ACCT: ASYMMETRIC CROSS-LLM CONTEXT TRANSFER

Anonymous authors
Paper under double-blind review

ABSTRACT

In real-world deployments, large language model (LLM) systems are increasingly evolving from single-model invocation toward heterogeneous deployments in which different models are dynamically invoked for inference. When the same context is reused across models, however, each newly selected target model typically has to process it again, repeatedly incurring substantial computational costs. Therefore, we propose ACCT (Asymmetric Cross-LLM Context Transfer), which targets context-state transfer between models of different scales within the same model family. ACCT transforms the internal representations produced by a source model when processing the context into target-native sparse KV states that can be directly consumed by the target model, thereby enabling cross-model reuse of context computation and decoupling context computation from downstream inference. ACCT consists of three lightweight components: the Cartographer identifies query-relevant context for each target layer and KV head; the Allocator assigns state capacity under a global budget; and the Compiler transforms selected source-side representations into sparse target-native KV states. The target model can therefore perform inference using the transferred states without reprocessing the original context. We evaluate ACCT in two complementary settings comprising five general-capability benchmarks and six highly context-dependent tasks. Experiments show that ACCT matches or exceeds the target model's full-context inference performance on selected tasks, with outcomes varying across model pairs and datasets. At a context length of approximately 19K tokens, ACCT's KV state conversion achieves a 6.5–$21.2\times$ speedup over target context prefill. These results demonstrate the feasibility of transferring context computation across models under a limited state budget, offering a new approach to context reuse and efficient inference in heterogeneous model systems.

1 INTRODUCTION

Large language model (LLM) systems are evolving from serving a single model to heterogeneous deployments comprising models of different scales and capabilities (Chen et al., 2024; Liu et al., 2024). Such systems may dynamically route requests according to task difficulty, latency requirements, and resource availability (Ong et al., 2025), or assign context processing and subsequent reasoning to different models. Meanwhile, conversation histories, retrieved documents, and task states often persist across requests, far outliving any individual model invocation (Packer et al., 2023; Gim et al., 2024). Thus, while the executing model may change, the context often remains the same. Yet when the same context is handed to a new target model, the target typically must process the entire context again to construct the internal states required for inference. As contexts grow and are used more frequently across models, the costs of repeated computation and state construction accumulate. Enabling models to reuse context computation performed by another model, is therefore an important challenge for efficient heterogeneous LLM systems.

The main limitation of existing mechanisms for reusing context computation is that cached internal states are tied to a particular model. KV caching allows subsequent generation to reuse computation over preceding content, while prefix caching extends this reuse across requests served by the same model (Kwon et al., 2023; Zheng et al., 2023). These states, however, depend jointly on the model's parameters and layer structure. Even models of different scales within the same family generally produce states that are not directly interchangeable when processing identical text (Liu et al., 2024;

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.

Fu et al., 2025). Allowing a target model to build on context computation already performed by a source therefore requires transforming source representations into inference states that the target can read directly. The transfer process should organize information around the target's reasoning needs, retaining content relevant to the current query under a specified state budget, and making it accessible through the target's own parameters.

To this end, we propose **ACCT (Asymmetric Cross-LLM Context Transfer)** for reusing context computation between models of different scales within the same family. ACCT formulates context-state transfer as a directed process of constructing states for the target, using context representations already produced by the source to generate sparse, target-native KV states that the target can consume directly. To reuse source context computation under a limited state capacity, ACCT connects information selection for the target, capacity allocation, and native state generation through three lightweight components. The **Cartographer** uses source context and query representations to identify and rank relevant content for every target layer and KV head. The **Allocator** assigns capacity according to learned preferences across layers and heads under a global state budget; together with the content rankings, these allocations determine which positions are retained. The **Compiler** converts selected source representations into native states for the corresponding target layers and KV heads. The three components are trained in stages: ACCT first learns information selection and capacity allocation for the target, then uses internal states and inference behavior from the frozen target to supervise cross-model state compilation. After mounting the transferred states, the target reads context information through its own attention mechanism, without re-encoding the raw context, and performs only query encoding and subsequent autoregressive generation. This interface integrates context computation already performed by the source into the target's regular inference process, allowing context encoding and downstream reasoning to be carried out by different models.

ACCT aims to preserve the target's reasoning capabilities while avoiding its re-encoding of the raw context. We evaluate ACCT on five general-capability benchmarks and six highly context-dependent tasks across six directed model pairs. Using **Target Full**, in which the target model processes the complete context, as the reference, ACCT closely matches its performance on many tasks and surpasses it on some. At a context length of approximately 19K tokens, ACCT's KV state conversion achieves a 6.5–$21.2\times$ speedup over target context prefill. These results show that a target can use context information through transferred states to support subsequent reasoning even without encoding the raw context itself. ACCT thus demonstrates the feasibility of treating context computation as a resource that can be reused across models, opening new possibilities for independently organizing context processing and reasoning in heterogeneous model systems.

Our main contributions are as follows:

- We propose ACCT, an asymmetric cross-LLM context transfer method that enables reuse of context computation between models of different scales within the same family. The target can build on context computation already performed by the source, decoupling context encoding from subsequent reasoning.
- We design a lightweight transfer interface comprising the Cartographer, Allocator, and Compiler. It unifies query-relevant content selection for each layer and head, capacity allocation under a global budget, and sparse target-native KV compilation into a process of constructing states for the target, enabling inference without re-encoding the raw context.
- We evaluate ACCT on five general-capability benchmarks and six highly context-dependent tasks across three model families in both transfer directions. The results identify tasks on which transferred states match or exceed Target Full and characterize performance differences across model pairs and state budgets. We further quantify KV state-conversion speedups over target context prefill.

2 Related Work

Heterogeneous LLM Serving. FrugalGPT (Chen et al., 2024) constructs model cascades to reduce inference costs, while RouteLLM (Ong et al., 2025) learns routing policies between models with different capabilities and costs. These approaches focus on selecting the executing model; ACCT complements them by enabling reuse of previously computed context representations when execution switches to a different-scale model.

Context Reuse in LLM Inference. Prompt Cache (Gim et al., 2024) reuses attention states across requests, while H_2O (Zhang et al., 2023), SnapKV (Li et al., 2024), and PyramidKV (Cai et al., 2024) reduce retained KV states through token selection or layer-wise budgeting. These methods reuse or compress states within the model that produced them. Extending reuse across models, DroidSpeak (Liu et al., 2024) shares caches between same-architecture variants of a common base model and recomputes critical context layers in the receiver. ACCT instead converts representations across model scales without target-side context recomputation. Beyond computational reuse, C2C (Fu et al., 2025) supports semantic communication across model sizes and families by projecting and fusing source KV with the receiver's own context KV, avoiding intermediate text generation. Its full method retains receiver-side context encoding, whereas ACCT constructs the target's context memory from source representations under a global state budget and runs the target only on the query and subsequent generation.Accordingly, neither method in its original formulation provides a matched baseline for our setting, which requires cross-scale context-state transfer without target-side context encoding.

3 METHODS

ACCT (Asymmetric Cross-LLM Context Transfer) converts source context representations into sparse KV states that the target can directly consume, allowing different models to perform context computation and downstream inference.

3.1 PROBLEM FORMULATION AND TRANSFER INTERFACE

Consider a source–target pair (F_S, F_T) within the same model family, a context c of length N, and a query q of length Q. Token semantics and positions must be compatible (Supplementary Material F.1). The target has L_T layers, H_T physical KV heads per layer, and head dimension d_T. Each unit $p = (\ell, h)$ identifies KV head h in layer ℓ; the $P = L_T H_T$ units receive memory indexed by original positions:

$$\widehat{\mathcal{C}}_{S\to T}(c,q) = \mathcal{G}_\Theta(F_S([c;q])) = \left\{(i, \widehat{K}_{p,i}, \widehat{V}_{p,i}) : i \in \mathcal{I}_p\right\}_{p=1}^{P}. \tag{3.1}$$

Here $\widehat{\mathcal{C}}_{S\to T}$ is the transferred target memory, $\mathcal{G}_\Theta$ the transfer interface with parameters Θ, and $[c;q]$ the concatenation of context and query. A causal source pass provides final-layer context and query states H_c, H_q, multi-layer context states H_S, and context KV. The Cartographer produces scores $S \in \mathbb{R}^{P\times N}$ from context and query information; the Allocator determines each retained position set $\mathcal{I}_p$; the Compiler synthesizes target KV $(\widehat{K}_{p,i}, \widehat{V}_{p,i})$ at original position i through an internal affine Bridge and a neural residual. After mounting memory, the target encodes only q and generates autoregressively. GQA (Ainslie et al., 2023) query heads sharing a physical KV head read the same memory. Both base models remain frozen. The transfer interface follows shared capacity rules; source context states are reusable, but new queries require reselection and recompilation.

3.2 CONSTRUCTING SPARSE TARGET STATES

3.2.1 CARTOGRAPHER: TARGET-CONDITIONED INFORMATION SELECTION

Source salience need not match target reasoning needs. The Cartographer first extracts shared query-relevant evidence, then conditions scores on target layers and heads. A shared projection maps H_c, H_q to X, Y. The m_M learnable slots A aggregate the query and read the context:

$$U = \mathrm{Attn}(A, Y, Y), \quad Z = \mathrm{LN}(UW_Q + \mathrm{Attn}(UW_Q, XW_K, XW_V)). \tag{3.2}$$

Attn is scaled dot-product attention with padding masked, LN denotes layer normalization, and $Z \in \mathbb{R}^{m_M\times d_M}$ contains shared evidence. Unit embedding e_p and mean evidence $\bar{Z}$ produce FiLM (Perez et al., 2018) parameters (γ_p, β_p). Let u_i be a projected position representation, $\mathbf{m}_{p,i}$ its matching vector against target-conditioned evidence slots, and ϕ_i relative position features:

$$S_{p,i} = f_o(\gamma_p \odot u_i + \beta_p + W_m \mathbf{m}_{p,i} + W_\phi \phi_i). \tag{3.3}$$

Here $W_Q, W_K, W_V, W_m, W_\phi$ are the Cartographer's learnable projections for queries, keys, values, matching vectors, and relative-position features, respectively. The shared nonlinear readout f_o maps

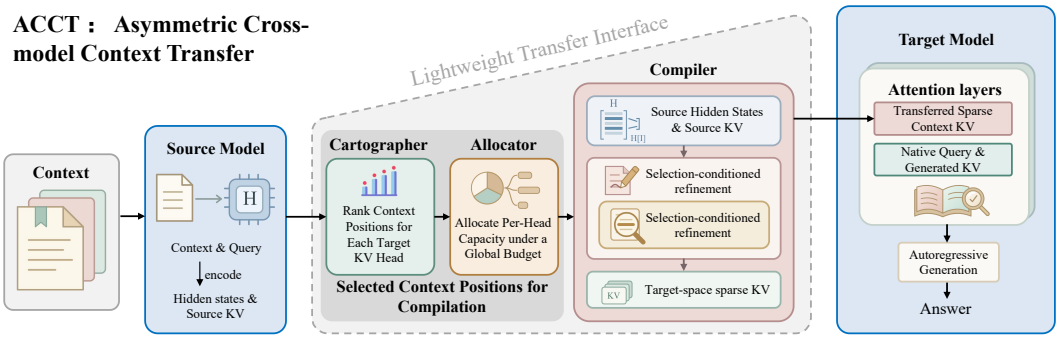

Figure 1: ACCT combines Cartographer selection, Allocator budgeting, and Compiler conversion to transfer source representations into sparse target KV for inference without context re-encoding.

fused features to a scalar. This shares query–context interactions while allowing the same position to rank differently across target units. Projections, matching, and positional encoding are specified in Supplementary Material A.1.

