READ ONCE, REUSE ACROSS LLM FAMILIES: KV CACHE SHARING WITH CACHEPORT

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-model LLM applications repeatedly process the same document or conversation when passing context between models. Reusing key–value (KV) caches could avoid this work, but differences in tokenization, layer and head structure, and learned representations prevent direct reuse across model families. We introduce **CachePort**, a framework for cross-family cache reuse that avoids constructing a complete native receiver cache. CachePort aligns states at shared text boundaries, translates them through measured layer correspondences and full-width affine maps, and combines a lightweight receiver adapter with limited native recomputation. Both models' pretrained weights remain frozen; only the translators and the adapter are trained. Experiments spanning Llama, Qwen, and Mistral demonstrate useful context transfer for language modeling, question answering, and summarization, with quality depending on the transfer direction and recomputation budget. A receiver adapter also supports a source excluded from its training when paired with newly fitted translators. For Llama-3.1-8B to Mistral-7B transfer, an optimized receiver-computation path achieves **1.2–5.3**$\times$ speedups over full prefill across tested context lengths, excluding serving-system overhead. Separately, a vLLM integration reduces receiver handoff latency from 123.9 to 75.8 ms at an 8k-token context with a perplexity increase of 0.30 relative to full prefill. These results establish a practical quality–latency trade-off for cross-family context reuse.

1 INTRODUCTION

Modern LLM applications increasingly coordinate multiple models through cascades and multi-agent workflows, where successive models often process the same document, code context, or conversation history (Chen et al., 2023; Aggarwal et al., 2024; Wang et al., 2025; Wu et al., 2023; Hadfield et al., 2025). Each model first performs *prefill* to construct a key–value (KV) cache that its attention layers consult during generation. This computation can dominate time-to-first-token for long inputs (Kwon et al., 2023; Patel et al., 2024; Zhong et al., 2024). However, KV states depend on the producing model's weights, architecture, and tokenizer, making them generally incompatible across models (Fu et al., 2026). As illustrated in Figure 1 (a), a conventional handoff passes the shared context as text, leaving the receiver to rebuild its own KV cache. The receiver thus repeats prefill over material already processed by the source, adding computation and delaying generation.

Prior work reduces repeated computation under several compatibility assumptions. Prefix caching reuses identical prefixes within one model (Zheng et al., 2024; Gim et al., 2024), while KVCOMM extends reuse to differing contexts among agents sharing a model (Ye et al., 2026). DroidSpeak transfers caches between fine-tuned variants of a shared base checkpoint (Liu et al., 2024), and Heo et al. (2026) fit linear mappings between checkpoints within the same model family. Cache fusion instead projects source states and combines them with a complete native receiver cache (Fu et al., 2026). We study **cross-family prefill replacement**: transferring caches between different model families, such as Llama and Qwen, without first constructing a complete native receiver cache. This distinction determines the computational benefit: if the receiver must still process the entire context itself, transferred states can at most refine its cache and cannot remove the prefill. Replacement, by contrast, lets the receiver's computation fall from the full context to a bounded number of positions, which is what makes sharing pay in a serving system. This setting requires handling differences in tokenization and learned representations beyond the compatibility assumed by within-family reuse.

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.

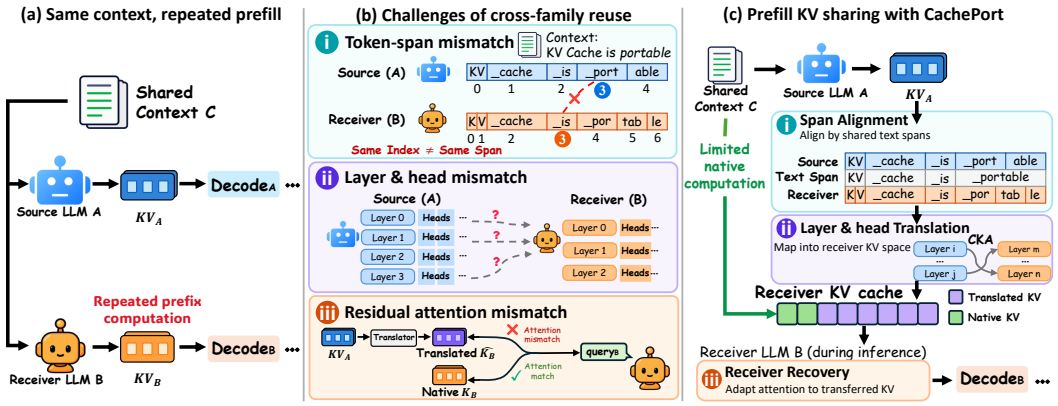


Figure 1: **From repeated prefill to cross-family KV cache sharing.** (a) In text-based handoffs, the source and receiver independently prefill the same context to construct their own KV caches. (b) Cross-family reuse must address mismatched token boundaries, layer and head correspondence, and residual translation errors that can alter how the receiver attends to the context. (c) CachePort aligns and translates the source cache, combines the transferred states with a bounded amount of receiver-native computation, and adapts the receiver to read the resulting hybrid cache without a full receiver prefill. Purple and green denote translated and native KV states, respectively. Token splits, layer structures, and cache proportions are illustrative.

Figure 1 (b) illustrates three obstacles to using a source model's cache in a different receiver. **First, token indices do not identify the same text across tokenizers.** Different token boundaries produce different cache lengths, so copying entries by index can assign source states to receiver positions representing different content. **Second, text alignment does not establish representation correspondence.** Models may differ in depth, head geometry, and feature coordinates; neither matching layer indices nor pairing heads directly determines how to translate their states. **Third, translated states may remain incompatible with the receiver's attention.** The receiver's queries were learned for its native cache: key errors change attention weights, while value errors change the retrieved information. Matching cache dimensions alone therefore does not preserve generation quality.

We introduce **CachePort** to address these challenges through a three-stage process illustrated in Figure 1 (c). First, CachePort aligns states at shared character-end boundaries and separates key translation from positional rotation. Second, it uses measured layer similarity to guide full-width affine maps, translating aligned states into the receiver's cache structure without imposing head-to-head pairing. Third, a lightweight receiver adapter learns to use the translated cache, while selective recomputation supplies receiver-computed states at a limited number of positions to improve quality. Both models' pretrained weights remain frozen; what is trained are the per-pair translators and one lightweight adapter per receiver, fitted offline from the two models' caches of unlabeled text and reused across requests. Together, these components enable generation without a full receiver prefill and provide an explicit trade-off between quality and computation.

Our contributions are:

- **A cache interface across tokenizers.** We align source and receiver states at shared character-end boundaries and separate key translation from rotary position encoding, preserving each model's tokenizer.
- **Translation guided by layer and head correspondence.** We combine measured layer routing with full-width affine maps. Analysis and ablations support this design: independently pretrained models exhibit distributed cross-head mappings, and constraining translation to fixed head pairs reduces quality.
- **Receiver adaptation with budgeted recomputation.** We jointly train source-specific translators and one adapter per receiver, shared across sources. Selective recomputation further improves quality, with query-selected refresh outperforming tail refresh on multi-hop QA at equal token budgets.

- **Evaluation of replacement quality and serving efficiency.** Language-modeling and downstream experiments demonstrate useful context transfer across model families. Our inference engine integration shows latency gains for sufficiently long contexts and aggregate throughput gains under shared GPU load, while identifying regimes where handoff overhead outweighs prefill savings.

2 Background and Related Work

Prefill replacement versus cache fusion. We distinguish two ways a receiver can use another model's KV states. *Replacement* lets the receiver generate from transferred states, supplemented by a bounded amount of native computation, without first building a complete native prefix cache. *Fusion* adds transferred states to a cache the receiver has already computed, so the receiver's prefill is retained. This paper studies replacement.

Model-pair classes. We classify model pairs by the compatibility a receiver inherits, since this determines which transfer methods apply. ***Same-checkpoint*** pairs share base weights (a base model and its instruction-tuned variant). ***Within-family*** pairs are pretrained separately but share a tokenizer, vocabulary, and architectural template, differing in scale (Qwen2.5-3B and 7B; Llama-3.1-8B and Llama-3.2-3B). ***Cross-family*** pairs come from independent pretraining lineages with different tokenizers and vocabularies (Llama, Qwen, Mistral). The boundary is the tokenizer and lineage, not the cache shape: Llama-3.1-8B and Mistral-7B-v0.3 have identical KV geometry yet are cross-family, whereas Qwen2.5-3B and 7B differ in depth and width yet are within-family.

Reuse within a model. Beyond identical-prefix reuse (Kwon et al., 2023; Zheng et al., 2024) and cross-context reuse among agents sharing a model (Ye et al., 2026), TurboRAG assembles precomputed context chunks, and CacheBlend and CacheClip recompute selected states to repair the discrepancies that recomposition introduces (Lu et al., 2025; Yao et al., 2025; Yang et al., 2025). These methods show that native computation is best spent on selected positions rather than the whole context, but the states they repair were produced by the model that consumes them. CachePort applies the same principle after cross-model translation.

Transfer across models. Same-checkpoint and within-family transfer exploit inherited compatibility: DroidSpeak reuses caches among fine-tuned variants and recomputes sensitive layers (Liu et al., 2024), SCD repairs same-architecture caches through subspace reconstruction (Ma et al., 2026), and Heo et al. (2026) fit closed-form linear maps within a family. PrefillShare and ICaRus construct compatibility by training models around a shared, frozen prefill module (Woo et al., 2026b;a), which changes the models themselves. Learned translation into another model's representation space has been studied for fusion, where C2C projects source states into a complete native receiver cache (Fu et al., 2026), and for conditioning generation on transformed states (Rossi et al., 2026; Chen et al., 2026; Dery et al., 2026). Closest to our setting, Dery et al. (2026) learn a shared latent space with one input and one output adapter per model, demonstrated within a common architecture family and tokenizer; the pool grows linearly in adapters, whereas CachePort's directional maps trade that scalability for translation guided by pair-specific layer correspondence. RelayCaching repairs decode-phase caches passed along agent chains (Geng et al., 2026), whereas CachePort replaces computation over a shared input prefix. None of these methods performs replacement between independently pretrained families with different tokenizers without modifying base weights, which is the setting of Table 1; Table 4 in the appendix summarizes the compatibility each method assumes and the receiver computation it requires.

3 CachePort

3.1 Problem Setup and Overview

We consider a source model A that has already processed a shared context C and a receiver model B that must generate conditioned on the same text. The models may differ in tokenizer, depth, KV-head geometry, and pretrained weights. CachePort constructs a receiver-compatible cache from the source states, allowing the receiver to generate without first computing a complete native prefix cache. Both models' pretrained weights remain frozen. The receiver may compute an initial segment

of S tokens and refresh up to B additional context positions; this limited computation complements the transferred states.

