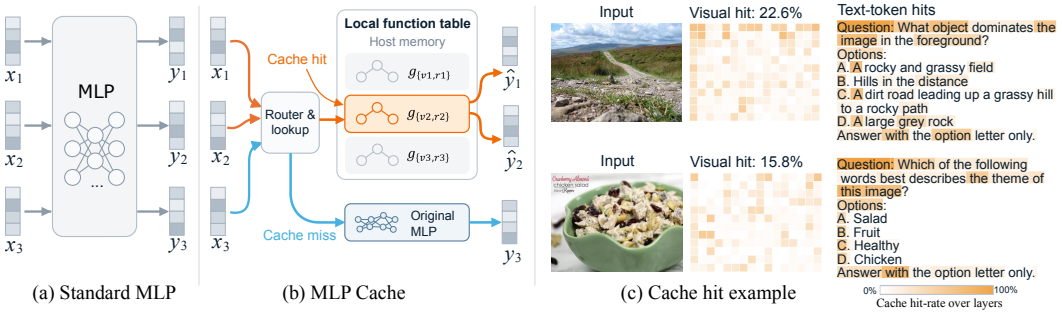
MLP CACHE: A POST-HOC MEMORY–COMPUTE TRADE-OFF FOR TRANSFORMER INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

MLPs impose substantial inference cost in Transformers by applying a full, high-capacity transformation to every token. Deployed workloads can revisit recurring local MLP input regions, there's opportunity for compute reuse beyond exact-match caching. We introduce MLP Cache, a post-hoc cache that stores compact local predictors of a frozen MLP without retraining the backbone. The central challenge is to allocate limited memory across region scale and predictor capacity: broader regions cover more inputs but may require larger predictors, finer regions are easier to approximate but require more entries. MLP Cache constructs a coarse-to-fine hierarchy using shared graph-spectral coordinates, whose signs define candidate regions and whose continuous values provide features for region-specific predictors. Tree dynamic programming selects region–predictor pairs to maximize workload coverage under memory and approximation-error budgets. On frozen Qwen3.5-9B, MLP Cache replaces 24–28% of MLP calls on MME-Coarse and MMStar-Coarse, with accuracy decreases of 1–2%. The cached model achieves a $1.10\times$ batch-one decoding wall-clock speedup on A100. Controlled MLP experiments show that cached prediction offers greater speedups for larger MLPs, the gains depend on the cost of uncached execution relative to cache lookup. For a 9B-shaped MLP, hit rates of 30% and 50% yield A100 prefill speedups of $1.24\times$ and $1.60\times$, and M5 Pro prefill speedups of $1.39\times$ and $1.61\times$.



(a) Standard MLP (b) MLP Cache (c) Cache hit example

Figure 1: A standard MLP use full-capacity function for every input, **MLP Cache** lookup a saved local function for recurring inputs, thus trade memory for compute. Cache hits appear on recurring image background and question-option templates, non-recurring question content don't use cache.

1 INTRODUCTION

The cache opportunity. A model is trained to cover a broad range of tasks and inputs, while a particular deployment typically serves a much narrower requests. Task-specific recurring prompt and visual patterns induce recurring MLP activation. yet every such activation is still processed by the same full-range MLP. Rather than recomputing the full MLP, can frequently visited portions of the MLP input–output be cached in a much smaller functions? MLP Cache explores this possibility by storing compact local functions of a frozen MLP. Figure 1 makes the opportunity concrete for VQA: across two distinct image–question pairs, cache hits cover background visual patches as well as recurring question template and answer-option structure.

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.

Existing memory–compute methods establish the value of memory, but the deployment locality above poses a different allocation problem: once the checkpoint is fixed, which recurring parts of its computation deserve the memory budget? Learned-memory systems such as PKM, UltraMem, and Engram [1, 2, 3] allocate memory as part of model training, so the allocation must reflect the workload anticipated during training or be adapted later. Table-based operator approximations such as MADDNESS [4] and LUTI-MLP [5] instead replace neural computation with lookup or interpolation, but apply a fixed approximation scheme to the operator. Neither approach directly answers the question raised by deployment locality: once a dense checkpoint is fixed and its serving workload is observed, which parts of the frozen MLP are worth materializing in memory?

A post-hoc cache must use its limited memory budgets where it covers the most recurring computation. Each cache entry combines a region of activation space with a compact predictor for the frozen MLP within that region (Figure 2). A broader region can replace more MLP evaluations, but the predictor must capture more variation in the function. Refining the region makes the local mapping easier to approximate, but divides the workload among more entries. Predictor capacity creates the same trade-off from the other direction: a larger predictor may preserve one broad region, while several smaller predictors may describe finer regions more efficiently. MLP Cache therefore chooses ***region extent*** and ***predictor capacity*** together, the goal is ***maximize hit-rate under approximation-error and memory budgets***.

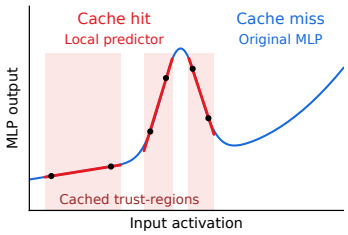


Figure 2: **Cache entries** store local functions over trust regions.

Storing a separate input projection for every local predictor would make the cache expensive as the number of regions grows. We therefore share a single input projection representation across regions and store only region-specific output projection. The difficulty is that the best shared representation is coupled with the cache structure itself: which representation are useful depends on the regions and predictor capacities being considered, while those choices in turn depend on the fitting error measured in that representation. Jointly optimizing the representation together with these discrete cache decisions would therefore entangle representation learning with region and capacity selection.