3.2.2 Allocator: Capacity Allocation under a Global Budget

ACCT separates content ranking from retained quantity: the Cartographer determines what to prioritize, while the Allocator learns capacity preferences for each model pair. Given retention ratio ρ and tail length τ, each head always retains the final $t = \min(\tau, N)$ positions. The average capacity k is $\lceil \rho N \rceil$ clipped to $[t, N]$, leaving prefix budget $B = P(k - t)$. For prefix length $n = N - t$ and integer allocations $k_p \in [0, n]$, selection is

$$\mathcal{I}_p = \mathrm{Top}_{k_p}(S_{p,0:n}) \cup \{N - t, \ldots, N - 1\}, \qquad \sum_p k_p = B. \tag{3.4}$$

The Allocator has only P trainable capacity parameters α_p. A shared adjustment λ maps them to continuous budgets:

$$b_p = n\sigma(\alpha_p - \lambda), \qquad \sum_p b_p = B. \tag{3.5}$$

Bisection solves for λ; flooring b_p and filling by fractional remainder gives integer capacities satisfying Eq. (3.4). Boundary budgets give zero or full capacity. For a fixed pair, length, and total budget, capacities are fixed while selected content varies with the context and query.

To train discrete selection, let $r_{p,i}$ be the zero-based prefix rank, with smaller values indicating greater importance. Soft gates preserve per-head capacity:

$$g_{p,i} = \sigma\left(\frac{\eta_p - r_{p,i}}{T_R}\right), \qquad \sum_{i=0}^{n-1} g_{p,i} = b_p. \tag{3.6}$$

Bisection sets threshold η_p, and temperature $T_R > 0$ controls smoothness; $b_p = 0, n$ give all-zero and all-one gates. The forward pass uses hard selection; backpropagation uses a soft-gate path that adds $\log g_{p,i}$ to context attention logits. This is a biased discrete-gradient surrogate (Supplementary Material C). Query and generated KV follow the native target path.

3.2.3 Compiler: Target KV Compilation with Shared Computation

The Compiler converts selected source representations into target-native KV. Its internal Bridge fits separate affine key and value maps between structurally aligned source and target layers and heads, producing base states $K^B_{p,i}, V^B_{p,i}$. Keys are aligned after native normalization and before RoPE (Su et al., 2021). Ridge least squares fits the Bridge on selected training positions, after which it is frozen.

The neural residual concatenates and projects context states from the final and three intermediate source layers into shared memory E. Let $\mathcal{U} = \bigcup_p \mathcal{I}_p$. Learnable latent slots $Z^{(0)}_W$ read the full

context, then positions in the selected union read those slots:

$$Z_W = \mathrm{LatentEncoder}(Z_W^{(0)}, E), \quad z_i = \mathrm{SlotDecoder}(E_i, Z_W), \quad i \in \mathcal{U}. \tag{3.7}$$

$\mathrm{LatentEncoder}$ aggregates memory with slot self-attention, context cross-attention, and feedforward layers, while $\mathrm{SlotDecoder}$ produces position representations with cross-attention to latent slots and feedforward layers; both are learnable modules. Each z_i is computed once and shared by all target units selecting that position, with no self-attention between selected positions. Retention density and original position form conditions $\xi_{p,i}$, giving $\widetilde{z}_{p,i} = z_i + W_\xi \xi_{p,i}$. Per-unit linear readouts then augment the base states:

$$\widehat{K}^{\mathrm{pre}}_{p,i} = K^B_{p,i} + \widetilde{z}_{p,i} W^K_p + b^K_p, \quad \widehat{V}_{p,i} = V^B_{p,i} + \widetilde{z}_{p,i} W^V_p + b^V_p. \tag{3.8}$$

Here W_ξ is the learnable conditioning projection, and each unit p has learnable key/value readout matrices W^K_p, W^V_p and biases b^K_p, b^V_p. Zero-initialized readouts and conditioning make compilation start from the Bridge. Mount KV after target RoPE at original positions; queries act through selection (Supplementary Material A.2).

3.3 Stagewise Learning

We learn selection and allocation with native target KV, then fix selection and learn compilation. Base weights remain frozen; target computation retains gradients to interface inputs.

Selection supervision. The Cartographer uses the local effect of deleting a position on target attention outputs as an importance proxy. Let $a_{j,i}, v_i, o_j$ denote native attention weights, values, and outputs. Fixing Q/K/V and renormalizing after deletion gives

$$u_{p,i} = \frac{1}{|\mathcal{O}_p|} \sum_{j \in \mathcal{O}_p} \left(\frac{a_{j,i}}{1 - a_{j,i}} \right)^2 \frac{\|v_i - o_j\|_2^2}{\|o_j\|_2^2 + \varepsilon_o}. \tag{3.9}$$

$\mathcal{O}_p$ collects observations from associated GQA query heads at the query end and during teacher continuation; ε_o stabilizes normalization. This utility supervises ranking through pairs sampled near the uniform-budget cutoff. The Cartographer is then frozen.

Capacity supervision. Using native target KV at selected positions, the Allocator learns α from the teacher-forced negative log-likelihood of annotated answers:

$$\mathcal{L}_{\mathrm{alloc}} = -\frac{1}{|y|} \sum_{t=0}^{|y|-1} \log p_T\big(y_t \mid q, y_{<t}; \mathcal{C}_T(c)|_{\mathcal{I}(\alpha)}\big). \tag{3.10}$$

Here y has $|y|$ annotated answer tokens, y_t is token t, $y_{<t}$ its prefix, and p_T the target distribution. $\mathcal{C}_T(c)|_{\mathcal{I}(\alpha)}$ restricts native target context KV to the selections determined by capacity parameters $\alpha = (\alpha_p)_{p=1}^P$. Soft gates propagate budget gradients; annotations enter only target supervision.

Compilation supervision. With the first two components fixed, we fit and freeze Bridge affine parameters β and optimize neural residual network parameters ψ:

$$\mathcal{L}_{\mathrm{comp}} = \mathcal{L}_{K,\mathrm{norm}} + 10^{-4}\mathcal{L}_{K,\mathrm{scale}} + \mathcal{L}_{K,\mathrm{attn}} + \mathcal{L}_V + \mathcal{L}_{\mathrm{beh}}. \tag{3.11}$$

The five terms respectively match normalized keys, log key scales, attention distributions, value geometry, and query-end vocabulary distributions; 10^{-4} is the fixed scale-loss weight. Attention fixes target queries and retains visible native query keys. In the behavioral KL divergence $D_{\mathrm{KL}}(p_T^{\mathcal{I}} \| p_\psi^{\mathcal{I}})$, the two distributions come from recomputing the query in the target with native and compiled KV, respectively, at the same selections $\mathcal{I}$. Target Full is the evaluation baseline; see Supplementary Material B–E.

3.4 Inference and Computational Properties

Algorithm 3.1 gives the complete inference procedure.

Algorithm 3.1 ACCT context transfer and inference

Require: Frozen source F_S and target F_T; trained interface Θ; context c, query q, retention ratio ρ, and tail length τ.

1: **Source encoding:** Encode $[c; q]$ with F_S to obtain the required hidden states and context KV.
2: **Selection and allocation:** The Cartographer scores positions; the Allocator selects positions for each target layer and KV head under the global budget, including the mandatory tail.
3: **Compilation and mounting:** The Compiler generates target KV at selected positions; apply target RoPE to the keys and mount KV at original positions.
4: **Target inference:** F_T encodes only q and generates y autoregressively, appending native KV during generation.

Ensure: Answer y.

Effective KV size. A full target context comprises a d_T-dimensional key and value for each of N positions in each of P layer–head units, totaling $2PNd_T$ effective KV elements. ACCT permits different capacities across units, while the budget constraint in Eq. (3.4) fixes their total retained slots, including the tail, at Pk. Its effective context KV therefore contains $2Pkd_T$ elements, with k/N denoting the retained fraction of effective KV elements. Union size and computational complexity are detailed in Supplementary Material F.2.

Execution cost. The target skips context prefill but still encodes queries, reads memory, and generates. Let T_S, T_{transfer}, and $T_{T,q}$ denote source encoding, the three components plus mounting, and target query encoding times. The full cost before generation is

$$T_{\text{ACCT}}^{\text{prefill}} = T_S([c; q]) + T_{\text{transfer}}(c, q) + T_{T,q}(q; \widehat{\mathcal{C}}). \tag{3.12}$$

Our efficiency evaluation reports only latency, focusing on the target side; Eq. (3.12) decomposes the full prefill cost, including source encoding and transfer.

4 Experiments

We evaluate ACCT in both transfer directions across three model families, examining whether transferred states support target inference and how state capacity affects different tasks.Although these target models also serve as source models in the reverse-direction pairs, we do not directly compare their results with those of the reverse-direction pairs here, because the two settings use different decoding models, and differences in task performance cannot be attributed solely to state transfer.

4.1 Experimental Setup

Models and datasets. We evaluate bidirectional transfer between Qwen3-4B and Qwen3-8B, Llama-3.2-3B and Llama-3.1-8B, and Qwen2.5-7B and Qwen2.5-32B, yielding six directed settings. Each pair shares a token-to-ID mapping and uses the target tokenizer and prompts.