We use p and q for source and receiver token positions, and i and j for their layers. The spans $s_A(p)$ and $s_B(q)$ refer to character offsets in the same text C. We write P for the receiver's context length and d_A, d_B for the concatenated KV-head widths. Keys and values are denoted by K and V, with $R_A(p)$ and $R_B(q)$ denoting the models' head-wise rotary transforms.

CachePort has three components. Content-anchored alignment pairs states at shared character endpoints and separates feature translation from rotary position encoding. Layer routing selects a source layer for each receiver layer, whose states are translated by full-width affine maps. Receiver adaptation and selective recomputation then recover quality: a shared receiver adapter learns to use translated states, while a limited set of positions is computed by the receiver itself.

Layer routes and translators are fitted offline for each directed model pair, together with one adapter shared across the sources of a receiver. At inference, CachePort aligns and translates the available source states, combines them with selectively computed receiver states, and generates from the resulting hybrid cache.

3.2 Content-Anchored Span Alignment

The same text can occupy different token positions under different tokenizers. To pair the source and receiver states, we use the character boundary at which each token ends. Tokens ending at the same boundary have consumed the same text prefix, even when their individual spans differ. These boundaries let us align the two caches while preserving each model's tokenization. We obtain offsets from both tokenizers on the same input string, excluding special tokens and zero-length spans. If several tokens share an endpoint, we retain the last nonempty token. We pair positions whose character endpoints coincide:

$$\mathcal{A}(C) = \{(p, q) : \mathrm{end}(s_A(p)) = \mathrm{end}(s_B(q))\}. \tag{1}$$

For example, consider `KV_cache_is_portable`, tokenized as `KV|_cache|_is|_port|able` by the source and `K|V|_cache|_is|_por|tab|le` by the receiver. The two streams share endpoints after `KV`, `KV_cache`, `KV_cache_is`, and the complete string. With zero-based indices, the anchors are $(0, 1)$, $(1, 2)$, $(2, 3)$, and $(4, 6)$. The receiver tokens `K`, `_por`, and `tab` have no source token ending at the same character offset. Shared endpoints cover 96–99.8% of receiver positions across our evaluated pairs; identical tokenization gives identity alignment over the retained tokens.

The aligned keys carry the positional rotation of the source model. Since RoPE is a known orthogonal transform, we can peel off this rotation analytically before translation and apply the receiver's rotation afterwards (Su et al., 2024). Values do not require rotary operation. For an aligned pair (p, q), the transfer takes the form of

$$\begin{aligned} \widehat{K}_{B,j,q} &= R_B(q)\mathcal{T}_j^K\left(R_A(p)^\top K_{A,i,p}\right), \\ \widehat{V}_{B,j,q} &= \mathcal{T}_j^V\left(V_{A,i,p}\right), (p, q) \in \mathcal{A}(C). \end{aligned} \tag{2}$$

where 3.3 specifies the selection of the source-layer and translators $\mathcal{T}_j^K, \mathcal{T}_j^V$, whose outputs follow the receiver's KV-head layout. This separates explicit rotary encoding from learned feature translation; positional information inherited from earlier layers remains in the states.

Positions without a shared endpoint have no directly aligned source state. Our reference construction omits these positions unless they are selected for native computation. Retained states preserve their original receiver position IDs for rotary encoding and causal attention, regardless of their physical cache slots.

3.3 Layer Routing and Full-Width Cache Translation

Text alignment pairs positions, but does not determine which source layer should supply each receiver layer or how their attention heads correspond. We examine these two questions across three model pairs: Llama-3.1-8B base and its instruction-tuned variant, Llama-3.1-8B and Llama-3.2-3B within the same family, and Llama-3.1-8B and Qwen2.5-7B across families. Figure 2 compares their layer correspondence and the head structure of fitted translation maps.

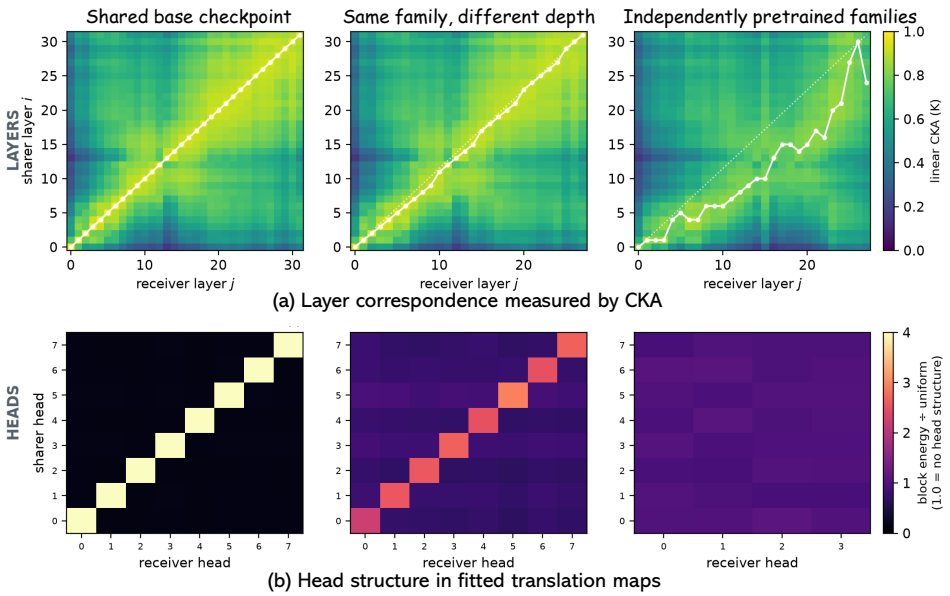


Figure 2: **Layer and head correspondence across model pairs.** Columns show a base/instruction-tuned pair, a within-family pair with different depths, and a cross-family pair. **(a)** Linear CKA between character-aligned pre-RoPE key states. Solid lines trace the highest-CKA source layer for each receiver layer; dotted lines show proportional-depth assignments. **(b)** Weight energy in source–receiver head blocks of fitted affine maps.

To identify corresponding layers, we compare the KV representations that the two models produce on the same text. We use centered kernel alignment (CKA), which measures similarity in the structure of these representations without requiring equal feature dimensions or matching feature coordinates (Kornblith et al., 2019). Figure 2(a) shows the resulting key-state similarities: the base/instruct pair follows a diagonal correspondence, while the within-family pair remains close to proportional depth. The cross-family route departs from this pattern, with several receiver layers selecting the same source layer and some assignments reversing layer order. Although the latter two pairs both have $32 \to 28$ layers, their routes differ substantially. We therefore estimate layer correspondence from aligned states for each model pair rather than prescribe it from layer indices or depth ratios.

We collect character-aligned training states and center each feature matrix across positions. For two such matrices, linear CKA is

$$\mathrm{CKA}(X, Y) = \frac{\|X^\top Y\|_F^2}{\|X^\top X\|_F \|Y^\top Y\|_F}, \tag{3}$$

where corresponding rows refer to aligned positions and the feature widths may differ. Each receiver layer selects its source layer independently:

$$\sigma^z(j) = \arg\max_i \mathrm{CKA}\left(\bar{X}^z_{A,i}, \bar{X}^z_{B,j}\right), \qquad z \in K, V. \tag{4}$$

The barred matrices contain aligned, centered states, with keys taken before RoPE. We estimate separate routes for keys and values, allowing repeated and non-monotone source-layer assignments. These routes are fixed after offline estimation.

Selecting a layer still leaves the correspondence between its heads unresolved. Figure 2(b) examines this structure through the weight energy of fitted affine maps, partitioned into source–receiver head blocks. The base/instruct pair has a strong diagonal pattern, and the within-family pair retains a weaker one. In the cross-family pair, the weights are spread much more evenly across source heads. This pattern motivates allowing each receiver head to use all source heads rather than imposing the head pairing seen between related models.

We concatenate the source KV heads and translate the selected state with a full-width affine map:

$$\mathcal{T}^z_j(x) = W^z_j x + b^z_j, \qquad W^z_j \in \mathbb{R}^{d_B \times d_A}, \quad z \in K, V. \tag{5}$$

Each map accommodates the source and receiver KV widths without requiring equal head counts or dimensions. We initialize it by ridge regression against aligned native receiver states, using pre-RoPE keys and unrotated values as targets, and then optimize it jointly with the receiver adapter described next. At inference, the selected source states pass through these fitted maps; their outputs are reshaped into receiver heads, with RoPE applied to the translated keys as in Equation 2.

3.4 Receiver Adaptation and Selective Recomputation

Layer routing and feature translation produce states in the receiver's cache layout, but do not fully recover its native representations. In our residual analysis (Appendix G), ridge correctors fitted to decode-time features remove about one point of an attention-output error worth 38% of the output norm. We therefore train the receiver to use the translated states and supplement them with a limited amount of native computation.

Adapting the receiver. The receiver's attention projections were trained to read its own cache. We add rank-64 LoRA adapters (Hu et al., 2021) to the query and attention-output projections so that the receiver can adjust both how it attends to translated keys and how it uses the retrieved values. Both models' pretrained weights remain frozen.

We train one adapter per receiver jointly with the source-specific translators. A frozen teacher uses the receiver's complete native cache, while the student uses the transferred cache and the adapter. Both predict the same continuation tokens under teacher forcing. The training objective is

$$\mathcal{L}_{\mathrm{KD}} = \mathbb{E}_{A,C,y}\left[\frac{1}{|y|}\sum_{t=1}^{|y|} w_t, D_{\mathrm{KL}}\left(p_B(\cdot \mid C, y_{<t}), \|, p_{\mathrm{CP}}(\cdot \mid C, y_{<t}; A)\right)\right], \tag{6}$$

where p_B is the native receiver distribution and p_{CP} is the distribution obtained with the transferred cache. The weights w_t, normalized to unit mean, emphasize tokens whose prediction benefits from the context. They are derived from the clipped difference between the receiver's token losses with a sink-only cache and with a full native cache.

Training samples multiple source models, sharing the receiver adapter while retaining separate translators for each directed pair. Layer routes remain fixed, and translators are updated at a lower learning rate than the adapter. Half of the training steps include a short native tail in addition to the sink and translated states, exposing the adapter to both refreshed and unrefreshed caches. No downstream task labels or source-generated responses are used.

The adapter is enabled for requests using transferred caches, including hybrid caches with native refresh. Requests using a complete native cache use the original receiver without the adapter. We also evaluate transfer to a source excluded from adapter training by fitting its translators while keeping the receiver adapter fixed.