We decouple the two by deriving a fixed coordinate system from calibration activations using **graph-spectral embedding**. The spectral coordinates serve two roles: their continuous values are shared inputs to the local predictors, while their successive signs define a nested, coarse-to-fine candidate regions. Once the coordinates is fixed, cache construction becomes a discrete selection problem over region–predictor pairs, choosing those that cover the most workload subject to approximation-error and memory budgets. The graph-spectral solve is expensive but entirely offline; we show that its coordinates can be accurately distilled into a lightweight MLP Router for fast inference.

Tunable memory–compute trade. On frozen Qwen3.5-9B, a higher-coverage configuration replaces 27.67% and 24.09% of MLP calls in cache-enabled layers on MME-Coarse and MMStar-Coarse, with accuracy decreases of 2.22 and 0.93 percentage points, respectively, using 96.36 GiB of host memory. The available reuse varies with the amount of repeated compute in the dataset, and additional memory increases coverage once sufficient fitting data support more local predictors. The current full-model implementation reduces batch-one decoding latency on A100 from 18.10 to 16.45 ms/token. Controlled MLP experiments further show that cached prediction offers greater speedups for larger MLPs, with the gains depending on the cost of uncached execution relative to cache lookup and reconstruction. For a 9B-shaped MLP, prescribed hit rates of 30% and 50% yield A100 prefill speedups of $1.24\times$ and $1.60\times$, and M5 Pro prefill speedups of $1.39\times$ and $1.61\times$.

Our contributions are threefold. First, we identify a post-hoc computation-reuse opportunity in frozen Transformers: deployment workloads can repeatedly revisit local portions of an MLP's input–output map that can be stored and reused without retraining the model. Second, we introduce *MLP Cache*, which jointly selects region granularity and predictor capacity under memory and approximation-error constraints. Shared graph-spectral coordinates define both the candidate region hierarchy and predictor inputs. Third, our experiments characterize the memory–compute trade. MLP Cache replaces 24–28% of Qwen3.5-9B MLP calls on MME-Coarse and MMStar-Coarse, with only 1–2% accuracy drop. Controlled benchmarks identify the speedup conditions: larger MLPs amortize cache-service costs more readily, hit rates of 30% and 50% yield A100 prefill speedups of $1.24\times$ and $1.60\times$, and M5 Pro prefill speedups of $1.39\times$ and $1.61\times$.

2 LOCAL FUNCTION CACHING

MLP Cache stores local approximations to an MLP and reuses them across different input activations. Let $f(\mathbf{x})$ denote the MLP, held fixed during cache construction, and let $\{(\mathbf{x}_i, f(\mathbf{x}_i))\}$ be calibration input–output pairs. Each cache entry associates an activation region $\mathcal{R}_v$ with a predictor $g_{v,r}$ of capacity r. The region determines which inputs the entry can serve, while the predictor determines its approximation error and storage cost. Construction generates candidates at multiple region scales and predictor capacities, then searches for a non-overlapping selection under memory and approximation-error budgets. Figure 3 illustrates this process.

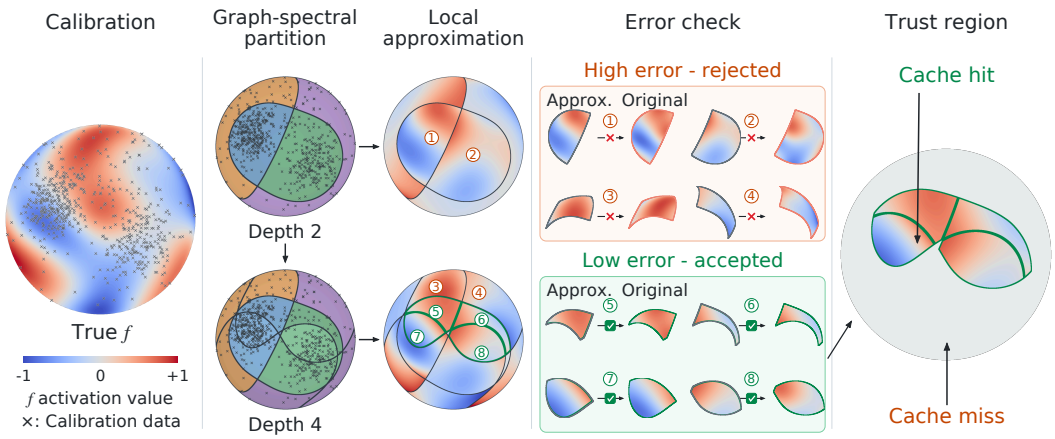


Figure 3: **Cache construction on calibration dataset for a frozen MLP function.** Coarse candidate regions that fail the error check can be refined into smaller alternatives; low-error regions become cached trust regions, while the remaining inputs use the original MLP. The schematic illustrates this coarse-to-fine construction for a scalar function.

2.1 CONSTRUCTING REGION–PREDICTOR CANDIDATES

A conventional rank-r local predictor takes the form $g_{v,r}(\mathbf{x}) = \mathbf{c}_{v,r} + \mathbf{A}_{v,r}\mathbf{B}_{v,r}^{\top}\mathbf{x}$, where $\mathbf{B}_{v,r}$ projects the input to r coefficients and $\mathbf{A}_{v,r}$ reconstructs the output. Storing a separate input projection for every region duplicates input-side parameters across the cache and requires an additional projection when an entry is evaluated. We instead share a coordinate map $\widehat{\mathbf{s}}_\phi(\mathbf{x}) \in \mathbb{R}^m$ across regions. Each region selects an index set $\mathcal{I}_{v,r} \subseteq \{1, \dots, m\}$, with $|\mathcal{I}_{v,r}| = r$, and predicts

$$g_{v,r}(\mathbf{x}) = \mathbf{c}_{v,r} + \mathbf{W}_{v,r}\widehat{\mathbf{s}}_{\phi,\mathcal{I}_{v,r}}(\mathbf{x}). \tag{1}$$

The coefficients $\mathbf{W}_{v,r} \in \mathbb{R}^{d_{\text{mlp}} \times r}$ and $\mathbf{c}_{v,r} \in \mathbb{R}^{d_{\text{mlp}}}$ are fitted separately for each region, where d_{mlp} is the MLP output dimension.

