THE PREFETCHABLE STORE: DECOUPLING MODEL CAPACITY FROM FAST MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

The largest model a machine can run at full speed is the largest one its fast memory holds. We argue that this is an addressing limit rather than a storage limit: a transformer chooses the parameters a token needs in the middle of its forward pass, so all of them must be resident. The **prefetchable store** is a storage-native architecture in which most of a model's trained parameters live on a commodity SSD as expert blocks whose addresses are known from the token id before any layer runs, so they stream under the model's own early layers. We state three requirements under which storage can hold trained capacity: the address must be known early enough to hide the transfer, the transfer must be shaped for the storage tier, and every address must receive enough training data to learn. We build a serving system that meets them and reproduces the resident model exactly through the disk path at equal precision, and we train models under the constraint from the first step against a dense frontier retrained at every budget under one recipe. Up to 8×10^9 training tokens, across three backbones and four store sizes, stored capacity lowers loss at every budget: a 220M backbone with a 3B-parameter store matches a fully resident 1B dense model under the same training budget while, served from NVMe, decoding faster in a third of the GPU memory. The price is total parameters, comparable to the overhead sparse models already pay, and it is paid on a drive rather than in fast memory. We further show that two ingredients behind the efficiency of frontier mixtures of experts, learned contextual selection and the combination of several fetched experts, work inside the prefetchable constraint. Extrapolating the measured scaling, a 16B resident backbone on a 24 GB consumer GPU with a 1T-parameter store on one 1 TB drive has the potential to match a dense model of several hundred billion parameters.

1 INTRODUCTION

Every deployable language model is sized by the fast memory that holds it. A frontier mixture of experts such as DeepSeek-V3 (DeepSeek-AI, 2024) activates 37B of its 671B parameters per token, yet all 671B must be resident. Because a transformer chooses the parameters a token will use from hidden states it computes during the forward pass, by the time the model knows which expert it needs, it needs it immediately. Storage is three orders of magnitude cheaper per byte and effectively unbounded, yet it contributes far less to model capacity than memory does. Offloading systems work around this by streaming weights on the critical path and paying in speed (Sheng et al., 2023; Alizadeh et al., 2024; Song et al., 2024b; Eliseev & Mazur, 2023), or by moving or predicting a router's decision ahead of the layer that consumes it (Hwang et al., 2024; Song et al., 2024a; Xue et al., 2024); most recently, Edge0 serves a 35B mixture of experts from SSD at about 20 tokens/s on a 24 GB machine by replacing its router with a cross-token prerouter and repairing the damage with recovery training (Lin et al., 2026). But these systems only mitigate the bottleneck, adapting a model trained for full residency to a fetch it never planned for. We address the root instead: how should a model allocate and learn its capacity when fast memory is a constraint from the start?

The motivation starts from two recent lines of work, applied to our storage use case. Hash Layers (Roller et al., 2021) showed that an expert selected by a fixed hash of the input token is competitive with learned routing; Engram (Cheng et al., 2026) showed that a large fraction of a model's parameters can live in a table addressed by surface n-grams. Together they imply that *an address computable from the surface tokens is computable at $t{=}0$, before any layer runs*. This creates a

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.

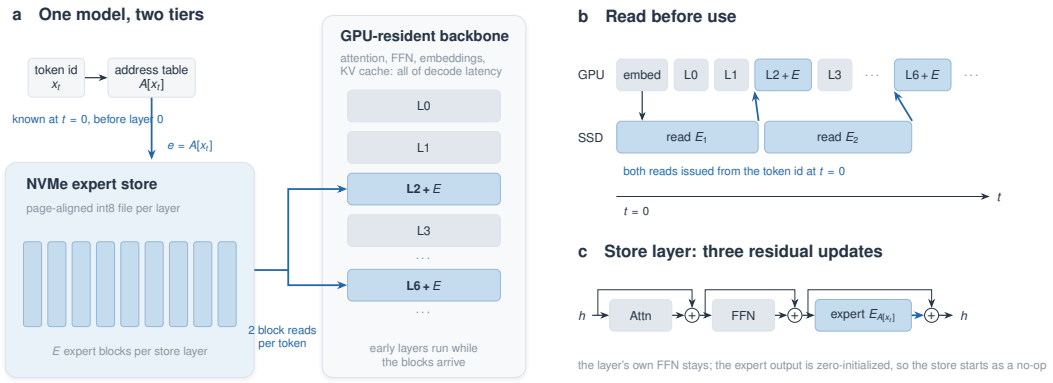

Figure 1: Capacity on disk. (a) A GPU-resident backbone accesses expert blocks on NVMe, with one expert per store layer selected by token id. (b) Both reads begin at t=0 and overlap early backbone computation, making the experts available before use. (c) Each store layer applies three sequential residual updates: attention, the backbone FFN and the fetched expert (Equation 1).

window in which the parameters can be fetched concurrently with computation. If the transfer completes within that window, storage latency is hidden from the decoding critical path, making secondary storage a viable tier for model capacity rather than a repository for cached state.

Motivated by this observation, we develop the **prefetchable store** (Figure 1), a storage-native architecture combining a small resident backbone with a large pool of token-addressed expert blocks on NVMe. The model learns to use this external capacity from initialization, while the serving system fetches each token's experts under the early layers of the same forward pass. We identify three requirements for storage to hold useful capacity, address lead time, tier-matched transfer granularity and per-address training viability, supported by analysis and measurements (§3). Our architecture and serving system meet these requirements, with bitwise parity through the disk path at equal precision and measured transfer overlap consistent with an analytic model (§4). Across three backbones, four store sizes and four token budgets, we evaluate quality and cost against a dense frontier retrained under the same recipe (§5.1–5.2): a 220M backbone with a 3B store matches a fully resident 1B dense model at a fifth of the residency, and decodes faster from NVMe in roughly a third of the GPU memory. Its total-parameter overhead is comparable to that reported for sparse models, and, unlike theirs, the additional capacity resides on a drive. We then adapt two ingredients of frontier MoE efficiency, contextual selection and multi-expert combination, to work under early addressing (§6.1). Their successful transfer shows that the baseline does not exhaust the architecture's potential; together with our measured scaling, these results motivate a projection of frontier-class capacity behind a single consumer GPU (§6.2). Our contributions are:

1. requirements for storage-resident capacity, and an architecture and serving system that meet them;
2. a systematic demonstration of capacity scaling beyond GPU residency, with measured quality, serving benefits and parameter overhead;
3. evidence of architectural headroom through frontier MoE ingredients, and a deployment scenario connecting the measured scaling to frontier-class capacity.

2 Related Work

Our work draws on sparse-model scaling, surface-addressed capacity and parameter offloading, bringing them together around the constraints of storage-resident computation.

Scaling laws. Dense scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022) motivate our reference models, retrained at every token budget. Routed-model and sparsity laws (Clark et al., 2022; Krajewski et al., 2024; Ludziejewski et al., 2025; Frantar et al., 2024), together with matched-data MoE comparisons (Dai et al., 2024; Tian et al., 2025; Yang et al., 2025), contextualize the trade between stored capacity, active computation and training data. **Sparse and surface-addressed capacity.** Conventional MoEs (Shazeer et al., 2017; Fedus et al., 2022; Dai et al., 2024) and learned memory layers (Lample et al., 2019; Berges et al., 2025; He, 2024; Huang et al., 2025) select capacity

from intermediate hidden states. Hash Layers (Roller et al., 2021) demonstrates token-id selection of executable experts, while Engram and its successors (Cheng et al., 2026; Chen et al., 2026; Ma et al., 2026) demonstrate surface-addressed lookup capacity; DeepSeek-V4 also uses token-id gates in early MoE layers (DeepSeek-AI, 2026). We turn these addressing principles into an architecture for storage-resident computation: balanced token assignments select residual expert blocks before the forward pass, jointly providing fetch lead time, efficient transfer granularity and consolidated training exposure. **Offloading and prefetch.** FlexGen, LLM-in-a-flash and PowerInfer (Sheng et al., 2023; Alizadeh et al., 2024; Song et al., 2024b) manage transfer costs through scheduling, reuse or selective execution; Pre-gated MoE, ProMoE and MoE-Infinity (Hwang et al., 2024; Song et al., 2024a; Xue et al., 2024) advance or predict expert selection to create overlap. Edge0 (Lin et al., 2026) replaces a pretrained model's routing with a cross-token prerouter and restores quality through recovery training. We eliminate the late-address dependency by construction during training, allowing transfers to leave the critical path when they fit the compute window. Our focus is therefore the allocation and learning of non-resident capacity from initialization, fundamentally removing the late-address bottleneck existing offloading research has been working around.