Allocating native computation. Adaptation improves the use of translated states but leaves a quality gap. CachePort therefore computes the first S context positions natively and selects up to B additional positions outside this initial segment for refresh. The initial segment preserves the receiver's attention-sink states; the additional budget targets context that is important to the current generation. A zero refresh budget still includes the native sink.

We consider two placement policies. Tail refresh selects the final B context positions, providing native states for the recent context used in continuation. Query-selected refresh instead directs computation toward evidence relevant to an available question. The question attends to the transferred cache, and we aggregate attention over its tokens, heads, and layers to score context positions. We then greedily select non-overlapping 64-token windows without exceeding the refresh budget.

Selected windows are recomputed from left to right, attending to the hybrid cache available at earlier positions. Their receiver-computed states replace translated entries or supply positions without anchors. These states need not equal the corresponding entries of a full native cache, since their left context may itself contain translated states. All entries retain their original receiver position IDs.

The budget B counts refreshed context positions rather than total execution cost. Query-selected refresh additionally requires a probing pass, and translation and cache assembly incur their own overhead. We compare placement policies at equal refresh budgets for downstream quality.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and transfer settings. Our primary pool contains five models from three families: Llama-3.1-8B (L8) and Llama-3.2-3B (L32) from Llama, Qwen2.5-7B (Q7) and Qwen2.5-3B (Q3) from Qwen, and Mistral-7B-v0.3 (MI) from Mistral. We classify transfers within the Llama or Qwen groups as within-family and transfers between these three groups as cross-family.

Tasks and data. We fit layer routes, initialize translators, and train receiver adapters on the WikiText-103 training split. Language-modeling evaluation uses 60 unseen test documents with 512-token receiver prefixes and up to 512 continuation tokens. Longer-context experiments use disjoint filtered test sets, with lengths and sample counts reported alongside the results. Downstream evaluation uses fixed subsets of SQuAD (959 examples), HotpotQA (500), and SAMSum (401); SQuAD and SAMSum are filtered by a minimum context-length requirement. We report perplexity for language modeling, normalized answer F1 for question answering, and ROUGE-L for summarization. No downstream task labels are used for training.

Execution and measurement. Live serving experiments run in vLLM on NVIDIA B200, with source and receiver engines sharing the GPU. The deployed path uses tail refresh and fills unanchored positions to construct a dense paged cache; the offline reference construction omits these positions. Downstream placement comparisons, including query-selected refresh, use the offline evaluation path. Appendix D describes the implementation. We measure median and p95 time to first token from cache handoff to the receiver's first generated token, including translation, cache assembly, native suffix computation, and adapter execution. The source's completed prefill precedes this interval. Under shared GPU load, we report source and receiver completion rates separately and their aggregate over the same measurement window. Quality comparisons use paired bootstrap confidence intervals over examples with 10,000 resamples. Experiment-specific budgets, sample counts, and workload settings accompany each result.

4.2 RESULTS

4.2.1 REPLACEMENT QUALITY

Table 1 places the transferred cache between the two references that define the task: the no-context floor and the receiver's own full prefill. With only the native sink, CachePort-B0 recovers 67–86% of the floor-to-full-prefill gap in every direction, and 128 refreshed tokens raise this to 83–92%; on L8→Q7 the refreshed cache reaches 6.935 against 6.738 with full prefill. Two controls separate transferred content from a favorable receiver prior: disabling the adapter on L8→Q7 raises PPL from 7.057 to 7.336, and a cache of the same shape from the wrong document raises it to 9.334, above the floor. Against prior transfer methods under the same protocol, CachePort-B0 has lower PPL in all five directions, by 0.5–1.1 PPL on the three cross-family pairs. Between fine-tune siblings its construction degenerates to verbatim reuse and performs like it, and a reduced configuration without adapter or refresh trails Family-Transfer on Q7→Q0.5B; both are reported in Appendix E together with every evaluated method and its caveats. The adapter also generalizes to a source it was not trained on: pairing Qwen3-8B with newly fitted closed-form maps and the unchanged Q7 adapter gives 7.608 at $B0$ and 7.317 at $B128$ against 6.740 with full prefill, about 0.3–0.5 PPL behind the in-training sources (Appendix F.4).

4.2.2 TASK FIDELITY AND REFRESH PLACEMENT

Near-native perplexity does not by itself guarantee that a task can still read what it needs. Table 2 evaluates three tasks on four directions with the offline path. Without refresh, the transferred cache lifts most task–direction cells above the no-context floor, with one exception (Q7→L8 on HotpotQA, 0.217 against a floor of 0.255), so refresh is not optional on hard retrieval. Native refresh then recovers most of the remaining gap, and where it is placed matters on multi-hop retrieval: at equal budgets, query-selected windows outperform tail refresh in all four HotpotQA directions, raising mean F1 from 0.331 to 0.381; on L8→Q7, F1 rises from 0.305 without refresh to 0.352 with a tail and 0.430 with query-selected windows, against 0.458 with full prefill. The two placements are equivalent on single-hop SQuAD and comparable on SAMSum. The remaining gap can nevertheless

Table 1: **Replacement quality against full prefill, no context, and prior transfer methods.** PPL↓ with 512-token prefixes. Floor is the receiver decoding from its native 16-token sink alone; Gold is the receiver's own full prefill. Prior is the strongest evaluated baseline in each row (Heo et al. (2026) for within-family scale transfer, Dery et al. (2026) for cross-family). CP-B0 and prior methods use the native sink without refresh; CP-B128 adds 128 refreshed tokens. Same-checkpoint pairs and the reduced Q7→Q0.5B configuration are in Appendix E.

Direction	Floor	Gold	Prior	CP-B0	CP-B128
Within-family scale transfer					
Q3→Q7	9.062	6.738	7.969	7.321	**7.089**
Q7→Q3	10.657	7.740	8.936	8.300	**8.087**
Cross-family transfer					
L8→Q7	9.062	6.738	8.129	7.057	**6.935**
Q7→L8	8.257	6.168	7.344	6.853	**6.519**
MI→L8	8.257	6.168	7.398	6.770	**6.505**

be substantial: Q7→L8 reaches 0.360 HotpotQA F1 with query-selected refresh against 0.500 with full prefill. Query-selected refresh is evaluated offline and is not scheduled in the serving engine, which uses tail refresh; equal refreshed-token budgets establish equal receiver computation, not equal runtime, since selecting windows requires the question before the cache is assembled.

Table 2: **Downstream fidelity and refresh placement.** Each cell reports Floor | Gold / CP-B0 / Tail / Query, where Floor is the receiver with its native sink only (no context). Metrics are F1 for SQuAD and HotpotQA and ROUGE-L for SAMSum. Tail and Query use equal budgets within each task. Results use the offline path. Bold marks the highest CachePort score in each cell, without implying statistical significance.

Direction	SQuAD $n = 959,\ B = 128$	HotpotQA $n = 500,\ B = 256$	SAMSum $n = 401,\ B = 128$
L8→Q7	.121 | .831 / .642 / .785 / **.792**	.177 | .458 / .305 / .352 / **.430**	.156 | .248 / .225 / **.240** / .239
Q3→Q7	.121 | .831 / .480 / **.748** / .735	.177 | .458 / .271 / .287 / **.351**	.156 | .248 / .208 / .235 / **.246**
Q7→L8	.315 | .725 / .395 / **.650** / .644	.255 | .500 / .217 / .322 / **.360**	.166 | .269 / .180 / **.258** / .254
L8→L32	.279 | .646 / .536 / **.624** / .614	.218 | .447 / .334 / .362 / **.384**	.158 | .252 / .226 / **.251** / .245

4.2.3 EFFICIENCY

Figure 3 compares CachePort's receiver-side computation with full prefill. On L8→MI, a 128-token refresh yields speedups ranging from $1.2\times$ to $5.3\times$ as the prefix length increases from 512 to 8k tokens. CachePort's latency grows from approximately 11 to 36ms over this range, substantially more slowly than full prefill: it translates the source cache and runs the receiver over only $S + B$ tokens. Refreshing up to 128 tokens adds little latency in the optimized implementation, while larger budgets increase the cost. Appendix I provides the measurement protocol, implementation breakdown, and refresh-budget sweep.

We also evaluate the end-to-end performance of CachePort integrated into the inference engine (Kwon et al., 2023). On L8→MI, handoff-to-first-token latency exceeds full prefill at 512 and 2k tokens, but selected refresh configurations become faster at 4k as prefill savings outweigh handoff overhead. At 8k, a 1,024-token refresh reduces median latency from 123.9 to 75.8ms ($1.6\times$), with PPL increasing from 4.660 to 4.961. On L8→Q7, a 128-token refresh improves aggregate source–receiver throughput by 1.03–$1.42\times$ at 2k, 1.17–$2.35\times$ at 4k, and 1.46–$3.47\times$ at 8k across the tested source loads. Under saturated source load, these gains accrue to the source rather than the receiver: at 8k, receiver throughput decreases from 4.13 to 3.70 requests/s, while source throughput increases from 4.39 to 8.72 requests/s. Thus, CachePort can improve overall serving throughput even when receiver throughput decreases. Appendix F.1 details the measurement scope, precomputed-sink setup, and full results.

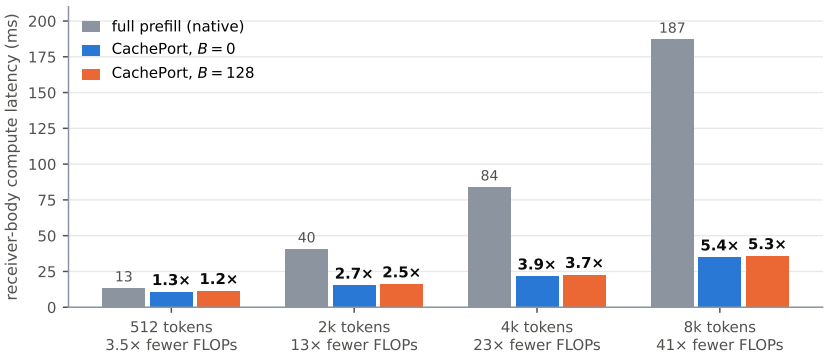


Figure 3: **Algorithmic speedup over native prefill.** Receiver-body compute latency, CachePort against a full prefill of the same P tokens. Bar labels give native $vs.$ CachePort; the axis labels give the analytic reduction in floating-point operations. Protocol in Appendix I.4.

