CACHET: How to Make KV-Cache Sharing Carry What the Larger Model Read

Anonymous authors
Paper under double-blind review

Abstract

Collaborative inference lets a small language model use the understanding of a larger one. Rather than passing that understanding through lossy text, recent methods share the larger model's KV cache, reshaped by a learned translator for the small model. When both models read the same prompt, accuracy rises, as if the small model had received what the larger one read. Yet the state-of-the-art C2C fuser still retains half of its gain when handed the cache of an unrelated question, and its updates track the question rather than the cache. The gain is largely an effect of added parameters adapting the small model's own cache, which the fuser also reads. The collaboration is then capped at what an adapter can add, however much the larger model understood. We propose CACHET, a translator designed so that the small model's answer depends on what the larger model read. It reads only the larger model's cache, closing that shortcut, and is trained to reproduce, jointly across all heads, keys, and values, the cache the small model would have built from the same input. Because most of the key cache is common to all inputs, this training emphasizes the part that differs between inputs. The loss is kept throughout answer training, which alone would pull the transmitted cache off target. When only the larger model sees the passage that determines the answer, a 0.6B model assisted by a 4B one reaches 97.3% without training on this test and drops to 1.0% when given the cache of a passage supporting a different answer. On four benchmarks, CACHET recovers 87.5% of the accuracy gap to the larger model, against 36.0% for the C2C fuser.

1 Introduction

Collaborative inference combines the efficiency of a small model (i.e., the receiver) with the contextual understanding of a stronger model (i.e., the sharer). While models typically communicate through text (She et al., 2025; Kim et al., 2025), this format may not fully convey the semantic information encoded in their internal representations (Fu et al., 2026). Cache translation instead transmits internal representations directly: learned mappings transform the sharer's key–value (KV) cache into representations that the receiver can incorporate into its attention computation (Fu et al., 2026; Rossi et al., 2026; Dery et al., 2026; Chen et al., 2026). The central question is: *How can we make KV-cache sharing carry what the larger model read?*

Prior work typically learns cache translation through end-to-end task supervision. C2C (Fu et al., 2026), for example, combines both models' caches for the same question using a fuser trained solely with answer loss. However, because the fuser also processes the receiver's own cache, it can improve answers by refining the receiver's internal representations without relying on information from the sharer's current cache. In this same-input setting, accuracy gains alone cannot separate this receiver-side adaptation from input-dependent transfer (Chen et al., 2026).

To test dependence on the sharer's input, we compare conditions for the sharer's cache while keeping the receiver's question fixed. Here, Matched uses the sharer's cache for that question (Figure 1(a)), whereas Swap replaces it with the cache of an unrelated question (Figure 1(b)). Despite this replacement, a released C2C fuser retains substantial accuracy gains over the receiver-only baseline. Much of the gain remains even when the sharer's cache is replaced with a fixed training-set average (Figure 1(c)); the fuser's only input-dependent cache is then the receiver's current-question cache. Thus, a substantial fraction of the fuser's improvement does not depend on the sharer's current input.

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.

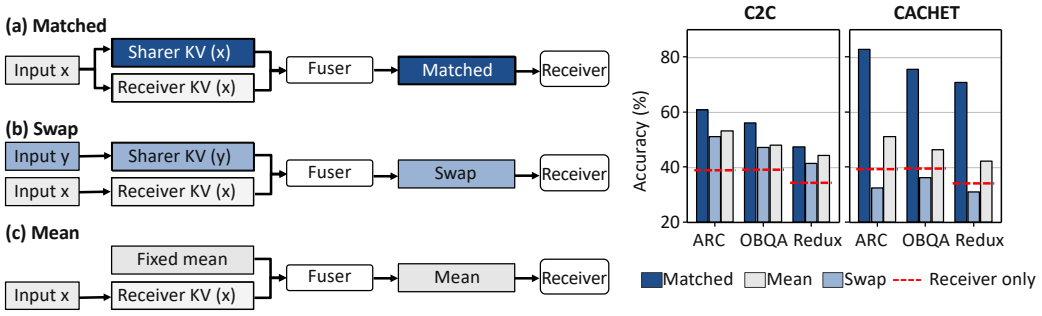


Figure 1: **Accuracy under cache replacement.** Left: C2C controls; right: Qwen3-4B to Qwen3-0.6B accuracy (%). Matched uses the current question's sharer cache; Swap uses an unrelated question's cache with receiver input fixed. Mean replaces C2C sharer caches or CACHET translated prefixes with their training-set average. Dashed: receiver-only accuracy (C2C's no-fusion path). C2C uses its public checkpoint; CACHET uses width 4096, seed 0. Controls: Appendix C.

These persistent gains motivate a closer look at the geometry of the fuser's cache updates as the sharer's input changes. For the same receiver question, the fuser's update to the receiver cache remains similar when the sharer's cache is replaced with one from an unrelated question.

Answer loss does not specify which details to retain from the sharer's input. The receiver's own caches for the same input (receiver-native caches) provide a representational reference (Chen et al., 2026). We therefore examine their structure and find that shared structure accounts for most of the key energy but much less of the value energy, leaving input-specific variation at different relative scales in the key and value caches. Relative to the respective residual energies of keys and values, our plain alignment baseline (block-normalized) places less weight on key-residual errors than on value-residual errors. The translator's architecture also determines which source information is available for reconstruction. A recent approach translates each head independently, and keys separately from values (Chen et al., 2026), so each receiver output can use only one source head's keys or values.

Building on these observations, we introduce CACHET, a source-only cache translator that produces a receiver prefix from the sharer's cache alone. To let each receiver output draw on information from any source head, the translator jointly maps the sharer's keys and values. We align the translated cache with the receiver's representational structure, using the receiver's cache for the same input as the target. To preserve input-specific variation, our K/V-aware mean–residual (MR) objective separates mean and residual errors, upweights residuals, and normalizes them separately for keys and values. We first train the translator with this objective and retain it during task training, since answer loss alone can rapidly move the translated cache away from its target.

A counterfactual passage task tests whether CACHET transfers information unavailable to the receiver. We reuse checkpoints trained with both models observing the same input, without training on this task. Only the sharer reads the answer-determining passage; the receiver's question and options are identical within each pair. With a 4B sharer and a 0.6B receiver, CACHET averages 97.3% accuracy across seven training seeds. Replacing the cache with one from a passage supporting a different answer reduces this to 1.0%.

Starting from the same alignment-pretrained checkpoints, removing alignment during task training lowers four-benchmark recovery from 87.8% to 29.5% across three seeds (Table 4). Across four benchmarks and seven seeds, CACHET recovers 87.5% of the solo accuracy gap between the 0.6B receiver and 4B sharer, compared with 73.0% for DLC-SI. Our contributions are:

- To distinguish receiver adaptation from input-dependent transfer, we combine cache-replacement tests with geometric analysis of cache updates.
- CACHET couples a K/V-aware mean–residual objective with a source-only translator that jointly processes keys and values from all source heads. The objective emphasizes input-specific variation in receiver-native targets and remains active during task training.
- Counterfactual passage tests demonstrate transfer of information unavailable to the receiver without training on these tasks. On four standard benchmarks, CACHET also recovers 87.5% of the standalone-model accuracy gap.

2 Background and Motivation

Why can accuracy improve without input-dependent transfer, and what should cache translation preserve? We examine C2C (Fu et al., 2026) through cache-replacement tests and update geometry, then analyze receiver-native cache structure to motivate the alignment objective and translator design.

2.1 Same-Input Cache Transfer

Same-input cache transfer aims to improve the receiver's answer using the sharer's interpretation of their shared input (Fu et al., 2026). A sharer M_s and a receiver M_r read the same question x_i, with target answer y_i and prefill caches $\mathbf{S}_i$ and $\mathbf{R}_i$, respectively.

C2C transmits the sharer's cache without generating answer tokens. A learned fuser reads both caches and adds a residual update U_θ to the receiver's cache at aligned question positions (Fu et al., 2026). The receiver answers from this fused cache. With both models frozen, only the fuser is trained to make the correct answer more likely:

$$\widetilde{\mathbf{R}}_i = F_\theta(\mathbf{R}_i, \mathbf{S}_i) = \mathbf{R}_i + U_\theta(\mathbf{R}_i, \mathbf{S}_i), \qquad \mathcal{L}_{\text{ans}}(\theta) = -\sum_i \log p_{M_r}(y_i \mid \widetilde{\mathbf{R}}_i). \tag{1}$$

However, the cache-replacement results above suggest that a substantial part of the accuracy gain comes from information already present in the receiver's cache. We next examine the geometry of cache updates to investigate this possibility.

2.2 Gains Survive Broken Correspondence

We test whether C2C's gains and cache updates depend on the sharer's current input. We replace the sharer's cache while keeping the receiver question and execution path fixed. Matched uses the corresponding cache $\mathbf{S}_i$, whereas Swap uses another question's cache $\mathbf{S}_{\pi(i)}$ under a fixed-point-free permutation computed independently within each benchmark's evaluation set. Mean uses a fixed training-set average of source caches. The receiver cache $\mathbf{R}_i$ is unchanged in all three conditions.

Gains are measured against a baseline that uses the same sectioned prefill with fusion disabled. Matched remains best on all three audited benchmarks. Appendix C specifies the mean construction and reports the underlying accuracies in Table B.4.

We now examine how the fuser's learned updates depend on the sharer's input and the receiver's question. Let $\mathbf{R}_i$ and $\mathbf{S}_j$ denote the caches of receiver question i and sharer input j, respectively. The fused cache $F_\theta(\mathbf{R}_i, \mathbf{S}_j)$, produced by the fuser with learned parameters θ, includes the receiver's original states. We subtract these states to isolate the residual update U_θ from Equation 1, denoting it by Δ_{ij}:

$$\Delta_{ij} = F_\theta(\mathbf{R}_i, \mathbf{S}_j) - \mathbf{R}_i = U_\theta(\mathbf{R}_i, \mathbf{S}_j). \tag{2}$$

Using the released 4B-to-0.6B fuser, we supply the same 16 source caches to each of 16 receiver questions, four per benchmark. This probe crosses receiver questions with source inputs from all four benchmarks, allowing us to vary the sharer's input while holding the receiver question fixed.

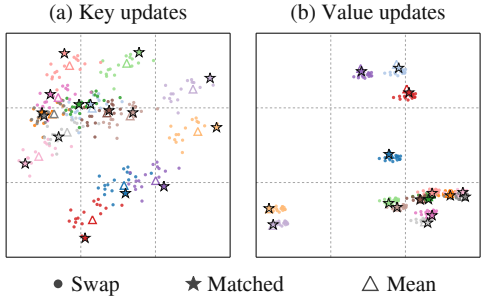


Figure 2: C2C updates for 16 receiver questions crossed with 16 source caches (4B→0.6B; Equation 2). Colors denote receiver questions; dots: Swap; stars: Matched; open triangles: Mean. PCA projections group by receiver, more clearly for values. Appendix C.4 details coordinates and explained variance.