Evaluation covers five general-capability benchmarks (HellaSwag, Winogrande, MMLU, PIQA, and WikiText2) and six context-dependent tasks (SearchQA, TriviaQA, BoolQ, RACE, WikiHop, and PubMedQA). We use 500 examples each from WikiText2 and PubMedQA and 1,000 from each remaining dataset, yielding the same fixed set of 10,000 evaluation examples for every direction.

Comparison setup. We evaluate the reasoning performance supported by transferred states when the target does not re-encode the context, together with the computational cost of constructing these states. Target Full provides a task-performance reference by processing the complete context natively, whereas ACCT supports target inference through cross-model state transfer. We assess this process in terms of both task performance and state-construction latency.DroidSpeak requires models with the same architecture, whereas C2C relies on the target model's own context prefill; neither directly satisfies our cross-scale setting without target-side context re-encoding, so we exclude them from the main-table comparison.

Training setup. We train a separate interface for each transfer direction while keeping both source and target models frozen. Main experiments use $4\times$ compression, retaining approximately one quarter of target context KV slots under the budget rule in Section 3, including the most recent 32 positions. The Cartographer and Allocator use 5,000 SearchQA training examples; the Compiler uses a fixed subset of 500 for four epochs. Training schedules, model sources, and interface capacity rules are provided in Supplementary Material D–E.

Evaluation metrics. We report token-level F1 for SearchQA and TriviaQA, PPL computed from mean token negative log-likelihood for WikiText2, and accuracy for the remaining tasks. General multiple-choice tasks use candidate-answer likelihoods; dataset sources and evaluation details appear in Appendix G. F1 and accuracy are reported as percentages (higher is better); PPL is reported on its original scale (lower is better). For efficiency, we compare KV state-conversion latency with native target context prefill latency; the measurement protocol is given in Appendix K.

4.2 Inference Efficiency

Longer contexts amplify reuse gains. Figure 2 shows faster state conversion in all six directions at every measured length. Speedups increase from 1.5–$8.6\times$ on the shortest contexts (approximately 1K tokens) to 6.5–$21.2\times$ on the longest (approximately 19K tokens), where conversion requires only 4.7–15.3% of target prefill time. Target prefill processes every context token through the full target network, whereas ACCT constructs target states from existing source representations using shared lightweight computation. Conversion latency grows more slowly with context length, making the savings from avoiding target re-encoding increasingly pronounced.

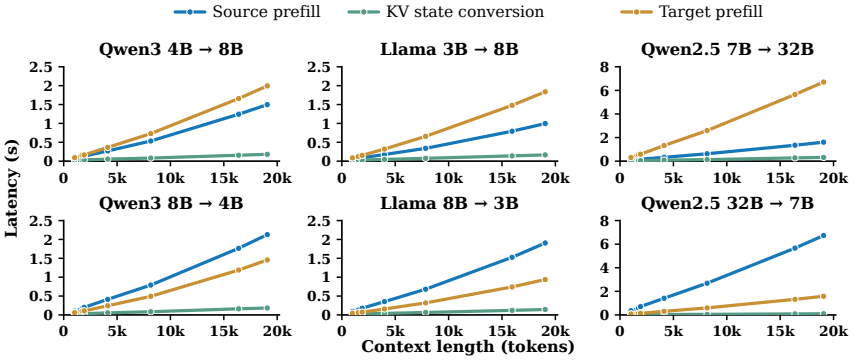

Figure 2: Source prefill, KV state conversion, and target prefill latency across six directions.

4.3 Task Performance

Table 1 summarizes task performance across three model families in both directions at $4\times$ compression. Alongside the substantial reduction in target context computation, ACCT incurs only modest performance losses on several tasks and exceeds Target Full on others.

Compact states effectively support target inference. ACCT partially preserves capabilities across diverse tasks through compressed source representations, with Qwen3 approaching or exceeding Target Full on multiple benchmarks. These results show that the target can use context information through transferred states without re-encoding the original context, connecting existing context computation to downstream inference.

Transfer capabilities extend across tasks. The interface uses only SearchQA for training, yet Qwen3 transfer also yields improvements over Target Full on TriviaQA, MMLU, WikiText2, and PubMedQA, spanning open-domain QA, knowledge reasoning, language modeling, and domain-specific QA. All six directions also retain much of the target's PIQA performance, demonstrating the potential for task generalization from a single training dataset.

The method's interface design is reusable across model families. We apply the same three-component architecture and capacity adaptation rules to within-family model pairs in Qwen3, Llama, and Qwen2.5, training a separate interface for each transfer direction. These experiments demonstrate the applicability of a shared interface design across model families, while context transfer itself is evaluated between models within the same family.

Table 1: Task performance (F1/accuracy in %; PPL ↓); bold marks improvements and Relative (Relative Performance) is ACCT/Target Full (inverted for PPL).

		General capability					Context-dependent tasks					
Model pair	Method	Hella Swag Acc.	Wino grande Acc.	MMLU Acc.	PIQA Acc.	Wiki Text2 PPL ↓	Search QA F1	Trivia QA F1	BoolQ Acc.	RACE Acc.	WikiHop Acc.	PubMed QA Acc.
Qwen3 4B→8B	Target Full	39.70	58.50	59.70	73.20	32.91	46.65	50.48	86.30	87.60	49.60	48.80
	ACCT	33.20	51.50	**65.70**	73.10	**25.94**	**67.07**	**66.31**	61.90	63.50	36.50	40.20
	Relative	0.84×	0.88×	1.10×	1.00×	1.27×	1.44×	1.31×	0.72×	0.72×	0.74×	0.82×
Qwen3 8B→4B	Target Full	37.10	51.00	63.60	69.20	81.29	38.56	37.68	84.00	87.10	45.80	34.00
	ACCT	36.90	**51.30**	58.20	69.00	98.92	**50.33**	**54.51**	60.20	66.30	32.50	**49.60**
	Relative	0.99×	1.01×	0.92×	1.00×	0.82×	1.31×	1.45×	0.72×	0.76×	0.71×	1.46×
Llama 3B→8B	Target Full	54.60	49.10	58.60	77.40	7.21	63.44	78.71	82.00	68.30	37.90	70.00
	ACCT	27.00	**50.10**	41.90	69.30	64.91	**66.88**	69.71	63.50	46.20	18.60	53.60
	Relative	0.49×	1.02×	0.72×	0.90×	0.11×	1.05×	0.89×	0.77×	0.68×	0.49×	0.77×
Llama 8B→3B	Target Full	50.20	49.30	45.20	73.80	9.04	37.96	69.50	69.00	57.60	29.60	66.00
	ACCT	27.00	**50.50**	33.00	68.70	65.36	**54.00**	56.45	42.60	32.20	16.80	33.20
	Relative	0.54×	1.02×	0.73×	0.93×	0.14×	1.42×	0.81×	0.62×	0.56×	0.57×	0.50×
Qwen2.5 7B→32B	Target Full	61.10	57.80	72.40	80.30	5.83	44.05	34.10	85.30	93.90	50.00	57.00
	ACCT	29.60	50.10	64.60	70.20	26.24	**46.47**	33.58	56.60	57.50	25.40	47.00
	Relative	0.48×	0.87×	0.89×	0.87×	0.22×	1.05×	0.98×	0.66×	0.61×	0.51×	0.82×
Qwen2.5 32B→7B	Target Full	57.00	51.80	54.30	76.80	8.40	46.63	41.15	84.60	85.30	33.00	64.60
	ACCT	38.50	49.30	50.90	69.10	26.35	22.35	25.48	63.00	47.30	18.50	54.80
	Relative	0.68×	0.95×	0.94×	0.90×	0.32×	0.48×	0.62×	0.74×	0.55×	0.56×	0.85×

Table 2: Performance of Qwen3 4B↔8B under 1×, 2×, and 4× compression.

Model pair	Budget	Hella Swag Acc.	Wino grande Acc.	MMLU Acc.	PIQA Acc.	Wiki Text2 PPL ↓	Search QA F1	Trivia QA F1	BoolQ Acc.	RACE Acc.	WikiHop Acc.	PubMed QA Acc.
Qwen3 4B→8B	1×	38.60	51.40	55.80	70.90	23.26	38.54	47.03	77.80	69.40	35.10	45.60
	2×	36.80	53.10	62.70	73.20	22.16	67.59	74.92	70.30	71.10	39.10	27.60
	4×	33.20	51.50	65.70	73.10	25.94	67.07	66.31	61.90	63.50	36.50	40.20
Qwen3 8B→4B	1×	37.40	50.50	44.70	69.30	60.74	12.54	14.11	68.00	56.00	4.80	48.40
	2×	32.80	52.50	53.60	70.10	142.09	61.54	57.65	62.60	66.80	36.10	55.20
	4×	36.90	51.30	58.20	69.00	98.92	50.33	54.51	60.20	66.30	32.50	49.60

Transfer effectiveness depends on the task. Gains are more pronounced in open-domain QA, while gaps from Target Full remain in reading comprehension, multi-hop reasoning, and some language-modeling settings. This suggests that task demands for context coverage and representation precision influence transfer.

4.4 Fine-Grained Analysis

Table 2 compares compression budgets for Qwen3 4B↔8B. Each setting is trained separately, with Compiler width, depth, and training schedule fixed within each direction.

The capacity budget offers flexible control over context use. Moving from 4× to 2× compression improves most tasks in both directions, including all six context tasks in reverse transfer. The budget thus provides a practical way to adjust how much context information ACCT retains and choose state capacity for different inference needs.

Information selection makes compact states more effective. The 1× setting jointly ablates the Cartographer and Allocator while retaining the Compiler. On open-domain QA in both directions, selective transfer at 2× and 4× outperforms full retention at 1×, showing that, on these open-domain QA tasks, selective transfer achieves better performance than full retention with fewer KV states.

Component ablations and state controls. At $4\times$ compression with the 32-token tail, *Random* replaces Cartographer rankings with random orderings and retrains the Allocator and Compiler; *Uniform* retains rankings, equalizes per-layer/head capacity, and retrains the Compiler. At unchanged positions, *Bridge-only* removes the neural residual and *Native KV* uses target-native states, both without retraining. Table 3 reports both directions; matched training settings and the 80,000-example evaluation protocol are detailed in Appendix J.

Table 3: Component ablations and state controls for Qwen3 at $4\times$ compression (F1/accuracy in %; PPL ↓).