3 Requirements for Stored Capacity

Turning storage into model capacity requires more than moving weights off the GPU. We find that a stored parameter's address must be known early enough to overlap its transfer with computation, its transfer must be shaped for the bandwidth of the storage tier rather than its IOPS, and its address must not split the training data so finely that the parameter never learns. We formulate these as three conditions (Figure 2), each supported by a measurement that guides the architecture.

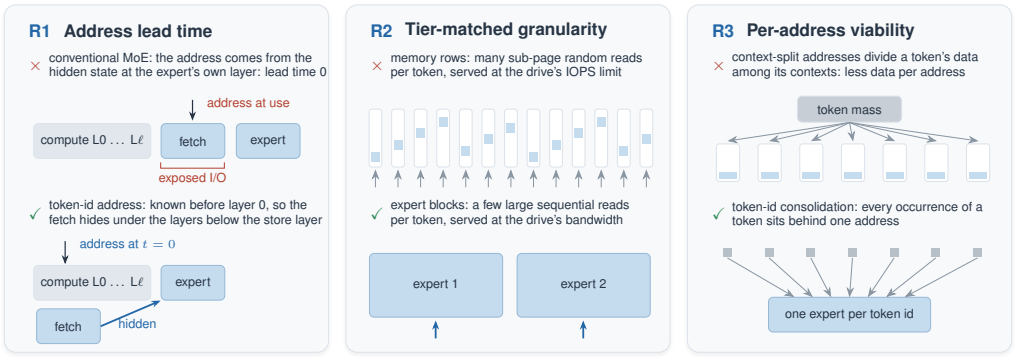

Figure 2: Three requirements. Each panel contrasts an access pattern that violates the requirement (×) with the corresponding design choice in the prefetchable store (✓).

R1: address lead time. *The address of every non-resident parameter must be known early enough for its fetch to complete under computation before it is used.* With one transfer per store layer the exposed stall is $\max(0,\ \text{fetch} - \text{window})$. Conventional mixture-of-experts routing (Shazeer et al., 2017; Fedus et al., 2022) computes the address from the hidden state at the expert's own layer, where the window is zero and the expert must already be in fast memory, which is why MoE offloading has been a prediction problem. Token-id addressing makes every expert address available before layer 0, allowing fetches to overlap the early layers during decode and to be scheduled across the known prompt during prefill. This lead time is not limited to static addressing: in §6.1, we train a learned router that selects experts using only information available before the forward pass begins.

R2: tier-matched transfer granularity. *The unit of transfer must allow the storage tier to operate near its bandwidth limit rather than its IOPS limit.* Engram-style memory rows (Cheng et al., 2026) satisfy **R1** but require 16 head-slot reads of ∼320 B at each of two layers: 32 scattered sub-page reads per token, an access pattern poorly suited to NVMe. Our trace study shows limited benefit from caching, with an LRU hit rate of $\leq 53\%$ at 10% of the footprint. Expert blocks replace these scattered accesses with a few multi-megabyte sequential reads per token, which the same device serves at 6.6 GB/s. Per-token traffic is then determined by the number and size of activated blocks, allowing the pool to grow by adding blocks without increasing traffic (§6.2).

R3: per-address training viability. *Every address must receive enough training tokens for the parameters behind it to learn.* Conditioning a token's address on context splits its training exposure across addresses. In our probes, the loss increase when disabling the store falls from 2.83 nats with token-id consolidation to 1.52 with a floor of 127k tokens per address, and to 0.65 at approximately 15k tokens per address (Appendix A). Balancing traffic across experts does not ensure sufficient exposure for each address. These results support a training-exposure explanation for Hash Layers' finding that bigram hashing underperforms unigram hashing (Roller et al., 2021, §5.3). Consistent with this requirement, every store's gain over its backbone increases with training data (§5.2).

4 Architecture and Serving

We now describe the architecture and serving stack built around **R1**–**R3**, then verify disk-path parity at equal precision and test the transfer-overlap model of **R1**.

Forward pass. The backbone is a standard pre-norm decoder (RMSNorm (Zhang & Sennrich, 2019), grouped-query attention (Ainslie et al., 2023), SwiGLU FFN (Shazeer, 2020)). At two layers, $\ell \in \{\lfloor 0.2L \rfloor, \lfloor 0.6L \rfloor\}$, fetched experts are added to the residual stream beside the layer's own FFN, which remains intact:

$$h \leftarrow h + \mathrm{Attn}(h); \qquad h \leftarrow h + \mathrm{FFN}(h); \qquad h \leftarrow h + \textstyle\sum_{j=1}^{k} w_j \, E_{a_j(x_t)}\big(\mathrm{RMSNorm}(h)\big), \quad (1)$$

where x_t is the token id at position t, each $a_j : \mathcal{V} \to \{1..E\}$ is a fixed lookup table, and E_e is a SwiGLU block with $d_{\text{ff}} = 853$ (1.3M parameters at the 27.7M backbone). The baseline fetches one expert per store layer (k=1, w_1=1); the extensions of §6 change how the a_j are chosen and how the k outputs are combined. Experts are zero-initialized on their output projection, so the store enters training as an exact no-op.

Balanced assignment. Each assignment table is built offline from training-corpus token counts. We assign token ids in decreasing frequency to the currently least-loaded expert, using longest-processing-time scheduling, whose maximum load is within $4/3$ of optimal (Graham, 1969); a load certificate is checked at every launch. The tables remain fixed during training and inference, requiring only a lookup at serving time. Every occurrence of a token selects the same expert at each store layer, consolidating its training exposure as motivated by **R3**.

Serving stack. Expert weights are stored in a page-aligned int8 file per store layer, with per-column absmax scales; a bf16 format supports parity checks. Reads use `O_DIRECT`, bypassing the page cache so that the measurements reflect device reads rather than page-cache hits. As soon as x_t is known, the prefetcher issues both expert reads, dequantizes the weights on a dedicated CUDA stream, and signals their availability to the store layers through an event. Decode uses a KV-cached session that also retains the previous-position state required by the learned router (§6.1).

We validate numerical correctness and transfer overlap. The bf16 disk path produces logits bitwise identical to the resident path; int8 quantization introduces a maximum logit difference of 0.023 in this parity test. Its effect on held-out loss is evaluated separately through the served weights (§5.2). Incremental decode matches the full forward pass within 3×10^{-6} for hash addressing and 2×10^{-5} for learned routing. The int8 synchronous-fetch ablation slows decode by $1.29\times$ relative to prefetched int8 execution, within the overlap model's predicted 1.25–$1.36\times$ range (Appendix E). Even the 27.7M backbone, which provides the shortest overlap window among the backbones tested, retains 0.881 of resident decode speed with the entire store on disk. Holding the measured $\sim$2 ms fetch and the model's other assumptions fixed, the overlap model predicts retention above 0.95 for backbone steps of at least 8.4 ms, a duration exceeded by the 220M backbone (§5.2).

5 Experiments

We evaluate how storage-resident capacity scales with store size, training tokens and backbone size under a shared training protocol. We then compare its parameter overhead with the sparse-model literature and measure the headline configuration's end-to-end serving performance.

5.1 SETUP

Data and training. All models train on FineWeb-Edu (sample-10BT; Penedo et al., 2024) withthe Llama-2 tokenizer (Touvron et al., 2023), a 32k vocabulary and a 2048-token context. Training budgets are 1, 2, 4 and 8×10^9 tokens, with larger datasets containing the smaller ones as exact prefixes. At each budget, models share the same recipe and batch order, isolating architectural differences (Appendix A).