The choice of shared representation affects which regions can be approximated accurately at a given capacity. To decouple representation learning from region and capacity selection, we first construct graph-spectral coordinates from calibration activations. For anchors $\{\mathbf{x}_i\}_{i=1}^{N_{\text{base}}}$, we define $A_{ij} = \frac{1}{2}\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/(2\sigma^2)) + \frac{1}{2}\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2/\sigma)$, with σ chosen so that the mean anchor affinity is 0.8. Writing $D_{ii} = \sum_j A_{ij}$, we form $\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ and solve $\mathbf{S}\mathbf{u}_j = \lambda_j \mathbf{u}_j$. The leading m nontrivial eigenvectors define coordinates on the anchors; an out-of-sample extension provides targets $\mathbf{s}^\star(\mathbf{x})$ for additional activations. We train a lightweight MLP Router to predict $\widehat{\mathbf{s}}_\phi(\mathbf{x}) \approx \mathbf{s}^\star(\mathbf{x})$ and then hold it fixed during candidate construction and selection.

Both construction and inference use the Router's predicted coordinates. Their continuous values provide the features in Equation 1, while their signs define a nested hierarchy of candidate regions. At depth h, the sign prefix is

$$\pi_h(\mathbf{x}) = \left(\text{sign}^+(\widehat{s}_{\phi,1}(\mathbf{x})), \dots, \text{sign}^+(\widehat{s}_{\phi,h}(\mathbf{x}))\right), \tag{2}$$

where $\text{sign}^+(t) = 1$ for $t \geq 0$ and 0 otherwise. Each observed prefix identifies a region $\mathcal{R}_v$; appending one sign subdivides it into at most two children.

For each eligible region, we fit predictors with capacities $r \in \{2, 4, 8, 16, 32\}$. Nested orthogonal matching pursuit selects r Router coordinates, and ridge regression estimates the corresponding $\mathbf{W}_{v,r}$ and $\mathbf{c}_{v,r}$. This produces a candidate bank indexed by (v, r), with alternatives spanning both region scale and predictor capacity.

2.2 Selecting the Cache Under Memory and Error Budgets

We measure each candidate by its coverage, approximation error, and storage cost. Let N be the total number of calibration MLP calls and $N_{v,r}^{\mathrm{hit}}$ the number served by candidate (v, r). The region's coverage is $\mathrm{Hit}(v, r) = N_{v,r}^{\mathrm{hit}}/N$, and the approximation error is

$$\mathrm{Error}(v, r) = \frac{1}{N} \sum_{i:\,\text{served by } (v,r)} \frac{\|g_{v,r}(\mathbf{x}_i) - f(\mathbf{x}_i)\|_2^2}{\max\{\|\mathbf{x}_i^{\mathrm{residual}} + f(\mathbf{x}_i)\|_2^2, \delta\}}. \tag{3}$$

Here $\mathbf{x}_i^{\mathrm{residual}}$ is the residual stream entering the MLP block and $\delta > 0$ is a numerical floor. Calls that fall back to the original MLP contribute zero local approximation error but remain in the denominator N. We use $\mathrm{Memory}(v, r)$ to denote serialized predictor storage.

A cache policy $\mathcal{P}$ contains selected region–predictor pairs (v, r). Selecting a predictor at a node excludes predictors at its descendants, so the retained regions are non-overlapping. Policy coverage, error, and storage are therefore the sums of the corresponding candidate quantities. Given an error budget τ and a predictor-memory budget B, the selection objective is

$$\max_{\mathcal{P}} \ \mathrm{Hit}(\mathcal{P}) \quad \text{subject to} \quad \mathrm{Error}(\mathcal{P}) \le \tau, \qquad \mathrm{Memory}(\mathcal{P}) \le B. \tag{4}$$

We search for a policy by assigning nonnegative prices λ_{err} and λ_{mem} to error and storage. At each node, retaining a larger predictor competes with retaining predictors in finer descendant regions. For example, a rank-16 parent stores 17 output vectors, whereas two rank-4 children store 10 in total. For a fixed price pair, the candidate utility and the optimal subtree utility are

$$\begin{aligned} u(v, r) &= \mathrm{Hit}(v, r) - \lambda_{\mathrm{err}}\mathrm{Error}(v, r) - \lambda_{\mathrm{mem}}\mathrm{Memory}(v, r), \\ U(v) &= \max\Big\{0,\ \max_r u(v, r),\ \sum_{c \in \mathrm{child}(v)} U(c)\Big\}. \end{aligned} \tag{5}$$

The inner maximum ranges over eligible predictors at v. The three options are fallback, retaining one predictor at v, and combining policies from its children. Bottom-up dynamic programming maximizes this utility for each price pair. We sweep a finite grid of prices and retain the highest-coverage policy found that satisfies the construction budgets.

Because the layer caches are selected independently, their approximation errors can interact during model execution. We therefore calibrate the enabled-layer set using next-token KL divergence from the original model on calibration trajectories. Starting with all layer caches enabled, we greedily disable the layer whose removal gives the lowest output KL divergence.