Model pair	Method	Hella Swag Acc.	Wino grande Acc.	MMLU Acc.	PIQA Acc.	Wiki Text2 PPL ↓	Search QA F1	Trivia QA F1	BoolQ Acc.	RACE Acc.	WikiHop Acc.	PubMed QA Acc.
Qwen3 4B→8B	ACCT	33.20	51.50	65.70	73.10	25.94	67.07	66.31	61.90	63.50	36.50	40.20
	Random	31.70	50.80	27.00	61.20	68.42	11.09	20.12	38.90	27.30	15.30	51.60
	Uniform	35.20	50.60	62.80	72.40	25.24	41.07	42.72	62.90	64.20	34.10	22.80
	Bridge-only	35.10	51.10	41.70	70.20	35.24	44.15	50.13	63.10	61.70	33.80	15.40
	Native KV	38.00	57.40	54.30	73.20	37.19	59.94	59.58	67.20	75.30	44.50	36.20
Qwen3 8B→4B	ACCT	36.90	51.30	58.20	69.00	98.92	50.33	54.51	60.20	66.30	32.50	49.60
	Random	29.20	50.00	23.50	57.30	172.93	3.67	5.20	38.90	24.90	3.70	19.40
	Uniform	31.90	51.50	51.10	69.00	145.68	57.82	59.44	61.20	64.10	35.50	58.00
	Bridge-only	31.30	48.50	41.20	64.80	155.10	36.28	34.98	49.60	58.80	31.90	17.40
	Native KV	32.50	50.90	62.20	69.20	117.63	56.26	49.92	51.50	74.80	43.00	16.80

Learned localization is central to effective transfer. Random selection substantially degrades diverse tasks in both directions even after downstream components are retrained. Learned localization maintains advantages in QA, knowledge reasoning, and language modeling, showing that the Cartographer retains more useful content under a fixed budget and provides an effective information basis for allocation and compilation.

Learned allocation benefits multiple benchmarks. At the same total budget, learned allocation improves performance on multiple benchmarks, including forward open-domain QA and MMLU in both directions, but shows no advantage over uniform allocation on some benchmarks.

Residual compilation strengthens cross-model state alignment. The full Compiler improves open-domain QA and MMLU and lowers language-modeling perplexity relative to Bridge-only in both directions; reverse-transfer gains cover every evaluated task. The residual network effectively complements affine mapping, making selected source representations more suitable for target consumption.

Compiled states approach or surpass native KV on multiple tasks. With positions and capacity fixed, ACCT nearly matches Native KV on PIQA and outperforms it on TriviaQA, PubMedQA, and WikiText2 in both directions. This control evaluates the quality of Compiler-generated states, showing that source representations can be compiled into target states with inference performance approaching or exceeding native KV on these tasks.

5 Conclusion

ACCT decouples context computation from inference by compiling source representations into sparse target-native KV. Bidirectional experiments across three model families show performance approaching or exceeding Target Full on multiple benchmarks. KV state conversion substantially outpaces target context prefill, especially for long contexts. These findings support context computation as a reusable resource across models, enabling flexible and efficient heterogeneous LLM systems.

AI USE STATEMENT

Generative AI tools were used to assist with literature search, manuscript drafting and language refinement, code implementation and debugging, experimental execution and reproduction, analysis of experimental results, and manuscript formatting. The authors reviewed and revised the AI-assisted text, code, and analyses, and take full responsibility for the research design, experimental procedures, reported results, references, and final content of this work, including all claims and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 3 and Appendices A–C describe the architecture, supervision, and budget optimization. Appendices D–F document training schedules, random seeds, model-pair capacity rules, and execution interfaces. Section 4 and Appendix G specify the datasets, input and decoding protocols, and evaluation metrics; Appendix J details the ablation controls. Appendix K reports hardware and execution settings, timing boundaries, repetition counts, and per-length latency measurements, including state construction with source encoding. Additional model-pair results are provided in Appendix I.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv preprint arXiv:1607.06450*, 2016. URL https://arxiv.org/abs/1607.06450.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. doi: 10.1609/aaai.v34i05.6239. URL https://arxiv.org/abs/1911.11641.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024. URL https://arxiv.org/abs/2406.02069.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=cSimKw5p6R.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. SearchQA: A New Q&A Dataset Augmented with Context from a Search Engine. *arXiv preprint arXiv:1704.05179*, 2017. URL https://arxiv.org/abs/1704.05179.

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. *arXiv preprint arXiv:2510.03215*, 2025. doi: 10.48550/arXiv.2510.03215. URL https://arxiv.org/abs/2510.03215.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, 2024. URL https://arxiv.org/abs/2311.04934.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016. URL https://arxiv.org/abs/1606.08415.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2009.03300.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015. URL https://arxiv.org/abs/1503.02531.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver IO: A General Architecture for Structured Inputs & Outputs. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2107.14795.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. PubMedQA: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2567–2577, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1259. URL https://aclanthology.org/D19-1259/.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

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*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 785–794, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1082. URL https://aclanthology.org/D17-1082/.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024. URL https://arxiv.org/abs/2404.14469.

Yuhan Liu, Yuyang Huang, Jiayi Yao, Shaoting Feng, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, Madan Musuvathi, and Esha Choukse. DroidSpeak: KV cache sharing for cross-LLM communication and multi-LLM serving. *arXiv preprint arXiv:2411.02820*, 2024. doi: 10.48550/arXiv.2411.02820. URL https://arxiv.org/abs/2411.02820.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1609.07843.

Meta. Llama 3.2 Model Card, 2024. URL https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Adam Paszke, Sam Gross, Francisco Massa, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1912.01703.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual Reasoning with a General Conditioning Layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. doi: 10.1609/aaai.v32i1.11671. URL https://arxiv.org/abs/1709.07871.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. doi: 10.1609/aaai.v34i05.6399. URL https://arxiv.org/abs/1907.10641.

Noam Shazeer. GLU Variants Improve Transformer. *arXiv preprint arXiv:2002.05202*, 2020. URL https://arxiv.org/abs/2002.05202.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arXiv preprint arXiv:2104.09864*, 2021. URL https://arxiv.org/abs/2104.09864.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, 2017. URL https://arxiv.org/abs/1706.03762.

Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, 6:287–302, 2018. doi: 10.1162/tacl_a_00021. URL https://aclanthology.org/Q18-1021/.

An Yang, Baosong Yang, Beichen Zhang, et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1910.07467.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023. URL https://arxiv.org/abs/2306.14048.

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. *arXiv preprint arXiv:2312.07104*, 2023. URL https://arxiv.org/abs/2312.07104.

A Component Parameterization and Bridge Fitting

A.1 Cartographer: query-conditioned ranking

Let $H_c \in \mathbb{R}^{N\times D_S}$ and $H_q \in \mathbb{R}^{Q\times D_S}$ denote final-layer source context and query states. A shared map f_θ applies LayerNorm (Ba et al., 2016), a linear projection, GELU (Hendrycks & Gimpel, 2016), and LayerNorm, producing width-w_M features $X = f_\theta(H_c)$ and $Y = f_\theta(H_q)$. Four learnable query slots $A \in \mathbb{R}^{4\times w_M}$ aggregate the query and then read context using scaled dot-product attention (Vaswani et al., 2017) with d_M-dimensional projections:

$$U = \mathrm{softmax}(AY^\top/\sqrt{w_M})Y, \qquad Z = \mathrm{LN}\left(UW_Q + \mathrm{softmax}\left(\frac{(UW_Q)(XW_K)^\top}{\sqrt{d_M}}\right)XW_V\right). \tag{A.1}$$

The learnable matrices $W_Q, W_K, W_V \in \mathbb{R}^{w_M\times d_M}$ project the features. The two softmax operations normalize over query and context positions, respectively, excluding padding. All target units share Z. For $p = (\ell, h)$, concatenate c_M-dimensional layer and head embeddings and project them to $e_p \in \mathbb{R}^{d_M}$. Together with $\bar{Z}$, this gives FiLM parameters $(\widetilde{\gamma}_p, \beta_p) = \mathrm{MLP}([e_p; \bar{Z}])$, where $\gamma_p = 1 + \tanh\widetilde{\gamma}_p$.

$$\begin{aligned} u_i &= W_x X_i + b_x, \qquad E_{p,a} = Z_a \odot [1 + \tanh(W_g e_p + b_g)], \\ m_{p,i,a} &= \langle u_i, E_{p,a}\rangle/\sqrt{d_M}, \qquad \phi_i = [i/N,\ \log(1+N-i)/\log(1+N)], \\ S_{p,i} &= w_o^\top \mathrm{Dropout}(\mathrm{GELU}(\gamma_p \odot u_i + \beta_p + W_m \mathbf{m}_{p,i} + W_\phi \phi_i)). \end{aligned} \tag{A.2}$$

Here $a \in \{1,\dots,4\}$ indexes query slots, $\mathbf{m}_{p,i}$ collects their four matching scores, and $i \in \{0,\dots,N-1\}$. The learnable W_x, b_x project tokens, W_g, b_g modulate slots, W_m, W_ϕ project matching and position features, and w_o reads out a scalar. Inference disables dropout and stably sorts non-tail positions by decreasing score. Scores determine content order; the Allocator determines capacity separately.

A.2 Compiler: affine Bridge and shared residual

A.2.1 Affine Bridge inside the Compiler

The source has L_S layers, H_S physical KV heads, and head dimension d_S. Relative-depth alignment and a modulo head map give $\pi_L(\ell) = \lfloor \ell(L_S-1)/(L_T-1) + 1/2 \rfloor$ and $\pi_H(h) = h \bmod H_S$; set $\pi_L(0) = 0$ if $L_T = 1$. These are fixed structural correspondences, not assumed semantic equivalences. Each target unit has separate affine key and value maps:

$$K^B_{p,i} = K^S_{\pi_L(\ell),\pi_H(h),i} A^K_p + a^K_p, \qquad V^B_{p,i} = V^S_{\pi_L(\ell),\pi_H(h),i} A^V_p + a^V_p. \tag{A.3}$$

The matrices $A^{K/V}_p \in \mathbb{R}^{d_S\times d_T}$ have d_T-dimensional biases. Keys are taken after native normalization and before RoPE. Fitting uses only valid positions in the frozen selections. For one head and vector type, let X, Y stack n_p source and target observations, with centered matrices X_c, Y_c. Then

$$\begin{aligned} &\min_{A,a} \|XA + \mathbf{1}a^\top - Y\|_F^2 + \lambda_B(n_p/d_S)\|A\|_F^2, \\ &\quad A = [X_c^\top X_c + \lambda_B(n_p/d_S)I]^{-1} X_c^\top Y_c, \qquad a = \bar{y} - \bar{x}A. \end{aligned} \tag{A.4}$$

We accumulate sufficient statistics and solve the linear system in double precision, without penalizing the bias. Set $\lambda_B = 10^{-2} d_T/128$. The Bridge is fitted on the Compiler's 500 training examples and frozen before optimizing the neural residual on the same subset.