Evaluation. Our primary quality measure is held-out loss on a fixed 0.5M-token slice, with the store enabled. Zero-shot LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019) and ARC-easy (Clark et al., 2018), evaluated through a shared harness adapter (Gao et al., 2023), test whether these comparisons extend to downstream tasks. The held-out data is a sequence-level split of the training stream; 1.6% of its 64-token windows recur verbatim in the 10^9-token training split. Models are compared at matched training exposure.

Dense reference and parameter overhead. We train dense models with 27.7M, 100.5M, 219.5M, 405.6M and 980.7M non-embedding parameters under the same recipe at every budget. Main comparisons use their measured losses directly. To quantify parameter efficiency, we also estimate the *dense-equivalent* size N_{eq} by fitting $L(N) = A + B\, e^{-c \ln(N/N_0)}$ to the five dense models at each budget and finding the size that matches a store model's loss. Its *overhead*, $H = P/N_{\text{eq}}$, is the ratio of total non-embedding parameters P to dense-equivalent parameters, enabling comparison with reported sparse-model overheads. Appendix B gives the equivalent exponent form used in §6.2. Backbone sizes exclude embeddings; GPU memory figures in §5.2 are measured. Quality evaluations use unquantized weights in fp32 unless identified as served-weight evaluations; serving uses bf16 computation with experts stored in int8.

Seeds and interpretation. 33 configurations have a second initialization seed (Appendix D), with loss differences of 0.000–0.028 nats for dense models, 0.001–0.013 for stores and 0.008–0.011 for the previous-step router. We emphasize differences of at least 0.01 nats or consistent trends across configurations, rather than smaller isolated margins. Cross-platform repeats of the same seed agree within 0.002 nats for store models and 0.007 for dense-27.7M.

5.2 MAIN RESULTS

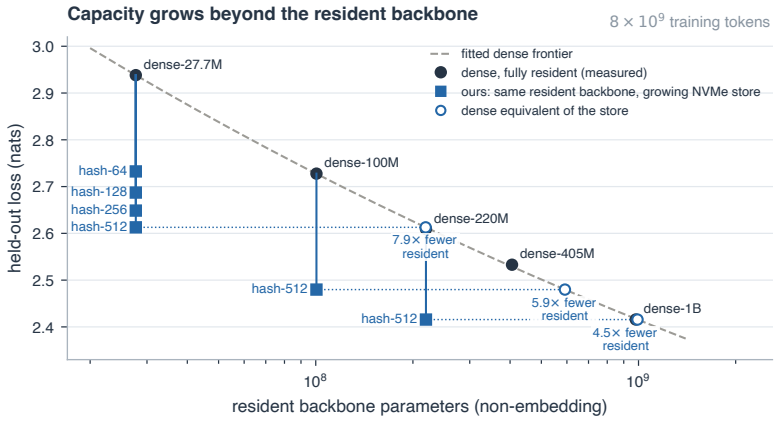

Figure 3: Capacity beyond GPU memory. Hollow markers indicate dense-equivalent sizes on the fitted frontier: the 27.7M, 100M and 220M backbones match the loss of dense models with 7.9×, 5.9× and 4.5× as many backbone parameters. All models train on 8×10^9 tokens under the same recipe.

Figure 3 summarizes the capacity result at 8×10^9 tokens and Figure 4 traces it along three axes. **The store axis.** Doubling the number of experts lowers loss at every budget. The average gain per doubling increases from 0.034 nats at 10^9 tokens to 0.040 at 8×10^9, widening the gap between the 64- and 512-expert stores from 0.099 to 0.120 nats. Token-id addressing also outperforms learned Switch top-1 routing over the same 512 experts by 0.098 and 0.075 nats at these budgets (Table 1).

model	backbone	active	total	held-out loss 10^9	held-out loss 8×10^9	zero-shot accuracy at 10^9 (%) LAMBADA	PIQA	HellaSwag	ARC-e
dense-27.7M	27.7M	27.7M	27.7M	3.2020	2.9382	15.4	57.6	27.2	42.0
dense-100M	100M	100M	100M	3.0239	2.7277	17.4	59.7	27.4	44.9
dense-220M	220M	220M	220M	2.9229	2.6092	21.0	60.0	28.1	46.4
dense-405M	406M	406M	406M	2.8714	2.5329	21.3	60.9	28.3	47.8
dense-1B	981M	981M	981M	2.7754	2.4158	22.6	62.1	29.3	50.0
27.7M + hash-64	27.7M	30.3M	195M	3.0162	2.7324	17.3	59.7	27.5	45.8
27.7M + hash-128	27.7M	30.3M	363M	2.9804	2.6872	17.1	59.2	27.8	44.9
27.7M + hash-256	27.7M	30.3M	699M	2.9429	2.6487	17.6	59.8	28.0	46.5
27.7M + hash-512	27.7M	30.3M	1.37B	**2.9170**	**2.6129**	18.4	61.0	28.2	45.0
27.7M + Switch MoE-512 (control)	27.7M	30.3M	1.37B	3.0155	2.6879	17.1	58.8	27.6	45.7
100M + hash-512	100M	105M	2.45B	**2.8037**	**2.4798**	20.8	62.2	28.9	48.8
220M + hash-512	219M	225M	3.24B	**2.7596**	**2.4157**	23.5	62.5	29.0	50.0

Table 1: Main comparisons. Counts exclude embeddings; active parameters include the backbone and experts executed per token. Held-out loss is at 10^9 and 8×10^9 tokens, and zero-shot accuracy (%) at 10^9. The resident Switch control uses learned top-1 routing over the same 512 experts per store layer.

The token axis. Across all four budgets, the 512-expert store on the 27.7M backbone remains within 0.006 nats of dense-220M, using approximately 30M active non-embedding parameters rather than 220M. Its loss reduction over its own backbone grows from 0.285 to 0.325 nats as training increases from 10^9 to 8×10^9 tokens. The advantage persists as both models train longer.

The backbone axis. The 100M backbone with a store outperforms dense-405M by 0.053–0.068 nats at every budget. The 220M configuration achieves lower loss than dense-1B at 10^9 tokens (2.7596 vs. 2.7754) and matches it at 8×10^9 (2.4157 vs. 2.4158), with 220M resident parameters rather than 981M. Its task scores are within one percentage point at 10^9 tokens (Table 1; all budgets in Appendix G). These results extend the capacity advantage to larger backbones, although backbone width, expert width and expansion ratio vary together rather than defining a self-similar scaling law.

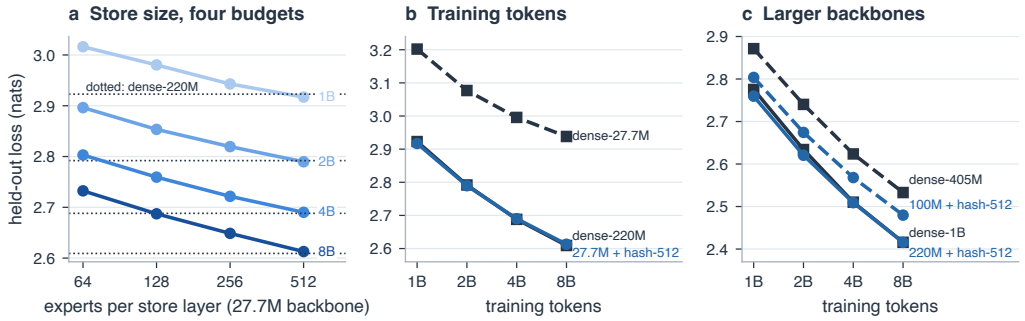

Figure 4: Three scaling axes. (a) Loss against store size at four budgets on the 27.7M backbone (dotted: dense-220M at the same budget). (b) The 512-expert store on the 27.7M backbone against dense-220M across an eightfold range of training tokens. (c) The 220M store matches dense-1B and the 100M store leads dense-405M at every budget. Single seed per point; spreads in Appendix D.