Figure 2 shows clearer receiver-question groups for values than for keys. For each receiver question, we average its updates over the 16 source inputs. The mean relative distance between these receiver-specific centers is 0.434 for keys and 0.948 for values, consistent with this K/V contrast. Appendix C.4 reports the Swap-to-Matched update distances and defines both measurements.

Receiver-specific update structure and much of the accuracy benefit therefore persist under source replacement. We refer to improvement that does not require the sharer's current input as *receiver adaptation*. Answer loss rewards improvements in the receiver's prediction, but does not specify which details the transmitted cache should retain. A receiver-native cache for the same input supplies a representational reference, a form of supervision also used by DLC (Chen et al., 2026). We next examine the structure of this reference.

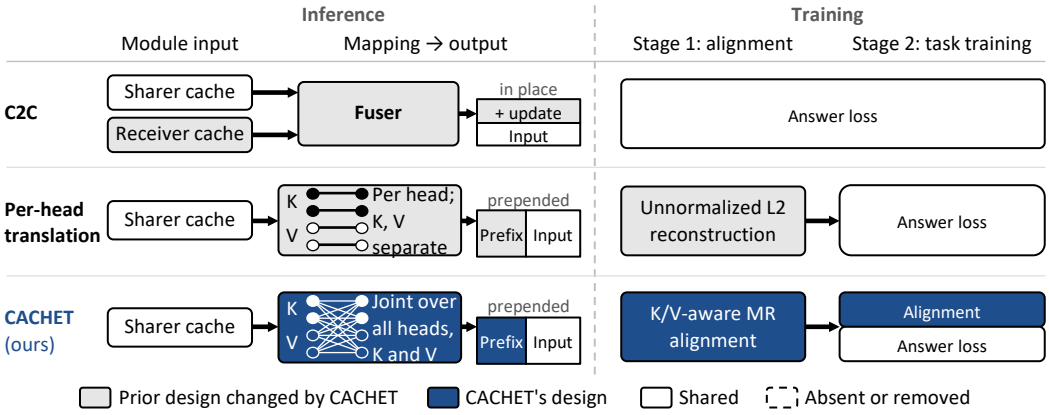


Figure 3: **CACHET against prior cache-transfer designs.** Rows: receiver-conditioned fusion (C2C), per-head translation (as in DLC; Chen et al., 2026), and CACHET. Left: C2C's fuser also reads the receiver's cache and updates it in place; both translators read only the sharer's cache and prepend a prefix at separate positions, mapping each source head's keys or values separately or all heads, keys, and values jointly. Right: Stage 1 aligns the prefix to the receiver-native cache of the same input, as unnormalized L2 error or with the K/V-aware mean–residual (MR) objective; Stage 2 adds answer loss, and only CACHET keeps the alignment term. C2C trains with answer loss alone.

2.3 Shared and Input-Specific Cache Structure

Shared structure accounts for a larger fraction of energy in receiver-native keys than in values. We quantify this difference on 2,000 training inputs for the 4B-to-0.6B pair. Within each batch, we compute the mean across valid examples at each cache coordinate and subtract it from each example to obtain a residual. We measure the mean's share of cache energy separately for each layer and K/V branch, then average the fractions over layers and batches (Appendix C.4).

At the training batch size of 16, the mean accounts for 79.7% of key energy and 25.7% of value energy. A batch-constant prediction equal to this target mean would leave only the residual component unreconstructed. Keys are thus dominated by structure common to the batch, whereas most value energy lies in variation across examples. These fractions describe the native targets at the training batch size.

This difference matters for objectives that normalize reconstruction error by total cache-block energy, such as the plain alignment of Section 3.2. For a given squared error relative to a branch's residual energy, the block-normalized penalty is scaled by that residual's fraction of total energy. The smaller key residual fraction therefore receives less emphasis. This motivates normalizing key and value residual errors by their own native scales, as developed in Section 3.2.

Preserving input-specific variation calls for an alignment objective that emphasizes differences between inputs. The mapping must also have access to the source information needed to reconstruct those differences. Restricting each output to one source head's keys or values assumes that the required information is available within that subset. The alignment objective and translator structure therefore need to be considered together.

3 CACHET: Alignment-Preserving Cache Translation

Better answers alone do not establish that a cache preserves what the sharer read. CACHET therefore anchors each translated cache to the receiver's own representation of the same input. Its translator reads only the sharer's cache and jointly processes keys and values from all source heads. The alignment objective emphasizes differences across inputs and remains active during answer training, keeping task optimization anchored to the information the cache should carry. Figure 3 contrasts these choices with receiver-conditioned fusion and per-head translation.

3.1 Source-Only Translation to Receiver-Native Targets

To construct a prefix from the sharer's cache alone, the sole trainable module g_θ maps $\mathbf{S}_i$ to a receiver-shaped cache block

$$\widehat{\mathbf{T}}_i = g_\theta(\mathbf{S}_i). \tag{3}$$

Restricting the translator's input to $\mathbf{S}_i$ makes its dependence on the current input pass through the transmitted cache. An interface with access to receiver-side states could instead satisfy alignment using the receiver's own representation of x_i. The receiver still reads its own input, but its current-input cache is outside the translator's inputs.

The translator selects one sharer layer for each receiver layer and applies a shared token-wise LayerNorm–MLP with layer-specific K/V output projections, without mixing tokens or layers. For each token, this shared MLP takes the keys and values of all eight source KV heads as a single concatenated input. Each receiver key or value can therefore draw on both keys and values from any source head (Section 2.3). Its input uses sharer keys before rotary position embedding (RoPE; Su et al., 2021) and sharer values; its output contains receiver keys in post-RoPE coordinates and receiver values. It receives no explicit position input and applies no rotary transform to its outputs. Task-only, CACHET w/o MR, and CACHET share this architecture; Appendix A gives the dimensions, layer mapping, and position conventions.

The translation target is the cache that the frozen receiver would produce from the same payload. We obtain this target in a separate pass over the raw payload tokens x_i, without chat markers or answer tokens, at prefix positions $\mathcal{P}_i$:

$$\mathbf{T}_i = \mathcal{C}_r(x_i; \mathcal{P}_i), \tag{4}$$

where $\mathcal{C}_r$ denotes receiver cache construction, yielding post-RoPE keys and unrotated values. Our model pairs share a tokenizer, so source, prediction, and target correspond to the same payload tokens; different tokenizers would require an additional token-alignment rule. Prediction and target also use the same receiver layers, prefix position IDs, and key/value coordinates. Padding and positions outside the transferred block are masked.

The translated block occupies a dedicated prefix, with one cache entry per transferred token. The receiver reads its own input at separate positions $\mathcal{Q}_i$ before generating an answer. The native target reaches the translator only through the training loss. At inference, the target-producing pass is removed: the sharer prefills its input, the translator constructs the prefix, and the receiver generates from the prefix and its own input. The sharer's prefill cache is the only cross-model payload, and translation is performed independently for each example.

3.2 K/V-Aware Mean–Residual Alignment

To emphasize input-specific variation, we separate prediction and target into batch means and residuals. Let c index a scalar cache coordinate, including layer, token position, key/value branch, head, and within-head channel, and let $\mathcal{V}_c$ contain the batch examples for which it is valid. We define

$$\mu_c = \frac{1}{|\mathcal{V}_c|}\sum_{i\in\mathcal{V}_c} T_{i,c}, \quad \widehat{\mu}_c = \frac{1}{|\mathcal{V}_c|}\sum_{i\in\mathcal{V}_c} \widehat{T}_{i,c}, \quad E_{i,c} = T_{i,c} - \mu_c, \quad \widehat{E}_{i,c} = \widehat{T}_{i,c} - \widehat{\mu}_c. \tag{5}$$

For each valid coordinate, squared error decomposes exactly as

$$\frac{1}{|\mathcal{V}_c|}\sum_{i\in\mathcal{V}_c} (\widehat{T}_{i,c} - T_{i,c})^2 = (\widehat{\mu}_c - \mu_c)^2 + \frac{1}{|\mathcal{V}_c|}\sum_{i\in\mathcal{V}_c} (\widehat{E}_{i,c} - E_{i,c})^2. \tag{6}$$

The mean term aligns the shared center; the residual term aligns differences across inputs. Since centering preserves pairwise displacements, $E_{i,c} - E_{j,c} = T_{i,c} - T_{j,c}$, residual matching supervises the relative arrangement of inputs' caches (Appendix C.1). The mean term alone can be minimized by assigning the target mean to every example in a batch, whereas matching nonzero residuals requires variation across examples.

Plain alignment: CACHET w/o MR. For each layer and K/V branch, mean and residual errors share a target-energy denominator. The mean error is weighted by valid-example counts to preserve the decomposition. Averaging over layers gives $E_{\mu,u}$ and $E_{r,u}$ for branch $u \in \{K, V\}$:

$$\mathcal{L}_{\text{align}}^{\text{w/o MR}} = \tfrac{1}{2}\sum_{u\in\{K,V\}} (E_{\mu,u} + E_{r,u}). \tag{7}$$

This objective already supervises differences across inputs. Its block-energy denominator, however, does not separately calibrate the mean and residual scales identified in Section 2.3.

K/V-aware residual weighting. MR normalizes each branch's residual error by its native residual scale and adjusts its weight relative to the mean. With mean scale d_m, residual scales d_K, d_V, and calibration constants q and κ, we define

$$M = \frac{E_{\mu,K} + E_{\mu,V}}{2d_m}, \qquad R_{\mathrm{KV}} = \frac{E_{r,K}/d_K + E_{r,V}/d_V}{2q}, \qquad \mathcal{L}_{\mathrm{align}}^{\mathrm{MR}} = \kappa\big(M + \lambda_{\mathrm{res}} R_{\mathrm{KV}}\big). \tag{8}$$

We use $\lambda_{\mathrm{res}} = 12$ for every model pair. The native scales are estimated once from training targets; q calibrates the residual aggregate and κ matches the overall loss scale at this reference. These constants remain fixed through both stages. Appendix C.1 gives their calculation and the masked reductions.

Separating mean and residual errors does not by itself balance the K/V branches. A pooled residual normalizer rescales both branches by the same factor, leaving their native scale difference intact. For the same error relative to each branch's native residual energy, the pooled objective assigns less weight to keys because their residual fraction is smaller. Dividing by d_K and d_V separately removes this scale imbalance before applying the common residual weight.

