Can internal representations of different LLMs be shared or mapped across models?

Jason Zuo, Ziyang Huang, Yingfei Xu, Hanxiang Qin

JOHNS HOPKINS
UNIVERSITY

Cache-to-Cache: Direct Semantic Communication Between Large Language Models

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, Yu Wang

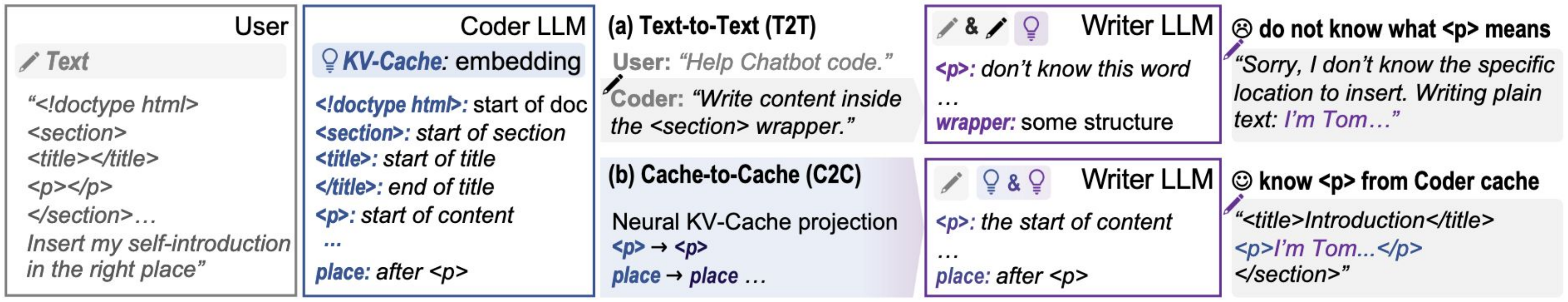
Background

- **Why Multiple LLMs?** Leveraging complementary strengths: Different LLMs excel in different domains (code, math, vision, etc.), so a team of specialized models can outperform any single model. Multi-LLM systems achieve performance and efficiency gains unattainable by a single model.
- **Current Communication Practice:** In today's multi-LLM pipelines, models collaborate by exchanging text messages (natural language). One model's output text is fed as input prompt to the next, mimicking human collaboration.
- **Drawbacks of Text-Only Exchange:** Both loses rich semantic information and incurs token-by-token generation latency

Challenges of Text-Based LLM Communication (Motivation)

- **Information Bottleneck:** Text is a low-bandwidth channel. One LLM’s high-dimensional internal representations have to be compressed into a linear sequence of tokens, then decompressed by the receiver. Rich semantic nuances get lost in this translation. If models have different knowledge or roles, some signals are irrecoverably dropped in text form.
- **Ambiguity**: Natural language is inherently ambiguous and underspecified. Idioms, vague references, or formatting (e.g. a token “<p>”) might confuse the receiver. Even structured protocols can’t eliminate all ambiguity in open-domain collaboration.
- **Latency:** Each message must be generated token-by-token by one model and then processed by the other. In multi-turn interactions, this sequential generation dramatically slows down overall inference.

🤔 Can LLMs communicate beyond text?



“<!doctype html>
<section>
<title></title>
<p></p>
</section>…
Insert my self-introduction in the right place”

<!doctype html>: start of doc
<section>: start of section
<title>: start of title
</title>: end of title
<p>: start of content
…
place: after <p>

User: “Help Chatbot code.”
Coder: “Write content inside the <section> wrapper.”

Neural KV-Cache projection
<p> → <p>
place → place …

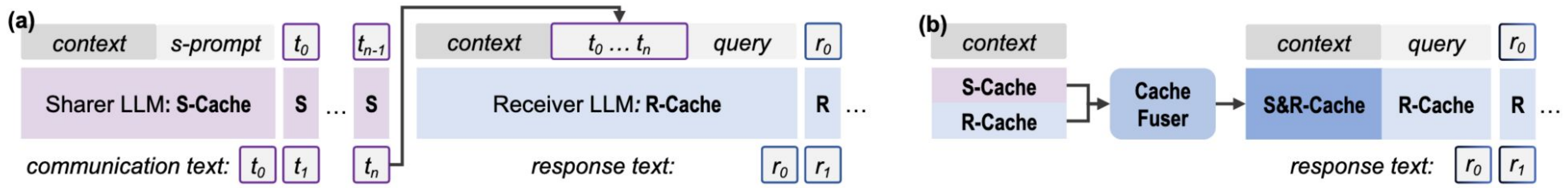
<p>: don’t know this word
…
wrapper: some structure