4.2.4 Module Ablation

Table 3 tests each design decision against the reference that isolates it: the closed-form recipe for the two decisions that change the maps, and the deployed recipe at the same refresh budget for the two recovery mechanisms. Without the content coordinate of M1 the cache is little better than no context at all, since translation has nothing correctly placed to translate. Constraining M2 to a head-to-head map costs 0.64 PPL even though this pair's head counts match, which is the behavioral counterpart of the flat head-energy pattern in Figure 2. Removing M3's adapter costs 0.42 PPL on the deployed maps, and 128 refreshed tokens recover a further 0.25; the native positions M3 manages are the most fragile part of the cache, since retaining only the BOS token natively costs another 0.13 and translating every position including BOS fails outright (Appendix H). The tested correction methods remove little of the remaining error: the best decode-time corrector fitted on the translated states recovers about one point of an attention-output error worth 38% of the output norm (Appendix G), which is why recovery is receiver-side.

Table 3: **One design decision at a time on MI→L8.** PPL on 60 documents with 512-token prefixes; full prefill 6.168, no-context floor 8.255. Each row replaces one decision and is compared with the reference that differs from it in that decision only: M1 and M2 arms refit the maps in closed form and are compared with the closed-form recipe (6.858); M3 arms toggle one recovery mechanism of the deployed co-trained recipe and are compared with it at the same budget. Every arm is in Appendix H.

Decision tested	Reference → ablation	PPL↓	Δ
M1: character anchors vs. token-index alignment	$6.858 \rightarrow$	7.987	$+1.129$
M2: full-width map vs. block-diagonal head map	$6.858 \rightarrow$	7.501	$+0.643$
M3: adapter on vs. off, both $B{=}0$	$6.767 \rightarrow$	7.191	$+0.424$
M3: adding 128 refreshed tokens	$6.767 \rightarrow$	6.520	-0.247

5 Conclusion

CachePort enables KV cache sharing across LLM families without a full receiver prefill. It combines text-based alignment, layer-guided translation, and receiver adaptation with selective recomputation, while keeping pretrained weights frozen. Experiments show that transferred caches preserve useful context for language modeling and downstream tasks. The optimized receiver-computation path achieves 1.2–$5.3\times$ speedups over full prefill when serving-system overhead is excluded, while vLLM integration demonstrates latency and aggregate source-plus-receiver throughput benefits for sufficiently long contexts. These results show that models from different families can reuse useful information from one another's caches, reducing repeated context processing in multi-model workflows through a controllable quality–compute trade-off.

AI USE STATEMENT

We used AI-assisted tools to improve the clarity, organization, and language of the manuscript, to assist with literature discovery during the early stages of the research, and to support the development and debugging of experimental code. All literature references, technical claims, and AI-assisted code were independently verified and reviewed by the authors. The authors revised all AI-generated suggestions as needed and take full responsibility for the final manuscript, including its accuracy, originality, experimental results, and citations.

REPRODUCIBILITY STATEMENT

We provide an anonymous code repository containing the implementation of CachePort, including cache alignment, layer routing, KV translation, receiver adaptation, selective recomputation, and the evaluation pipeline. The repository also includes the configurations and scripts needed to reproduce the main experiments, ablations, and serving measurements reported in the paper. We use publicly available pretrained models and benchmark datasets, and specify the model pairs, training data, evaluation subsets, refresh budgets, hardware, and measurement protocol in Section 4 and the appendix. We will release the final code and experiment configurations publicly upon publication.

ETHICS STATEMENT

This work studies efficient KV-cache reuse across different families of large language models. Our experiments use existing benchmark datasets and involve no human subjects or collection of personal data. Because KV caches are derived from model inputs and may retain information about the underlying context, CachePort should not be interpreted as providing privacy or confidentiality guarantees; deployed systems should protect transferred caches according to the sensitivity of the original inputs. Transferred caches can also introduce a quality gap relative to native prefill, so task-specific validation is important for high-stakes applications.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *Advances in Neural Information Processing Systems*, 37:131000–131034, 2024.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Siyi Chen, Xiaoyan Zhang, Meng Wu, Jonathan Tremblay, Valts Blukis, Stan Birchfield, Rene Vidal, Alvaro Velasquez, Sijia Liu, and Qing Qu. See what i see, know what i think: Dense latent communication across heterogeneous agents. *arXiv preprint arXiv:2606.13594*, 2026.

Lucio M Dery, Zohar Yahav, Henry Prior, Qixuan Feng, Jiajun Shen, and Arthur Szlam. Latent space communication via kv cache alignment. *arXiv preprint arXiv:2601.06123*, 2026.

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. Cache-to-cache: Direct semantic communication between large language models. In *International Conference on Learning Representations*, volume 2026, pp. 43130–43158, 2026.

Yingsheng Geng, Yuchong Gao, Weihong Wu, Guyue Liu, and Jiang Liu. Relaycaching: Accelerating llm collaboration via decoding kv cache reuse. *arXiv preprint arXiv:2603.13289*, 2026.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. How we built our multi-agent research system. Anthropic Engineering Blog, June 2025. `https://www.anthropic.com/engineering/multi-agent-research-system`.

Taekyung Heo, Rasoul Shafipour, Ritchie Zhao, Maximilian Golub, Mohammad Mahdi Kamani, Ritika Borkar, Makesh Tarun Chandran, Pantea Zardoshti, and Bita Darvish Rouhani. Cross-model kv cache transfer in llm families: A closed-form linear mapping for prefill reuse. *arXiv preprint arXiv:2608.03893*, 2026.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pp. 3519–3529. PMlR, 2019.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph 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*, pp. 611–626, 2023.

Yuhan Liu, Yuyang Huang, Jiayi Yao, Shaoting Feng, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, et al. Droidspeak: Kv cache sharing for cross-llm communication and multi-llm serving. *arXiv preprint arXiv:2411.02820*, 2024.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. Turborag: Accelerating retrieval-augmented generation with precomputed kv caches for chunked text. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6599–6612, 2025.

Qianli Ma, Zhiqing Tang, Hanshuai Cui, Zhi Yao, and Weijia Jia. Semantic cache distillation: Efficient state transfer via reuse and selective patching. *arXiv preprint arXiv:2606.07684*, 2026.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

Maximillian Rossi, Prajwal Raghunath, and Eugene Wu. Latent cache flow: Model-to-model communication without text. *arXiv preprint arXiv:2605.22863*, 2026.

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

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, volume 2025, pp. 33944–33963, 2025.

Sunghyeon Woo, Jaeeun Kil, Hoseung Kim, Minsub Kim, Joonghoon Kim, Ahreum Seo, Sungjae Lee, Minjung Jo, Jiwon Ryu, Baeseong Park, et al. Icarus: Identical cache reuse for efficient multi model inference. *arXiv preprint arXiv:2603.13281*, 2026a.

Sunghyeon Woo, Hoseung Kim, Sunghwan Shim, Minjung Jo, Hyunjoon Jeong, Jeongtae Lee, Joonghoon Kim, Sungjae Lee, Baeseong Park, Se Jung Kwon, et al. Prefillshare: A shared prefill module for kv reuse in multi-llm disaggregated serving. *arXiv preprint arXiv:2602.12029*, 2026b.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Bin Yang, Qiuyu Leng, Jun Zeng, and Zhenhua Wu. Cacheclip: Accelerating rag with effective kv cache reuse. *arXiv preprint arXiv:2510.10129*, 2025.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the twentieth European conference on computer systems*, pp. 94–109, 2025.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, et al. Kvcomm: Online cross-context kv-cache communication for efficient llm-based multi-agent systems. *Advances in Neural Information Processing Systems*, 38:17882–17928, 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 193–210, 2024.

A LIMITATIONS.

On difficult retrieval tasks, CachePort's KV translation does not fully preserve native-cache quality. And short contexts may not amortize handoff overhead. Additionally, translators are fitted per directed model pair, increasing adaptation and storage requirements as the model pool grows.

B COMPARISON WITH PRIOR KV REUSE

Table 4: Positioning of KV reuse methods. "Full" means that the receiver still prefills the complete context; "token" and "layer" denote the axis of selective native recomputation.

method	model boundary	use of source	recompute	fixed pool?
CacheBlend (Yao et al., 2025)/CacheClip (Yang et al., 2025)	same model	replace	token	no
DroidSpeak (Liu et al., 2024)/SCD (Ma et al., 2026)	fine-tune/arch.	replace	layer	no
PrefillShare (Woo et al., 2026b)/ICaRus (Woo et al., 2026a)	co-trained	replace	none	yes
C2C (Fu et al., 2026)	cross-family	fuse	full	no
Latent transfer (Dery et al., 2026)	mostly same family	replace	none	no
CACHEPORT	**cross-family**	**replace**	**token**	**no**

C SUPPLEMENTARY ANALYSIS

C.1 MODEL SCALE AND CAPABILITY EFFECTS

On the within-family ladder into Qwen2.5-7B (Gold PPL 6.74, test corpus, n=60), KV-share PPL decreases from 7.91 with a 0.5B sharer to 7.53 at 1.5B, 7.32 at 3B, and 7.19 at 14B. The trend is monotone in log-size and saturates near the receiver's own scale; there is no "cache upgrade," as the 14B cache does not improve on Gold. Capability at fixed size contributes little: Qwen1.5-7B (MMLU $\approx$61) gives PPL 7.31, close to the same-generation size curve despite matching the benchmark capability of Qwen2.5-1.5B (PPL 7.53, MMLU $\approx$60). Small receivers are also usable: L8$\rightarrow$Qwen2.5-0.5B gives PPL 13.62 versus Gold 12.26, and L8$\rightarrow$Qwen2.5-1.5B gives 9.42 versus Gold 8.80. Size and source-KV dimensionality are collinear in this pool, so this experiment does not separate their effects.

C.2 LONG-CONTEXT POSITIONAL ATTRIBUTION

Replacing one equal-length block of an otherwise-gold 4k cache yields approximately 1% PPL damage for each 512-token block except the final block, which contributes 3.4%. Block damages are nearly additive ($\sum = 11.1$ points versus 12.0 when the entire middle is translated), and per-position map R^2 is flat. The length effect is therefore not position-distribution shift or strongly super-additive attention noise; it is a broad translation background plus a recency spike.

This attribution predicts the measured policies. Against Gold PPL 6.58, a 25% proportional budget gives 6.91, while a fixed 256-token budget with long-context distillation gives 7.02. Progressive front-to-back write-back reaches 6.79 after roughly eight turns and converges to 6.62 at a constant per-turn cost. Finally, a refresh-only control obtains deceptively strong perplexity because that metric emphasizes recent tokens, but cannot support content tasks without the translated middle. Long-context QA makes the distinction explicit: mid-context answers suffer a 66% relative drop, motivating the query-aware policy used in the main results.