A.2.2 Shared residual and native KV mounting

The residual network concatenates the final source layer and three intermediate layers at $\ell_j = \mathrm{clip}(\mathrm{round}(jL_S/4), 1, L_S)$ for $j = 1, 2, 3$, indexed in the returned hidden-state sequence. A lin-

ear projection gives $E \in \mathbb{R}^{N \times w_W}$. Following the latent read/write attention pattern of Perceiver IO (Jaegle et al., 2022), starting from m_W learnable latent slots, each of D_Z blocks applies slot self-attention, cross-attention to all of E, and a SwiGLU (Shazeer, 2020) feedforward transformation, with RMSNorm (Zhang & Sennrich, 2019) and residual connections at every sublayer.

$$\begin{aligned} Z' &= Z + \mathrm{SA}(\mathrm{RMSNorm}(Z)), \\ Z'' &= Z' + \mathrm{CA}(\mathrm{RMSNorm}(Z'), E), \\ Z^{+} &= Z'' + \mathrm{SwiGLU}(\mathrm{RMSNorm}(Z'')), \qquad \mathrm{SwiGLU}(x) = [\mathrm{SiLU}(xW_1) \odot xW_2]W_3. \end{aligned} \tag{A.5}$$

The feedforward matrices are $W_1, W_2 \in \mathbb{R}^{w_W \times f_W}$ and $W_3 \in \mathbb{R}^{f_W \times w_W}$; SA and CA have learnable attention projections. For each position in $\mathcal{U} = \bigcup_p \mathcal{I}_p$, initialize with E_i and apply D_U blocks containing cross-attention to latent slots and feedforward transformations, followed by RMSNorm to obtain z_i. Selected positions do not self-attend. All target units selecting the same position share its z_i.

$$\begin{aligned} \xi_{p,i} &= [\delta_p, \delta_p^2, i/\max(1, N-1), 1], \quad \delta_p = |\mathcal{I}_p|/N, \quad \widetilde{z}_{p,i} = z_i + W_\xi \xi_{p,i}, \\ \widehat{K}^{\mathrm{pre}}_{p,i} &= K^B_{p,i} + \widetilde{z}_{p,i} W_p^K + b_p^K, \qquad \widehat{V}_{p,i} = V^B_{p,i} + \widetilde{z}_{p,i} W_p^V + b_p^V. \end{aligned} \tag{A.6}$$

The matrix W_ξ projects the four conditions; $W_p^K, W_p^V \in \mathbb{R}^{w_W \times d_T}$ and b_p^K, b_p^V are unit-specific readouts. Conditioning and readout weights and biases are zero-initialized, so the initial output equals the Bridge. The Compiler does not separately read query hidden states; the query affects its output through selection.

Before mounting, apply target RoPE at original positions, $\widehat{K}_{p,i} = \mathrm{RoPE}_T(\widehat{K}^{\mathrm{pre}}_{p,i}, i)$, retaining positions and validity masks. Query positions start at N and generation continues at $N + Q$. External context and visible native query/generated KV share the target attention normalizer. Mandatory-tail states are also compiled, without additional native target context computation.

B Supervision and Training Objectives

B.1 Local deletion utility and ranking

Fix Q/K/V for a target attention head. Let $a_{j,i}$ be the weight on position i in the visible history, v_i its value, and $o_j = \sum_i a_{j,i} v_i$. Removing position i and renormalizing gives the exact local identity:

$$o_j^{\setminus i} = \frac{o_j - a_{j,i} v_i}{1 - a_{j,i}}, \qquad o_j^{\setminus i} - o_j = \frac{a_{j,i}}{1 - a_{j,i}}(o_j - v_i). \tag{B.1}$$

$$u_{p,i} = \frac{1}{|\mathcal{O}_p|} \sum_{j \in \mathcal{O}_p} \left(\frac{a_{j,i}}{1 - a_{j,i}} \right)^2 \frac{\|v_i - o_j\|_2^2}{\|o_j\|_2^2 + \varepsilon_o}. \tag{B.2}$$

The set $\mathcal{O}_p$ collects associated GQA query heads at up to eight final query positions and eight greedy teacher-continuation positions, sampling at most four target layers per example. The implementation floors $1 - a_{j,i}$ and scales the stabilizer ε_o using output energy. This utility measures a local change with fixed inputs, rather than the task loss from rerunning the entire model after deletion. Teacher continuation is used only for supervision.

$$\mathcal{L}_M = \mathrm{Mean}_p \, \mathrm{Mean}_{(i,j) \in \mathcal{P}_p} \, \mathrm{softplus}\left(\frac{S_{p,j} - S_{p,i}}{\max(\mathrm{Std}(S_{p,0:n}), 10^{-4})} \right). \tag{B.3}$$

Pairs satisfy $u_{p,i} > u_{p,j}$. Each valid head supplies 32 pairs: half straddle the uniform-budget utility cutoff and half come from local bands on either side. Ties in utility are excluded. Score-standard-deviation normalization limits improvements obtained merely by increasing score magnitude.

B.2 Allocation objective and teacher definitions

After freezing the Cartographer, the Allocator uses selected native target KV and minimizes teacher-forced answer NLL. It uses the first canonical answer, capped at 64 tokens; answers enter target supervision but not source-side ranking inputs. The extra output-KL weight is zero. The Compiler instead uses a same-selection native-KV teacher, extracting $\mathcal{I}$ from native target caches and recomputing the query:

$$p_T^{\mathcal{I}}(\cdot \mid q, c) = p_T(\cdot \mid q; \mathcal{C}_T(c)|_{\mathcal{I}}), \qquad p_\psi^{\mathcal{I}}(\cdot \mid q, c) = p_T(\cdot \mid q; \widehat{\mathcal{C}}_\psi|_{\mathcal{I}}). \tag{B.4}$$

The Compiler aligns these distributions to learn representation conversion at fixed positions and capacity. Both differ from the evaluation reference Target Full, which uses the entire original context. Selection and synthesis are separated in training design; this does not imply additive errors in downstream task metrics.

B.3 Compiler geometry, attention, and behavior losses

Let K^*, V^* denote native target pre-RoPE keys and values at selected positions. Geometry terms average valid slots within each example before averaging examples. Define $r(v) = \sqrt{\|v\|_2^2/d_T + \epsilon}$ and $\nu(v) = \max(\|v\|_2, \epsilon)$, with $\epsilon = 10^{-6}$.

$$\begin{aligned}
\mathcal{L}_{K,\text{norm}} &= \text{Mean}_{p,i} \frac{1}{d_T} \left\| \frac{\widehat{K}^{\text{pre}}_{p,i}}{r(\widehat{K}^{\text{pre}}_{p,i})} - \frac{K^*_{p,i}}{r(K^*_{p,i})} \right\|_2^2, \\
\mathcal{L}_{K,\text{scale}} &= \text{Mean}_{p,i} [\log r(\widehat{K}^{\text{pre}}_{p,i}) - \log r(K^*_{p,i})]^2, \\
\mathcal{L}_V &= \text{Mean}_{p,i} \left[\left\| \frac{\widehat{V}_{p,i}}{\nu(\widehat{V}_{p,i})} - \frac{V^*_{p,i}}{\nu(V^*_{p,i})} \right\|_2^2 + [\log \nu(\widehat{V}_{p,i}) - \log \nu(V^*_{p,i})]^2 \right].
\end{aligned} \tag{B.5}$$

Attention supervision uses up to the final 32 query positions from the native full-prompt forward pass. Fix teacher query vectors q_j^* and visible native query keys $K^*_{q,\le j}$, replacing only selected context keys:

$$\begin{aligned}
a_j^* &= \text{softmax}\Big(q_j^* [K_{\mathcal{I}}^{*,\text{rot}}; K^*_{q,\le j}]^\top / \sqrt{d_T} \Big), \\
\widehat{a}_j &= \text{softmax}\Big(q_j^* [\widehat{K}_{\mathcal{I}}; K^*_{q,\le j}]^\top / \sqrt{d_T} \Big), \qquad \mathcal{L}_{K,\text{attn}} = \text{Mean}_{\ell,h_q,j} \, D_{\text{KL}}(a_j^* \| \widehat{a}_j).
\end{aligned} \tag{B.6}$$

Here h_q indexes query heads, $\mathcal{I}$ is the selection of the corresponding physical KV head, and rot denotes target RoPE. Joint normalization preserves competition between context and query history. Following distribution-based knowledge distillation (Hinton et al., 2015), behavioral supervision mounts native and compiled selections separately, recomputes the query, and compares full-vocabulary distributions at up to eight final query positions:

$$\mathcal{L}_{\text{beh}} = \text{Mean}_{j \in \mathcal{J}} \, D_{\text{KL}}(p_{T,j}^{\mathcal{I}} \| p_{\psi,j}^{\mathcal{I}}), \qquad |\mathcal{J}| = \min(8, Q). \tag{B.7}$$

This KL uses temperature one and a stopped-gradient teacher; the compiled path retains input gradients. It uses no answer annotations and does not require generating eight answer tokens. The final objective is the five-term sum in the main compilation-supervision equation, with weight 10^{-4} only on key scale. Joint residual training begins from the Bridge plus zero residual, without separate key/value pretraining.

C Budget Conservation and Gradient Derivations

C.1 Continuous allocation, rounding, and uniqueness

Let $t = \min(\tau, N)$ be the mandatory tail length, $k = \min(N, \max(t, \lceil \rho N \rceil))$ the average total capacity per head, $n = N - t$ the ranked-prefix length, and $B = P(k - t)$ the prefix budget. For $0 < B < Pn$, static learnable preferences α_p and a shared adjustment λ give capacities:

$$b_p = n\sigma(\alpha_p - \lambda), \qquad \sum_{p=1}^{P} b_p = B. \tag{C.1}$$

For finite α_p, total capacity is continuous and strictly decreasing in λ, with limits Pn and zero. Thus each interior budget has a unique solution, obtained by bisection. Boundaries directly return zero or full capacity. The mapping is also the first-order condition of the strictly concave problem below, where $\mathcal{H}(z) = -z \log z - (1 - z) \log(1 - z)$:

$$\max_{0 \le b_p \le n,\ \sum_p b_p = B} \sum_p [\alpha_p b_p + n\mathcal{H}(b_p/n)]. \tag{C.2}$$

This optimization characterizes the budget map, not an optimum of the downstream task loss. Rounding starts with $f_p = \lfloor b_p \rfloor$ and assigns one extra slot to the $r = B - \sum_p f_p$ non-full heads with largest fractional parts, breaking ties deterministically. Since B is integral, the resulting k_p satisfy $\sum_p k_p = B$; adding tails gives $\sum_p |\mathcal{I}_p| = Pk$ without losing capacity through rounding.