2.3 Selective Online Execution

For a new activation $\mathbf{x}$, the Router predicts $\widehat{\mathbf{s}}_\phi(\mathbf{x})$. An auxiliary support score rejects inputs with low predicted affinity to the calibration anchors before lookup. For an accepted input, the coordinate signs determine the prefix in Equation 2, and lookup returns a retained entry if one exists.

To limit extrapolation within a matched entry (v, r), we also check the fitted range of every coordinate in $\mathcal{I}_{v,r}$. A width parameter α scales each interval about its midpoint: $\alpha = 1$ preserves the fitting range, $\alpha < 1$ narrows it, and $\alpha > 1$ expands it. The predictor is used only when all selected coordinates lie within their scaled intervals; otherwise, the original MLP executes.

The Router, prefix directory, coordinate indices, and support metadata reside on the GPU, while the region-specific coefficients $\mathbf{W}_{v,r}$ and $\mathbf{c}_{v,r}$ reside in host memory. For a supported hit, the GPU sends the predictor identity and selected coordinates to the CPU for host memory lookup and prediction.

3 EXPERIMENTS

We evaluate MLP Cache on frozen Qwen3.5-9B [6], with caches constructed from Cauldron General VQA [7]. The primary evaluation uses 240 MME questions on existence, count, position, and color, denoted MME-Coarse, and 250 MMStar coarse-perception questions, denoted MMStar-Coarse [8, 9]. Answer accuracy is averaged over three sampling generation runs.

Cache configurations. The two principal full-model configurations use $B = 1$ GiB per enabled layer with $\tau = .001$, and $B = 4$ GiB per enabled layer with $\tau = .003$. Both enable caching in the 28 calibration-selected layers. The budget B covers stored predictor coefficients and scales; reported hit rates count replaced MLP calls across the enabled layers.

3.1 WHERE DOES THE REUSE HAPPEN?

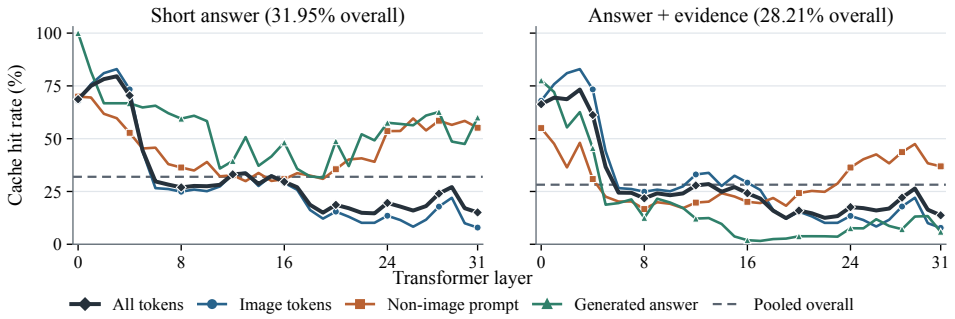


Figure 4: **Longer generation reveals a strong layer-wise separation between prompt and answer reuse.** With short answers, generated tokens remain highly reusable in later layers; when the model must also provide answer and evidence, later-layer answer reuse is low while prompt reuse is high. Curves show image, non-image prompt, generated-answer, and pooled hit rates at each layer. Plots are on 240 MME-Coarse questions and cache configuration $B = 4$ GiB/layer, $\tau = .003$.

Reusable computation is structured across layers. At the higher-coverage $B = 4$ GiB/layer, $\tau = .003$ operating point, MLP Cache serves 27.67% and 24.09% of MLP calls in the 28 cache-enabled layers on MME-Coarse and MMStar-Coarse, respectively (Table 3). To inspect the full layer profile, Figure 4 evaluates the corresponding per-layer caches with all 32 layers enabled and separates image, non-image prompt, and generated-answer calls. In both response formats, image-token hit rates peak in the shallow stack and fall later; prompt hit rates decline through the middle layers, then rise again toward the end.

Answer reuse changes sharply with the requested output. In Figure 4, the two panels use the same 240 MME-Coarse questions, images, cache, and sampling seeds; only the final response instruction changes from a short yes/no answer to an answer with brief visual evidence. With short answers, generated-token hits remain frequent even in later layers. Across layers 16–31, their hit rate is 49.09%, close to the 45.80% prompt hit rate. With evidence responses, the answer hit rate across layers 16–31 falls to 6.17%, while prompt calls still hit at 31.67%. The short-answer format can therefore obscure the separation between late prompt and answer reuse: its short, highly constrained generated tokens remain frequently accepted, whereas tokens in evidence responses rarely do.

This pattern reveals a layer asymmetry between prompt and answer reuse. One interpretation is that early layers are still forming a useful representation of the image and question, while deeper layers may devote more input-dependent transformation to the evolving answer when generation requires visual evidence. Hit rate alone does not establish layer importance or a causal decomposition of the network. It does raise the question of whether input processing and generation should traverse the same number of layers. DeepSeek-V4.1-Flash makes a related architectural choice: its causal encoder–decoder lets prefill prompt tokens bypass full decoder computation [10].

3.2 HOW SHOULD REUSABLE LOCAL FUNCTIONS BE REPRESENTED?

Shared spectral coordinates target small predictors. We compare shared Spectral coordinates with an independent low-rank projection from the full MLP input on the same layer-16 regions and

held-out examples, using approximately matched parameter bytes per predictor. Table 1 shows a crossover. At the smallest matched budget, shared Spectral predictors have lower held-out error at every reported depth; as more parameters are devoted to one region, the more flexible full-X predictor eventually becomes more accurate. This is the regime we care about. MLP Cache does not seek the most expressive predictor for one region; it seeks to cover many recurring regions under a fixed total memory budget. Cheap entries allow the same budget to retain more regions or allocate more capacity elsewhere. In addition, small predictors are fast for cache lookups in inference.