At the target-only calibration reference, each objective is evaluated with zero predictions on 2,000 native training targets in batches of four. The mean, key-residual, and value-residual terms then account for 70%, 6%, and 24% of plain alignment. The MR weights yield 8%, 46%, and 46%, respectively (Table C.1). MR thus particularly increases the emphasis on key displacements. These are reference proportions; the training loss shares depend on the prediction errors.

Batch means and block energies are recomputed over valid examples. Singleton coordinates receive only mean-error supervision because their residuals are zero; coverage is reported in Appendix C.1. All batch statistics are absent at inference.

3.3 Alignment-Preserving Task Training

Answer training should improve predictions while preserving alignment to each input's native cache. In the same-input setting, the receiver also reads x_i, so answer loss alone leaves open which details the prefix must retain. Alignment pretraining supplies an initialization, while retaining its objective continues to supervise each input's native representation as the prefix adapts to the task.

Stage 2 step	Keep	Drop
Native-cache reconstruction error		
0	0.2657	0.2657
25	0.3287	0.7926
200	0.3144	2.8267
Answer cross-entropy		
200	0.9672	0.9736

Table 1: Early Stage 2 continuation from the same Stage 1 checkpoint (4B→0.6B, seed 0). Keep retains alignment; Drop removes it. Reconstruction is measured with the plain block-relative objective.

Table 1 shows how quickly the two continuations diverge. Removing alignment raises native-cache reconstruction error to 2.4 times the retained-alignment value after 25 updates and 9.0 times after 200, while answer cross-entropy at step 200 is nearly identical. Answer training can therefore move the prefix away from the native reference early in optimization while achieving a similar answer loss.

Let $\ell_i^{\mathrm{matched}}$ be cross-entropy on reference answer tokens when the receiver uses its own input and the prefix translated from the corresponding sharer cache. The task objective is

$$\mathcal{L}_{\mathrm{task}} = \frac{1}{N} \sum_{i=1}^{N} \ell_i^{\mathrm{matched}}. \tag{9}$$

Both language models remain frozen, and only g_θ is updated. Let $\mathcal{L}_{\mathrm{align}}$ denote the selected alignment objective: Equation 8 for CACHET or Equation 7 for CACHET w/o MR. Stage 1 optimizes alignment; Stage 2 continues from its checkpoint and adds the task objective:

$$\mathcal{L}^{(1)} = \mathcal{L}_{\mathrm{align}}, \qquad \mathcal{L}^{(2)} = \mathcal{L}_{\mathrm{align}} + \lambda_{\mathrm{task}} \mathcal{L}_{\mathrm{task}}. \tag{10}$$

We use $\lambda_{\mathrm{task}} = 0.3$. Gradients pass through the frozen receiver and update only g_θ. The alignment term acts as a soft anchor to the per-input native target, while the answer loss can favor task-dependent

Method	4B→0.6B		8B→1.7B		14B→0.6B	
	4 bench.	MMLU-Pro	4 bench.	MMLU-Pro	4 bench.	MMLU-Pro
Receiver	36.0 (0.00)	13.9 (0.00)	65.5 (0.00)	29.5 (0.00)	36.0 (0.00)	13.9 (0.00)
T2T	44.9 (21.1)	19.0 (18.1)	67.9 (14.4)	33.9 (24.6)	44.7 (17.3)	19.7 (16.5)
C2C	51.2 (36.0)	15.3 (5.0)	64.8 (-4.1)	31.1 (8.9)	57.6 (42.9)	17.3 (9.7)
DLC-SI	66.9 (73.0)	25.2 (40.1)	68.3 (16.7)	30.6 (6.3)	74.3 (75.7)	29.5 (44.2)
CACHET	**73.1 (87.5)**	**31.2 (61.7)**	**75.7 (59.9)**	**36.5 (39.3)**	**80.8 (88.7)**	**33.9 (56.9)**
Sharer	78.4 (100.0)	42.0 (100.0)	82.5 (100.0)	47.4 (100.0)	86.6 (100.0)	49.1 (100.0)

Table 2: Main results with CACHET at width 4096. Cells report accuracy (PGR), both in percent. Four-benchmark results use macro accuracies. PGR uses common receiver-solo and sharer-solo baselines. CACHET and DLC-SI use seven seeds; C2C uses three for 4B→0.6B on the four benchmarks and one for the other available results. T2T follows C2C's text-relay protocol (one deterministic run; the sharer generates 22–48 tokens on average). C2C PGRs derived from rounded accuracies are approximate.

departures. Section 4.3 evaluates the alignment objective and its retention during Stage 2; Appendix A gives the training schedules.

Although training uses the same underlying input for both models, the interface does not require their inputs to be identical at inference. The prefix is constructed from the sharer's payload, while the receiver processes its own input at separate positions. Section 4.2 evaluates this setting with passages available only to the sharer, using the same checkpoints without additional training.

4 Experiments

Establishing that CACHET carries what the sharer read requires evidence beyond accuracy gains. We measure those gains on same-input benchmarks (Section 4.1), then test whether the receiver can answer from sharer-only information and whether its answers depend on the supplied cache (Section 4.2). Section 4.3 examines the effects of the training objective, translator architecture, and model size.

Experimental setup. CACHET uses translator width 4096 unless an architectural variant is specified. Task-only and CACHET w/o MR use the same translator as CACHET, with answer loss alone and plain receiver-native alignment, respectively. We evaluate frozen Qwen3 (Yang et al., 2025) sharer–receiver pairs on ARC-C (Clark et al., 2018), OpenBookQA (Mihaylov et al., 2018), MMLU-Redux (Gema et al., 2025), C-Eval (Huang et al., 2023), and two MMLU-Pro subsets (Wang et al., 2024). Comparisons use identical evaluation items and a common zero-shot, answer-only scoring protocol. We compare with C2C (Fu et al., 2026) and DLC-SI, our same-input reimplementation of DLC (Chen et al., 2026) (Appendix A). DLC-SI keeps DLC's cache transformation and two-phase losses rather than the CACHET translator, and is trained on the same data. We choose the same-input variant to match CACHET's training condition; Appendix A.5 reports a focused comparison with mixed Phase-II training on 4B→0.6B. Main CACHET and DLC-SI results use seven training seeds; ablation and C2C seed counts are specified in the tables. Performance-gap recovery (PGR) measures improvement over the solo receiver relative to the solo sharer–receiver accuracy gap. We compute it from macro-averaged accuracies over the four benchmarks or the two MMLU-Pro subsets.

4.1 Main Results

CACHET improves over DLC-SI on both the four-benchmark aggregate and MMLU-Pro for 4B→0.6B, 8B→1.7B, and 14B→0.6B (Table 2). For the representative 4B→0.6B pair, it recovers 87.5% of the four-benchmark solo gap and 61.7% of the MMLU-Pro gap, with accuracy gains of approximately six percentage points over DLC-SI on both aggregates. T2T, in which the sharer writes background text that the receiver reads (C2C's protocol), recovers 21.1% of the gap on this pair; generating that text makes it 5.2–8.8× slower than CACHET at batch size one on a single GPU (Table B.3). Under cache replacement on this pair (Figure 1), another question's cache lowers CACHET below its receiver-only accuracy, and a fixed output-mean prefix retains 19–27% of the matched gain (Appendix C.2).

Across the nine pairs with 0.6B or 1.7B receivers, CACHET achieves higher four-benchmark mean accuracy than DLC-SI in all nine pairs and higher MMLU-Pro accuracy in eight (Table B.1).

4.2 Transfer of Information Unavailable to the Receiver

Can the receiver answer using information it never reads directly? We test same-input checkpoints on two passage tasks without training on either task. Only the sharer sees the passage; Matched and Swap conditions test whether answers depend on its content (Table 3, 4B→0.6B).

Counterfactual passage transfer. The first task contains paired passages supporting different answers, while the receiver's question and four options remain identical within each pair. CACHET reaches 97.3% mean accuracy with the matched cache; replacing it with the partner passage's cache lowers accuracy against the original answer to 1.0%. DLC-SI also succeeds on this task, reaching 95.9% matched accuracy and 1.3% under Swap. These results show that both methods can transfer passage information absent from the receiver's input after same-input training.

Passage retrieval with more answer choices. The second task tests transfer with eight answer options. CACHET and DLC-SI average 88.3% and 64.4% accuracy, respectively, across seven seeds; CACHET w/o MR averages 70.7% across three seeds. Replacing the sharer's cache with the entire cache of an unrelated input lowers CACHET's accuracy to 8.6%. This control tests dependence on the supplied input; it differs from the paired-passage intervention above.

Within the retrieval-task family, increasing the options from four to eight widens the 4B→0.6B accuracy gap over DLC-SI from 2.1 to 23.9 percentage points (Table B.6). CACHET has higher eight-option mean accuracy in all six evaluated pairs; the gap grows in four and narrows in two (Table B.6). These results support an advantage on the evaluated eight-option tasks, with the effect of increasing option count varying by pair.

Method	Matched	Swap
Counterfactual passage: four options		
C2C	22.6 ± 4.0	21.8 ± 5.9
DLC-SI	95.9 ± 5.2	1.3 ± 1.6
CACHET w/o MR	$\mathbf{97.8 \pm 2.0}$	0.8 ± 0.7
CACHET	97.3 ± 5.8	1.0 ± 2.0
Separated	87.3 ± 10.6	4.3 ± 3.1
Passage retrieval: eight options		
C2C	46.3 ± 29.9	13.7 ± 2.9
DLC-SI	64.4 ± 5.3	10.7 ± 2.5
CACHET w/o MR	70.7 ± 5.9	10.3 ± 0.6
CACHET	$\mathbf{88.3 \pm 12.6}$	8.6 ± 1.3
Separated	52.0 ± 9.0	13.3 ± 2.1

Table 3: Passage-transfer accuracy (%), 4B→0.6B. CACHET variants use width 4096 except Separated. Mean ± sample SD over 7 seeds (CACHET, DLC-SI) or 3 (others). Swap: counterfactual partner passage (4 options) or an unrelated input's full cache (8 options); scores use the original target answer. Bold: highest matched mean per task.

4.3 Ablation Studies

Our ablations separate the effects of the training objective and translator architecture. On 4B→0.6B, we vary training components with the translator fixed, then restrict source access with MR and alignment retention fixed (Tables 4 and 3). Both studies keep the language models frozen and use the same data and update budget. The controlled comparisons use batch size 16 and 1,320 updates per stage, with seeds 0–2. Task-only uses answer loss throughout; CACHET w/o MR and CACHET use plain and MR alignment, respectively, in both stages. We denote CACHET's configuration by MR&KV12. MR&KV1 sets $\lambda_{\text{res}} = 1$ and recomputes κ; MR&GLOBAL12 uses a shared K/V residual scale at weight 12. MR → CE-only starts from the same MR Stage 1 checkpoint and removes alignment during Stage 2. We then vary model size across nine pairs using the same CACHET training recipe and translator width.