C.2 Derivatives of constrained allocation and soft rank gates

Define $\omega_p = n\sigma(\alpha_p - \lambda)[1 - \sigma(\alpha_p - \lambda)]$. Differentiation gives $\mathrm{d}b_p = \omega_p(\mathrm{d}\alpha_p - \mathrm{d}\lambda)$. Conservation of the total budget yields

$$\mathrm{d}\lambda = \frac{\sum_p \omega_p \, \mathrm{d}\alpha_p}{\sum_p \omega_p}, \qquad \frac{\partial b_p}{\partial \alpha_j} = \omega_p \mathbf{1}[p = j] - \frac{\omega_p \omega_j}{\sum_r \omega_r}. \tag{C.3}$$

Every first-order capacity change therefore sums to zero. For upstream gradient $v_p = \partial \mathcal{L}/\partial b_p$, backpropagation gives $\partial \mathcal{L}/\partial \alpha_p = \omega_p[v_p - \sum_j v_j \omega_j / \sum_j \omega_j]$, with zero gradients at boundaries. With frozen ranking and rank $r_{p,i}$, define soft gates whose total is b_p:

$$g_{p,i} = \sigma((\eta_p - r_{p,i})/T_R), \quad \sum_i g_{p,i} = b_p, \qquad \frac{\partial g_{p,i}}{\partial b_p} = \frac{g_{p,i}(1 - g_{p,i})}{\sum_j g_{p,j}(1 - g_{p,j})}. \tag{C.4}$$

Set $T_R = 2$ and solve η_p by bisection; boundary capacities give all-zero or all-one gates. Implicit differentiation cancels the explicit temperature factor, while temperature still changes gates and the gradient distribution. The forward pass uses hard selection at integer capacities. Backpropagation uses a soft path that adds $\log g_{p,i}$ to context attention logits, retaining native query-history normalization. This is a biased surrogate for discrete selection, not the exact derivative of top-k.

D Training Configuration and Data Boundaries

We train a separate interface for each direction, freezing both base models throughout. Main runs use $\rho = 0.25$ and $\tau = 32$. The Cartographer and Allocator use a fixed pool of 5,000 SearchQA training examples; the Compiler uses a fixed 500-example subset. The Cartographer additionally uses 1,000 validation examples for generation evaluation at the end of its stage.

All three trainable stages use AdamW (Loshchilov & Hutter, 2019). The Cartographer's final batch contains eight examples. Four Compiler epochs correspond to 2,000 example presentations and 500 updates; Bridge fitting makes an additional pass over 500 examples. The Allocator and Compiler

Table 4: Recorded training schedules.

Setting	Cartographer	Allocator	Compiler residual
Training examples	5,000	5,000	500
Epochs	1	1	4
Global batch	16	4	4
Optimizer updates	313	1,250	500
Optimizer	AdamW	AdamW	AdamW
Learning rate	10^{-4}	10^{-2}	10^{-4}
Weight decay	0.01	0	0.01
Gradient clip	1	1	1
Seed	17	434781	469601542

retain their fixed training endpoints without an additional best-epoch search; the Cartographer uses stage-end validation generation metrics. Data and optimization costs are reported by stage: the complete pipeline is not trained on only 500 examples.

Source forwards use BF16; Cartographer ranking and its teacher use FP32. The Allocator and Compiler use the frozen target's BF16 path. Compiler forwards use BF16 autocast, with critical normalizations and losses in FP32. Freezing target parameters does not remove gradients to interface inputs. We freeze θ after Cartographer training, α after Allocator training, and β after Bridge fitting, then optimize only ψ.

Compression settings are trained separately, keeping Compiler width, depth, and training schedule fixed within each direction. The $1\times$ setting bypasses ranking and allocation but still learns compilation, so it is not Target Full. Results represent the recorded single run per configuration; no unmeasured standard deviations, significance tests, or confidence intervals are supplied.

E Pair Capacity Rules and Parameter Counts

Model sources are the Qwen3 (Yang et al., 2025), Qwen2.5 (Yang et al., 2024), and Llama 3 (Grattafiori et al., 2024) technical reports and the official Llama 3.2 model card (Meta, 2024).

E.1 Capacity from model geometry

Source hidden width D_S determines input projections; $P = L_T H_T$ and target head dimension d_T determine output geometry. To adapt the shared trunk as well, let D_T be target hidden width and define the following preset rule, where $\lceil x \rceil_a = a\lceil x/a \rceil$ denotes upward alignment:

$$\begin{aligned} s &= \max\{1, \sqrt{D_S/1024}, \sqrt{D_T/4096}, \sqrt{P/288}\}, \\ (w_M, d_M, c_M) &= (\lceil 256s \rceil_{32}, \lceil 64s \rceil_{16}, \lceil 32s \rceil_8), \qquad m_M = 4, \\ w_W &= \lceil 256s \rceil_{32}, \quad m_W = \lceil 16s \rceil_4, \quad D_Z = D_U = \max(2, \lceil 2s \rceil), \quad f_W = \lceil 2w_W \rceil_8. \end{aligned} \tag{E.1}$$

Compiler attention uses dimension 32 per head, hence $w_W/32$ heads, and retains three intermediate taps. The Allocator always contains only P parameters. This rule standardizes configurations rather than asserting an empirically optimal scaling law. Source width, target geometry, and shared capacity can all change in the reverse direction, whose interface is trained separately.

Limitations under strong size asymmetry. The current rule sets interface capacity from source and target hidden widths and target KV geometry, without explicitly accounting for the directional model-size ratio, representation mismatch, or their training requirements. When the source is much larger than the target, a high source-to-target size ratio may lead to degradation under the current configuration; this ratio refers to model parameter counts, not KV compression. The Qwen3 $14\text{B} \to 4\text{B}$ results in Appendix I show that the current capacity rule does not ensure contextual-task transfer quality. Our cross-model parameterization therefore remains incomplete. Future work will investi-

gate how asymmetric source and target scales affect context reuse, and how interface capacity, depth, and training budget should be adapted jointly.

E.2 Exact counts and validation limits

$$\begin{aligned} P_W &= (4D_S + 1)w_W + 4w_W + m_W w_W \\ &\quad + D_Z(8w_W^2 + 3w_W f_W + 3w_W) \\ &\quad + D_U(4w_W^2 + 3w_W f_W + 2w_W) + w_W + 2P(w_W d_T + d_T), \\ P_M &= D_S w_M + 2D_S + 7w_M + 4w_M d_M \\ &\quad + (L_T + H_T)c_M + 2c_M d_M + 5d_M^2 + 15d_M, \\ P_R &= P, \qquad P_B = 2Pd_T(d_S + 1). \end{aligned} \tag{E.2}$$

Here P_W counts only the Compiler residual network and P_B its internal Bridge. The complete Compiler has $P_W + P_B$ parameters; total interface size is $P_M + P_R + P_W + P_B$, of which $P_M + P_R + P_W$ are optimized by gradients. Base-model weights are excluded. Unit-specific readouts already scale with target geometry and receive no additional model-size multiplier.

$$C_M = \left\lceil 10^6 \max(s^2, P/288) \right\rceil, \qquad C_W = 10^6 \left\lceil 1.25 P_W / 10^6 \right\rceil. \tag{E.3}$$

The limits C_M, C_W validate instantiated parameter counts; they introduce no parameters and are not target sizes. The limit C_W applies to the residual network; the complete Compiler's corresponding limit is $C_W + P_B$. Command-line overrides of widths, depths, or slot counts must be saved together with the resulting validation.

Table 5: Recorded geometry for six directions; D is the shared encoder/decoder depth.

Pair	D_S	(L_T, H_T)	(w_M, d_M, c_M)	(w_W, m_W, D)
Qwen3 4B→8B	2560	(36,8)	(416,112,56)	(416,28,4)
Qwen3 8B→4B	4096	(36,8)	(512,128,64)	(512,32,4)
Llama 3B→8B	3072	(32,8)	(448,112,56)	(448,28,4)
Llama 8B→3B	4096	(28,8)	(512,128,64)	(512,32,4)
Qwen2.5 7B→32B	3584	(64,8)	(480,128,64)	(480,32,4)
Qwen2.5 32B→7B	5120	(28,4)	(576,144,72)	(576,36,5)

Table 6: Exact interface parameter counts; Compiler includes the fitted, frozen Bridge.

Pair	Cartographer	Allocator	Compiler	Total
Qwen3 4B→8B	1,338,768	288	61,151,168	62,490,224
Qwen3 8B→4B	2,474,112	288	80,917,504	83,391,904
Llama 3B→8B	1,665,424	256	62,676,608	64,342,288
Llama 8B→3B	2,473,600	224	70,398,976	72,872,800
Qwen2.5 7B→32B	2,081,440	512	108,981,440	111,063,392
Qwen2.5 32B→7B	3,424,048	112	71,890,624	75,314,784

Table 7: Compiler decomposition and gradient-optimized parameters of the entire interface.

Pair	Bridge	Residual	Gradient-optimized total
Qwen3 4B→8B	9,510,912	51,640,256	52,979,312
Qwen3 8B→4B	9,510,912	71,406,592	73,880,992
Llama 3B→8B	8,454,144	54,222,464	55,888,144
Llama 8B→3B	7,397,376	63,001,600	65,475,424
Qwen2.5 7B→32B	16,908,288	92,073,152	94,155,104
Qwen2.5 32B→7B	3,698,688	68,191,936	71,616,096

Counts are derived from the actual dimensions in six recorded configurations; residual counts match instantiation preflights. All six directions use $d_S = d_T = 128$. The general rectangular-map

count does not establish execution support for every unequal-head-dimension pair. The interface is lightweight relative to billion-parameter bases; latency also depends on activations, readouts, and mounting.