<p>: the start of content
…
place: after <p>

“Sorry, I don’t know the specific location to insert. Writing plain text: I’m Tom…”

“<title>Introduction</title>
<p>I’m Tom...</p>
</section>”

Why Go Beyond Text?



- Illustration of text vs. cache-based communication:
  - (a) In standard Text-to-Text (T2T) collaboration, a Sharer LLM must output a verbal message (purple tokens) that the Receiver LLM reads as input — losing some internal context in the process.
  - (b) In Cache-to-Cache (C2C), the Sharer's internal KV-cache (its hidden state representations) is directly injected into the Receiver via a learned fuser, transferring semantics without intermediate text. C2C thus bypasses natural language, conveying precise structural information that text might miss.

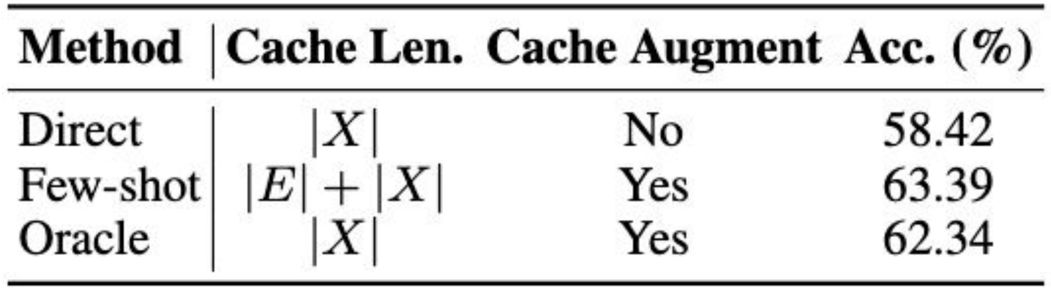
Oracles Experiments – Can Caches Be Shared?

Oracle 1 – Cache Enrichment:

Can a model's capabilities be improved through KV-Cache semantic enrichment without extending sequence length?

- (1) *Direct* – prefill on X only and decode with C(X)
- (2) *Few-shot* – prefill on E⊕X and decode with C(E⊕X) (longer cache)
- (3) *Oracle* – prefill on E⊕X but uses only the question portion of the cache

  The Oracle setup improved accuracy even though the final cache length was unchanged, proving that a cache enriched with semantic content yields better responses. In other words, it's not just more tokens that help – it's the semantic richness in the cache

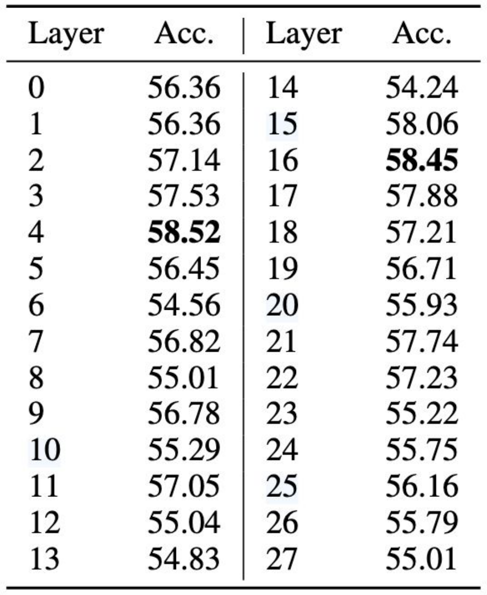


Method	**Cache Len.**	**Cache Augment**	**Acc. (%)**
Direct	$\lvert X\rvert$	No	58.42
Few-shot	$\lvert E\rvert + \lvert X\rvert$	Yes	63.39
Oracle	$\lvert X\rvert$	Yes	62.34