C.3 ANCHOR COVERAGE AND GOLD-DROP BY DIRECTION

C.4 CROSS-MODEL HEAD CORRESPONDENCE

The affine map in Equation 5 is learned over the concatenated KV-head width. To inspect whether it preserves a one-to-one head structure, we reshape each fitted matrix to $[n_a, d_h, n_b, d_h]$, compute the squared Frobenius mass of every source–receiver head block, average over layers, and normalize

Table 5: Character-anchor coverage and the measured cost of dropping unanchored positions (test corpus, n=60; 512-token prefixes). Gold-drop removes exactly the unanchored positions from the receiver's own cache. Same-tokenizer directions (Qwen↔Qwen, Llama↔Llama) have ≈100% coverage and are omitted.

direction	coverage	mean / max gap	gold-drop ΔPPL
L8→Q7, L8→Q3	96.1%	1.6 / 2	+0.11%, +0.12%
MI→Q7, MI→Q3	99.2%	1.1 / 3	+0.13%, +0.10%
Q7→L8, Q7→L32	99.8%	1.5 / 5	+0.01%, −0.01%
MI→L8, MI→L32	99.0%	1.1 / 3	+0.07%, +0.09%
L8→MI	87.3%	1.2 / 3	+0.82%
Q7→MI	90.9%	1.1 / 5	+0.79%

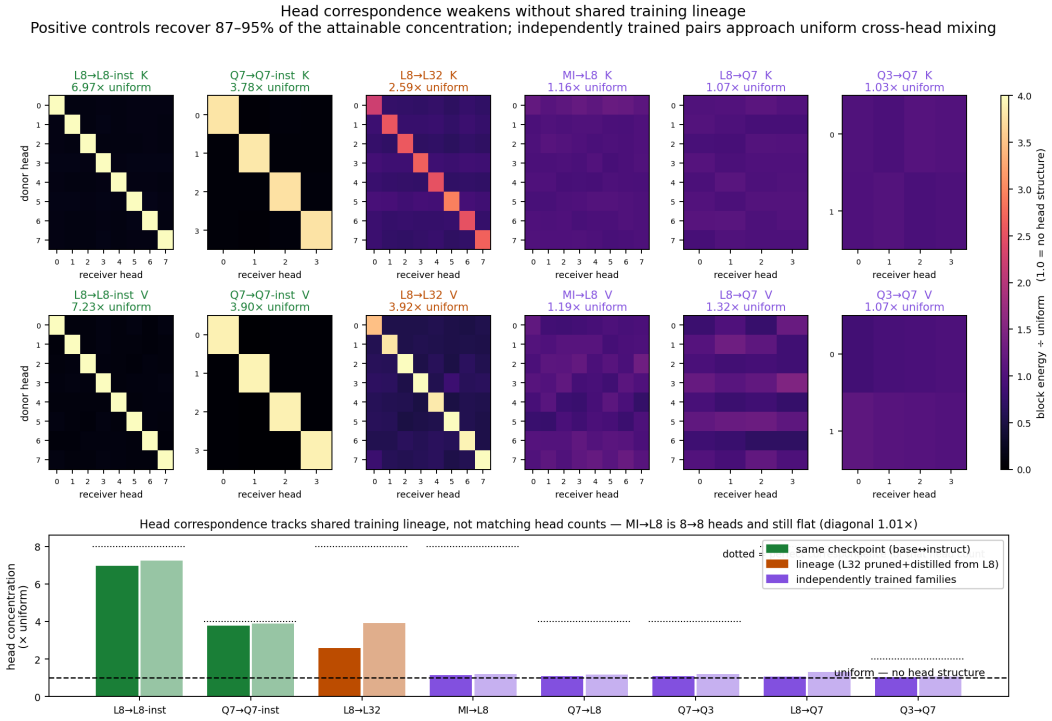


Figure 4: **Head correspondence follows shared training lineage.** Same-checkpoint controls recover 87–95% of the maximum possible concentration and form diagonal maps, validating that the statistic detects correspondence when it exists. The related L8→L32 pair retains intermediate structure. Independently trained pairs are close to uniform mixing (1.03–1.32×), including MI→L8, whose equal $8 \rightarrow 8$ head counts nevertheless have diagonal concentration 1.01×. The analysis supports not imposing a fixed source–receiver head pairing; it does not preclude a more complex nonlinear correspondence.

within each receiver head. Concentration is the strongest source head's mass divided by the uniform value $1/n_a$.

The full-width map therefore adapts to both regimes: it can recover a block-diagonal solution when head identity is inherited and can mix source heads when independent training removes that structure. This observation is a design rationale rather than a causal performance comparison with every possible head-constrained translator.

D System Integration

Section 3 determines what the hybrid cache contains. This section describes how a serving engine produces that cache and runs a request through it, using vLLM (Kwon et al., 2023) with a source engine and a receiver engine on one GPU. Neither engine's model code, attention kernels, or scheduler is modified.

D.1 Cache Handoff and Assembly

The source runs unchanged weights. When it prefills a context, a hook exports the K/V projections of the anchored positions and publishes them under a content key derived from the source's token IDs; only anchor rows are published, not the whole cache. A live engine stores keys after rotation, so the exported keys still carry the source's RoPE at the source's positions.

The receiver's connector maps those rows directly out of the source's memory through CUDA IPC, so on a shared GPU nothing is copied; across devices the same rows would travel over the interconnect. It then applies §3.3 as a single pass rather than layer by layer: the keys are un-rotated with the *source's* rotary table at the source's true positions, gathered by the measured route, multiplied by the full-width maps, reshaped into the receiver's heads, and rotated with the receiver's table at the anchor positions q. Because several receiver layers read the same source layer, their maps are concatenated so that one GEMM per distinct source layer yields all of them.

Assembly follows the layout of §3.4: positions $[0, S)$ receive the receiver's own sink rows, stored with the alignment table; the translated rows fill $[S, n_{\text{ext}})$; and positions $[n_{\text{ext}}, P)$ are left for the receiver to compute. Here

$$n_{\text{ext}} = \left\lfloor (P - B)/\beta \right\rfloor \beta \tag{7}$$

is the number of positions the connector supplies, with β the engine's block size, so that the refresh budget B becomes literally the number of prefix tokens the receiver still computes.

Two facts about a paged cache shape the write. Every stored row keeps its logical position ID q; positions are never renumbered. But slots and positions must be one-to-one, so the unanchored positions that the offline construction omits cannot be left empty: they are filled from the nearest *preceding* anchor by a single cumulative-max gather. A filled row carries only content the receiver has already read, so causality is preserved, and the in-engine/offline agreement check of Section 4.1 bounds the cost of filling relative to omission. All n_{ext} rows are then written to their pages through the engine's own paged-write kernel.

D.2 Execution in the Serving Engine

Offline, a directed pair computes its layer routes and maps and a receiver trains its adapter; these are installed by placing the maps next to the engine and loading the adapter as an ordinary LoRA. Online, only the alignment table, the gather, the GEMMs, and the page writes occur.

A request enters the receiver engine as a token sequence. The connector derives its content key, checks whether a source has published the corresponding rows, and admits the request to the sharing path only if $P \geq P_{\min}$: every handoff has a fixed cost that the replaced prefill must outweigh, and $P_{\min}$ is read off the measured length–load surface of Table 10(b) rather than set by hand ($P_{\min} = 2048$ on the B200 path). For an admitted request the connector reports n_{ext} to the scheduler.

To the scheduler, CachePort is then a prefix-cache hit of n_{ext} tokens. It skips those positions exactly as it would skip a locally cached prefix and prefills the remainder $[n_{\text{ext}}, P)$ with its ordinary chunked prefill, attending over the injected pages. This is how tail refresh is realized: the native segment is the engine's own prefill over the positions the connector did not supply, computed against the hybrid left context by construction, with no code path of its own. Query-selected refresh needs the question before the segment is chosen; it is evaluated in the offline path, and scheduling that dependency inside the engine is left to future work. The adapter needs no invocation either: it is a LoRA the engine applies inside `q_proj` and `o_proj` for every token of the request, during the native prefill and during decoding alike; the full-prefill and source arms of our measurements do not load it. Once the native segment is prefilled the cache is complete and decoding proceeds as for any other request; time to first token covers the entire path.

D.3 Per-Request Cost, Gates, and Adapter Obligation

Cost of the handoff. At an 8k prefix on one B200 the sharing path costs about 25 ms per request (L8→Q7): roughly 3 ms to open the source's IPC handles, 20 ms to map and gather the anchor rows, and 2 ms for un-rotation, the fused GEMMs, re-rotation, and the page writes. The translation itself is a small fraction of the path; the movement of rows dominates even when no copy is made. At 512 tokens the same path costs 13 ms (open 3.6, fetch 8.5, transform 1.1), while the receiver's native prefill costs 10 ms, which is the per-request view of the admission threshold.

Alignment tables. The control plane needs only the two tokenizations of a document. Per document it stores the anchor pairs (p, q) restricted to the translated span, the receiver's S native sink rows for every layer, and the source's head count, a few hundred kilobytes in total and no KV tensors. Because a finer source tokenization can place anchors beyond the receiver's P, the source prompt is extended to cover the last anchor rather than truncated at P; gathering a row the source never computed is a device-side assertion that terminates the engine.

Gates on the live path. Three failure modes on the live path produce plausible caches and no error, and each is guarded by a check that a run must pass before its numbers are accepted. (i) Un-rotating the source's keys with the receiver's rotary table: Llama-3.1 uses $\theta = 5{\times}10^5$ with frequency scaling and Qwen2.5 uses $\theta = 10^6$, so the result is structured noise; the gate compares in-engine and offline PPL for the same documents. (ii) Un-rotating at the row index instead of the true source position when rows have been pre-gathered by the producer, which rotates every key by the wrong angle and roughly halves quality. (iii) A rotary table shorter than the positions it is indexed at, an out-of-range gather that surfaces one kernel later as an opaque device-side assertion; the translator checks table length against n_{ext} and the largest source anchor. A fourth control, *wrong document*, points each receiver document at the next document's source export; quality must collapse toward the no-context floor, otherwise the connector is not actually reading the cache.

Adapter obligation. A translated cache is valid only for a receiver that has loaded the matching adapter. The connector carries this as an explicit obligation with three settings, refuse, warn, or ignore, so that a misconfigured receiver fails loudly instead of decoding from a cache it was never adapted to. All reported CachePort arms run under *refuse*. When source and receiver share a tokenizer, the anchor set is the identity and the alignment table degenerates to the sink rows; same-family pairs run through exactly this path, with no special casing.