Parameter overhead. These gains come with a total-parameter overhead comparable to that reported for sparse models, except the prefetchable store moves this cost to a cheaper storage tier. Table 2 compares our overhead with sparse models evaluated against same-data dense counterparts: the 220M configuration's 3.3× lies within a factor of 1.5 of the reported frontier range, despite far fewer training tokens. For our smallest backbone, the joint MoE law of Ludziejewski et al. (2025) implies an overhead of 6.3–7.0×, placing our 6.3× near the lower end (Appendix B).

Served end to end. We serve the 220M checkpoint from Table 1 (10^9 training tokens) from local NVMe and compare it with the resident dense-1B trained on the same tokens (Figure 5; Table 10, Appendix E). Experts are stored in int8 and dequantized to bf16 on device; both models compute in bf16 using the same decode implementation. Each paired comparison runs on the same node at batch 1 for 1,024 decode steps, repeated across 4 nodes. The store is 1.31–1.40× faster in every repeat (mean 1.36×), using 0.61 GB of peak GPU memory versus 2.11 GB. Its 3.1 ms fetch overlaps early-layer computation within the 9.8 ms decode step. Held-out loss through the served weights is

model	inactive parameters live in	total	matched dense	total / dense	training tokens
DeepSeekMoE-16B (Dai et al., 2024)	accelerator memory	16.4B	6.9B	2.4×	2T
DeepSeekMoE-145B (Dai et al., 2024)	accelerator memory	144.6B	67B	2.2×	245B
Ling-mini-beta (Tian et al., 2025)	accelerator memory	17.5B	6.1B	2.9×	1T
Qwen3-30B-A3B (Yang et al., 2025)	accelerator memory	30.5B	14.8B	2.1×	36T
27.7M + hash-512 (ours)	NVMe	1.37B	219M	6.3×	8B
100M + hash-512 (ours)	NVMe	2.45B	592M	4.1×	8B
220M + hash-512 (ours)	NVMe	3.24B	993M	3.3×	8B

Table 2: Parameter overhead. Overhead divides total parameters by the same-data dense comparator reported in prior work, or the fitted dense equivalent at the same token budget for our stores. Our stores place this additional capacity on disk.

2.7673 versus 2.7800, an improvement of 0.013 nats. The same-model ablation at 27.7M isolates the benefit of overlap: prefetching recovers 71% of the synchronous-fetch penalty (Figure 5c).

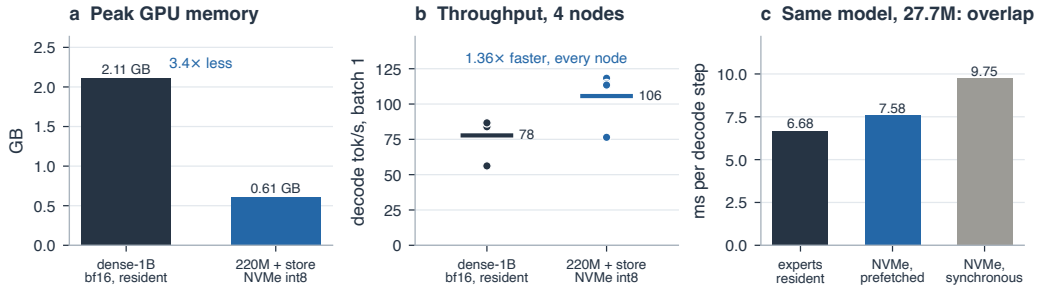

Figure 5: Serving and overlap. (a, b) Peak GPU memory and batch-1 decode throughput for the 220M backbone with a 3.0B store on NVMe (3.24B total), compared with resident bf16 dense-1B trained on the same 10^9 tokens. (c) The same 27.7M model with resident, prefetched and synchronously fetched experts, testing the overlap model of **R1**.

6 Toward the Frontier

The preceding results establish useful scaling with a simple storage-resident architecture. We now test whether ingredients used by efficient frontier MoEs can improve this architecture while preserving early addressing, then combine the experimental evidence with a resource budget to examine what frontier-scale deployment could look like.

6.1 Adapting Frontier Ingredients

The overhead measured in §5.2 belongs to a deliberately simple baseline: one hashed expert per store layer, without learned selection. Modern MoEs use contextual selection and combinations of multiple experts to make better use of sparse capacity (Dai et al., 2024; Krajewski et al., 2024). We test whether these ingredients transfer to the prefetchable store. Contextual selection appears to conflict with **R1**, since conventional routing uses state computed after the fetch window begins; multi-expert combination tests whether fixed addresses can support richer use of the stored pool. Both experiments use the 27.7M backbone and hash-512 reference configuration from §5.2.

Prefetchable learned routing. Our control uses conventional fine-grained routing: 2,048 quarter-width experts per store layer, one always-on shared expert and the top three routed experts per token, with dropless execution. The expert pool and selected expert width match the hash store's stored and active expert capacity. The router selects experts from the current store-layer state, so it is not prefetchable. With more training, this control moves from 0.020 nats behind the token-id store at 10^9 tokens to parity at 4×10^9 and 0.010 nats ahead at 8×10^9 (Figure 6a).

The **previous-step router** retains this expert architecture but changes the information used for selection. It reads the current token's embedding and the store-layer state of the *previous* position:

$$a_{\ell,t} = \mathrm{TopK}\big(W_\ell\,[\mathrm{RMSNorm}(e(x_t));\ \tilde{h}_{\ell,t-1}]\big). \tag{2}$$

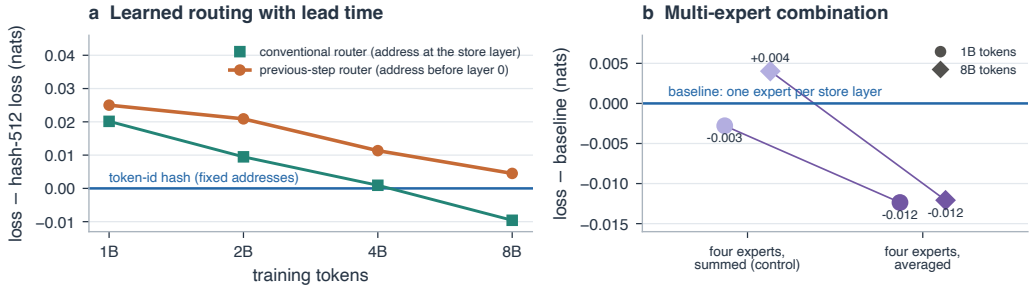

Figure 6: Prefetchable frontier ingredients. (a) Loss relative to the token-id store for the conventional router (addresses at the store layer) and the previous-step router (addresses before layer 0) across four training budgets. (b) Loss relative to the single-expert baseline for four experts summed or averaged. All configurations use the 27.7M backbone; second seeds are in Appendix D.

Both inputs are available before layer 0 of step t, providing the same address lead time as token-id routing while the selected experts still process the current state. Relative to the conventional control, the loss cost is 0.005 nats at 10^9 tokens and 0.014 at 8×10^9; second-seed differences for the previous-step router are 0.011 and 0.008 nats. At the longer budget it retains 96% of conventional routing's gain over the dense backbone, and its gap to token-id addressing narrows from 0.025 to 0.005 nats. Incremental decode reproduces the trained model within 2×10^{-5}, with 2.1M additional resident router parameters. Contextual selection can therefore retain nearly all of the conventional router's benefit while making every expert address available before the forward pass.

Multi-expert combination. The second experiment preserves token-based addressing and changes how much capacity each token uses. Four independent balanced tables select four full-width experts per store layer, whose outputs are averaged: $\Delta h = \frac{1}{4} \sum_{j=1}^{4} E_{a_j(x_t)}(\mathrm{RMSNorm}(h))$. Stored capacity is unchanged; the cost is 26% more active non-embedding parameters and four expert reads per store layer. At 10^9 tokens, loss falls from 2.9170 to 2.9047 (-0.012 nats; a second seed gives 2.9085). The improvement persists at 8×10^9 tokens, from 2.6129 to 2.6008 (-0.012 nats; overhead 6.3× to 5.7×). The averaged store also matches the conventional learned router's loss (2.6033) while retaining fixed addresses (Figure 6b).