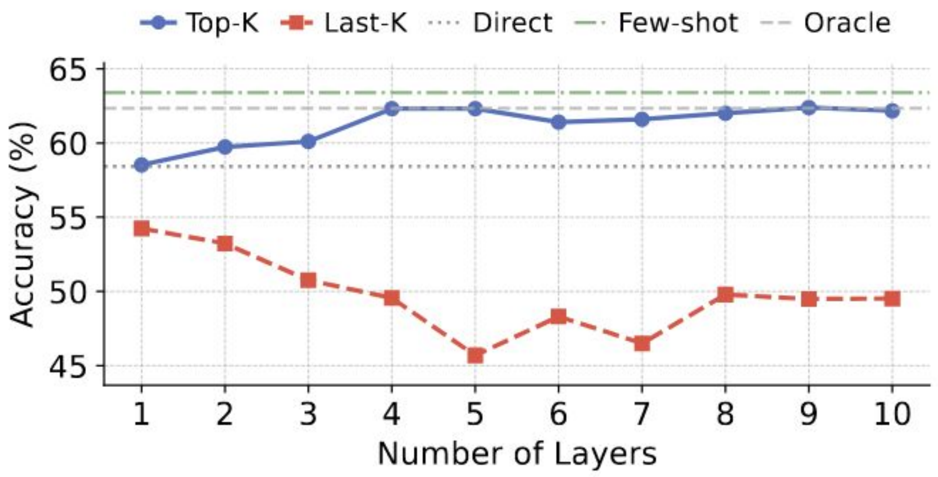
Oracles Experiments – Can Caches Be Shared?

Layer-wise Effects:

- Not all transformer layers benefited equally from the enriched cache. Enriching certain layers gave big accuracy gains while others actually hurt performance. Augmenting only the top 10 best-performing layers beat augmenting all layers, whereas augmenting the worst layers caused accuracy to drop.
- This insight directly informed C2C's design by introducing **gating mechanism** to select which layers to fuse.



Layer	Acc.	Layer	Acc.
0	56.36	14	54.24
1	56.36	15	58.06
2	57.14	16	**58.45**
3	57.53	17	57.88
4	**58.52**	18	57.21
5	56.45	19	56.71
6	54.56	20	55.93
7	56.82	21	57.74
8	55.01	22	57.23
9	56.78	23	55.22
10	55.29	24	55.75
11	57.05	25	56.16
12	55.04	26	55.79
13	54.83	27	55.01

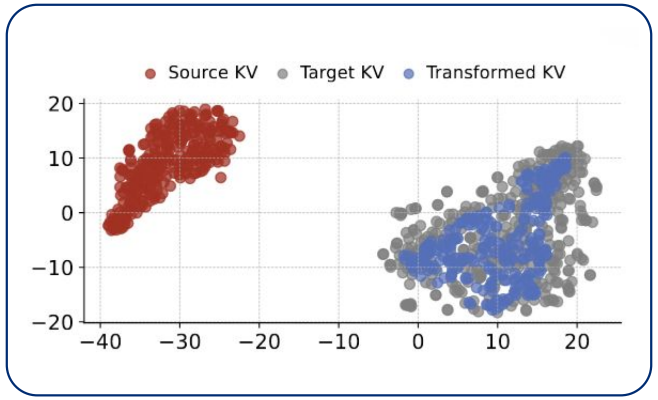


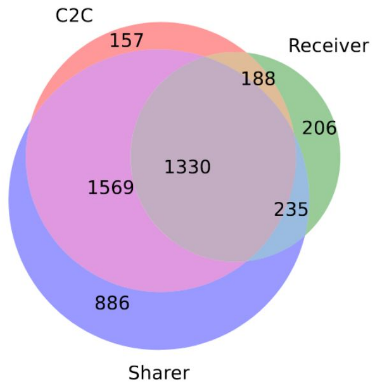
Oracles Experiments – Can Caches Be Shared?

Oracle 2 – Cache Transformation:

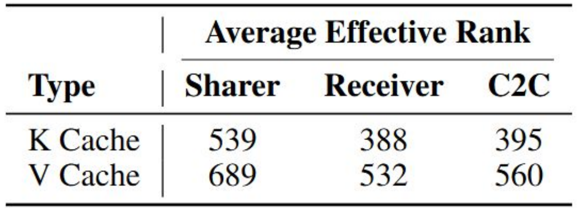
Can the KV-Cache of one model be effectively utilized by another model?