Shared coordinates enable earlier fallback. Full-X support checks depend on the region-specific projection $B_v^\top x$, so support-rejected queries cannot enter the dense MLP path until that projection completes; a host-side projection additionally requires transferring the full activation. MLP Cache instead checks support directly in Router coordinates already available on the GPU.

Approximation error: MSE ($\times 10^{-3}$)			
Partition depth	Predictor payload (KiB) 80	144	272
24	**17.40**/17.93	15.59/**15.30**	13.88/**12.85**
60	**1.53**/1.58	1.31/**1.28**	1.10/**0.99**
96	**0.42**/0.43	0.36/**0.35**	0.30/**0.27**

Table 1: Each entry are Spectral / full-X fitting error; the 80/144/272 KiB columns pair ranks 4/2, 8/4, and 16/8. Shared Spectral coordinates has lower-error at small payload size.

Holding Spectral prediction coordinates fixed, Table 2 compares SimHash, PCA, Bisecting k-means, Non-bisecting Spectral, and Bisecting Spectral. Bisecting Spectral provides the highest observed coverage at both tested budgets, reaching 17.49% at 1 GiB and 24.38% at 4 GiB. However, Bisecting Spectral is substantially harder to learn with a lightweight Router: in our preliminary attempts, sign prediction was substantially less accurate, with agreement below 80%, compared with 99.46% for the Non-bisecting Spectral Router. We therefore use Non-bisecting Spectral in the current system. The higher coverage of Bisecting Spectral suggests that improving hierarchical routing could unlock additional reuse.

Partition method	Hit rate (%) 1 GiB	4 GiB
SimHash	14.15	18.67
PCA	14.58	23.56
Bisecting k-means	17.03	22.78
Non-bisecting Spectral	15.77	23.38
Bisecting Spectral	**17.49**	**24.38**

Table 2: Cache constructed for layer-16 with error budget $\tau = .003$, hit rate are on MME and MMStar questions. Bisecting Spectral has the best hit rate.

3.3 What limits cache coverage?

Three resources constrain the cache: approximation error, calibration data size, and predictor memory.

The error budget controls how aggressively the MLP is approximated. Figure 5 varies predictor-storage ceilings from 1 to 32 GiB per enabled layer and construction targets $\tau \in \{.001, .002, .003\}$, while keeping the disabled-layer mask fixed. More permissive targets generally expose higher-coverage policies, but benchmark accuracy does not change monotonically with storage or coverage. The sweep therefore exposes distinct accuracy–reuse operating points.

Unused memory budget. The 1 GiB/layer, and 4 GiB/layer configurations have nominal ceilings of 28 and 112 GiB across the 28 cached layers, yet use 17.77 and 96.36 GiB of predictor payload. The construction procedure treats memory as a ceiling rather than filling it mechanically.

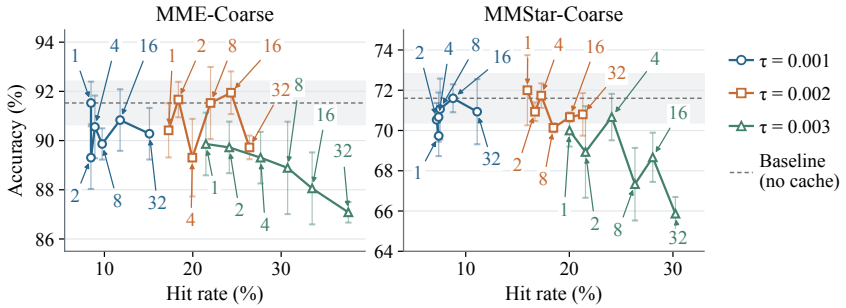


Figure 5: **Error and memory budgets expose the accuracy–reuse operating points.** Each curve varies the memory budgets from 1 to 32 GiB per layer at a fixed construction target τ.

The memory ceiling and calibration data size. In Figure 6, we study the memory budgets at layer 16 at fixed error budget $\tau = 0.003$, we construct cache while varying the number of calibration data

size and the memory budgets. With small fitting populations, the 1, 4, and 16 GiB budgets select the same policy: additional memory has no candidate regions to store. Near 100M fitting activations, the budgets diverge. Increasing storage from 1 to 16 GiB raises coverage from 17.62% to 32.93% on MME-Coarse and from 14.99% to 23.69% on MMStar-Coarse. Conversely, at 1 GiB, adding fitting data beyond roughly one million activations yields no further measured coverage gain.

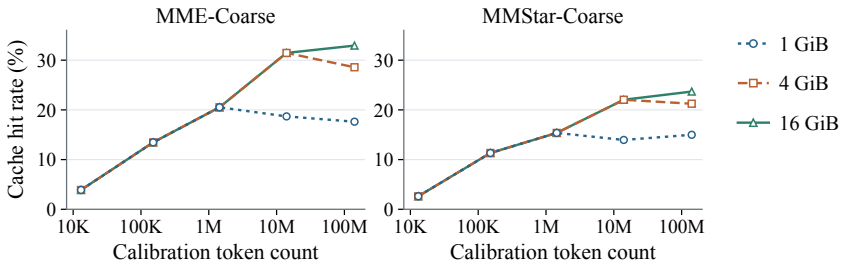


Figure 6: **Larger memory budgets need more calibration data.** At 10K–1M fitting activations, the 1, 4, and 16 GiB budgets select the same policy; toward 10M–100M, the storage budgets begin to separate in coverage. This layer-16 study fixes the Spectral representation and $\tau = .003$ and evaluates all policies on the same MME-Coarse and MMStar-Coarse inputs.