F Execution Interface, Complexity, and Reuse

F.1 Token and position interfaces

We verify token-to-ID semantics within each Qwen3, Llama, and Qwen2.5 pair and use the target tokenizer consistently. Llama tokenizer JSON files need not have identical file hashes, but their actual vocabulary mappings must agree. In-range IDs alone do not establish semantic compatibility. Context, query, and generation boundaries are matched across training, native Full, and mounted execution.

We retain native chat templates where available. Llama Base contexts contain a single BOS, context text, and two newlines. Contextual QA uses completion-v2, appending the continuation cue "Answer: The answer is" to the query, or only "The answer is" after an existing "Answer:". General choice and language-modeling tasks retain plain-v1 and stored token sequences. This dataset-specific protocol is matched between the method and Full.

Mounting native target KV at every position controls interface consistency. We test a representative example per dataset plus the longest general-task example, comparing up to four generated tokens and EOS behavior; candidate-token NLL differences must not exceed 0.05 in absolute value. This checks cache, position, and attention implementation, without requiring compiled KV to equal native KV.

F.2 Effective state size and computation

$$\mathrm{Elements}(\widehat{\mathcal{C}}) = 2d_T \sum_p |\mathcal{I}_p| = 2Pkd_T, \qquad \frac{\mathrm{Elements}(\widehat{\mathcal{C}})}{2PNd_T} = \frac{k}{N}. \tag{F.1}$$

For $N = 1024$, $\rho = 0.25$, $\tau = 32$, and $P = 288$, there are 73,728 effective slots versus 294,912 in the full cache, with the tail included. For union size $U = |\bigcup_p \mathcal{I}_p|$, we have $\max_p |\mathcal{I}_p| \le U \le \min(N, Pk)$. Retaining one quarter per head does not imply a union covering only one quarter of the context.

Cartographer scores have size $O(PN)$; fixed query slots do not construct an $N \times N$ target attention matrix. Let $S_{\max} = \max_p |\mathcal{I}_p|$. The main residual-network cost is

$$\begin{aligned}\mathcal{O}\big(&ND_S w_W + D_Z[(N+m_W)w_W^2 + m_W N w_W + m_W^2 w_W] \\ &+ D_U[(U+m_W)w_W^2 + U m_W w_W] + PS_{\max} w_W d_T\big).\end{aligned} \tag{F.2}$$

The shared context and union trunk carry no additional factor P, while per-unit readouts depend on target geometry. Current readouts pad heads to a common width, hence $PS_{\max}$. Bridge affine maps add their own per-slot cost.

Latent slots read the complete provided context and can aggregate context positions later than a given slot. Those positions are available for queries after the context, as evaluated here; this does not establish suitability for online causal prediction within the context itself.

F.3 Complete and source-state reuse paths

$$\begin{aligned}T_{\mathrm{ACCT}}^{\mathrm{prefill}} &= T_S([c;q]) + T_{\mathrm{transfer}}(c,q) + T_{T,q}(q;\widehat{\mathcal{C}}), \\ T_{\mathrm{ACCT}}^{\mathrm{incremental}} &= T_{S,q}(q;\mathcal{R}_S(c)) + T_{\mathrm{transfer}}(c,q) + T_{T,q}(q;\widehat{\mathcal{C}}).\end{aligned} \tag{F.3}$$

The first line is the main paper's complete-cost decomposition. Transfer includes the three components, sorting, mounting, and required data movement. The second line assumes that $\mathcal{R}_S(c) = (H_c, H_S, K_S, V_S)$ already exists and still includes source query computation.

When the query changes, causal source context states can be reused if weights, tokens, and positions are unchanged, but ranking, selection, and compilation must be repeated. Allocator counts remain fixed at the same length and budget while content can change.

G Data and Evaluation Metrics

Table 8: The fixed 10,000-example evaluation panel.

Group	Dataset	Examples	Metric
General	HellaSwag (Zellers et al., 2019)	1,000	Accuracy
General	Winogrande (Sakaguchi et al., 2020)	1,000	Accuracy
General	MMLU (Hendrycks et al., 2021)	1,000	Accuracy
General	PIQA (Bisk et al., 2020)	1,000	Accuracy
General	WikiText2 (Merity et al., 2017)	500	PPL
Context	SearchQA (Dunn et al., 2017)	1,000	Token F1
Context	TriviaQA (Joshi et al., 2017)	1,000	Token F1
Context	BoolQ (Clark et al., 2019)	1,000	Accuracy
Context	RACE (Lai et al., 2017)	1,000	Accuracy
Context	WikiHop (Welbl et al., 2018)	1,000	Accuracy
Context	PubMedQA (Jin et al., 2019)	500	Accuracy

SearchQA and TriviaQA use token F1; all other tasks except WikiText2 use accuracy. WikiText2 PPL is the exponential of mean NLL over evaluated tokens, not a mean of example-level perplexities. General multiple-choice tasks use candidate-sequence NLL, preserving exact token IDs, options, and budgets in stored rows. Contextual tasks primarily score generated answers.

Generation-based evaluations use greedy decoding with a 32,768-token input limit, 32 reserved tokens, and at most 32 generated tokens, following shared EOS handling and answer normalization. All six directions use the same fixed examples. GSM8K is excluded.

Target Full results are reused only when target-weight identity, actual input tokens, positions, precision, prompts, candidates, decoding budgets, and scoring code match. A source-model change alone does not require rerunning an identical Full contract, while method outputs and interface training remain specific to each run.

H Additional Analysis of Recorded Results

The following table is computed from the updated main-table entries. Retention is ACCT/Full for higher-is-better metrics and Full/ACCT for PPL. We average equally across five general benchmarks and separately across six contextual tasks. These are descriptive summaries, not a claim that heterogeneous metrics form a single statistical quantity or establish significance.

Table 9: Descriptive retention and task wins derived from the recorded main-table entries.

Pair	General retention	Context retention	Wins over Full
Qwen3 4B→8B	101.7%	95.9%	4/11
Qwen3 8B→4B	94.7%	106.6%	4/11
Llama 3B→8B	64.7%	77.5%	2/11
Llama 8B→3B	67.2%	74.7%	2/11
Qwen2.5 7B→32B	66.8%	77.5%	1/11
Qwen2.5 32B→7B	75.7%	63.4%	0/11

Both Qwen3 directions improve open-domain QA, while Llama and Qwen2.5 have substantial gaps on some language-modeling, reading-comprehension, and multi-hop tasks. The results support directed-interface execution and task-specific transfer gains, not a uniform retention rate over all pairs and tasks. In particular, original-scale PPL remains necessary so inverse-ratio summaries do not hide large losses.

In the compression study, increasing capacity from $4\times$ to $2\times$ improves all six contextual tasks in the reverse direction, while full retention is not uniformly best. The $1\times$ configuration also changes whether selection is active, so it compares complete budget configurations.

I Per-Dataset Results for Additional Model Pairs

Table 10 supplements the main table with seven Qwen3 directions: $0.6\text{B}\rightarrow 4\text{B}$, $0.6\text{B}\rightarrow 8\text{B}$, $0.6\text{B}\rightarrow 14\text{B}$, $4\text{B}\leftrightarrow 14\text{B}$, and $8\text{B}\leftrightarrow 14\text{B}$. These runs use the mainline configuration of $4\times$ compression and 500 Compiler training examples for four epochs, with the same 10,000 evaluation examples and main-table metrics. All entries come from completed evaluations with zero execution failures; Target Full is the native target reference for the corresponding run.

Table 10: Additional Qwen3 results on 11 datasets (F1/accuracy in %; PPL ↓), with bold marking ACCT improvements over the corresponding Target Full.

		General capability					Context-dependent tasks					
Model pair	Method	Hella Swag Acc.	Wino grande Acc.	MMLU Acc.	PIQA Acc.	Wiki Text2 PPL ↓	Search QA F1	Trivia QA F1	BoolQ Acc.	RACE Acc.	WikiHop Acc.	PubMed QA Acc.
Qwen3	Target Full	37.10	51.00	63.60	69.20	81.29	38.56	37.68	84.00	87.10	45.80	34.00
$0.6\text{B}\rightarrow 4\text{B}$	ACCT	34.60	49.70	60.70	65.40	**46.66**	**44.29**	**45.63**	68.10	53.20	32.60	**49.00**
Qwen3	Target Full	39.70	58.50	59.70	73.20	32.91	46.65	50.48	86.30	87.60	49.60	48.80
$0.6\text{B}\rightarrow 8\text{B}$	ACCT	33.50	50.40	59.00	65.30	38.10	40.29	41.70	62.80	53.90	28.20	43.40
Qwen3	Target Full	36.60	59.20	72.30	73.90	19.74	65.45	52.83	88.60	91.10	49.10	47.40
$0.6\text{B}\rightarrow 14\text{B}$	ACCT	36.30	52.00	68.20	69.80	21.37	57.43	**56.44**	64.00	60.20	36.50	19.00
Qwen3	Target Full	36.60	59.20	72.30	73.90	19.74	65.45	52.83	88.60	91.10	49.10	47.40
$4\text{B}\rightarrow 14\text{B}$	ACCT	**36.90**	49.30	52.40	68.20	24.02	53.91	**57.13**	67.10	59.10	35.10	39.80
Qwen3	Target Full	37.10	51.00	63.60	69.20	81.29	38.56	37.68	84.00	87.10	45.80	34.00
$14\text{B}\rightarrow 4\text{B}$	ACCT	25.60	50.40	25.50	55.40	98.38	0.31	0.66	0.60	13.80	0.60	0.60
Qwen3	Target Full	36.60	59.20	72.30	73.90	19.74	65.45	52.83	88.60	91.10	49.10	47.40
$8\text{B}\rightarrow 14\text{B}$	ACCT	**37.40**	49.50	69.00	70.40	**19.03**	58.93	**61.59**	69.10	59.00	37.40	39.80
Qwen3	Target Full	39.70	58.50	59.70	73.20	32.91	46.65	50.48	86.30	87.60	49.60	48.80
$14\text{B}\rightarrow 8\text{B}$	ACCT	30.60	50.90	55.80	65.10	53.96	20.20	26.94	48.90	49.50	26.90	**54.00**

Performance across directions. The $0.6\text{B}\rightarrow 4\text{B}$ configuration exceeds Target Full on both open-domain QA tasks, and all three additional directions targeting 14B improve TriviaQA. Qwen3 $14\text{B}\rightarrow 4\text{B}$ remains below 1% accuracy on several contextual tasks. Its contextual-task degradation should therefore be considered separately from language modeling, and transfer quality remains dependent on the pair and task.