- Trained a small 3-layer MLP to map KV (Qwen3-4B) into (Qwen3-0.6B).
- t-SNE:





(b) Sharer: Q3-4B, Receiver: Q3-0.6B

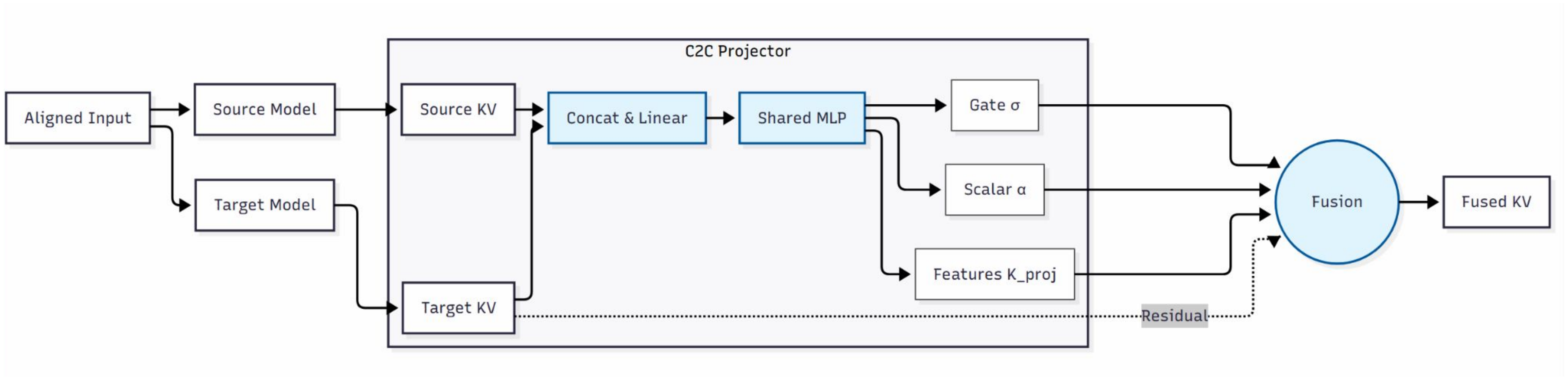


	Average Effective Rank		
Type	**Sharer**	**Receiver**	**C2C**
K Cache	539	388	395
V Cache	689	532	560

Table 2: Average effective rank of KV-Cache from Sharer, Receiver, and the C2C fused one.

- Complementary but Not Identical:
  - Transformed cache cannot fully span the target's representational space.
  - Each model encodes some unique aspects of the context that the other doesn't.
- **Hypothesis:** If we can successfully fuse two models' caches, we might leverage their complementary knowledge for better results

C2C Architecture & Training



- The Sharer and Receiver LLMs themselves are **kept frozen** during training
- Only the small C2C fuser network is trained.
- The training objective: Next-token prediction on the Receiver's output, given fused caches.
- In practice, they take a supervised dataset of question→answer pairs. For each training example:
  - (1) run both models on the question to get their caches;
  - (2) fuse the caches and feed into the Receiver model to generate an answer;
  - (3) compute loss against the ground-truth answer and backpropagate through the fuser (the LLM weights don't change).

Models & Alignment

Models:

- cross-generation (e.g. Qwen3 vs Qwen2.5 series)
- cross-family (e.g. Qwen vs LLaMA vs Gemma LLMs)
- cross-size (from 0.6B up to 14B parameters)
- different specializations（general-purpose, code, math models)
- different training stages (pretrained and instruction fine-tuned models)

Token Alignment:

- Different LLMs might tokenize the same text differently.
- C2C must ensure the Sharer's cache lines up with the Receiver's token positions.
- for each token in the Receiver's input, they decode it to text and then re-encode it with the Sharer's tokenizer.
- **Max coverage**: If a single Receiver token maps to multiple Sharer tokens, they choose the Sharer token that covers the largest portion of the text.