3.4 When does reuse improve wall-clock?

A cache hit removes MLP computation, but its latency benefit depends on how that replacement changes the execution path. We combine full-model measurements with controlled MLP benchmarks to distinguish the amount of reusable computation from the conditions under which it saves time.

Full-model measurements show different outcomes for prefill and decoding. Table 3 reports answer quality, cache coverage, predictor storage, GPU activity, and phase wall-clock time. For BF16 Qwen3.5-9B, the higher-coverage cache reduces batch-one decode wall-clock from 18.099 to 16.447 ms/token, a $1.10\times$ speedup, while GPU activity decreases from 16.043 to 13.380 ms/token. Prefill also performs less GPU work, decreasing from 332.89 to 317.03 ms per 4,096-token cohort, but its wall-clock increases from 414.11 to 465.98 ms. Understanding these different outcomes requires examining how hits replace computation in the two phases.

Method	Predictor GiB	MME-Coarse Acc. / Hit (%)	MMStar-Coarse Acc. / Hit (%)	Prefill (ms) GPU / Wall	Decode (ms) GPU / Wall
BF16	0.00	91.53$\pm$0.87 / 0.00	71.60$\pm$1.20 / 0.00	332.9 / 414.1	16.0 / 18.1
+ Cache (τ=.001, 1GiB)	17.77	91.53$\pm$0.87 / 8.48	69.73$\pm$1.01 / 7.39	341.0 / 456.1	14.4 / 17.4
+ Cache (τ=.003, 4GiB)	96.36	89.31$\pm$1.05 / 27.67	70.67$\pm$1.15 / 24.09	317.0 / 466.0	13.4 / 16.4
GPTQ W4A16	0.00	91.25$\pm$0.42 / 0.00	71.47$\pm$1.01 / 0.00	346.8 / 435.4	9.2 / 11.2
+ Cache (τ=.003, 4GiB)	96.36	89.31$\pm$1.34 / 27.48	68.67$\pm$1.15 / 23.94	335.9 / 488.3	8.6 / 11.8
SmoothQuant W8A8	0.00	92.64$\pm$0.24 / 0.00	71.20$\pm$0.00 / 0.00	274.6 / 396.6	10.8 / 13.1
+ Cache (τ=.003, 4GiB)	96.36	89.31$\pm$2.71 / 27.49	69.60$\pm$1.06 / 23.90	267.9 / 469.7	9.8 / 13.1

Table 3: **Full-model results separate cache coverage from realized execution savings.** The higher-coverage BF16 cache reduces batch-one decode wall-clock, while prefill and quantized compositions show that lower GPU activity need not produce an incremental latency gain. GPU and wall-clock columns measured on A100; prefill is per 4,096-token cohort and decode per batch-one step.

Controlled prefill exposes the balance between cache service and GPU execution. We benchmark synthetic 4B-, 9B-, and 27B-shaped SwiGLU MLPs with prescribed hit rates and rank-4 cached predictors. Complete-path timing includes routing, lookup, support checks, CPU reconstruction, data movement, GPU fallback, and output assembly. In the prefill path, CPU reconstruction serves hits while the GPU processes misses. At low hit rates, too little GPU work is removed to offset routing and scheduling overhead. At intermediate rates, reconstruction can overlap the remaining GPU computation. At sufficiently high rates, cache service can become the longer branch: the GPU finishes its miss batch but must wait for CPU cache hits. Figure 7 shows these regimes; at 30% hits, the 9B-shaped MLP achieves cold-cache prefill speedups of 1.24× on A100 and 1.39× on M5 Pro.

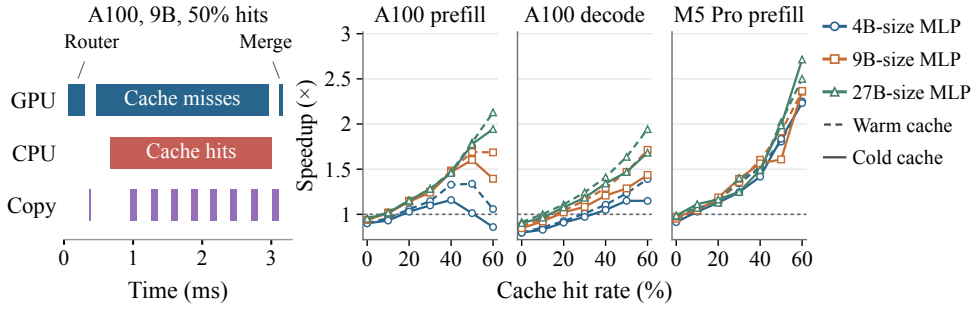


Figure 7: **A cache hit yield speedup when cache service is cheaper than the MLP.** Left: CPU reconstruction of hits overlaps GPU execution of misses in the controlled A100 path. Right: larger MLP and slower hardware yield more speedup. Cold cache increase cache service time cost.

Layerwise variation makes average hit rate insufficient to predict prefill latency. Unlike a controlled benchmark at one prescribed rate, the learned cache exhibits substantial variation across layers, with hit rates ranging from 10% to 70%. Low-coverage layers incur routing and scheduling costs while retaining most of their GPU MLP work; high-coverage layers can instead become limited by reconstruction and transfer. These costs occur at different layers and are obscured by a workload-average hit rate. Within a layer-synchronous execution path, spare capacity at one layer cannot compensate for a longer cache branch at another. The hit rate imbalance helps explain why the controlled MLP gains do not carry over to full-model prefill, even when GPU activity decreases.

Batch-one decoding instead replaces complete MLP invocations. Each layer processes one activation, so a call either takes the cache path or executes the original MLP. A hit bypasses the dense-MLP kernel launches and their weight reads from GPU memory, rather than merely reducing the number of rows in a still-required MLP batch. There is no simultaneous GPU miss subbatch to balance against CPU reconstruction. Hit probabilities can still vary across layers, but they change how often complete MLP invocations are replaced, not the relative lengths of two concurrent branches.