Most of the gain comes from how the outputs are combined rather than from the extra fetch: summing the same four experts without rescaling gives losses of 2.9143 and 2.6169 at the two budgets. Learning the combination weights provides no further improvement over uniform averaging (2.9052 and 2.6027; Appendix C). Thus, richer expert use improves parameter efficiency at fixed stored capacity, and the gain survives an eightfold increase in training tokens.

These experiments demonstrate architectural headroom within the prefetchable constraint: early contextual routing preserves nearly all of conventional routing's gain, while multi-expert combination directly reduces parameter overhead. The baseline's efficiency is not an architectural ceiling.

6.2 Frontier Projection

We combine measured exchange rates, reported frontier MoE comparisons and an explicit execution budget to construct a frontier-scale scenario. Consider a 16B int8 backbone on a 24 GB consumer GPU and a 1T-parameter int8 store on one 1 TB drive ($90). Its expansion of 62.5× is close to the 49× we trained, although the backbone and store are two orders of magnitude larger.

To extrapolate across expansion ratios, we express overhead as an *exchange rate*, $r = 1 - \ln H / \ln(P/B)$. At a given expansion, this is equivalent to H; holding r fixed across scales projects a dense-equivalent size of $B(P/B)^r$ (Appendix B). Using the token-id store's measured rates of 0.53–0.56 at 8×10^9 tokens (Table 4) gives 143–163 B dense-equivalent parameters, approximately 150 B (Figure 7a). Rates of 0.65–0.67, reported for Ling-mini and Qwen3-30B-A3B against same-data dense comparators, give 235–255 B; DeepSeek-V3's benchmark-parity rate gives 495 B. Our measured rates define the baseline scenario, while reported frontier MoE rates illustrate the potential of higher parameter efficiency. The experiments in §6.1 demonstrate that parameter efficiency can improve within the prefetchable architecture.

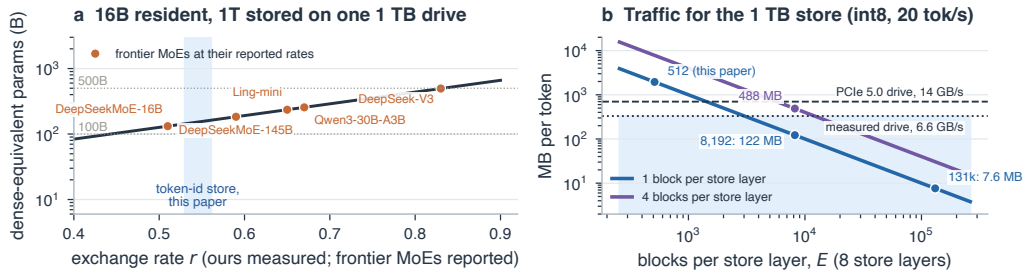

Figure 7: Frontier deployment scenario. (a) Projected dense-equivalent size of a 16B backbone with a 1T-parameter store: shading marks our measured token-id exchange rates at 8×10^9 tokens; points use rates inferred from reported frontier MoE comparisons. (b) Traffic versus blocks per store layer, with bandwidth budgets at 20 tokens/s for the measured drive and a PCIe 5.0 consumer-drive scenario.

The storage budget determines the architecture's shape. Distribute the pool across eight store layers, every fifth layer of a 40-layer backbone, with 8,192 blocks of 15M parameters per store layer. Reading one block per layer transfers 122 MB per token, requiring 2.4 GB/s on average at 20 tokens/s and 3.1 GB/s at the first layer deadline, within the bandwidth measured in this paper. Reading four blocks per layer, as in §6.1, transfers 488 MB and requires 9.8 GB/s on average and 12.2 GB/s at the first deadline, within the PCIe 5.0 consumer-drive class budgeted in Figure 7b. Distributing the store layers through the depth permits transfers to meet successive layer deadlines under the assumed compute and bandwidth schedule (Appendix F). The corresponding resident-memory budget, including backbone weights, embeddings, KV cache, transfer buffers and dequantized expert copies, is 20.5 GB at 8k tokens of context, within a 24 GB card.

The central quality assumption is that the exchange rate transfers to this larger architecture. Three observations support it: the measured rate remains approximately stable across an eightfold training-token range (§5.2), the proposed expansion is close to the tested range, and at a 14.8T-token budget each stored parameter would receive 118–474 presentations, compared with 12–48 in our experiments, although still tens of times below DeepSeek-V3's (Appendix F). The larger backbone, store and eight-layer placement remain untested, and across our 27.7M grid the rate fell slowly as the store grew (Appendix F). The measured scaling, transferable architectural improvements and resource budget establish a path to investigate frontier-class capacity behind consumer GPU memory.

7 Limitations

Our measurements cover models with up to 3.2B total parameters trained on up to 8×10^9 tokens using one corpus and tokenizer. The frontier scenario in §6.2 extrapolates measured parameter efficiency to a larger backbone, store and activation pattern, with an assumed execution budget. Evaluation uses a sequence-level split of the training stream, with the overlap reported in §5.1. Most configurations use a single seed; 33 seed pairs characterize observed variability. Serving measurements cover batch-1 decode, and the learned router's decode path is validated for numerical parity but not benchmarked from NVMe.

8 Conclusion

The prefetchable store separates trained capacity from GPU residency by making expert addresses available early enough to overlap storage transfers with computation. Across every axis tested, stored capacity improves quality; served end to end, it matches a larger dense model while decoding faster in less GPU memory. Its parameter overhead is comparable to established sparse-model tradeoffs, and the transfer of frontier MoE ingredients demonstrates room to improve beyond the baseline. Together, these results motivate a concrete deployment scenario for frontier-class capacity behind a consumer GPU and a 1 TB drive, if the measured efficiency transfers to scale. A model's total parameters need not be its resident parameters.

REPRODUCIBILITY STATEMENT

Code and results are available at `https://anonymous.4open.science/r/prefetchable-store-release-52CE/`. A single script regenerates every table, figure and in-text number from the per-run result files in the repository, which cover Appendices B–G. The repository includes the training script with the command line of every reported run (Appendix A), a corpus builder for the public FineWeb-Edu dataset that verifies its output against checksums of the training corpus, the serving stack and serving scripts (§4, Appendix E), and the task-scoring code. Checkpoints are omitted because of their size.

AI USE STATEMENT

We used generative AI tools, including LLM-based coding agents under the authors' direction, to implement and refactor experiment, serving, testing and corpus-building code; launch and monitor training and serving jobs; and aggregate and analyze results. These tools also assisted with experimental suggestions, scientific figures, literature summaries, manuscript drafting and editing, and reference formatting. No synthetic training or evaluation data were generated. The authors substantially wrote and revised the final manuscript, reviewed all AI-assisted work, and determined the research questions, design requirements and interpretation of the results. Reported results are traceable to committed result files, serving and decode-parity checks are covered by the test suite, and the authors reviewed the frozen decision rules and their outcomes. The authors take responsibility for all content, claims and artifacts in this work.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. LLM in a flash: Efficient large language model inference with limited memory. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. In *International Conference on Machine Learning*, 2025.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2020.

Yilong Chen, Yanxi Xie, Zitian Gao, He Xin, Yihao Xiao, Jason Klein Liu, Haoming Luo, Yifan Luo, Zhengmao Ye, Tingwen Liu, Xin Zhao, Ran Tao, and Bryan Dai. Beyond N-gram: Data-aware X-GRAM extraction for efficient embedding parameter scaling. *arXiv preprint arXiv:2604.21724*, 2026.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Han Zhang, Yu-Kun Li, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models. In *Annual Meeting of the Association for Computational Linguistics*, 2026.

Aidan Clark, Diego de las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew J. Johnson, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Marc'Aurelio Ranzato, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models. In *International Conference on Machine Learning*, 2022.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

DeepSeek-AI. DeepSeek-V4: towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238*, 2023.

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.

