmod{E}$ ensures non-identical modular mappings. These conditions establish full modular reachability and mapping diversity, but not statistical independence by themselves; the latter also depends on the empirical distribution of n-gram residues. Therefore, Eqs. 29–38 provide a probabilistic explanation of the observed behavior rather than a deterministic guarantee for every corpus.

D ADDITIONAL EXPERT-BALANCE ANALYSIS

D.1 EXPERT-BALANCE MAXVIO ANALYSIS

We report the *expert-balance maximum violation* (MaxVio) to measure the degree of expert overloading. Let L_e denote the number of routed assignments received by expert e, and let $\bar{L} = \frac{1}{E}\sum_{e=1}^{E} L_e$ be the mean expert load. MaxVio is defined as

$$\mathrm{MaxVio} = \max_e \left(\frac{L_e}{\bar{L}} - 1\right) = \frac{\max_e L_e - \bar{L}}{\bar{L}}. \tag{40}$$

A value of zero indicates perfectly uniform utilization, while a value of 0.5, for example, means that the most heavily loaded expert receives 50% more assignments than the average expert.

Figure 4(a) reports the per-micro-batch MaxVio throughout pretraining. For each MoE layer, expert-assignment counts are aggregated across the parallel group, MaxVio is computed for the current micro-batch, and the logged value is averaged over MoE layers and logging steps. The results show that PROGRAM$_{123}$ rapidly reduces the initial assignment skew and reaches a stable level after

approximately 15B training tokens. It remains substantially better balanced than LEARNED-MOE, whose MaxVio remains above 0.7 at the end of training.

The Dense-First baselines continue to reduce MaxVio during later training and eventually become slightly lower than PROGRAM$_{123}$. This comparison, however, involves different sets of MoE layers. Dense-First-1 and Dense-First-3 remove one and three shallow MoE layers, respectively, and therefore exclude the layers closest to the embedding space, where token distributions are typically the most concentrated. Their remaining learned routers can also continue adapting under the load-balancing objective, whereas the expert identities in the hash-routed layers of PROGRAM$_{123}$ are fixed. The late-stage MaxVio advantage of the Dense-First variants should therefore not be interpreted as evidence that replacing shallow MoE layers with dense layers produces better routing behavior under an otherwise identical layer set.

Figure 4(b) provides a complementary global-batch measurement at the 100B-token checkpoint. Here, expert-assignment counts are accumulated over all micro-batches within a global batch before MaxVio is computed, and the result is averaged over the first three hash-routed layers. The fixed-order variants exhibit a clear monotonic trend: MaxVio decreases from 1.258 for Token-ID Hash to 0.169, 0.032, and 0.011 for the fixed 2-, 3-, and 4-gram variants, respectively. This result is consistent with the routing-key concentration analysis: increasing the n-gram order partitions frequent tokens into a larger number of lower-probability contextual keys and thereby produces more dispersed expert assignments.

The progressive schedules lie between the corresponding fixed-order extremes. PROGRAM$_{123}$ obtains a global-batch MaxVio of 0.486, substantially improving over Token-ID Hash but remaining less uniform than the fixed higher-order variants. PROGRAM$_{234}$ further reduces MaxVio to 0.071 because it uses contextual routing from the first hash layer and therefore avoids the concentrated unigram assignment present in PROGRAM$_{123}$.

Importantly, lower MaxVio does not necessarily imply better language modeling or downstream performance. Exact load uniformity is an operational objective for preventing expert collapse and computational hot spots, but it is neither necessary nor sufficient for effective expert specialization. Tokens and local contexts are not semantically interchangeable, and forcing all experts to receive identical traffic can remove useful structure from the routing assignment. PROGRAM$_{123}$ therefore does not optimize load uniformity alone. Instead, it retains a lexical routing anchor in the first layer and progressively introduces richer contextual keys in subsequent layers, trading a moderate degree of load concentration for a depth-aligned transition from lexical identity to contextual specialization. The MaxVio results should thus be interpreted jointly with language-modeling loss and downstream performance rather than as a standalone model-selection criterion.