The cost of the original MLP determines the margin for savings. Larger MLPs remove more dense computation per successful cache hit and yield more speedup. At 60% cold-cache hits, the 27B shape reaches 1.94× A100 prefill and 1.68× A100 decode speedup. Quantization makes the original MLP cheaper and more challenging for cache to speedup: adding the same cache to GPTQ and SmoothQuant reduces GPU time, but does not improve wall-clock (Table 3).

These results motivate execution-aware cache selection. The current construction maximizes coverage under approximation-error and memory budgets. A latency-oriented extension should also account for routing overhead, predictor rank and memory locality, and the layer-specific balance between cache service and GPU fallback. The objective is to retain a hit rate in the 20–50% range that shorten wall-clock, rather than to increase hit rate uniformly.

3.5 How does the cache adapt to workload shift?

A shifted workload creates two problems: old predictors may begin extrapolating beyond their fitted support; the new workload may contain recurring local behavior absent from the original cache.

Restrict extrapolation. We keep the 4 GiB/layer cache fixed and vary only its predictor-support width on MMStar Math and Logical. Reducing the width from 1.4 to 1.0 raises Math accuracy from 51.60% to 60.67% while reducing coverage from 9.44% to 7.12%. On Logical, accuracy rises from 55.87% to 60.00% while coverage falls from 14.85% to 12.19%. The predictors and hierarchy do not change; more queries simply fall back to the original MLP.

Update cache. We expand the cache tree to cover the shifted workload. The graph-spectral coordinates constructed on the original distribution may leave parts of the new distribution insufficiently subdivided. Recomputing spectral coordinates with the new data can reveal finer subdivisions in those regions. We therefore augment the original anchor graph with new-workload inputs and obtain updated coordinates $s^\star_{\text{new}}(x)$. On separate calibration inputs, we fit a linear alignment R so that $R^\top s^\star(x)$ approximates $s^\star_{\text{new}}(x)$. We extract a few leading eigen directions of the residual $s^\star_{\text{new}}(x) - R^\top s^\star(x)$ and append their projections to the aligned original coordinates. The number of added coordinates is selected by coordinate-reconstruction error. We use the extended coordinates to

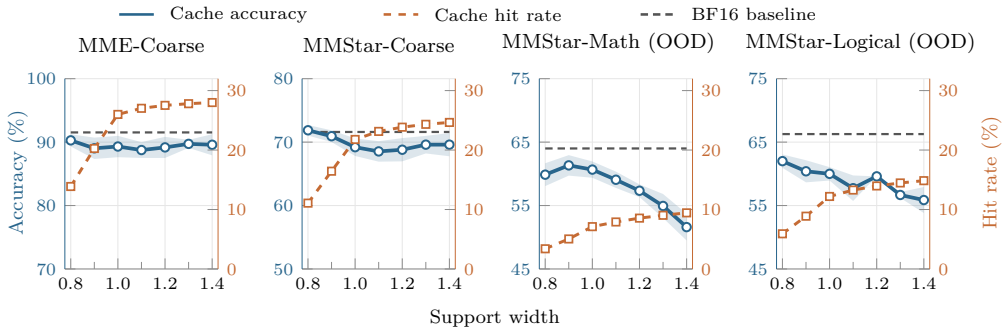


Figure 8: **Tightening predictor support width under workload shift.** Narrower support rejects more calls on MMStar Math and Logical, recovering accuracy while reducing cache coverage. The $B = 4$ GiB/layer, $\tau = .003$ cache is fixed and only predictor-support width varies. Blue shows accuracy, orange hit rate, and the gray dashed line the uncached BF16 baseline.

build a supplemental tree and fit local predictors on the new workload. The original cache remains unchanged. In the experiment at Qwen3.5-9B layer 16, with error budget $\tau = 0.003$ and memory budget $B = 4$ GiB for the old cache tree, $B = 0.1$ GiB for the added tree, the new tree use 3 added coordinates, the combined tree raises cache hit rate on MMStar Math from 8.02% to 11.68%.

4 Related Work

Exact reuse in LLM inference. KV caching reuses previously computed attention keys and values; vLLM's PagedAttention manages and shares the cached state [11]. SGLang's RadixAttention reuses KV states for identical prefixes [12], and RadixMLP deduplicates shared-prefix computation within a forward pass [13]. Such exact reuse preserves existing computation results. MLP Cache instead approximates new MLP outputs by sharing predictor parameters across distinct activations.

Learned memory and conditional computation. PKM, UltraMem, and LookupFFN combine learned addressing with stored values or parameters [1, 2, 14]; Engram uses token n-gram lookup [3]; MoLE and MemoryLLM obtain token-indexed precomputation by making the relevant component inputs depend on token embeddings [15, 16]. Sparse MoE and PEER likewise access only part of a larger parameter collection for each input [17, 18]. Learned memory and sparse experts establish the broader memory–computation trade-off. Related approaches also start from existing models: MoEfication partitions pretrained FFN parameters and routes inputs to subsets [19]. MLP Cache instead fits separate local predictors to a fixed operator's observed outputs, retains the original MLP as fallback.

Spectral representations for addressing and regression. Normalized cuts and spectral hashing use graph spectra for partitioning and binary addressing [20, 21]; spectral series regression uses eigenfunctions as prediction features [22]. We use coordinate signs to define nested regions and continuous coordinates to fit local MLP predictors.