Training components. Task-only has low mean recovery and high seed variance. Removing alignment in Stage 2 lowers four-benchmark PGR from 87.8 to 29.5 despite starting from the same MR Stage 1 checkpoint. The early continuation diagnostic (Table 1) shows that native-cache reconstruction error already diverges within 200 updates while answer cross-entropy remains nearly identical. Among CACHET recipes that retain alignment, CACHET has the highest mean PGR and the lowest observed seed SD on both aggregates.

Its within-seed gain over CACHET w/o MR is 4.0 points on the four-benchmark aggregate and 11.9 points on MMLU-Pro; the MMLU-Pro margins over MR&KV1 and MR&GLOBAL12 are 8–10 points. In the three-seed counterfactual comparison, retaining alignment raises DLC-SI's matched accuracy from $94.8 \pm 8.2\%$ to $98.5 \pm 0.7\%$.

Translator architecture. With MR and alignment retention fixed, we test whether joint source processing improves performance. Restricting each output to one source head's keys or values (Separated) lowers benchmark recovery, increases seed variation (Table 4), and reduces eight-option passage accuracy (Table 3). Thus, residual-sensitive alignment does not remove the cost of restricted source access. For DLC-SI, the benefits of retaining alignment and applying MR vary across model pairs; even with both changes, it remains below CACHET on 4B→0.6B (Table D.1).

Model size. Four-benchmark accuracy improves as the sharer grows up to 14B for both receivers (Figure 4). At 32B, both accuracy and PGR decline under the same training recipe. The advantage over DLC-SI also varies by model pair, narrowing at 32B for the 1.7B receiver. Larger sharers therefore do not consistently yield better recovery or a greater advantage over DLC-SI. Full pairwise results are in Appendix B.

Training recipe	4-bench	MMLU-Pro
Task-only	2.7 ± 26.1	5.0 ± 16.3
CACHET w/o MR	83.8 ± 4.4	49.5 ± 11.3
MR&KV1	82.3 ± 3.7	51.2 ± 6.1
MR&GLOBAL12	80.8 ± 6.1	53.1 ± 4.1
MR → CE-only	29.5 ± 6.4	20.5 ± 13.3
Separated	62.0 ± 14.2	33.1 ± 4.0
CACHET	$\mathbf{87.8 \pm 2.2}$	$\mathbf{61.3 \pm 2.3}$
DLC-SI	73.2 ± 3.1	40.3 ± 1.6
DLC-SI + align.	74.3 ± 4.8	38.8 ± 4.1

Table 4: CACHET ablations (width 4096 except Separated) and DLC-SI controls on 4B→0.6B. PGR (%), mean ± sample SD over three seeds. DLC-SI + align. retains reconstruction during Phase 2. The seven-seed CACHET mean is 87.5 (Table 2).

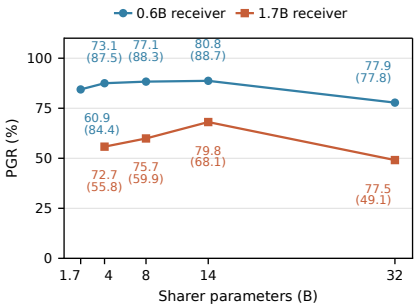


Figure 4: CACHET four-benchmark PGR (%) at width 4096. Labels show accuracy (PGR), both in %.

5 Related Work

Cross-model cache translation. C2C learns to project and fuse sharer and receiver caches via response supervision (Fu et al., 2026). Dery et al. (2026) align models using a shared cache space; SoftCoT projects assistant hidden states into target embeddings (Xu et al., 2025). Dense Latent Communication reconstructs receiver-native caches before generation training, including settings where the receiver lacks the source context (Chen et al., 2026). CACHET uses receiver-native supervision with joint processing of source heads and K/V branches and a K/V-aware objective that weights shared structure and input-specific variation and remains active during task training.

Source dependence and cache structure. Prefix-tuning shows that input-independent continuous prefixes can adapt a frozen model (Li & Liang, 2021; Lester et al., 2021). Recent audits examine source dependence using latent-message and cache substitutions (Zhang & Emu, 2026; Cheng et al., 2026). We combine cache substitutions with training ablations and passage tasks requiring information absent from the receiver input. KIVI treats keys and values differently to accommodate their distributions (Liu et al., 2024), while cache compression uses token selection (Zhang et al., 2023; Li et al., 2024) or low-rank attention (Saxena et al., 2024). CACHET uses shared and input-specific cache structure across examples to guide its alignment objective.

6 Conclusion

To carry what the sharer read, CACHET combines joint source processing with K/V-aware mean–residual alignment to per-input receiver-native targets throughout answer training. Across evaluated model pairs, it improves accuracy and transfers passage information unavailable to the receiver without additional training for asymmetric inputs. Ablations support these design choices, and cache replacements show dependence on the supplied information. Cross-family transfer beyond shared-tokenizer models, open-ended generation, and end-to-end communication efficiency remain future work.

AI USE STATEMENT

In this work, we used generative AI tools (Anthropic Claude Fable) for the following tasks with required disclosure. (i) Implementing methods: AI tools assisted in writing and modifying training and evaluation code, including the rule-based generator for the passage tasks (Appendix C.3); the passages, questions, and options are produced by that script, not by a language model. (ii) Generating synthetic data and cleaning datasets: we used Claude Fable to generate the auxiliary PLAN and teacher-prediction fields attached to the evaluation pools; these fields affect only which items are retained during preprocessing and are ignored by the evaluator (Appendix A). (iii) Providing feedback on methodology and experiments and interpreting results: AI tools critiqued the experimental controls, suggested priorities among additional experiments, and audited whether each stated claim is supported by the reported numbers. We have not used generative AI tools to propose the research hypotheses or the method design, to develop theoretical or conceptual frameworks, to formulate mathematical claims, or to assist with proofs; translation and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to draft and edit parts of the manuscript, to check consistency between the text, tables, and appendix, and to search for related work.

We have reviewed all AI-assisted work. AI-assisted code was reviewed and tested by the authors, all reported results come from author-run training and evaluation, and the reported numbers were checked by the authors against the logged outputs. AI-suggested interpretations and wording were adopted only after the authors verified them against the data, and every paper identified with AI assistance was read by the authors before citation. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

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. doi: 10.48550/arXiv.2606.13594. URL https://arxiv.org/abs/2606.13594.

Jiaming Cheng, Subhransu Das, and Rajiv Ramnath. When does latent communication pay? a causal audit of relayed KV caches in multi-agent LLMs. *arXiv preprint arXiv:2608.04893*, 2026. doi: 10.48550/arXiv.2608.04893. URL https://arxiv.org/abs/2608.04893.

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

Lucio M. Dery, Zohar Yahav, Henry Prior, Qixuan Feng, Jiajun Shen, and Arthur Szlam. Latent space communication via K-V cache alignment. *arXiv preprint arXiv:2601.06123*, 2026. doi: 10.48550/arXiv.2601.06123. URL https://arxiv.org/abs/2601.06123.

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*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/474ada926b331d78f06d95e8913111cc-Abstract-Conference.html.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, Claire Barale, Robert McHardy, Joshua Harris, Jean Kaddour, Emile Van Krieken, and Pasquale Minervini. Are we done with MMLU? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5069–5096. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.262. URL https://aclanthology.org/2025.naacl-long.262/.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. C-Eval: A multi-level multi-discipline chinese evaluation suite for

foundation models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 62991–63010. Curran Associates, Inc., 2023. doi: 10.52202/075280-2749. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/c6ec1844bec96d6d32ae95ae694e23d8-Abstract-Datasets_and_Benchmarks.html.

Yujin Kim, Euiin Yi, Minu Kim, Se-Young Yun, and Taehyeon Kim. Guiding reasoning in small language models with LLM assistance. *arXiv preprint arXiv:2504.09923*, 2025. doi: 10.48550/arXiv.2504.09923. URL https://arxiv.org/abs/2504.09923.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.243. URL https://aclanthology.org/2021.emnlp-main.243/.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bz.html.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Maximillian Rossi, Prajwal Raghunath, and Eugene Wu. Latent cache flow: Model-to-model communication without text. *arXiv preprint arXiv:2605.22863*, 2026. doi: 10.48550/arXiv.2605.22863. URL https://arxiv.org/abs/2605.22863.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for KV cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15332–15344. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.899. URL https://aclanthology.org/2024.findings-emnlp.899/.

Jianshu She, Zhuohao Li, Zhemin Huang, Qi Li, Peiran Xu, Haonan Li, and Qirong Ho. Hawkeye: Efficient reasoning with model collaboration. *arXiv preprint arXiv:2504.00424*, 2025. doi: 10.48550/arXiv.2504.00424. URL https://arxiv.org/abs/2504.00424.

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.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290. Curran Associates, Inc., 2024. doi: 10.52202/079017-3018.

URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT: Soft chain-of-thought for efficient reasoning with LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23336–23351, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1137. URL https://aclanthology.org/2025.acl-long.1137/.

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

Huixiang Zhang and Mahzabeen Emu. Do latent channels actually communicate? a causal audit of latent multi-agent LLM. *arXiv preprint arXiv:2607.26773*, 2026. doi: 10.48550/arXiv.2607.26773. URL https://arxiv.org/abs/2607.26773.

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. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract.html.

A Training and Evaluation Details

All CACHET results use translator width $d_b = 4096$.

A.1 Model pairs and translator architecture

Table A.1 lists the twelve frozen Qwen3 pairs. Each pair has its own translator and uses the receiver tokenizer for both models, with thinking disabled.

Let L_s and L_r denote the sharer and receiver layer counts. For zero-based receiver layer j, the selected one-based sharer layer is

$$s(j) = \min\{L_s, \max\{1, \mathrm{round}((j+1)L_s/L_r)\}\}. \tag{A.1}$$

Rounding uses Python's ties-to-even rule. Sharer keys are extracted after `k_norm`, before RoPE; values come from `v_proj`. Concatenating flattened keys and values (eight heads of dimension 128 each) gives 2,048 features per payload token.

The trunk uses affine LayerNorm ($\epsilon = 10^{-5}$), biased $2048 \to 4096 \to 4096$ linear layers, and exact GELU after each linear layer. Each receiver layer has separate biased $4096 \to 1024$ K/V heads. The trunk acts independently on each token and selected layer, without dropout or explicit position input. The 4B→0.6B translator has approximately 260M parameters; all controlled ablations share this architecture.