E Detailed Baseline Results

All tables in this section report absolute PPL. Baseline rows use a native 16-token sink and no tail refresh; CachePort-$B0$ is the directly comparable arm, while CachePort-$B128$ reports the complete 512-token recipe. LSC (Dery et al., 2026), MoT, Family-Transfer (Heo et al., 2026), and DroidSpeak (Liu et al., 2024) lack public implementations matching our setting and are reduced reimplementations from their descriptions. C2C-Project evaluates the projector of C2C (Fu et al., 2026) as a replacement cache; this is deliberately harder than C2C's native fusion setting, which retains both models during decoding. These results should be read as controlled comparisons in our protocol, not reproductions of the papers' headline results. CachePort numbers are live-engine measurements; baseline numbers use the research harness. Engine/offline full-prefill gates agree within 0.02 PPL.

E.1 Same-Checkpoint Transfer

CachePort is not intended to improve inherited compatibility; in this domain its construction degenerates to verbatim reuse. The result is accordingly close to DroidSpeak rather than uniformly better.

Table 6: Fine-tune sibling pairs. Each entry is full-prefill / reused-cache PPL. KVComm uses $n = 30$, DroidSpeak $n = 40$, and CachePort the corresponding historical subset, so full-prefill values are shown inside each cell rather than silently equated across subsets.

method	L8→L8-Instruct	Q7→Q7-Instruct
KVComm	6.996 / 7.580	7.011 / 7.986
DroidSpeak	6.907 / 7.083	6.931 / 6.780
CachePort (verbatim limit)	6.907 / 7.108	6.931 / **6.776**

Table 7: Same-family, different-scale transfer ($n = 60$). Lower PPL is better. The Q7→Q0.5B CachePort cell uses direct maps without a receiver adapter or refresh and is therefore not a full-recipe comparison.

direction	full prefill	Family-Transfer	MoT	CP-$B0$	CP-$B128$
Q3→Q7	6.738	7.969	8.448	7.321	**7.089**
Q7→Q3	7.740	8.936	9.353	8.300	**8.087**
Q7→Q0.5B	12.264	**14.420**	15.587	14.779†	–

E.2 Same-Family Scale Transfer

E.3 Cross-Family Transfer

E.4 Transform Cost and Protocol Debt

Three protocol debts remain. First, learned reimplementations currently have one training seed. Second, KVComm and DroidSpeak should be rerun on all 60 documents. Third, trainable parameters, training GPU-hours, and one-shot end-to-end B200 latency should be reported for every baseline. None changes the live CachePort measurements, but all would strengthen claims about relative efficiency.

F Supplementary Evaluation

F.1 Live-Engine Quality–Latency and Throughput

Table 10(a) reports quality and latency for the same live-engine configurations at 4k. Increasing the refresh budget from 128 to 1,024 tokens improves PPL from 6.803 to 6.562, compared with 6.269 for full prefill. B1024 lowers median TTFT from 53.2 to 43.7 ms, although its p95 is higher at 74.1 versus 55.9 ms; B128 is slower than full prefill at 64.2 ms median. The budget changes both native computation and the number of cache rows translated and written, so latency need not increase with refresh size. Short contexts do not amortize the handoff overhead: at 512 tokens, B128 takes 38.4 ms versus 9.6 ms for native prefill.

Table 10(b) measures aggregate source and receiver throughput with B128 under shared GPU load. Sharing improves aggregate throughput at every load from 2k upward while reducing it at 512 tokens, and the gain may accrue to the source rather than the receiver: at 4k and full producer duty, receiver throughput falls from 10.21 to 5.18 requests/s while source throughput rises from 5.87 to 13.72 requests/s, a 17% aggregate increase. A separate run generating 64 receiver tokens observes gains of 25% at half duty and 17% at full duty (Table 12).

F.2 System Throughput: Full Decomposition and the 8k Row

Table 11 lists the source and receiver rates behind every ratio in Table 10(b), including the 8k row. Two regimes are visible. At 512 tokens CachePort lowers the receiver's own rate at every load and the aggregate ratio rises with load only because the source absorbs the freed compute; the ratio never reaches one. From 2k upward the receiver itself is faster at idle and partial load, and at full duty the gain moves entirely to the source: at 8k and full duty the receiver alone drops from 4.13 to 3.70 requests/s while the source rises from 4.39 to 8.72, for an aggregate gain of 46%. The 8k row uses

Table 8: Cross-family replacement ($n = 60$). LSC uses its head-wise OT stabilization for L8→Q7 because the direct fit has PPL above 1,000.

direction	full prefill	LSC	MoT	C2C-Project	CP-$B0$	CP-$B128$
L8→Q7	6.738	8.129	8.548	10.379	7.057	**6.935**
Q7→L8	6.168	7.344	7.495	9.511	6.853	**6.519**
MI→L8	6.168	7.398	7.479	9.113	6.770	**6.505**

Table 9: Median transform-stage latency at 512 tokens on B200. These numbers exclude shared cache assembly, native sink/refresh computation, and receiver decoding; Table 10(a) is the end-to-end CachePort result.

method	ms	method	ms
KVComm/verbatim	1.2	DroidSpeak	0.6
Family-Transfer	3.7	CachePort	2.4
LSC	13.8	C2C-Project	17.2
MoT	76.2		

the twelve WikiText-103 test documents long enough to supply it and measures throughput only; quality is verified to 4k in Table 10(a) and is not part of this run.

An earlier, independent run at 4k with four source duties (Table 12) gives the same picture: an aggregate gain of $+120\%$ at idle falling to $+18\%$ at full duty, with the receiver's own rate halving at full duty while the source's more than doubles. A variant of that run generating 64 receiver tokens instead of one gives $+25\%$ at 50% duty and $+17\%$ at full duty.

F.3 A Second Direction from 512 to 8k Tokens: L8→ MI

Table 13 repeats the live-engine protocol of Table 10(a) for Llama-3.1-8B→Mistral-7B-v0.3 at four prefix lengths, with Mistral as the receiver for the first time in the engine. Every live-path check passes at every length. Two observations carry over from L8→Q7 and one is stronger. The crossover in time to first token again lies between 2k and 4k, and at 8k the handoff is $1.6\times$ faster than native prefill (75.8 versus 123.9 ms) while B=1024 trails full prefill by 0.30 PPL. Time to first token is nearly flat in B up to 1,024 tokens, so the refresh is not where the handoff's time goes. And at 4k three refresh budgets (B=256, 512, 1024) beat native prefill, against one for L8→Q7, because Mistral's native prefill is slightly slower than Qwen's while the handoff cost is the same.

F.4 An Unseen Source with a Frozen Receiver Adapter

Qwen3-8B applies a per-head normalization to keys before rotation, so its exported key states are four-dimensional where every earlier source's were two-dimensional; a single shape coercion in the export hook is the only code change it required, verified bit-identical on the existing sources. Its routes and maps were fitted in closed form on the standard training rows and paired with the unchanged Q7 adapter; no adapter training was run. On the 512-token replacement protocol the arms give full prefill 6.740, CachePort-$B0$ 7.608, CachePort-$B128$ 7.317, and wrong document 9.148; all live-path gates pass. The gap to the in-training sources of Table 1 is about 0.3–0.5 PPL at $B0$ and 0.4 at $B128$, the price of an adapter that has never seen this source's residual.

G Residual Error Analysis

The remaining error is not predictable from what the receiver sees. Figure 5 summarizes the decode-time corrector probe referred to in §3.4. For each receiver layer and decode position, the query is taken from the shared run, and the target is the difference between the attention output over the full-prefill cache and over the translated cache. Per-head ridge correctors are fitted on 200 training documents (32 positions each) from four feature sets, the corrupted output alone, the query

Table 10: **Quality, latency, and serving trade-offs in vLLM.** L8→Q7 on one B200. (a) PPL and handoff-to-first-token latency at 4k tokens ($n = 24$); lower is better; Floor is the no-context reference. (b) CachePort/full-prefill aggregate throughput ratio with B128, summing source and receiver completion rates over the same 60-second window; columns denote source prefill duty. The 8k row measures throughput only; Table 11 gives every cell's source and receiver rates.

(a) Quality–latency at 4k

		TTFT (ms)	
Config.	PPL	p50	p95
Floor	8.868	–	–
Gold	**6.269**	53.2	**55.9**
CP-B128	6.803	64.2	71.5
CP-B256	6.726	66.9	80.6
CP-B512	6.669	69.0	76.0
CP-B1024	6.562	**43.7**	74.1

(b) Throughput with B128

	Source duty		
Prefix	Idle	50%	100%
512	0.41	0.47	0.58
2k	1.42	1.31	1.03
4k	2.35	1.84	1.17
8k	3.47	2.53	1.46

Table 11: Requests per second, receiver + source = aggregate, L8 source and Q7 receiver on one B200, $B = 128$, 60-second window after a 30-second warmup, one receiver token generated. Source load is its own prefill duty cycle.

prefix	load	full prefill	CachePort	ratio
512	idle	$152.2 + 0.0 = 152.2$	$62.8 + 0.0 = 62.8$	0.41
512	50%	$135.1 + 6.9 = 142.1$	$40.5 + 25.8 = 66.2$	0.47
512	100%	$119.0 + 14.1 = 133.1$	$25.4 + 51.3 = 76.7$	0.58
2048	idle	$38.3 + 0.0 = 38.3$	$54.5 + 0.0 = 54.5$	1.42
2048	50%	$30.7 + 4.2 = 34.9$	$32.8 + 13.0 = 45.7$	1.31
2048	100%	$24.8 + 8.7 = 33.5$	$8.6 + 25.8 = 34.4$	1.03
4096	idle	$18.9 + 0.0 = 18.9$	$44.4 + 0.0 = 44.4$	2.35
4096	50%	$14.3 + 2.9 = 17.3$	$25.0 + 6.8 = 31.8$	1.84
4096	100%	$10.2 + 5.9 = 16.1$	$5.2 + 13.7 = 18.9$	1.17
8192	idle	$8.8 + 0.0 = 8.8$	$30.6 + 0.0 = 30.6$	3.47
8192	50%	$6.3 + 2.4 = 8.6$	$17.5 + 4.4 = 21.9$	2.53
8192	100%	$4.1 + 4.4 = 8.5$	$3.7 + 8.7 = 12.4$	1.46

and the output, the two plus their elementwise product, and a 256-dimensional PCA of both across heads, and scored on 60 held-out documents. The error is 16–58% of the output norm (mean 38%); held-out R^2 is 0.01–0.04 overall, 0.08–0.11 in shallow layers, and negative in the deepest layers, where every fitted corrector is worse than none. Neither the explicit multiplicative term nor cross-head mixing helps. Mechanically this is expected: the residual is $\sum_p a_p^* v_p^* - \sum_p \tilde{a}_p \tilde{v}_p$, and neither the corrupted sum nor the query contains the correct one. An error that cannot be inferred from the cache has to be supplied by the receiver, which is the rationale for the adapter and the native refresh of §3.4.