J Component Ablation Protocols

All controls use the formal Qwen3 $4\text{B}\leftrightarrow 8\text{B}$ configurations with a $4\times$ budget and the mandatory 32-token tail. Random deterministically generates independent prefix rankings for each layer/head from a fixed seed and the exact prompt tokens; the same rule is used in training and evaluation, without reference answers. We retrain the Allocator on the original 5,000 SearchQA examples, then refit the Bridge and train the Compiler. Uniform retains the original Cartographer, assigns identical counts to all layer/head units, and refits the Bridge and retrains the Compiler. Both use the original 500-example subset, four epochs, 500 updates, and direction-specific Compiler dimensions.

Bridge-only and Native KV fix the original Cartographer, Allocator, and retained layer/head positions without retraining. Bridge-only retains the fitted affine Bridge and removes only the neural residual. Native KV gathers target-native context KV at identical positions, preserving existing key

RoPE without a second rotation; it still requires target context prefill. These controls assess residual correction and state quality at fixed selection, rather than retraining after removing the entire Compiler.

All eight runs complete 10,000 examples with zero execution failures or duplicate samples. Across batches, we verify sample identities, exact input tokens, context/query boundaries, total retained slots, and reused per-example ACCT/Full baselines; the two Compiler controls also match exact selected positions. Each method uses a fresh target continuation cache. F1 and accuracy are averaged over examples; WikiText2 PPL is the exponential of total token NLL divided by total scored tokens.

K Inference Latency Protocol and Results by Context Length

Hardware and concurrency. We launch six independent workers on a machine with eight NVIDIA A800 80GB GPUs, with one process per transfer direction. The four Qwen3 $4\text{B} \leftrightarrow 8\text{B}$ and Llama $3\text{B} \leftrightarrow 8\text{B}$ directions each occupy one of GPUs 0–3, placing the source, target, and three components on the same device. Each Qwen2.5 $7\text{B} \leftrightarrow 32\text{B}$ direction uses two GPUs, in groups 4–5 and 6–7. Within each process, the source and components reside on its first visible GPU and the target on its second. This is concurrency across directions; individual directions use neither tensor nor data parallelism. We use PyTorch (Paszke et al., 2019) with BF16 and SDPA, disable `torch.compile` (`SLOT_COMPILE=0`), and set OMP, MKL, and OpenBLAS CPU thread counts to 4.

Samples and timing. Each fixed sample runs independently with batch size 1 and input shape $[1, L]$. Measurements use $\rho = 0.25$ and a 32-token tail. We measure eight context samples per direction; their actual lengths are listed below. Source prefill, Cartographer, Allocator, Compiler, and target prefill are each warmed up and then timed according to Table 11, reporting the median. Before timing, `cuda.synchronize` synchronizes all GPUs visible to the process; `time.perf_counter()` records wall-clock time. Repetition counts apply to each stage of each sample.

Table 11: Warm-up and timed repetitions for each stage.

Context length L	Warm-up runs	Timed runs
$L < 4096$	2	3
$4096 \leq L < 8192$	1	3
$L \geq 8192$	1	2

Comparison scope and results. KV state-conversion latency is the sum of synchronized wall-clock medians for Cartographer, Allocator, and Compiler. Speedup is target context prefill latency divided by this sum. This component metric assumes available source representations and excludes source prefill, packing, positional rotation, query processing, and generation. Source prefill is tabulated separately; Section 3 and Appendix F.3 give the complete prefill cost decomposition. Tables 12–14 report stage times and three-component totals in ms; Cart., Alloc., and Comp. denote the Cartographer, Allocator, and Compiler. Figure 2 uses all 48 measurements. Tabulated times and speedups are rounded to one and two decimal places, respectively.

Table 12: Qwen3 stage latencies (ms) and state-conversion speedups by context length.

Direction	L	Cart.	Alloc.	Comp.	Convert	Source	Target	Speedup
4B → 8B	1001	7.0	7.4	12.8	27.2	69.4	91.1	3.35×
	1024	7.0	7.4	12.9	27.3	69.4	91.9	3.37×
	1729	7.0	7.9	17.0	31.9	106.1	159.0	4.98×
	1918	7.1	7.4	18.1	32.6	131.6	165.3	5.07×
	4126	15.6	7.5	30.6	53.7	270.2	363.9	6.78×
	8158	20.3	7.8	53.3	81.4	533.7	729.8	8.97×
	16389	43.5	7.8	105.0	156.3	1242.7	1658.2	10.61×
	19070	48.0	7.7	124.6	180.3	1496.7	1993.4	11.06×
8B → 4B	1001	7.0	7.4	13.0	27.4	102.5	69.3	2.53×
	1024	6.9	7.3	12.9	27.1	103.2	57.5	2.12×
	1729	7.0	7.3	17.2	31.5	170.9	98.4	3.12×
	1918	7.0	7.3	18.3	32.6	200.8	107.6	3.30×
	4126	15.5	7.4	31.3	54.2	412.8	242.6	4.48×
	8158	20.4	7.4	55.9	83.7	791.0	492.6	5.89×
	16389	43.0	8.6	110.5	162.1	1765.4	1188.5	7.33×
	19070	46.4	8.2	128.1	182.7	2127.7	1456.3	7.97×

Table 13: Llama stage latencies (ms) and state-conversion speedups by context length.

Direction	L	Source	Cart.	Alloc.	Comp.	Convert	Target	Speedup
3B → 8B	995	42.6	6.2	7.4	12.3	25.9	86.4	3.34×
	996	43.8	6.2	7.4	12.3	25.9	86.4	3.34×
	1551	64.5	6.2	7.4	15.3	28.9	123.8	4.28×
	1900	88.2	6.2	7.4	17.1	30.7	153.1	4.99×
	3989	176.3	9.9	7.4	28.3	45.6	319.9	7.02×
	7848	339.0	17.6	7.4	49.4	74.4	656.8	8.83×
	15947	792.9	33.4	7.7	98.3	139.4	1477.9	10.60×
	19047	995.8	40.8	7.7	117.1	165.6	1837.5	11.10×
8B → 3B	995	90.7	7.5	7.3	11.3	26.1	39.8	1.52×
	996	93.8	5.5	7.5	11.1	24.1	45.7	1.90×
	1551	129.4	5.6	7.0	13.5	26.1	60.7	2.33×
	1900	178.0	5.6	7.0	15.1	27.7	73.3	2.65×
	3989	351.8	8.8	6.9	24.3	40.0	156.4	3.91×
	7848	681.6	15.7	6.9	42.4	65.0	318.2	4.90×
	15947	1526.1	29.1	7.2	84.2	120.5	742.1	6.16×
	19047	1905.4	36.0	7.4	100.4	143.8	937.4	6.52×

Table 14: Qwen2.5 stage latencies (ms) and state-conversion speedups by context length.

Direction	L	Source	Cart.	Alloc.	Comp.	Convert	Target	Speedup
7B → 32B	1001	85.7	10.7	8.0	17.2	35.9	308.5	8.59×
	1024	86.4	10.8	8.3	17.3	36.4	310.9	8.54×
	1729	127.3	11.0	8.2	24.4	43.6	552.9	12.68×
	1918	162.2	10.9	8.3	26.2	45.4	584.1	12.87×
	4126	323.6	25.9	8.3	48.6	82.8	1324.1	15.99×
	8158	625.1	34.0	8.4	89.1	131.5	2592.0	19.71×
	16389	1348.4	73.8	8.9	193.1	275.8	5653.4	20.50×
	19070	1607.3	83.3	9.0	223.7	316.0	6706.3	21.22×
32B → 7B	1001	355.4	3.9	7.1	10.7	21.7	76.6	3.53×
	1024	357.6	3.9	7.4	10.7	22.0	76.3	3.47×
	1729	571.1	4.0	7.1	12.5	23.6	123.4	5.23×
	1918	706.7	4.1	7.4	12.9	24.4	131.6	5.39×
	4126	1413.2	7.6	7.2	18.2	33.0	303.4	9.19×
	8158	2689.8	12.8	7.4	29.9	50.1	595.7	11.89×
	16389	5664.2	24.7	7.5	53.7	85.9	1322.7	15.40×
	19070	6732.5	28.2	7.5	61.6	97.3	1580.7	16.25×

Choosing a source when states are unavailable. When no source states are available and the goal is to construct target KV states from raw context, source encoding must also enter the latency budget. A speedup requires source prefill plus state transfer to be faster than direct target prefill. A smaller source leaves more room for transfer overhead and is therefore the preferred starting point for this use case. Table 15 sums the measured stage medians for source prefill, the three components, packing, and positional rotation; subsequent target query encoding and generation are outside this state-construction metric.

At approximately 19K tokens, Qwen3 4B → 8B, Llama 3B → 8B, and Qwen2.5 7B → 32B achieve 1.16×, 1.53×, and 3.35× speedups after accounting for source encoding, reducing latency by 13.9%, 34.8%, and 70.1%, respectively. All three directions remain faster on every measured context from approximately 4K to 19K tokens. The benefit depends on context length: Qwen3 4B → 8B is slower on the shortest sample. In all measured reverse-direction cases, source prefill alone exceeds target prefill, so newly encoding with a larger source offers no latency benefit. These results favor smaller-to-larger transfer for constructing target states from scratch; transferring already available states remains useful in both directions, as evaluated above.

Table 15: Target-state construction latency (ms), including source encoding, on the shortest and longest contexts.

Direction	L	Source	Transfer	Total	Target	Speedup
Qwen3 4B → 8B	1001	69.4	36.4	105.8	91.1	0.86×
	19070	1496.7	219.6	1716.3	1993.4	1.16×
Llama 3B → 8B	995	42.6	33.6	76.2	86.4	1.13×
	19047	995.8	202.9	1198.7	1837.5	1.53×
Qwen2.5 7B → 32B	1001	85.7	52.3	138.0	308.5	2.24×
	19070	1607.3	397.4	2004.7	6706.3	3.35×
Qwen3 8B → 4B	1001	102.5	36.5	139.0	69.3	0.50×
	19070	2127.7	222.2	2349.9	1456.3	0.62×
Llama 8B → 3B	995	90.7	35.4	126.1	39.8	0.32×
	19047	1905.4	173.9	2079.3	937.4	0.45×
Qwen2.5 32B → 7B	1001	355.4	28.8	384.2	76.6	0.20×
	19070	6732.5	114.7	6847.2	1580.7	0.23×

Transfer includes Cartographer, Allocator, Compiler, packing, and positional rotation. Total adds source prefill; speedup is Target/Total. Totals are computed from the reported stage medians before final display rounding.