Receiver	Sharers	Pair identifiers
0.6B	1.7B, 4B, 8B, 14B, 32B	P1, P2, P3, P7, P9
1.7B	4B, 8B, 14B, 32B	P4, P5, P8, P10
4B	8B, 14B, 32B	P6, P11, P12

Table A.1: Model pairs. CACHET and DLC-SI use seven seeds per pair, except P12 (three). Other seed counts accompany their results.

A.2 Payloads, native targets, and prefix positions

The *payload* is the raw prompt span, excluding chat markers and generated answers. Its n_i valid tokens produce n_i prefix positions; batches pad to the maximum payload length and mask padding. The frozen receiver reads these tokens at positions $0, \ldots, n_i - 1$ to supply detached native targets: post-RoPE keys and unrotated values. The translator predicts these coordinates directly, without another RoPE transform. The target-producing pass is used only in training.

The sharer reads the payload inside its full chat prompt. If the payload begins at source position a_i, its token j is processed at source position $a_i + j$, whereas the native target uses position j. The translator receives the corresponding pre-RoPE key and value features, with no position input or position-indexed parameters. Its key output is inserted directly into the cache in the target's post-RoPE coordinates.

Predicted K/V states enter a bf16 `DynamicCache`. Prefix and receiver-prompt masks are concatenated; receiver position IDs count valid tokens and start at n_i. Training and evaluation share this prefix path. Models use bf16, translator parameters and loss reductions use fp32, and the deterministic evaluator uses eager attention.

A.3 Training data and input construction

OpenHermes-2.5's 1,001,551 rows are shuffled with `random.Random(0)`. From human–assistant exchanges, the builder selects 9,000 general examples (12–220-word prompts; responses of at least six words) and 1,900 multiple-choice examples (line-initial lettered options; responses of at most 120 words). Multiple-choice rows are repeated to 3,000, the pool is reshuffled, and responses are capped at 90 words.

Requiring 16–300 prompt tokens and at least four response tokens leaves 10,574 rows. The loader caps responses at 64 tokens, reapplies these criteria, and removes fifteen unlocatable payload spans. All remaining 10,559 rows (8,407 general; 2,152 multiple-choice, including repeats) are used for training.

For example k, the sharer cycles through identity, math, and C2C-general templates using $k \bmod 3$ and applies the Qwen chat template. Native targets use the raw payload; answer cross-entropy uses response tokens only. Both models see the same underlying prompt during training.

A.4 Optimization and training recipes

Each stage makes two file-order traversals of the 10,559 rows, restarting at the first row. Batch size 16 without accumulation gives 1,320 updates per stage (2,640 total), with fourteen rows in the final batch. Initialization follows the training seed. Each stage retains its last checkpoint; Stage 2 resets the optimizer and scheduler.

Setting	Value
Optimizer	AdamW; reset at the stage boundary
Learning rate, Stages 1 / 2	5×10^{-4} / 2×10^{-4}
Updates, Stages 1 / 2	1,320 / 1,320
Batch / gradient accumulation	16 / none
Weight decay / gradient clipping	0.01 / 1.0
Adam (β_1, β_2); ϵ	$(0.9, 0.999)$; 10^{-8}
Schedule / warm-up	Cosine / 50 updates per stage
Checkpoint	Last update of each stage

Table A.2: Optimization settings for CACHET and its controlled training ablations at width 4096.

For zero-based update s within a stage, the learning-rate multiplier is

$$\min\left(1, \frac{s+1}{50}\right) \frac{1 + \cos(\pi \min(1, s/1320))}{2}. \tag{A.2}$$

Table A.3 specifies the objectives; Appendix C.1 defines the alignment terms. Task gradients pass through the frozen receiver and update only the translator.

Recipe	Stage 1	Stage 2
Task-only	$0.3\mathcal{L}_{\text{task}}$	$0.3\mathcal{L}_{\text{task}}$
CACHET w/o MR	$\mathcal{L}_{\text{align}}^{\text{w/o MR}}$	$\mathcal{L}_{\text{align}}^{\text{w/o MR}} + 0.3\mathcal{L}_{\text{task}}$
MR & KV1	$\mathcal{L}_{\text{align}}^{\text{KV1}}$	$\mathcal{L}_{\text{align}}^{\text{KV1}} + 0.3\mathcal{L}_{\text{task}}$
MR & GLOBAL12	$\mathcal{L}_{\text{align}}^{\text{GLOBAL12}}$	$\mathcal{L}_{\text{align}}^{\text{GLOBAL12}} + 0.3\mathcal{L}_{\text{task}}$
MR $\rightarrow$ CE-only	$\mathcal{L}_{\text{align}}^{\text{MR}}$	$0.3\mathcal{L}_{\text{task}}$
CACHET	$\mathcal{L}_{\text{align}}^{\text{MR}}$	$\mathcal{L}_{\text{align}}^{\text{MR}} + 0.3\mathcal{L}_{\text{task}}$

Table A.3: Controlled training recipes. CACHET is the MR & KV12 configuration. MR $\rightarrow$ CE-only continues from the corresponding seed's MR Stage 1 checkpoint.

DLC-SI. No official DLC implementation was available at the time of submission; we implement DLC-SI from the paper's description (Chen et al., 2026). DLC-SI uses DLC's cache transformation rather than the CACHET translator. Receiver layer l reads sharer layer $\text{round}\big(l(L_s-1)/(L_r-1)\big)$ (their Eq. 10, zero-based), and the KV-group map π_l is the identity because every Qwen3 model used here has eight KV heads. Each receiver layer and KV group has separate key and value MLPs ($d_S \rightarrow 16\,d_S \rightarrow d_R$, that is, $128 \rightarrow 2{,}048 \rightarrow 128$, with GELU and biases), and one learnable gate $\gamma^{(l,g)} = \sigma(\cdot)$ scales both outputs (their Eq. 11–12); gate logits start at zero, so $\gamma = 0.5$ initially. The transformation acts on position-free keys and values, and the receiver's RoPE is applied to the gated keys afterwards at the prefix positions (their Eq. 8–9). Because sharer keys are read before RoPE (Appendix A.1), sender-side RoPE removal is the identity; a unit test confirms that re-applying RoPE to these keys reproduces the sharer's own cached keys. The translated states enter the receiver through the same prefix path and the same evaluator as CACHET.

Phase 1 sums unnormalized squared reconstruction error over layers, KV heads, channels, and K/V, divided by valid-token count (their Eq. 5 up to this normalization); since RoPE is an isometry, this

error is the same before and after the receiver's RoPE is re-applied. Phase 2 starts from the Phase 1 weights and uses cross-entropy on response tokens alone, with weight 1.0 and no reconstruction term, as in DLC. DLC-SI is trained on the same 10,559 rows as CACHET, in the same order and with the same sharer-template rotation and 64-token response cap (Appendix A.3). The optimizer, schedule, warm-up, clipping, batch size, and update budget (1,320 updates per phase) are copied from CACHET (Table A.2, Equation A.2), so that the comparison reflects architecture and objective rather than optimization settings.

Learning rates are the exception. For each pair and phase, we select among three candidates by held-out loss after 150 updates with seed 0: $\{5, 10, 20\} \times 10^{-4}$ in Phase 1 and $\{2, 5, 10\} \times 10^{-4}$ in Phase 2. Phase 1 selects 2×10^{-3}, the largest candidate, for all twelve pairs. Phase 2 selects 10^{-3} for P1, P5, P8, and P9, 5×10^{-4} for P2, P4, P6, P10, and P12, and 2×10^{-4} for P3, P7, and P11 (pair identifiers in Table A.1); seven of these twelve choices lie at an end of the candidate set. CACHET uses the fixed learning rates of Table A.2 for every pair.

DLC-SI departs from DLC in three respects. First, it is trained on our data with cross-entropy on response tokens, whereas DLC trains on GSM8K, MATH-algebra, and ARC-Challenge with receiver-generated reasoning traces as targets. Second, in Phase 2 the DLC-SI receiver always reads its own prompt, whereas DLC samples prompts with and without the receiver-side context with equal probability (their Eq. 6–7 and their Appendix A). Third, DLC reports 2,000 Phase-II updates and does not state the Phase-I count; DLC-SI uses 1,320 updates in each phase to match CACHET. Table A.4 summarizes the settings.

Setting	DLC-SI
Optimizer	AdamW, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$;
Learning rate, Phase 1	2×10^{-3} (all pairs; pilot-selected)
Learning rate, Phase 2	2×10^{-4}, 5×10^{-4}, or 10^{-3} by pair (pilot-selected; see text)
Updates, Phases 1 / 2	1,320 / 1,320
Batch / gradient accumulation	16 / none
Weight decay / gradient clipping	0.01 (all parameters) / 1.0
Schedule / warm-up	Equation A.2: cosine to zero times a 50-update warm-up, per phase
Phase 2 loss	response-token cross-entropy, weight 1.0
Checkpoint	Last update of each phase
Precision	fp32 transformation and losses; frozen models in bf16
MLP	$128 \to 2{,}048 \to 128$ ($16\,d_S$ hidden), exact GELU, biased
Layer map	DLC Eq. 10 (zero-based)
KV-group map π_l	identity
Gate	$\gamma = \sigma(\text{logit})$, logit initialized to 0; shared by K and V
Parameters	235.9M (0.6B and 1.7B receivers); 303.2M (4B receivers)

Table A.4: DLC-SI settings. All 80 checkpoints share them except for the Phase 2 learning rate and the receiver-dependent parameter count. The DLC paper specifies the architecture (their Eq. 8–12) and 2,000 Phase-II updates; it does not state the optimizer, learning rate, batch size, warm-up, schedule, Phase-I update count, gate initialization, or π_l.

A.5 Mixed Phase-II training

Method	4-bench	MMLU-Pro
CACHET	73.20 ± 0.93	31.13 ± 0.64
CACHET-MIX	72.82 ± 1.06	28.37 ± 1.54
DLC-SI	67.01 ± 1.32	25.23 ± 0.46
DLC-MIX	65.00 ± 1.13	25.00 ± 0.66

Table A.5: Mixed Phase-II training on 4B→0.6B. Accuracy (%), three-seed mean ± SD.

Table A.5 compares same-input and mixed Phase-II training on the 4B→0.6B pair used in our controlled ablations. On this pair, neither method attains higher mean accuracy under MIX on either evaluation. The effect is pair-dependent: on 8B→1.7B, MIX increases four-benchmark accuracy from 75.10 to 77.62 for CACHET and from 69.08 to 75.05 for DLC, also over three seeds.