Layer Alignment:

- Models may have different numbers of layers or architectures.
- "**terminal layer alignment**" strategy:
  - They align the top (final output) layers of Sharer and Receiver, then the next-to-last layers, and so on, pairing layers from the top down until they run out of layers on the smaller model.
  - This means the deepest layers of both models are fused first, and shallower layers of the larger model may not have a counterpart (if model sizes differ).

Experiments

Baselines:

- *Text-to-Text (T2T) Communication:*
  - Receiver (question, sharer_analysis) -> reply
- *Query-Level Routing(Ong et al., 2024):*
  - Route query to model based on query difficulty.

Benchmark Tasks:

- *OpenBookQA* – science fact and common sense questions (open-domain reasoning)
- *MMLU-Redux* – a challenging academic and common knowledge test covering various subjects
- *ARC-Challenge* – grade-school science exam questions that require complex reasoning
- *C-Eval* – a comprehensive Chinese academic exam dataset, to test non-English and multi-domain knowledge

Evaluation Metrics:

- Average accuracy, inference latency (per-question runtim on a single NVIDIA A100 GPU)
- All models were run in zero-shot mode with deterministic decoding (temperature 0) for consistent comparison

Training dataset: first 500k samples of OpenHermes-2.5 Dataset

Results

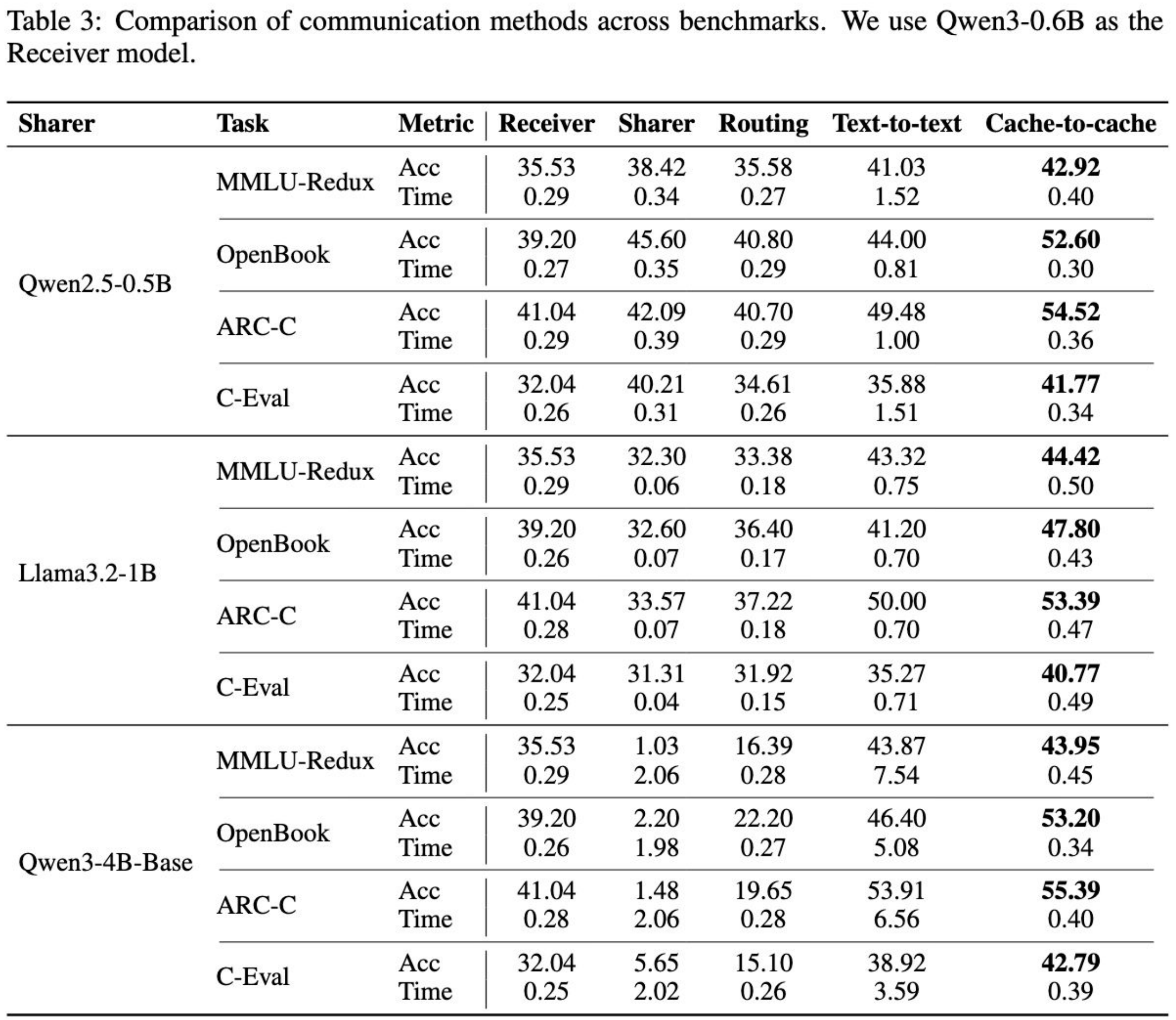


Table 3: Comparison of communication methods across benchmarks. We use Qwen3-0.6B as the Receiver model.