Elias Frantar, Carlos Riquelme Ruiz, Neil Houlsby, Dan Alistarh, and Utku Evci. Scaling laws for sparsely-connected foundation models. In *International Conference on Learning Representations*, 2024.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient sparse training with mixture-of-experts. In *Proceedings of Machine Learning and Systems*, 2023.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. Zenodo, 2023. Version v0.4.0, doi:10.5281/zenodo.10256836.

Ronald L Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

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

Xu Owen He. Mixture of a million experts. *arXiv preprint arXiv:2407.04153*, 2024.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, 2022.

Zihao Huang, Qiyang Min, Hongzhi Huang, Defa Zhu, Yutao Zeng, Ran Guo, and Xun Zhou. Ultra-sparse memory network. In *International Conference on Learning Representations*, 2025.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *International Symposium on Computer Architecture*, 2024.

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.

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

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems*, 2019.

Yu Lin, Yiming Wang, Runyuan Cai, Hanze Liu, and Xiaodong Zeng. The other half of the memory wall: Serving 35B MoEs from SSD with trained routing prediction. *arXiv preprint arXiv:2609.18063*, 2026.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

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

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

Yutang Ma, Kecheng Huang, Xikun Jiang, and Zili Shao. TF-Engram: A train-free Engram with SSD-backed memory for large language models. *arXiv preprint arXiv:2607.07388*, 2026.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Annual Meeting of the Association for Computational Linguistics*, 2016.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2024.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. Hash layers for large sparse models. In *Advances in Neural Information Processing Systems*, 2021.

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.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning*, 2023.

Xiaoniu Song, Zihang Zhong, Rong Chen, and Haibo Chen. ProMoE: Fast MoE-based LLM serving using proactive caching. *arXiv preprint arXiv:2410.22134*, 2024a.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. PowerInfer: Fast large language model serving with a consumer-grade GPU. In *Proceedings of the ACM SIGOPS Symposium on Operating Systems Principles*, 2024b.

Changxin Tian, Kunlong Chen, Jia Liu, Ziqi Liu, Zhiqiang Zhang, and Jun Zhou. Towards greater leverage: Scaling laws for efficient mixture-of-experts language models. *arXiv preprint arXiv:2507.17702*, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. MoE-Infinity: Efficient MoE inference on personal machines with sparsity-aware expert cache. *arXiv preprint arXiv:2401.14361*, 2024.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics*, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019.

A Experimental Details

setting	value
corpus / tokenizer	FineWeb-Edu sample-10BT / Llama-2, 32k
sequence length; sequences per step	2,048; 64 (131,072 tokens per step)
steps at $1/2/4/8 \times 10^9$ tokens	7,629 / 15,258 / 30,516 / 61,032
optimizer	AdamW (Loshchilov & Hutter, 2019), $\beta = (0.9, 0.95)$, weight decay 0.1, clip 1.0
learning rate	peak 6×10^{-4}, 300 warmup steps, cosine decay (Loshchilov & Hutter, 2017) to $0.1\times$ peak
precision	bf16 training autocast; fp32 evaluation; int8 store, bf16 serving
learned routers	load-balancing coefficient 0.01; dropless (Gale et al., 2023)
held-out evaluation	524,288 tokens of the main split

Table 3: Shared training recipe. Each token budget is a separate run with its own cosine schedule; larger training sets contain the smaller sets as exact prefixes.

Models. Backbones are listed as non-embedding parameter count with (d_{model}, layers, d_{ff}, query/KV heads), head dimension 64 throughout: 27.7M (512, 10, 1376, 8/2); 100.5M (896, 12, 2432, 14/2); 219.5M (1152, 16, 3072, 18/3); 405.6M (1408, 20, 3776, 22/2); and 980.7M (1792, 30, 4800, 28/2). Store layers are placed at $\lfloor 0.2L \rfloor$ and $\lfloor 0.6L \rfloor$, with RMSNorm on each expert input. Hash experts use SwiGLU with $d_{\text{ff}} = 853$, giving approximately 1.3M, 2.3M and 2.9M parameters per expert on the 27.7M, 100M and 220M backbones. The learned routers in §6.1 use 2,048 experts of width 213 per store layer, with one shared expert and top-3 routed experts whose routing weights are renormalized over the selected set. The previous-step router adds 2.1M resident parameters. Dense controls use the same training recipe.

Training viability (R3). The probes cited in §3 use the 27.7M backbone trained for 10^9 tokens under the earlier recipe without store-input normalization. Disabling the experts increases held-out loss by 2.83 nats for token-id addressing, 1.52 nats for bigram addressing with a minimum of 127k tokens per address, and 0.65 nats for mass-balanced bigram hashing at approximately 15k tokens per address. These probes examine address-level training exposure; the main capacity and architectural-extension studies use the normalized recipe.

B Capacity and Overhead

For resident backbone size B, total non-embedding parameters P and fitted dense-equivalent size N_{eq}, overhead is $H = P/N_{\text{eq}}$. We also report the *exchange rate* $r = \ln(N_{\text{eq}}/B)/\ln(P/B)$. At a fixed expansion, these are equivalent descriptions: $H = (P/B)^{1-r}$. Extrapolation in §6.2 assumes that r transfers across scales.

backbone + store	resident B	total P	active	loss	nearest dense	N_{eq}	overhead P/N_{eq}	r
27.7M + hash-64	27.7M	195M	30.3M	2.7324	100M (2.728)	97.2M	2.0×	0.64
27.7M + hash-128	27.7M	363M	30.3M	2.6872	100M (2.728)	131M	2.8×	0.60
27.7M + hash-256	27.7M	699M	30.3M	2.6487	220M (2.609)	170M	4.1×	0.56
27.7M + hash-512	27.7M	1.37B	30.3M	2.6129	220M (2.609)	219M	6.3×	0.53
100M + hash-512	100M	2.45B	105M	2.4798	406M (2.533)	592M	4.1×	0.56
220M + hash-512	220M	3.24B	225M	2.4157	981M (2.416)	993M	3.3×	0.56

Table 4: Capacity ledger at 8×10^9 tokens. Parameter counts exclude embeddings; dense references train on the same tokens.

backbone + store	1×10^9 tokens loss	N_{eq}	2×10^9 tokens loss	N_{eq}	4×10^9 tokens loss	N_{eq}	8×10^9 tokens loss	N_{eq}
27.7M + hash-64	3.0162	106M	2.8964	102M	2.8030	101M	2.7324	97.2M
27.7M + hash-128	2.9804	142M	2.8535	143M	2.7594	137M	2.6872	131M
27.7M + hash-256	2.9429	195M	2.8195	189M	2.7216	182M	2.6487	170M
27.7M + hash-512	2.9170	246M	2.7897	242M	2.6901	230M	2.6129	219M
100M + hash-512	2.8037	742M	2.6743	682M	2.5681	607M	2.4798	592M
220M + hash-512	2.7596	1.20B	2.6206	1.15B	2.5090	1.00B	2.4157	993M
dense-27.7M	3.2020	–	3.0769	–	2.9958	–	2.9382	–
dense-100M	3.0239	–	2.9005	–	2.8067	–	2.7277	–
dense-220M	2.9229	–	2.7920	–	2.6881	–	2.6092	–
dense-405M	2.8714	–	2.7403	–	2.6238	–	2.5329	–
dense-1B	2.7754	–	2.6341	–	2.5103	–	2.4158	–

Table 5: Held-out loss and fitted dense-equivalent size for every store and token budget, using the five-point dense frontier trained on the same tokens.

Table 5 lists every store at every budget. The 512-expert stores' loss reductions over their backbones grow from 0.285 to 0.325 nats on 27.7M, 0.220 to 0.248 on 100M, and 0.163 to 0.193 on 220M as training increases from 10^9 to 8×10^9 tokens. Dense-equivalent sizes can nevertheless decline because the dense references also improve: the loss gap between dense-27.7M and dense-220M widens by 18%. At 8×10^9 tokens, the 27.7M store's exchange rate decreases from approximately 0.64 at 64 experts to 0.53 at 512 experts.