Quantization and execution kernels. GPTQ, SmoothQuant, AWQ, and FLUTE reduce the cost of operator evaluation through quantization and kernel design [23, 24, 25, 26]. Quantization reduce the cost of an evaluation, whereas a cache replaces selected evaluations with local predictions. We evaluate combinations in which hits use cached predictors and misses use the quantized backbone.

5 Conclusion and Future Work

Conclusion. MLP Cache turns deployment-specific redundancy in frozen MLPs into a post-hoc memory–compute trade by materializing recurring local functions. The amount of reusable computation depends on the workload and cache budget. Speedup grows when each avoided MLP call is more expensive, as in larger models or hardware with lower MLP throughput. On Qwen3.5-9B, MLP Cache reuses about 25% of enabled-layer MLP calls with only 1–2% accuracy drop.

Future work. Future work should make cache construction service-aware and adaptive to workload shift. MLP Cache complements quantization and optimized kernels, which make operators cheaper, and sparse or mixture-of-experts models, which allocate computation conditionally; post-hoc caching instead asks which recurring computation is worth materializing.

6 AI USE STATEMENT

In this work, we used generative AI tools for implement methods, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis. We have not used generative AI tools for help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, interpret results, and generate synthetic data sets, assist in the writing of proofs are not applicable to this work. Additionally, we used generative AI tools for create or modify scientific figures or images, create or edit software code, creation of artifacts, draft parts of a research paper, summarize or analyse existing literature, brainstorming, sourcing/searching for information, edit a research paper to improve readability, identify relevant literature, format references, suggest a structure for a research paper, propose a title or keywords for a research paper. We have reviewed all AI-assisted work. LLM-generated paper drafts were manually corrected, and LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

[1] 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. URL https://arxiv.org/abs/1907.05242.

[2] 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. URL https://arxiv.org/abs/2411.12364.

[3] Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, Han Zhang, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026. URL https://arxiv.org/abs/2601.07372v1.

[4] Davis Blalock and John Guttag. Multiplying matrices without multiplying. In *International Conference on Machine Learning*, 2021. URL https://proceedings.mlr.press/v139/blalock21a.html.

[5] Yusuke Sekikawa and Teppei Suzuki. Irregularly tabulated MLP for fast point feature embedding. *arXiv preprint arXiv:2011.09852*, 2020. URL https://arxiv.org/abs/2011.09852.

[6] Qwen Team. Qwen3.5-9B model card. Hugging Face, 2026. URL https://huggingface.co/Qwen/Qwen3.5-9B. Accessed September 4, 2026.

[7] HuggingFaceM4. The cauldron dataset card. Hugging Face, 2024. URL https://huggingface.co/datasets/HuggingFaceM4/the_cauldron. Accessed September 4, 2026.

[8] Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, Yunsheng Wu, Rongrong Ji, Caifeng Shan, and Ran He. MME: A comprehensive evaluation benchmark for multimodal large language models. *arXiv preprint arXiv:2306.13394*, 2023. URL https://arxiv.org/abs/2306.13394.

[9] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. Are we on the right way for evaluating large vision-language models? *arXiv preprint arXiv:2403.20330*, 2024. URL https://arxiv.org/abs/2403.20330.

[10] DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026. URL https://arxiv.org/abs/2609.19969.

[11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180.

[12] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2312.07104.

[13] Michael Feil and Julius Lipp. RadixMLP—intra-batch deduplication for causal transformers. *arXiv preprint arXiv:2601.15013*, 2026. URL https://arxiv.org/abs/2601.15013.

[14] Zhanpeng Zeng, Michael Davies, Pranav Pulijala, Karthikeyan Sankaralingam, and Vikas Singh. LookupFFN: Making transformers compute-lite for CPU inference. In *International Conference on Machine Learning*, 2023. URL https://proceedings.mlr.press/v202/zeng23a.html.

[15] Shibo Jie, Yehui Tang, Kai Han, Yitong Li, Duyu Tang, Zhi-Hong Deng, and Yunhe Wang. Mixture of lookup experts. In *International Conference on Machine Learning*, 2025. URL https://proceedings.mlr.press/v267/jie25b.html.

[16] Ajay Jaiswal, Lauren Hannah, Han-Byul Kim, Duc Hoang, Arnav Kundu, Mehrdad Farajtabar, and Minsik Cho. MemoryLLM: Plug-n-play interpretable feed-forward memory for transformers. *arXiv preprint arXiv:2602.00398*, 2026. URL https://arxiv.org/abs/2602.00398.

[17] 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. URL https://arxiv.org/abs/1701.06538.

[18] Xu Owen He. Mixture of a million experts. *arXiv preprint arXiv:2407.04153*, 2024. URL https://arxiv.org/abs/2407.04153.

[19] Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. MoEfication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, 2022. URL https://aclanthology.org/2022.findings-acl.71/.

[20] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. doi: 10.1109/34.868688.

[21] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, 2008. URL https://proceedings.neurips.cc/paper/2008/hash/d58072be2820e8682c0a27c0518e805e-Abstract.html.

[22] Ann B. Lee and Rafael Izbicki. A spectral series approach to high-dimensional nonparametric regression. *Electronic Journal of Statistics*, 10(1):423–463, 2016. doi: 10.1214/16-EJS1112. URL https://arxiv.org/abs/1602.00355.

[23] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.17323.

[24] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, 2023. URL https://proceedings.mlr.press/v202/xiao23c.html.

[25] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Conference on Machine Learning and Systems*, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

[26] Han Guo, William Brandon, Radostin Cholakov, Jonathan Ragan-Kelley, Eric P. Xing, and Yoon Kim. Fast matrix multiplications for lookup table-quantized LLMs. *arXiv preprint arXiv:2407.10960*, 2024. URL https://arxiv.org/abs/2407.10960.