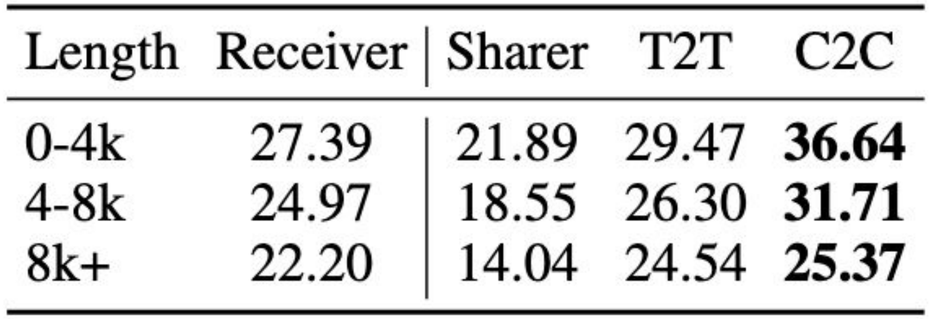
Sharer	Task	Metric	Receiver	Sharer	Routing	Text-to-text	Cache-to-cache
Qwen2.5-0.5B	MMLU-Redux	Acc	35.53	38.42	35.58	41.03	**42.92**
		Time	0.29	0.34	0.27	1.52	0.40
	OpenBook	Acc	39.20	45.60	40.80	44.00	**52.60**
		Time	0.27	0.35	0.29	0.81	0.30
	ARC-C	Acc	41.04	42.09	40.70	49.48	**54.52**
		Time	0.29	0.39	0.29	1.00	0.36
	C-Eval	Acc	32.04	40.21	34.61	35.88	**41.77**
		Time	0.26	0.31	0.26	1.51	0.34
Llama3.2-1B	MMLU-Redux	Acc	35.53	32.30	33.38	43.32	**44.42**
		Time	0.29	0.06	0.18	0.75	0.50
	OpenBook	Acc	39.20	32.60	36.40	41.20	**47.80**
		Time	0.26	0.07	0.17	0.70	0.43
	ARC-C	Acc	41.04	33.57	37.22	50.00	**53.39**
		Time	0.28	0.07	0.18	0.70	0.47
	C-Eval	Acc	32.04	31.31	31.92	35.27	**40.77**
		Time	0.25	0.04	0.15	0.71	0.49
Qwen3-4B-Base	MMLU-Redux	Acc	35.53	1.03	16.39	43.87	**43.95**
		Time	0.29	2.06	0.28	7.54	0.45
	OpenBook	Acc	39.20	2.20	22.20	46.40	**53.20**
		Time	0.26	1.98	0.27	5.08	0.34
	ARC-C	Acc	41.04	1.48	19.65	53.91	**55.39**
		Time	0.28	2.06	0.28	6.56	0.40
	C-Eval	Acc	32.04	5.65	15.10	38.92	**42.79**
		Time	0.25	2.02	0.26	3.59	0.39

- After applying C2C, we see an average increase of accuracy around ~10% across three different Sharers.
- Compared with T2T communication, C2C achieves an average accuracy increase ~4%.
- It also achieves obvious speedups from 3.46× to 14.41×, thanks to the waiving of intermediate text message generation.
- In contrast, query-level routing prioritizes efficiency but limits accuracy to the better of the two original models.