Comparison with scaling laws. The joint MoE law of Ludziejewski et al. (2025) models

$$L(N_{\mathrm{act}}, D, E) = a\,\hat{E}^{\delta} N_{\mathrm{act}}^{\alpha+\gamma \ln \hat{E}} + b\,\hat{E}^{\omega} D^{\beta+\zeta \ln \hat{E}} + c,$$

where $\hat{E}$ saturates expert count. The fit covers over 280 Switch models with one active expert, $E \leq 32$ and 0.5–80×10^9 tokens. Without refitting, we evaluate the published width-512 configuration and solve $L(N, D, 1) = L(N_{\mathrm{act}}, D, E)$ for the corresponding dense size. The range spans $E \in \{16, 32\}$, with and without embeddings.

This gives $r \approx 0.25$–0.28 at 10^9 tokens and 0.50–0.53 at 8×10^9; Table 6 lists the rates that frontier reports imply. Applying the latter range to our smallest backbone's expansion gives overhead $(P/B)^{1-r} = 6.3$–$7.0\times$. The different corpus, tokenizer and architecture make this a contextual comparison rather than a calibrated prediction for our models.

model	active	total	matched dense model	training tokens	r	comparison
DeepSeekMoE-145B	22.2B	144.6B	67B	0.245T	0.59	same training data
Ling-mini-beta	0.85B	17.5B	6.1B	1T	0.65	same training data
DeepSeekMoE-16B	2.8B	16.4B	6.9B	2T	0.51	same training data
Qwen3-30B-A3B	3.3B	30.5B	14.8B	36T	0.67	same training data
DeepSeek-V3	37B	671B	405B	14.8T	0.83	different data (benchmark parity)

Table 6: Reported MoE comparisons expressed as $r = \ln(N_{\mathrm{dense}}/N_{\mathrm{active}}) / \ln(N_{\mathrm{total}}/N_{\mathrm{active}})$. The first four rows use same-data dense comparators from the same research group; DeepSeek-V3 uses benchmark parity with a separately trained model, Llama 3.1 405B (Grattafiori et al., 2024). Literature rates use active parameters as the reference size, whereas our store rates use the resident backbone.

C ARCHITECTURAL EXTENSIONS

Table 7 lists every routing arm and Table 8 every multi-expert arm of §6.1.

router (2,048 experts per store layer, 1 shared + top-3, dropless)	10^9	2×10^9	4×10^9	8×10^9
token-id hash (reference)	2.9170	2.7897	2.6901	2.6129
conventional (store-layer state, not prefetchable)	2.9372	2.7991	2.6911	2.6033
previous-step (prefetchable)	**2.9421**	**2.8105**	**2.7015**	**2.6174**
previous-step − hash	+0.025	+0.021	+0.011	+0.005
previous-step − conventional	+0.005	+0.011	+0.010	+0.014
gain over dense-27.7M: hash / conventional / previous-step	0.285 / 0.265 / 0.260	0.287 / 0.278 / 0.266	0.306 / 0.305 / 0.294	0.325 / 0.335 / 0.321

Table 7: Held-out loss for token-id addressing and learned routing on the 27.7M backbone. Second seeds for both routers are listed in Table 9.

hash-512 store, 27.7M backbone	fetches / token	loss (10^9)	overhead	loss (8×10^9)	overhead
one expert per store layer (baseline)	2	2.9170	5.6×	2.6129	6.3×
four experts, outputs summed (control)	8	2.9143	5.4×	2.6169	6.4×
four experts, outputs averaged	8	**2.9047**	**5.0**×	**2.6008**	**5.7**×
four experts, averaged, learned weights	8	2.9052	5.0×	2.6027	5.8×

Table 8: Multi-expert combinations and controls on the 27.7M backbone with hash-512 storage. Four-expert configurations use independent balanced assignment tables. Learned averaging weights are computed by a softmax gate on the store input, initialized to uniform averaging. The averaged arm's second-seed loss at 10^9 tokens is 2.9085.

D SEED VARIATION

arm	tokens	seed 0	seed 1	$\lvert\Delta\rvert$
dense-27.7M	1×10^9	3.2020	3.2206	0.019
dense-27.7M	2×10^9	3.0769	3.0977	0.021
dense-27.7M	4×10^9	2.9958	3.0110	0.015
dense-27.7M	8×10^9	2.9382	2.9407	0.002
dense-100M	1×10^9	3.0239	3.0258	0.002
dense-100M	2×10^9	2.9005	2.9004	0.000
dense-100M	4×10^9	2.8067	2.8074	0.001
dense-100M	8×10^9	2.7277	2.7321	0.004
dense-220M	1×10^9	2.9229	2.9328	0.010
dense-220M	2×10^9	2.7920	2.7982	0.006
dense-220M	4×10^9	2.6881	2.6874	0.001
dense-220M	8×10^9	2.6092	2.5991	0.010
dense-405M	1×10^9	2.8714	2.8439	0.028
dense-405M	8×10^9	2.5329	2.5147	0.018
dense-1B	1×10^9	2.7754	2.7610	0.014
hash-64	1×10^9	3.0162	3.0296	0.013
hash-64	8×10^9	2.7324	2.7381	0.006
hash-128	1×10^9	2.9804	2.9834	0.003
hash-128	8×10^9	2.6872	2.6920	0.005
hash-256	1×10^9	2.9429	2.9494	0.007
hash-256	8×10^9	2.6487	2.6494	0.001
hash-512	1×10^9	2.9170	2.9187	0.002
hash-512	2×10^9	2.7897	2.7936	0.004
hash-512	4×10^9	2.6901	2.6927	0.003
100M + hash-512	1×10^9	2.8037	2.8084	0.005
four averaged	1×10^9	2.9047	2.9085	0.004
conventional router	1×10^9	2.9372	2.9552	0.018
conventional router	2×10^9	2.7991	2.8149	0.016
conventional router	8×10^9	2.6033	2.6101	0.007
previous-step router	1×10^9	2.9421	2.9534	0.011

arm	tokens	seed 0	seed 1	$\lvert\Delta\rvert$
previous-step router	2×10^9	2.8105	2.8194	0.009
previous-step router	4×10^9	2.7015	2.7129	0.011
previous-step router	8×10^9	2.6174	2.6249	0.008

Table 9: Configurations with a second initialization seed. Cross-platform repeats are not independent samples.

Cross-platform repeats at 10^9 tokens give losses of 3.0296 versus 3.0298 for hash-64, 2.9834 versus 2.9855 for hash-128, 2.9494 versus 2.9483 for hash-256, and 3.2020 versus 3.1951 for dense-27.7M. The hash-64 pair compares H100 and L40S execution.

E SERVING MEASUREMENTS

Both models use the same decode implementation: greedy generation from token id 1, 32 warmup steps and $1{,}024$ timed incremental steps at batch 1. Each follows its own generated sequence. Comparisons are paired on the same node; Table 10 summarizes them, Table 11 lists every repeat, and Table 12 gives the same-model offload study.

	dense-1B, bf16, fully resident	220M backbone + store on NVMe (int8)
held-out loss through the served weights	2.7800	**2.7673**
decode speed (tok/s)	78 ± 13	**106** $\pm$ **17** (1.36×)
step time (ms)	13.3	9.8 (3.1 fetch, overlapped)
peak GPU memory (GB)	2.11	**0.61** (3.4×)
board power (W)	143	110
resident parameters (incl. embeddings)	1038M	**256M**
on-disk store (int8)	–	3.03 GB

Table 10: Serving comparison across 4 nodes, with mean $\pm$ standard deviation where shown. Both models compute in bf16; store experts are read in int8 and dequantized on device. Loss is evaluated through served weights, and resident parameter counts include embeddings.

repeat	store compute	dense-1B bf16 tok/s	store tok/s	store / dense	store step (ms)	store fetch (ms)	store peak GB
1	bf16	84.4	118.3	1.40	8.5	2.8	0.61
2	bf16	56.2	76.4	1.36	13.1	3.8	0.61
3	bf16	83.9	114.5	1.36	8.7	2.8	0.61
4	bf16	86.7	113.4	1.31	8.8	3.0	0.61
1	fp32	47.7	71.4	1.50	14.0	3.9	1.15
2	fp32	85.9	125.1	1.46	8.0	2.3	1.15
3	fp32	84.4	121.6	1.44	8.2	2.6	1.15
4	fp32	84.1	115.2	1.37	8.7	2.7	1.15

Table 11: Individual paired serving runs on L40S nodes, including earlier runs with fp32 store computation.