C2C. The released checkpoint used in the introductory audit is `nics-efc/C2C_Fuser`, subfolder `qwen3_0.6b+qwen3_4b_Fuser/final`. It has approximately 529M parameters and maps sharer layers 8–35 to receiver layers 0–27. Locally trained results are reported separately. The recorded local recipe uses 10,468 rows from a random 99% split, shuffled epochs, maximum length 2,048, batch eight with accumulation two, and 1,310 updates over two epochs. AdamW uses learning rate 10^{-4}, weight decay 0.01, a linear schedule with 10% warm-up, clipping 1.0, projector annealing 300, and `last_aligned` mapping.

A.6 Evaluation populations and answer scoring

Benchmark	Items	Evaluated population
ARC-C	493	Fixed 500-item test pool after seven exclusions.
OpenBookQA	484	Test split after sixteen exclusions.
MMLU-Redux	442	Fixed twelve-subject pool after formatting and sanitization exclusions.
C-Eval	1,345	Validation split across 52 subjects; one overlength chat input excluded.
MMLU-Pro	$500 + 500$	Two fixed subsets, A and B, with ten answer options.

Table A.6: Fixed evaluation populations. Within each comparison, all methods use the same items and labels.

ARC-C uses the first 500 seed-0-shuffled test items; OpenBookQA starts from its 500-item test split. Redux takes 500 seed-0-shuffled `error_type=="ok"` items from twelve subjects. Preprocessing removes fourteen teacher-format failures in Redux, then seven ARC-C, sixteen OpenBookQA, and forty-four Redux items via PLAN sanitization: generated PLAN text is cut at answer cues (including gold-dependent cues), and items with fewer than 24 remaining tokens are excluded. The evaluator ignores PLAN and teacher-prediction fields. C-Eval excludes one chat input exceeding the 320-token eligibility limit. ARC-C and OpenBookQA also informed development; overlap screening covered exact stem-prefix matches and found none.

Evaluation is zero-shot, answer-only, and uncalibrated. The sharer's 320-token payload cap truncates nineteen Redux inputs; the receiver's C2C-formatted question and options are not subject to this cap. The scorer teacher-forces `The correct answer is` and takes the largest next-token logit over bare and space-prefixed answer letters.

Accuracy, PGR, and seed aggregation. Let $a_{j,b}$ be seed j's accuracy on benchmark b, with fixed receiver/sharer solo accuracies $a_{r,b}, a_{s,b}$. Their macro averages $\bar{a}_r, \bar{a}_s$ use the same benchmark set $\mathcal{B}$:

$$\bar{a}_j = |\mathcal{B}|^{-1} \sum_{b \in \mathcal{B}} a_{j,b}, \qquad \mathrm{PGR}_j = 100 \frac{\bar{a}_j - \bar{a}_r}{\bar{a}_s - \bar{a}_r}. \tag{A.3}$$

ARC-C, OpenBookQA, MMLU-Redux, and C-Eval receive equal weight; MMLU-Pro averages A/B. PGR uses unrounded macro accuracies without clipping. Tables give mean and sample SD across seeds (divisor $n - 1$); paired differences are computed within seeds. Single-seed and solo results have no seed SD.

C2C reference paths. C2C's no-fusion reference retains sectioned prefill. The released checkpoint audit reports *gain retention* for condition u:

$$\rho_{u,b} = \frac{a_{u,b} - a_{\text{no-fusion},b}}{a_{\text{matched},b} - a_{\text{no-fusion},b}}. \tag{A.4}$$

This is the retained fraction of the matched gain. PGR instead uses the common sharer–receiver solo endpoints above.

B Results by Model Pair and Training Recipe

All CACHET and controlled ablations use width 4096. Tables report accuracy and PGR (%), seed count n, and sample SD across seeds.

Tables B.1 and B.2 report performance across model pairs and individual benchmarks. The first covers twelve pairs; the second gives benchmark-level scores and solo references for 4B→0.6B and 8B→1.7B. C2C uses locally trained checkpoints.

Pair	Method	n	4-bench acc.	PGR	MMLU-Pro acc.	PGR
1.7B→0.6B	DLC-SI	7	55.5 ± 3.6	66.1	19.7 ± 1.2	37.4
	CACHET	7	60.9 ± 2.1	84.4	24.0 ± 1.3	64.5
4B→0.6B	DLC-SI	7	66.9 ± 1.0	73.0	25.2 ± 0.9	40.1
	CACHET	7	73.1 ± 0.9	87.5	31.2 ± 0.6	61.7
8B→0.6B	DLC-SI	7	72.1 ± 1.2	77.7	28.1 ± 0.9	42.4
	CACHET	7	77.1 ± 0.5	88.3	33.9 ± 1.9	59.8
14B→0.6B	DLC-SI	7	74.3 ± 2.7	75.7	29.5 ± 1.9	44.2
	CACHET	7	80.8 ± 1.1	88.7	33.9 ± 1.9	56.9
32B→0.6B	DLC-SI	7	65.1 ± 3.0	54.1	23.2 ± 1.4	23.0
	CACHET	7	77.9 ± 3.5	77.8	25.0 ± 1.0	27.4
4B→1.7B	DLC-SI	7	68.7 ± 1.5	24.6	30.0 ± 1.1	4.3
	CACHET	7	72.7 ± 1.8	55.8	33.7 ± 1.8	33.9
8B→1.7B	DLC-SI	7	68.3 ± 3.1	16.7	30.6 ± 1.3	6.3
	CACHET	7	75.7 ± 1.2	59.9	36.5 ± 1.0	39.3
14B→1.7B	DLC-SI	7	71.2 ± 2.6	27.0	31.7 ± 1.6	11.4
	CACHET	7	79.8 ± 1.0	68.1	37.8 ± 1.3	42.5
32B→1.7B	DLC-SI	7	77.3 ± 1.6	48.2	32.9 ± 2.4	13.7
	CACHET	7	77.5 ± 1.7	49.1	32.4 ± 1.2	11.7
8B→4B	DLC-SI	7	81.7 ± 0.7	–	41.5 ± 1.4	–
	CACHET	7	83.1 ± 0.5	–	42.9 ± 1.1	–
14B→4B	DLC-SI	7	80.5 ± 0.7	–	39.2 ± 0.8	–
	CACHET	7	83.6 ± 0.4	–	41.7 ± 0.9	–
32B→4B	DLC-SI	3	79.2 ± 0.4	–	37.9 ± 0.4	–
	CACHET	3	80.5 ± 0.3	–	39.7 ± 1.0	–

Table B.1: Model-pair results. The four-benchmark score averages ARC-C, OpenBookQA, MMLU-Redux, and C-Eval; MMLU-Pro averages A/B. The model-size PGR comparison covers 0.6B and 1.7B receivers; 4B rows report accuracy.

Method	n	ARC-C	OBQA	MMLU-R	C-Eval	MMLU-Pro
4B→0.6B						
Receiver	–	39.4	39.5	34.2	30.9	13.9
C2C	3/1	59.2 ± 7.7	54.5 ± 6.1	47.5 ± 4.8	43.7 ± 3.7	15.3
DLC-SI	7	77.3 ± 0.9	68.1 ± 1.4	63.4 ± 2.3	58.8 ± 1.7	25.2 ± 0.9
CACHET	7	82.7 ± 1.0	74.1 ± 2.0	70.3 ± 1.2	65.2 ± 1.0	31.2 ± 0.6
Sharer	–	88.2	80.2	74.2	70.9	42.0
8B→1.7B						
Receiver	–	76.3	66.3	61.5	57.8	29.5
C2C	1	74.8	67.8	58.6	58.0	31.1
DLC-SI	7	78.4 ± 3.0	71.8 ± 3.1	63.3 ± 3.8	59.8 ± 2.9	30.6 ± 1.3
CACHET	7	85.8 ± 1.2	79.0 ± 1.0	70.7 ± 1.6	67.1 ± 1.9	36.5 ± 1.0
Sharer	–	91.3	85.1	78.7	74.9	47.4

Table B.2: Benchmark-level accuracy (%). CACHET uses width 4096. For C2C on 4B→0.6B, $3/1$ denotes three seeds on the four benchmarks and one on MMLU-Pro; C2C on 8B→1.7B has one seed throughout. CACHET and DLC-SI have seven seeds. Solo rows are fixed references.

T2T protocol and latency. T2T reproduces C2C's two-stage text relay: the sharer writes background text (C2C's prompt, greedy decoding, 256-token cap, never reached) and the receiver reads that text before answering. It involves no training, so each value is a single deterministic run. Latency is measured at batch size one on a single, otherwise idle GPU hosting both models, from item input to answer letter, with CUDA-event timing, three warm-up and five timed runs per item (median), over 100 items per pair. It excludes any cost of moving caches between devices.

Pair	CACHET (ms)	T2T (ms)	T2T p10 / p90	Sharer tokens	T2T / CACHET
4B→0.6B	78.7	406.5	276 / 585	22.4	5.17×
8B→1.7B	79.4	702.0	519 / 929	43.1	8.84×
14B→0.6B	80.7	617.2	475 / 816	31.9	7.64×

Table B.3: Batch-one latency, median over 100 items. T2T latency follows the number of sharer tokens generated; CACHET requires one sharer prefill and no decoding.

Table B.4 reports the released-C2C audit alongside width-4096 CACHET, seed 0; Appendix C defines the mean controls.

Condition	ARC-C	OBQA	MMLU-R
Released C2C			
Matched	60.65	56.20	47.29
Swap	51.12	47.11	41.40
Input mean	53.14	48.14	44.12
No fusion	38.95	39.05	34.39
CACHET, width 4096, seed 0			
Matched	82.96	75.41	70.81
Swap	32.45	36.16	31.00
Output mean	51.1	46.3	42.1
Receiver only	39.35	39.46	34.16

Table B.4: Cache-replacement accuracy (%) on 4B→0.6B. Each block uses one checkpoint. C2C no fusion retains sectioned prefill; CACHET receiver only has no translated prefix. Output-mean values are available to one decimal place. Gain retention and PGR use the distinct definitions in Appendix A.6.

Passage-transfer results appear in Tables B.5–B.7. Same-input checkpoints are evaluated without additional training. The paired task changes the passage while fixing the receiver question; the additional tasks vary distractors, options, or requested values and use whole-input cache swaps. Appendix C.3 defines the tasks and scoring.