Concern: the accuracy for the 4B Base seemed to be too low…

Results -- Scaling with Sequence Length

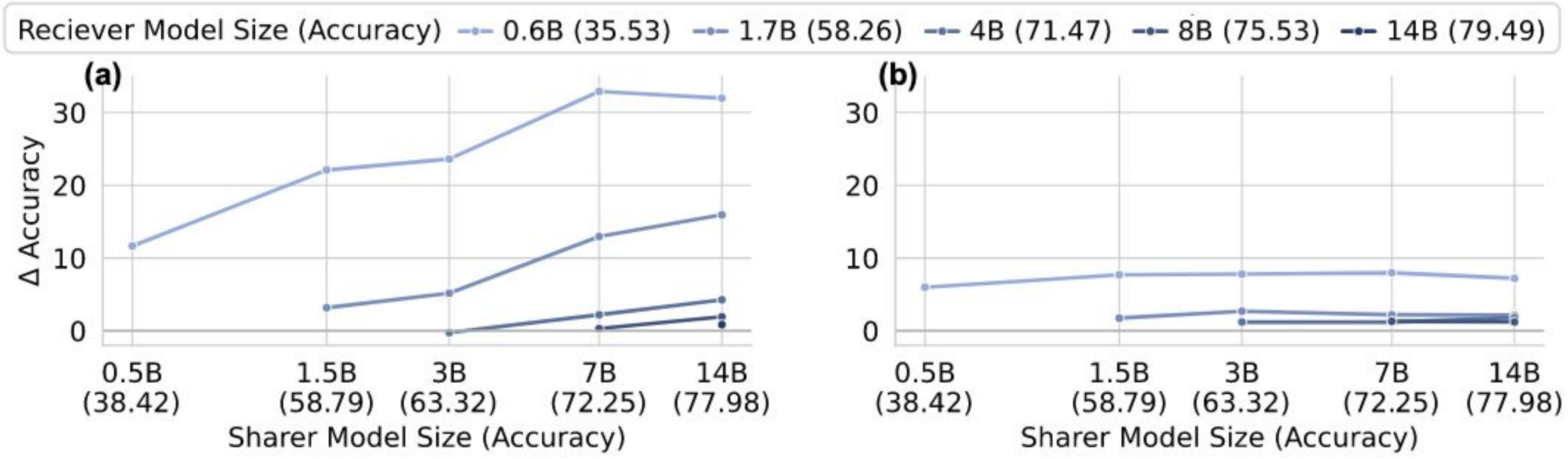
- On LongBench (documents up to 8k+ tokens)
  - C2C outperformed text-based communication consistently across short, medium, and long context lengths Notably, the advantages of C2C held even as input length grew, indicating it scales well with context size
  - Direct cache fusion does not suffer from the compounding token-by-token cost that text methods do for long prompts



Length	Receiver	Sharer	T2T	C2C
0-4k	27.39	21.89	29.47	**36.64**
4-8k	24.97	18.55	26.30	**31.71**
8k+	22.20	14.04	24.54	**25.37**