Each component addresses a distinct failure. On L8→Q7 at 4k (validation long-document set, $n = 14$; full-prefill PPL 6.58, a different document set from the 4k test set of Table 10), the full method gives 7.17. Removing the adapter raises PPL to 7.42, removing the native refresh to 7.37, and removing the 16-token sink to 8.93. On the 512-token protocol, the ridge-initialized affine translator with the adapter gives 7.12 against 6.74 for full prefill, whereas an equal-budget three-layer MLP gives 8.68. The gains come from aligning the right states and training for receiver behavior, not from translator capacity.

The remaining error originates primarily at the source. A two-way decomposition of the full direction matrix attributes 67% of PPL variance to source identity and only 5% to receiver identity after adapter training. A linear regression from three concatenated sources explains 73–85% of receiver keys but only 49–71% of values, leaving 43–51% of deep-value variance unexplained by this probe. Receiver recomputation supplies these states directly, which gives the native refresh a

Table 12: Independent four-duty run at a 4k prefix. Entries are Q7 receiver + L8 source = aggregate requests/s; CachePort uses $B = 128$.

source duty	full prefill	CachePort	aggregate change
idle	$18.64 + 0 = 18.64$	$41.00 + 0 = 41.00$	$+120\%$
25%	$16.38 + 1.43 = 17.81$	$32.30 + 3.44 = 35.74$	$+101\%$
50%	$14.29 + 2.91 = 17.20$	$22.94 + 6.72 = 29.66$	$+72\%$
100%	$9.99 + 5.78 = 15.77$	$5.10 + 13.52 = 18.63$	$+18\%$

Table 13: **Live length ladder for L8→MI** (Mistral-7B-v0.3 receiver, B200). PPL is document-mean; TTFT is per-request median/p95 in ms and covers the full handoff path. B is the native tail; the best CachePort arm per length is in bold. Wrong-document caches sit 1.2–1.9 PPL above full prefill at every length. *Floor*: the receiver's native sink only, no context (research harness). $n = 60/39/24/12$ documents.

	512		2048	
receiver computation	PPL	TTFT	PPL	TTFT
no context (floor)	7.125	–	7.624	–
full prefill	**5.313**	**10.1** / 10.5	**5.484**	**29.5** / 34.1
no adapter, $B0$	5.970	–	6.308	–
CachePort-$B0$	5.668	–	6.021	–
CachePort-$B64$	5.549	42.5 / 51.7	5.868	51.9 / 56.3
CachePort-$B128$	5.510	43.6 / 53.2	5.802	50.9 / 59.7
CachePort-$B256$	–	–	5.740	50.4 / 53.2
CachePort-$B512$	–	–	5.689	50.4 / 62.1
wrong document, $B0$	7.087	–	7.406	–
	4096		8192	
receiver computation	PPL	TTFT	PPL	TTFT
no context (floor)	7.650	–	6.006	–
full prefill	**5.243**	59.8 / 62.7	**4.660**	123.9 / 130.7
no adapter, $B0$	6.399	–	5.409	–
CachePort-$B0$	6.033	–	5.231	–
CachePort-$B128$	5.793	62.1 / 65.7	5.113	76.4 / 79.0
CachePort-$B256$	5.732	56.7 / 67.5	–	–
CachePort-$B512$	5.625	56.2 / 63.7	5.010	76.9 / 83.0
CachePort-$B1024$	5.546	**54.0** / 56.8	4.961	**75.8** / 80.6
CachePort-$B2048$	–	–	4.903	88.7 / 89.5
wrong document, $B0$	7.259	–	5.826	–

representational rather than merely engineering role; the result does not rule out recovery by richer future translators.

H Module Ablation: Per-Decision Breakdown

Table 14 expands the three rows of Table 3 into every design decision, with references matched to how each arm was produced. Module 1 and 2 arms replace one decision and refit the maps in closed form (ridge, $\alpha = 10^{-2}$, at most 8,192 anchored rows on the same training documents), so they are compared with the closed-form recipe; Module 3 arms disable one recovery mechanism of the deployed, co-trained recipe and are compared with it. Two further Module 1 and 2 decisions appear here only: fitting the key map on rotated keys instead of un-rotated ones costs 0.13 PPL, and terminal-layer routing costs 0.40, while proportional-depth routing costs only 0.08 on this $32 \to 32$ pair because its measured route is itself near-diagonal (the non-monotone L8→Q7 route is where measured routing matters). Co-training the maps with the adapter is worth 0.09 PPL over the closed-form fit ($6.858 \to 6.767$) and the 128-token refresh a further 0.25.

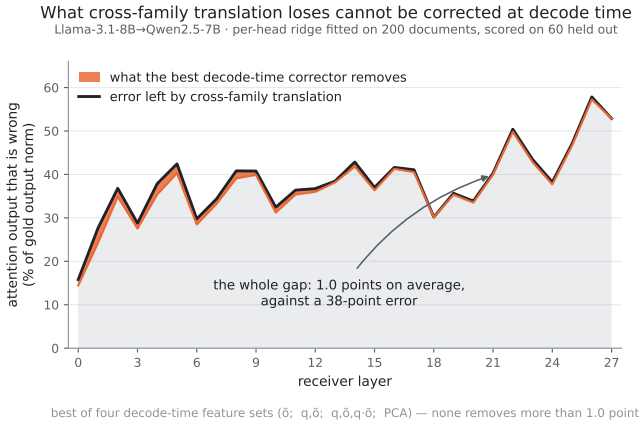


Figure 5: **The residual is not predictable from the translated states.** Per receiver layer, the share of the attention output that is wrong under the translated cache (line), and the part of it that the best decode-time corrector fitted on the same translated states removes (band between the line and the corrected curve). The band is about one point against an error of roughly 38; almost all of the residual survives any corrector. L8→Q7, 200 fitting and 40 held-out documents.

Table 14: **Module ablation on MI→L8** (512 tokens, $n = 60$, PPL). Full prefill 6.168; no-context floor 8.255. R is the fraction of the floor-to-full-prefill gap recovered. Module 1 and 2 arms replace one design decision and refit the maps in closed form (ridge, same data and α), so they are compared with the closed-form recipe; Module 3 arms disable one recovery mechanism of the deployed recipe and are compared with it.

module	ablation	PPL	Δ	R
Reference: closed-form recipe (ridge maps, CKA route, full width, adapter on, B=0)				
	recipe, closed form	6.858	–	.669
1 Span alignment	index alignment instead of character anchors	7.987	+1.129	.128
	map fitted on rotated keys (no un-/re-RoPE)	6.988	+0.130	.607
2 Layer routing, full-width map	block-diagonal per-head map (8×128 blocks)	7.501	+0.643	.361
	terminal route (every layer reads the last sharer layer)	7.262	+0.404	.476
	proportional-depth route[†]	6.941	+0.083	.630
Reference: deployed recipe (co-trained maps, adapter on)				
	recipe, B=128	6.520	–	.831
	recipe, B=0	6.767	+0.247	.713
3 Receiver recovery (B=0)	adapter off	7.191	+0.424	.509
	sink reduced to BOS only (S=1)	6.894	+0.127	.652
	no native positions (BOS absent, S=0)	867.2	–	–
	wrong-document cache (control)	7.867	+1.100	.186

[†]On this 32 → 32 pair the measured CKA route is itself near-diagonal, so proportional depth is a weak ablation here; the routing evidence for a non-monotone pair (L8→Q7) is in §4.2.4. Module 3 Δ is against the deployed recipe at the same B (B=0: 6.767; the B=128 row against B=0).

I Where the Speedup Comes From: Stack, Kernels, Dispatch

The main text reports time to first token inside vLLM. This section asks how much of that number is the method and how much is the engine, by re-running the same two paths in progressively less optimized stacks on the same hardware and direction (L8→MI). Three findings recur: CachePort's own path costs roughly the same everywhere, the replaced prefill does not, and what remains of CachePort's cost at long prefixes is arithmetic rather than dispatch.

Setup shared by all tables. One B200; bf16; receiver Mistral-7B-v0.3 with its matrix adapter, source Llama-3.1-8B; the same documents as Table 13 (WikiText-103 test; n per table below); prefix $P \in \{512, 2048, 4096, 8192\}$. In every CachePort arm the source's prefill is not timed (it has happened in deployment) and the source's rows are already resident, mirroring the IPC path; the receiver's LoRA adapter is on for CachePort and off for the baseline; no torch.compile anywhere. TTFT covers everything from handoff to the first-token logit; "compute" excludes the LM head and any cache bookkeeping and is stated where used.

I.1 Stack Dependence of Time to First Token

Table 15: **TTFT under three stacks** (ms, per-request median; CachePort B=128). *Engine*: vLLM with paged cache, fused kernels and CUDA-IPC handoff, $n = 60/39/24/12$ (Table 13). *HF sdpa*: HuggingFace transformers, PyTorch SDPA attention, `DynamicCache`, Python loop over layers for the translation, LM head included, $n = 8$. *HF eager*: the same with unfused attention (explicit $P \times P$ scores). Ratios are native / CachePort.

	Engine (vLLM)		HF sdpa		HF eager	
P	native	CachePort	native	CachePort	native	CachePort
512	10.1	43.6 (0.23×)	17.3	37.2 (0.46×)	23.1	50.2 (0.46×)
2048	29.5	50.9 (0.58×)	39.3	52.3 (0.75×)	79.3	56.3 (1.41×)
4096	59.8	62.1 (0.96×)	82.6	58.2 (1.42×)	248.0	62.7 (3.95×)
8192	123.9	76.4 (1.62×)	184.4	82.8 (2.23×)	902.5	90.7 (9.95×)

Across the three stacks the CachePort column at 8k moves from 76 to 83 to 91 ms while the native column moves from 124 to 184 to 903 ms. The engine optimizes the prefill it replaces far more than it optimizes the small forwards and per-layer maps that CachePort runs, so the ratio grows as the stack gets less efficient. The eager column is reported for completeness; its ratio is dominated by the quadratic cost of materialized attention and should not be read as the method's speedup.