C Definitions and Control Protocols

C.1 Native alignment and K/V-aware mean–residual loss

For receiver layer ℓ and branch $u \in \{K, V\}$, let predicted and target states be $Z, T \in \mathbb{R}^{B\times H\times P\times d}$ and payload validity be $w_{ip} \in \{0, 1\}$. With $n_p = \sum_i w_{ip}$, the batch means and block energy are

$$\mu^Z_{hpc} = \frac{\sum_i w_{ip} Z_{ihpc}}{\max(n_p, 1)}, \qquad \mu^T_{hpc} = \frac{\sum_i w_{ip} T_{ihpc}}{\max(n_p, 1)}, \tag{C.1}$$

$$D_{\ell u} = \max\left(\sum_{i,h,p,c} w_{ip} T^2_{ihpc}, 10^{-9}\right). \tag{C.2}$$

Pair	Method	n	Matched	Passage swap
1.7B→0.6B	DLC-SI	7	87.1 ± 16.9	4.8 ± 6.2
	CACHET	7	93.9 ± 4.8	1.9 ± 1.5
4B→0.6B	DLC-SI	7	95.9 ± 5.2	1.3 ± 1.6
	CACHET	7	97.3 ± 5.8	1.0 ± 2.0
	CACHET w/o MR	3	97.8 ± 2.0	0.8 ± 0.7
	C2C	3	22.6 ± 4.0	21.8 ± 5.9
8B→0.6B	DLC-SI	7	99.6 ± 0.3	0.1 ± 0.1
	CACHET	7	98.7 ± 3.1	0.4 ± 0.9
14B→0.6B	DLC-SI	7	99.6 ± 0.2	0.2 ± 0.2
	CACHET	7	98.9 ± 2.5	0.4 ± 0.8
32B→0.6B	DLC-SI	7	77.1 ± 12.8	7.8 ± 4.7
	CACHET	7	89.7 ± 8.4	3.5 ± 3.0
4B→1.7B	DLC-SI	7	84.0 ± 15.3	5.5 ± 5.1
	CACHET	7	96.9 ± 3.1	1.4 ± 0.9
8B→1.7B	DLC-SI	7	61.0 ± 26.3	12.7 ± 8.3
	CACHET	7	99.6 ± 0.2	0.1 ± 0.1
14B→1.7B	DLC-SI	7	57.4 ± 15.5	15.2 ± 5.2
	CACHET	7	98.6 ± 1.8	0.5 ± 0.6
32B→1.7B	DLC-SI	7	87.7 ± 8.4	5.0 ± 3.0
	CACHET	7	81.2 ± 4.8	6.6 ± 1.8
8B→4B	DLC-SI	7	100.0 ± 0.0	0.0 ± 0.0
	CACHET	7	100.0 ± 0.0	0.0 ± 0.0
14B→4B	DLC-SI	7	95.6 ± 4.7	1.9 ± 1.8
	CACHET	7	100.0 ± 0.0	0.0 ± 0.0
32B→4B	DLC-SI	3	61.7 ± 5.1	12.5 ± 3.3
	CACHET	3	73.8 ± 4.9	8.8 ± 1.5

Table B.5: Paired counterfactual passage accuracy (%) on 400 items with four options. CACHET and w/o MR use width 4096. n denotes training seeds. Receiver question and options are identical within each pair; only the passage changes the answer. Both conditions are scored against the original gold answer.

Pair	Method	Distractor $d = 8$	4-option matched	8-option matched	8-option swap	Multi-value $k = 3$
4B→0.6B	DLC-SI	93.4 ± 5.3	95.3 ± 5.9	64.4 ± 5.3	10.7 ± 2.5	58.7 ± 8.8
	CACHET	98.7 ± 3.0	97.4 ± 5.2	88.3 ± 12.6	8.6 ± 1.3	76.7 ± 6.8
8B→0.6B	DLC-SI	99.3 ± 1.5	99.7 ± 0.5	73.6 ± 7.8	8.7 ± 1.4	73.7 ± 7.6
	CACHET	97.9 ± 3.5	97.7 ± 4.1	76.6 ± 10.4	9.7 ± 1.2	85.7 ± 5.8
8B→1.7B	DLC-SI	60.1 ± 25.7	61.3 ± 25.8	36.6 ± 20.1	11.6 ± 3.1	42.0 ± 20.9
	CACHET	99.6 ± 0.8	99.7 ± 0.8	84.9 ± 4.6	10.3 ± 0.8	66.9 ± 7.7
14B→0.6B	DLC-SI	99.6 ± 1.1	98.0 ± 1.8	66.3 ± 5.2	11.6 ± 1.0	98.6 ± 1.1
	CACHET	99.3 ± 1.9	99.0 ± 2.6	74.9 ± 6.9	9.9 ± 1.7	98.0 ± 1.0
14B→1.7B	DLC-SI	59.1 ± 17.0	55.6 ± 15.6	35.7 ± 15.6	13.7 ± 2.3	63.9 ± 23.3
	CACHET	96.6 ± 4.6	99.4 ± 0.8	75.6 ± 8.3	12.9 ± 2.3	87.0 ± 7.0
32B→0.6B	DLC-SI	78.4 ± 14.3	74.9 ± 16.3	44.1 ± 8.5	12.9 ± 1.9	52.6 ± 11.4
	CACHET	90.9 ± 7.8	91.0 ± 8.5	53.1 ± 5.8	11.1 ± 0.9	86.7 ± 8.8

Table B.6: Additional passage-task accuracy (%), seven-seed mean ± sample SD. CACHET uses width 4096. Distractor and multi-value columns report matched accuracy. The four- and eight-option columns belong to the same retrieval-task family, distinct from the paired task in Table B.5. The eight-option test has 100 items and chance accuracy 12.5%; its swap condition replaces the complete input cache from another family.

Pair	Method	n	Matched	Whole-input swap
4B→0.6B	C2C	3	46.3 ± 29.9	13.7 ± 2.9
	CACHET w/o MR	3	70.7 ± 5.9	10.3 ± 0.6
8B→1.7B	C2C	3	43.3 ± 36.6	12.7 ± 6.7
	Task-only	3	12.7 ± 0.6	13.0 ± 0.0
	CACHET w/o MR	3	71.3 ± 5.8	10.3 ± 1.5

Table B.7: Eight-option control accuracy (%). C2C uses three seeds per pair. Full CACHET and DLC-SI results appear in Table B.6.

The mean and residual errors use the same block denominator:

$$e_{\mu,\ell u} = \frac{\sum_{h,p,c} n_p (\mu^Z_{hpc} - \mu^T_{hpc})^2}{D_{\ell u}}, \tag{C.3}$$

$$e_{r,\ell u} = \frac{\sum_{i,h,p,c} w_{ip} [(Z - \mu^Z) - (T - \mu^T)]^2_{ihpc}}{D_{\ell u}}. \tag{C.4}$$

The n_p weighting makes their sum equal masked reconstruction error divided by $D_{\ell u}$. Layer averages give $E_{\mu,u}$ and $E_{r,u}$. CACHET w/o MR uses

$$\mathcal{L}^{\text{w/o MR}}_{\text{align}} = \tfrac{1}{2} \sum_{u \in \{K,V\}} (E_{\mu,u} + E_{r,u}). \tag{C.5}$$

This is block-relative reconstruction; DLC-SI uses unnormalized error.

Fixed normalization constants. The first 2,000 training examples, in file order and batches of four, set target-only constants. With $Z = 0$, fp64 layer/batch averages give $v_u = \text{mean}_b\, E^{(b)}_{r,u}$, $m = \text{mean}_b(E^{(b)}_{\mu,K} + E^{(b)}_{\mu,V})/2$, $v = (v_K + v_V)/2$, and $P_0 = m + v$. These constants are independent of translator width and fixed through both stages. Define

$$d_u = \max(v_u, 0.01v), \qquad d_m = \max(m, 0.01(m + v)), \tag{C.6}$$

$$q = \tfrac{1}{2}(v_K/d_K + v_V/d_V), \qquad \kappa = \frac{P_0}{m/d_m + \lambda_{\text{res}}}, \tag{C.7}$$

$$M = \frac{E_{\mu,K} + E_{\mu,V}}{2d_m}, \qquad R_{KV} = \frac{E_{r,K}/d_K + E_{r,V}/d_V}{2q}, \tag{C.8}$$

$$R_G = \frac{E_{r,K} + E_{r,V}}{2v}, \qquad \mathcal{L}^{\text{MR}}_{\text{align}} = \kappa(M + \lambda_{\text{res}} R_{KV}). \tag{C.9}$$

CACHET (MR & KV12) uses $\lambda_{\text{res}} = 12$ for every model pair. This fixed hyperparameter is separate from the target-derived normalization statistics. The multiplier κ sets the target-only reference loss to P_0. MR & KV1 uses weight one and its corresponding κ, giving $\mathcal{L}^{\text{KV1}}_{\text{align}}$. MR & GLOBAL12 replaces R_{KV} with R_G at weight twelve, giving $\mathcal{L}^{\text{GLOBAL12}}_{\text{align}}$. With inactive floors and $P_0 = 1$, $q = 1$ and $\kappa = 1/(1 + \lambda_{\text{res}})$.

Receiver	v_K	v_V	m
0.6B	0.124244	0.476519	0.699618
1.7B	0.1261	0.4824	0.6957
4B	0.1736	0.4758	0.6753

Table C.1: Receiver-native normalization statistics. The 1.7B and 4B entries are rounded for display; training uses the saved constants.

Training centers the current single-GPU minibatch (normally sixteen examples). Detached bf16 targets are cast to fp32; gradients pass through Z and μ^Z. Empty positions contribute zero; singleton positions retain only mean error. Block denominators update each step. Inference requires neither batch statistics nor alignment loss.

Pairwise interpretation of residual matching. For a valid coordinate c, centering each cache by its batch mean preserves pairwise differences. The residual term in Equation (6) can therefore be written as

$$\frac{1}{|\mathcal{V}_c|}\sum_{i\in\mathcal{V}_c}(\widehat{E}_{i,c}-E_{i,c})^2 = \frac{1}{2|\mathcal{V}_c|^2}\sum_{i,j\in\mathcal{V}_c}\left[(\widehat{T}_{i,c}-\widehat{T}_{j,c})-(T_{i,c}-T_{j,c})\right]^2.$$

The identity follows because the residual errors have zero mean.

Reference weighting of the variants. At the same target-only reference used in Section 3.2, MR&KV1 assigns 50%, 25%, and 25% of the loss to the mean, key-residual, and value-residual terms; MR&GLOBAL12 assigns 8%, 19%, and 73%, respectively. The constants above give $\kappa = 1/2$ for MR&KV1 when the floors are inactive and $P_0 = 1$. These percentages describe the reference evaluation, not fixed loss proportions during training.