In the parity test, int8 quantization introduces a maximum logit difference of 0.023. Evaluations from bf16-stored checkpoints are approximately 0.005 nats above the training-time fp32 evaluations. The int8 round-trip of expert weights changes loss by +0.0000 nats; casting dense-1B to bf16 changes it by -0.0004 nats, at the reported precision.

configuration (hash-512, 27.7M backbone)	tok/s	step (ms)	fetch (ms)	logit Δ
experts resident in GPU memory	149.8	6.68	–	–
NVMe bf16, prefetched	137.2	7.29	2.25	0.0 (bitwise)
NVMe int8, prefetched	132.0	7.58	1.64	0.023
NVMe int8, synchronous (ablation)	102.6	9.75	2.43	–

Table 12: Same-model offload comparison on the 27.7M backbone. Synchronous int8 fetching is $1.29\times$ slower than prefetched int8 execution, within the overlap model's $1 + \mathrm{fetch}/\mathrm{step}$ range of 1.25–$1.36\times$, using the two measured fetch times. The bf16 disk and resident paths produce bitwise-identical logits.

Measurement exclusions. All four offload measurement attempts are preserved in the repository. Attempt 1 waited for fetched weights before starting the step, eliminating overlap. Attempt 2 used different compute paths for resident and offloaded execution and was excluded. Attempt 3

matched the compute paths and supplies the reported comparison. Attempt 4 added CUDA-event side-stream synchronization and retained 0.802 of resident speed on a faster node; its synchronous ablation exceeded the acceptance bound fixed before serving experiments and was classified as an I/O-contention anomaly. Reported speed ratios compare runs within the same attempt and node.

F Frontier Deployment Budget

Resident memory. The budget includes a 16B int8 backbone (16 GB), a 128k-entry int8 embedding table of width 5120 (0.7 GB), and a bf16 KV cache for 40 layers with eight KV heads of dimension 128 (1.3 GB at 8k context). Expert storage includes double-buffered int8 transfers across eight store layers (244 MB for one block per layer or 977 MB for four), bf16 working copies (244 or 977 MB), and 0.5 GB of workspace. Totals at 2k, 8k and 32k context are 18.0, 19.0 and 23.0 GB for one block per layer, and 19.4, 20.5 and 24.5 GB for four, so at 32k context the four-block point exceeds a 24 GB card and the one-block point is the one that fits. Quantization metadata and allocator overhead require validation in an implementation.

Transfer schedule. Assuming equal compute time per layer in a 50 ms token, store layers $5, 10, \ldots, 40$ require their blocks at $5, 11.25, \ldots, 48.75$ ms. For total payload Q bytes per token, eight equal transfers scheduled in layer order require sustained bandwidth

$$b \geq \max_{j=1,\ldots,8} \frac{jQ/8}{1.25(5j-1) \times 10^{-3}} = \frac{Q}{0.04}.$$

The first deadline is the tightest, requiring 25% more bandwidth than the token-average rate $Q/0.05$. The one- and four-block configurations therefore require approximately 3.1 and 12.2 GB/s, respectively, before allowing for I/O startup, dequantization and synchronization. The average traffic rates in Figure 7b are necessary bandwidth budgets; fully hidden transfer additionally requires these layer deadlines to be met. With two store layers and half the four-block payload due at 10 ms, the corresponding requirement would be approximately 24 GB/s.

Sensitivity to expansion. Across the 27.7M grid at 8×10^9 tokens, the exchange rate decreases by 0.040 per doubling of expansion, from 0.64 at 7× to 0.53 at 49× (Table 4). Extending this observed trend to 62.5× gives $r = 0.52$, or 135 B dense-equivalent parameters. This provides a sensitivity case alongside the constant-rate projection in §6.2.

Training exposure. For D training tokens, k selected blocks per store layer, n_s store layers and P_{store} stored parameters, balanced routing gives normalized exposure $q = Dk\,n_s/P_{\text{store}}$. This is the number of tokens received by an expert divided by its parameter count. The hash-512 baseline receives approximately 1.5 at 10^9 tokens and 12 at 8×10^9; the four-expert configuration receives 48 at the longer budget. At 14.8T tokens, the proposed deployment receives 118 and 474 for one and four blocks per layer, respectively, compared with approximately 10,500 for a DeepSeek-V3 routed expert (DeepSeek-AI, 2024). Exposure alone does not determine quality: the summed and averaged four-expert configurations share q but differ in loss. This comparison motivates longer training (Frantar et al., 2024; Ludziejewski et al., 2025), without attributing the observed parameter overhead solely to exposure.

G Task Scores

model	tokens	LAMBADA	PIQA	HellaSwag	ARC-e
dense-27.7M	1×10^9	15.4	57.6	27.2	42.0
dense-100M	1×10^9	17.4	59.7	27.4	44.9
dense-220M	1×10^9	21.0	60.0	28.1	46.4
dense-405M	1×10^9	21.3	60.9	28.3	47.8
dense-1B	1×10^9	22.6	62.1	29.3	50.0
27.7M + hash-64	1×10^9	17.3	59.7	27.5	45.8
27.7M + hash-128	1×10^9	17.1	59.2	27.8	44.9

model	tokens	LAMBADA	PIQA	HellaSwag	ARC-e
27.7M + hash-256	1×10^9	17.6	59.8	28.0	46.5
27.7M + hash-512	1×10^9	18.4	61.0	28.2	45.0
100M + hash-512	1×10^9	20.8	62.2	28.9	48.8
220M + hash-512	1×10^9	23.5	62.5	29.0	50.0
27.7M + hash-512, previous-step router	1×10^9	19.5	61.2	28.2	47.1
27.7M + hash-512, four averaged	1×10^9	18.1	60.5	28.3	47.0
dense-1B	2×10^9	27.1	63.3	30.8	52.2
27.7M + hash-64	2×10^9	21.6	60.2	28.4	46.4
27.7M + hash-128	2×10^9	21.2	61.8	28.4	46.1
27.7M + hash-256	2×10^9	21.6	61.9	29.1	47.8
27.7M + hash-512	2×10^9	22.7	62.3	29.2	49.1
100M + hash-512	2×10^9	24.8	64.2	29.9	50.6
220M + hash-512	2×10^9	27.0	64.3	30.8	52.7
dense-1B	4×10^9	33.2	65.0	32.7	55.0
27.7M + hash-64	4×10^9	23.3	61.4	28.8	49.2
27.7M + hash-128	4×10^9	23.9	62.9	29.2	49.3
27.7M + hash-256	4×10^9	25.7	62.8	30.0	50.2
27.7M + hash-512	4×10^9	24.2	62.9	30.2	50.8
100M + hash-512	4×10^9	27.6	64.9	31.5	55.7
220M + hash-512	4×10^9	31.7	64.5	32.6	56.2
dense-1B	8×10^9	35.9	67.6	35.2	60.7
27.7M + hash-64	8×10^9	24.9	61.8	29.5	50.8
27.7M + hash-128	8×10^9	26.2	63.5	30.2	50.7
27.7M + hash-256	8×10^9	26.6	63.2	30.6	51.4
27.7M + hash-512	8×10^9	26.5	64.4	31.1	53.2
100M + hash-512	8×10^9	29.3	66.3	33.3	59.3
220M + hash-512	8×10^9	33.5	67.1	34.8	60.6
27.7M + hash-512, previous-step router	8×10^9	26.9	65.7	31.0	54.5

Table 13: Zero-shot accuracy (%) for scored checkpoints; standard errors are 0.5–1.2 percentage points.

At 4 and 8×10^9 tokens, the 220M store trails dense-1B on LAMBADA by 1.6 and 2.3 percentage points, respectively, while remaining within 1.3 points on the other tasks. Thus, close language-modeling losses do not imply uniform parity across downstream tasks.