I.2 Compute under Identical Kernels

Table 16: **Arithmetic cost with the system removed** (ms, median of 30 trials after 50 warmups, 3 documents per length). Both paths run the receiver's transformer body only (no LM head) through the same HF modules with PyTorch's FlashAttention SDPA backend forced for unmasked calls (the chunk-over-cache tail call falls back to the efficient kernel, recorded); no IPC, no paged writes, no CUDA graphs. CachePort "compute" = sink forward (16 tokens) + translation (un-RoPE with the source's table, one affine map per layer, receiver RoPE) + tail forward over B tokens against the assembled cache; `DynamicCache` assembly (0.6–4 ms) is excluded. "FLOPs" is the analytic ratio of the two paths' floating-point operations from the model configs and the measured anchor count.

	native	CachePort compute (ms) / measured ratio			FLOPs ratio		
P	body (ms)	$B0$	$B128$	$B1024$	$B0$	$B128$	$B1024$
512	13.0	37.0 (0.35×)	38.0 (0.34×)	–	24	3.5	–
2048	39.8	40.8 (0.98×)	41.0 (0.97×)	53.3 (0.75×)	62	13	2.0
4096	82.1	48.4 (1.70×)	53.3 (1.54×)	71.8 (1.14×)	84	23	3.9
8192	183.2	64.3 (2.85×)	78.9 (2.32×)	108.6 (1.69×)	107	41	7.5

The algorithm removes 96–98% of the prefill arithmetic (24–107× fewer operations at B=0), of which the un-fused research implementation realizes 1.7× at 4k and 2.9× at 8k. The gap is visible in the breakdown: the sink forward costs ≈ 15 ms at every length and the tail forward 15–30 ms, both far above their arithmetic, while the translation grows linearly with the anchor count (5.8 → 33.6 ms for 0.4k→ 7.1k rows, about 4.7 μs per row over 64 small GEMMs). The two forwards are memory-bound reads of the receiver's weights plus per-kernel dispatch; the next table separates the dispatch part.

Table 17: **CachePort compute with and without CUDA graphs** (ms, same protocol as Table 16 but with HF `StaticCache` so that shapes are static; each path is captured in one graph and replayed). The native column repeats the FlashAttention body time of Table 16: with a static cache HF materializes a mask and SDPA leaves the flash kernel, so a static-cache native baseline would not be the same kernel. Ratios are native / CachePort.

	native	$B0$		$B128$	
P	body (ms)	eager	graph	eager	graph
512	13.0	39.9 (0.33×)	20.3 (0.64×)	39.3 (0.33×)	20.3 (0.64×)
2048	39.8	39.6 (1.01×)	31.2 (1.28×)	39.0 (1.02×)	31.7 (1.26×)
4096	82.1	51.8 (1.58×)	45.9 (1.79×)	53.8 (1.53×)	48.1 (1.71×)
8192	183.2	78.7 (2.33×)	75.1 (2.44×)	81.4 (2.25×)	77.9 (2.35×)

I.3 Dispatch Overhead: CUDA Graphs

Removing dispatch helps only where the replaced prefill is itself short: graphs save 19 ms at 512 tokens, 7 at 2k, 6 at 4k and 3.5 at 8k, because kernel launches are asynchronous and are hidden behind GPU work once the kernels are long. They lift the 512-token ratio from $0.33\times$ to $0.64\times$ and move the crossover to about 1.5k tokens, but change the 8k ratio by $0.1\times$. What remains at long prefixes is arithmetic: the per-layer translation GEMMs (FLOP-bound in non-tensor-core fp32) and two memory-bound small forwards; §I.4 removes both.

I.4 Optimizing the Research Path

Table 18: **CachePort compute after three implementation changes** (ms, same protocol as Table 16: B200, bf16, receiver body only, 3 documents × 50 warmups + 30 trials; native is the FlashAttention body time measured in the same run). *fp32 loop*: the Python loop of 64 fp32 GEMMs and two receiver forwards of Table 16. *fused TF32*: receiver layers reading the same source layer share one GEMM (41 instead of 64) on TF32 tensor cores, the matmul mode vLLM uses; the fp32 loop was FLOP-bound, not launch-bound, at 8k. *one forward*: sink and tail tokens run as a single receiver forward with a 4-D mask that encodes the logical causal structure (sink sees earlier sink; tail sees sink, all translated rows, earlier tail) and logical position ids; the translated rows are written to the static cache first. *graph*: the one-forward path captured in a CUDA graph. Ratios are native / CachePort. Equivalence: the one-forward path reproduces the two-forward final hidden state to a relative error of 1.1–1.8×10^{-2}, the same order as two SDPA kernels on the same input (1.05–1.25×10^{-2}); the two-forward path repeated is bit-identical.

	native	CachePort, B=128				B=0
P	body	fp32 loop, 2 fwd	fused TF32, 2 fwd	one fwd, eager	**one fwd, graph**	one fwd, graph
512	13.2	38.0 (0.35×)	39.6 (0.33×)	21.6 (0.61×)	**11.2 (1.18×)**	10.5 (1.26×)
2048	40.5	41.0 (0.99×)	39.3 (1.03×)	21.6 (1.88×)	**15.9 (2.55×)**	15.0 (2.70×)
4096	83.8	53.3 (1.57×)	41.3 (2.03×)	24.2 (3.46×)	**22.5 (3.72×)**	21.4 (3.92×)
8192	187.0	78.9 (2.37×)	62.6 (2.99×)	37.4 (5.00×)	**35.6 (5.25×)**	34.7 (5.39×)

Protocol of Figure 3. The main-text figure plots the two bold columns of Table 18 (one forward, CUDA graph, B=128 and B=0) against the native column. Direction L8→MI on one B200 in bf16; receiver transformer body only (no LM head); median of 30 trials after 50 warmups on 3 WikiText-103 test documents per length. Native is one FlashAttention prefill over P tokens. CachePort is the complete receiver-side path from resident source rows to the logit: un-RoPE with the source's table, fused affine translation on TF32 tensor cores, receiver RoPE, and a single receiver forward over the 16 sink tokens and the B refreshed tokens against the assembled cache, captured in a CUDA graph. The source's prefill and the cross-process transport are excluded, as in every table of this section. The FLOP ratios on the figure's axis are the analytic counts of Table 16.

Three changes take the research path from the fp32 loop to within a factor of the FLOP bound. Running the translation GEMMs on TF32 tensor cores removes the arithmetic bottleneck at long prefixes (8k: 79 → 63 ms), which is why fusing the GEMMs alone (Table 16's loop was already

FLOP-bound in non-tensor-core fp32) had changed little. Merging the sink and tail into one forward removes one of the two memory-bound weight reads at every length (8k: $63 \rightarrow 37$ ms), and is the change that turns the 512-token cell positive. CUDA graphs then contribute 1–10 ms, most at short prefixes. The resulting path costs 11–36 ms from 512 to 8k tokens against a native prefill of 13–183 ms: $1.2\times$ at 512, $2.6\times$ at 2k, $3.7\times$ at 4k and $5.3\times$ at 8k, with a crossover below 512 tokens. These are compute numbers on the unoptimised stack; the engine's remaining gap to them is the handoff (Section I.1), not arithmetic.

Table 19: **Tail budget B: quality against compute, per prefix length** (L8→MI). PPL is the in-engine E1 measurement of Table 13 (full prefill $5.313/5.484/5.243/4.660$; no-context floor $7.125/7.624/7.650/6.006$); ms is the optimized research path of Table 18 (one forward, fused TF32 translation, CUDA graph; 3 documents $\times$ 50 warmups $+$ 30 trials; native body $13.0/40.3/83.4/186.6$ ms). Ratio is native / CachePort. Dashes: not measured in the engine at that length.

	512		2048		4096		8192	
B	PPL	ms	PPL	ms	PPL	ms	PPL	ms
0	5.668	10.5 (1.24×)	6.021	15.0 (2.69×)	6.033	21.4 (3.90×)	5.231	34.7 (5.38×)
16	5.612	10.6 (1.23×)	–	15.2 (2.65×)	–	21.6 (3.86×)	–	34.9 (5.35×)
32	5.591	10.5 (1.24×)	–	15.2 (2.65×)	–	21.8 (3.83×)	–	35.3 (5.29×)
64	5.549	10.8 (1.20×)	5.868	15.5 (2.60×)	–	22.2 (3.76×)	–	35.2 (5.30×)
128	5.510	11.2 (1.16×)	5.802	15.9 (2.53×)	5.793	22.5 (3.71×)	5.113	35.6 (5.24×)
256	–	12.9 (1.01×)	5.740	18.6 (2.17×)	5.732	26.6 (3.14×)	–	42.6 (4.38×)
512	–	–	5.689	21.6 (1.87×)	5.625	29.4 (2.84×)	5.010	45.3 (4.12×)
1024	–	–	–	32.3 (1.25×)	5.546	44.6 (1.87×)	4.961	69.0 (2.70×)
2048	–	–	–	–	–	75.9 (1.10×)	4.903	118.0 (1.58×)

The tail is free up to about 128 tokens at every length (the single forward is memory-bound on the receiver's weights, and 16 or 128 tokens ride on the same weight read) and then grows roughly linearly, at 14–17 μs per tail token at 2–4k and $\approx 40\,\mu$s at 8k, where each tail token attends over the full cache. Read against the PPL column, this places the useful operating points: B=128 buys 0.12–0.24 PPL over B=0 for 1–5% of the time, B=512 a further 0.10 PPL for $1.3\times$ the time, and B=1024–2048 approaches full-prefill quality at a cost that at 4k and 8k still leaves a 1.6–$2.7\times$ margin.

I.5 A Larger Receiver's Prefill

Table 20: **Native prefill time of a 14B receiver** (ms, median of 8 documents, one B200, bf16, HF transformers, LM head included, one untimed warmup per length). Ministral-3-14B-Base (13.9B parameters, loaded as its text-and-vision class and fed text only) next to Mistral-7B-v0.3 under the same code. No CachePort artefacts exist for Ministral; the table bounds the prefill that a handoff to a receiver of this size would replace.

	HF sdpa		HF eager	
P	Mistral-7B	Ministral-14B	Mistral-7B	Ministral-14B
512	17.3	29.6	23.1	36.2
2048	39.3	62.6	79.3	111.7
4096	82.6	128.8	248.0	333.4
8192	184.4	280.7	902.5	1177.5

The 14B prefill is 1.5–$1.7\times$ the 7B prefill under sdpa and 1.3–$1.6\times$ under eager attention, whose long-context cost is set by layer count and sequence length rather than parameter count. CachePort's own path would scale with the receiver's per-token cost as well, so the ratio, not the absolute saving, is what a larger receiver would preserve; peak memory for the eager 8k prefill was 47.8 GiB against 28.3 GiB with sdpa.