Batch support for residual supervision. The batch-four reference fit determines the fixed scales, while training uses batches of sixteen and recomputes means over valid examples. When $|\mathcal{V}_c| = 1$, both centered residuals are zero: the coordinate receives mean-error supervision but no additional signal from residual upweighting. In the measured batches, singleton coordinates account for 40.6% of active coordinates at batch size four and 21.4% at batch size sixteen.

C.2 Cache replacements

The *source cache* is the sharer's prefill cache supplied to the interface. For CACHET, Matched and Swap use $g_\theta(S_i)$ and $g_\theta(S_{\pi(i)})$ for fixed-point-free π; C2C uses $F(R_i, S_i)$ and $F(R_i, S_{\pi(i)})$ with R_i fixed. Receiver inputs and labels stay fixed for both methods. Under Swap, CACHET uses the swapped cache's length and mask.

For the released-C2C accuracy control, source-input permutations are computed independently within each benchmark's evaluation set. ARC-C, OpenBookQA, and MMLU-Redux use seed 0 and contain 493, 484, and 442 evaluated items, respectively. The C-Eval control uses seed 20260912 on 1,345 items and additionally excludes source inputs from the receiver's subject. The realized mappings contain no fixed points or cross-benchmark replacements, and matched, swapped, and no-fusion conditions use identical evaluation items.

Each C2C source cache is constructed at its source input's positions and then re-indexed onto the receiver's real-token columns. Longer source sequences are truncated to the receiver's columns; shorter ones repeat their last real cache column. The receiver cache and mask are retained, so the fuser output keeps the receiver's cache length.

Input mean (C2C). C2C averages post-RoPE full-chat sharer caches from the first 2,000 raw training rows with plain-chat prompts, using fp64 accumulation. The current receiver cache is retained; current-question sharer prefill is not required.

Output mean (CACHET). CACHET averages translated prefixes from fifty fixed OpenHermes prompts over valid tokens at each position; uncovered positions use the pooled mean for that layer, head, and channel. The bf16 prefixes are averaged in fp32 and stored in bf16. The control retains the evaluated item's mask. These two mean controls use different reference sets and act on different representations.

C.3 Passage tasks and the meaning of swap

Paired counterfactual passage task. The 400 examples form 200 WikiText-103-based pairs with random-digit facts (generator seed 20260912), balanced across four answer letters. Paired receiver questions, token IDs, and options are identical; equal-length passages support different answers. Only the sharer reads the passage; both models receive the question and options. Evaluation reuses same-input checkpoints without additional training, using batch eight and the letter scorer. Payloads are 298–302 tokens under a 2,048-token cap. Swap uses the partner's cache and mask; both conditions are scored against the original answer.

Additional passage retrieval tasks. Variants change options (four/eight), distractors ($d = 0, 4, 8$), or requested values ($k = 1, 2, 3$). The eight-option test has 100 items and 12.5% chance accuracy. Table B.6 reports $d = 8$, four/eight options, and $k = 3$. Distractor effects vary; $k = 2$ has a reported receiver-only option cue. Whole-input swap uses another family's cache and mask at the same level, including a different question. Receiver inputs and gold labels stay fixed. Evaluation covers matched and whole-input swap conditions.

C.4 Cache-geometry measurements

The released-C2C geometry probe independently samples sixteen receiver questions and sixteen source inputs, four per benchmark in each set, using seed 20260914, and crosses every receiver with every source input. These cross-benchmark substitutions are separate from the within-benchmark permutations used for the accuracy control. Matched and fixed-source-mean conditions are evaluated separately.

We record $\Delta_{ij} = F(R_i, S_j) - R_i$ at eight stem positions; each selected stem contains at least eight tokens. For each K/V branch, the recorded updates are concatenated across gate-retained receiver layers, KV heads, these eight positions, and channels, without per-layer rescaling. Let $\mathbf{d}_{ij}$ denote this vector for receiver i and source input j, and $\mathbf{d}_i^{\mathrm{mat}}$ the corresponding matched update. The swapped-to-matched update distance is

$$D_{\mathrm{src}} = \frac{1}{256} \sum_{i=1}^{16} \sum_{j=1}^{16} \frac{\|\mathbf{d}_{ij} - \mathbf{d}_i^{\mathrm{mat}}\|_2}{\max(\|\mathbf{d}_i^{\mathrm{mat}}\|_2, 10^{-12})}. \tag{C.10}$$

The center for receiver i averages its updates over the sixteen source inputs, excluding the separately recorded matched and source-mean conditions. We also report the distance between receiver centers:

$$\mathbf{c}_i = \frac{1}{16} \sum_{j=1}^{16} \mathbf{d}_{ij}, \qquad D_{\mathrm{ctr}} = \frac{1}{120} \sum_{i<k} \frac{\|\mathbf{c}_i - \mathbf{c}_k\|_2}{(\|\mathbf{c}_i\|_2 + \|\mathbf{c}_k\|_2)/2}. \tag{C.11}$$

Norms are unsquared, and ratios are formed per pair before averaging. D_{src} measures source-cache replacements relative to each receiver's matched update; D_{ctr} uses a symmetric normalization of receiver centers. They have different denominators and are reported as separate statistics.

Statistic	Keys	Values
Swapped-to-matched D_{src}	0.331267	0.410198
Between-center D_{ctr}	0.433597	0.947764

Table C.2: Relative distances recomputed from the recorded released-C2C updates. Equations C.10 and C.11 use distinct normalizations.

Native mean-energy fractions use Equation C.3 with zero predictions on 2,000 training inputs, centered in batches of sixteen and averaged over layers separately for K/V. Normalization constants instead use the batch-four fit above.

Figure 2 shows two-dimensional PCA projections of the recorded updates, computed separately for keys and values. The first two principal components explain 17.0% of the key-update variance and 16.1% of the value-update variance.

D Additional Translator and Alignment Diagnostics

We report the Separated translator configuration, DLC+MR comparison, mean–residual reconstruction diagnostic, and three-seed passage-transfer controls, complementing the source-access ablation in Table 4.

Separated translator implementation. We replace the joint trunk with 16 independent head/branch groups: each receiver key or value reads only the same-index source head and the same branch. Hidden transformations, output projections, and LayerNorm remain within each group; layer mapping, RoPE handling, and prefix insertion follow CACHET. Each group has hidden width 2582, giving 260,161,792 trainable parameters versus 260,116,480 for the joint translator at width 4096; the two variants are parameter-matched within 0.02%. Both variants use the same learning-rate schedule,

without retuning. This ablation removes cross-head and cross-branch access together and does not separate their individual effects.

	4B→0.6B		8B→1.7B		14B→0.6B	
Method	4-bench	Pro	4-bench	Pro	4-bench	Pro
DLC-SI	73.21	40.33	21.09	10.99	73.46	41.57
DLC-SI + align.	74.32	38.79	52.50	10.43	72.39	39.77
DLC+MR	72.55	35.71	47.60	22.16	81.07	44.70
CACHET w/o MR	83.84	49.47	50.72	33.15	89.05	54.17
CACHET	87.80	61.33	56.47	37.43	88.99	55.30

Table D.1: PGR (%), three-seed means from the completed DLC+MR comparison. Pro denotes the MMLU-Pro aggregate. DLC-SI + align. retains its reconstruction objective during task training: its Phase 2 loss is $\mathcal{L}_{\text{rec}}/R_0 + 0.3\mathcal{L}_{\text{task}}$, where $\mathcal{L}_{\text{rec}}$ is DLC-SI's Phase 1 reconstruction loss and R_0 its value at the end of Phase 1 for the same seed, so that the two terms have comparable scale; DLC+MR uses MR in both stages. These are three-seed comparisons, separate from the seven-seed main results.

Applying MR to DLC-SI does not close the gap to CACHET on either aggregate for any of the three model pairs (Table D.1). Relative to DLC-SI + align., MR improves both aggregates on 14B→0.6B; on 8B→1.7B, it raises MMLU-Pro PGR from 10.43 to 22.16 while lowering four-benchmark PGR from 52.50 to 47.60.

For the reconstruction diagnostic, prediction and target are decomposed into their own batch means and centered residuals. The four error terms use the common plain alignment definition in Equation 7; the total is $\frac{1}{2}(E_{\mu,K} + E_{r,K} + E_{\mu,V} + E_{r,V})$. Comparisons use receiver post-RoPE key coordinates and receiver value coordinates, with padding masked. The target is the native cache for the payload at prefix positions, not the receiver's subsequent question cache.

Method	Stage	$E_{\mu,K}$	$E_{r,K}$	$E_{\mu,V}$	$E_{r,V}$	Plain
DLC-SI	1	0.0204	0.0843	0.0981	0.4468	0.3248
	2	0.3389	0.1240	0.1926	0.5168	0.5862
DLC-SI + align.	2	0.0237	0.0880	0.1404	0.4931	0.3726
DLC+MR	1	0.0345	0.0828	0.1062	0.4397	0.3316
	2	0.0777	0.0836	0.1369	0.4448	0.3715
CACHET w/o MR	1	0.0485	0.0861	0.0740	0.3408	0.2748
	2	0.0556	0.0948	0.0822	0.3537	0.2932
CACHET	1	0.0679	0.0716	0.0720	0.3080	0.2597
	2	0.1102	0.0732	0.1259	0.3146	0.3120

Table D.2: Receiver-native reconstruction errors on 4B→0.6B at the ends of Stages 1 and 2. Lower is better for each error measure. Values are rounded separately, so sums may differ in the last digit. These are alignment diagnostics, not direct measurements of task-relevant information transfer.

Within CACHET, MR lowers both Stage 2 residual errors relative to plain alignment while increasing the mean errors. Consequently, the MR checkpoint has higher total plain error despite better residual reconstruction. Across architectures, CACHET w/o MR has lower value-residual error but higher key-residual error than DLC+MR. The diagnostic therefore does not support attributing all performance differences to a single aggregate reconstruction error.

Eight-option passage controls. Table D.3 reports the three-seed comparison on 4B→0.6B, with 100 evaluation examples per seed. Separated reaches 52.00% matched accuracy and 13.33% after replacing the source cache with an unrelated input's cache; chance accuracy is 12.5%. The joint translator reaches 91.33% matched accuracy under this three-seed comparison.

Method	Matched	Swap
Separated	52.00 ± 9.00	13.33 ± 2.08
CACHET	91.33 ± 10.26	8.33 ± 0.58
DLC-SI	65.33 ± 5.77	12.33 ± 3.06
C2C	46.33 ± 29.91	13.67 ± 2.89

Table D.3: Eight-option passage accuracy (%) for 4B→0.6B, mean ± sample SD over three seeds. Swap replaces the entire source cache with one from an unrelated input and scores against the original target answer. CACHET and DLC-SI results here use three seeds, whereas their main results in Table 3 use seven.