Results -- Scaling with Model Size

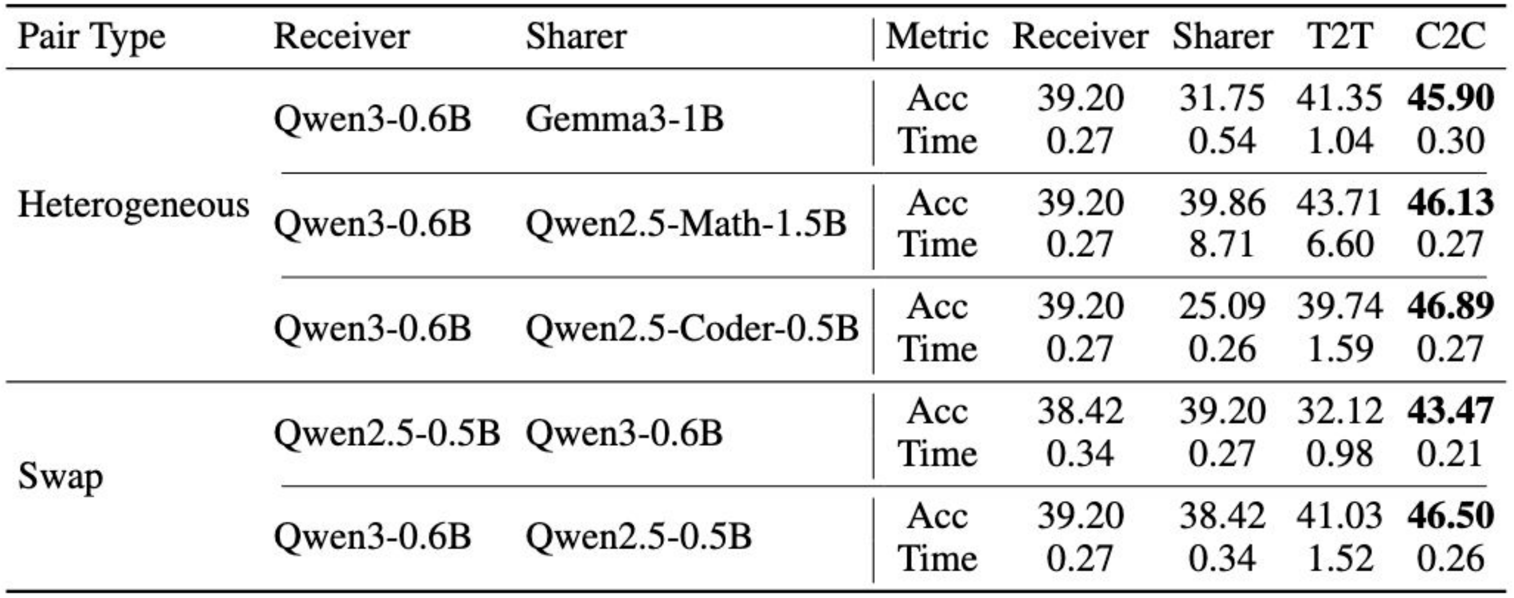
- Using a bigger Sharer generally yielded bigger accuracy gains for the Receiver (provided the Sharer had relevant knowledge).
- Even a relatively small Receiver (e.g. 0.6B) can boost a lot when paired with a much larger Sharer.
    - Going beyond its 'small brain' by making use of the larger model's cache.
    - An potential use-case:
        - A small model could handle easy parts of a task but call upon a large model's semantics when needed, via cache transfer, instead of always running the large model fully.



C2C T2T

Results: Generality of Cache-to-Cache

- **Across Model Families:**
  - C2C's paradigm proved effective across heterogeneous models.
  - Despite differences in architecture and pretraining, as long as token and layer alignment was done, the cache fuser could learn to bridge them.
- **Commutativity --** C2C works in **either direction**
  - One can swap the roles of Sharer and Receiver and still see gains (depending on the task)
  - The decision of which model should be "Sharer" can be flexible or even dynamic.
- **Potential**:
  - Two LLMs could mutually share caches in a multi-turn process.



Pair Type	Receiver	Sharer	Metric	Receiver	Sharer	T2T	C2C
Heterogeneous	Qwen3-0.6B	Gemma3-1B	Acc	39.20	31.75	41.35	**45.90**
			Time	0.27	0.54	1.04	0.30
	Qwen3-0.6B	Qwen2.5-Math-1.5B	Acc	39.20	39.86	43.71	**46.13**
			Time	0.27	8.71	6.60	0.27
	Qwen3-0.6B	Qwen2.5-Coder-0.5B	Acc	39.20	25.09	39.74	**46.89**
			Time	0.27	0.26	1.59	0.27
Swap	Qwen2.5-0.5B	Qwen3-0.6B	Acc	38.42	39.20	32.12	**43.47**
			Time	0.34	0.27	0.98	0.21
	Qwen3-0.6B	Qwen2.5-0.5B	Acc	39.20	38.42	41.03	**46.50**
			Time	0.27	0.34	1.52	0.26

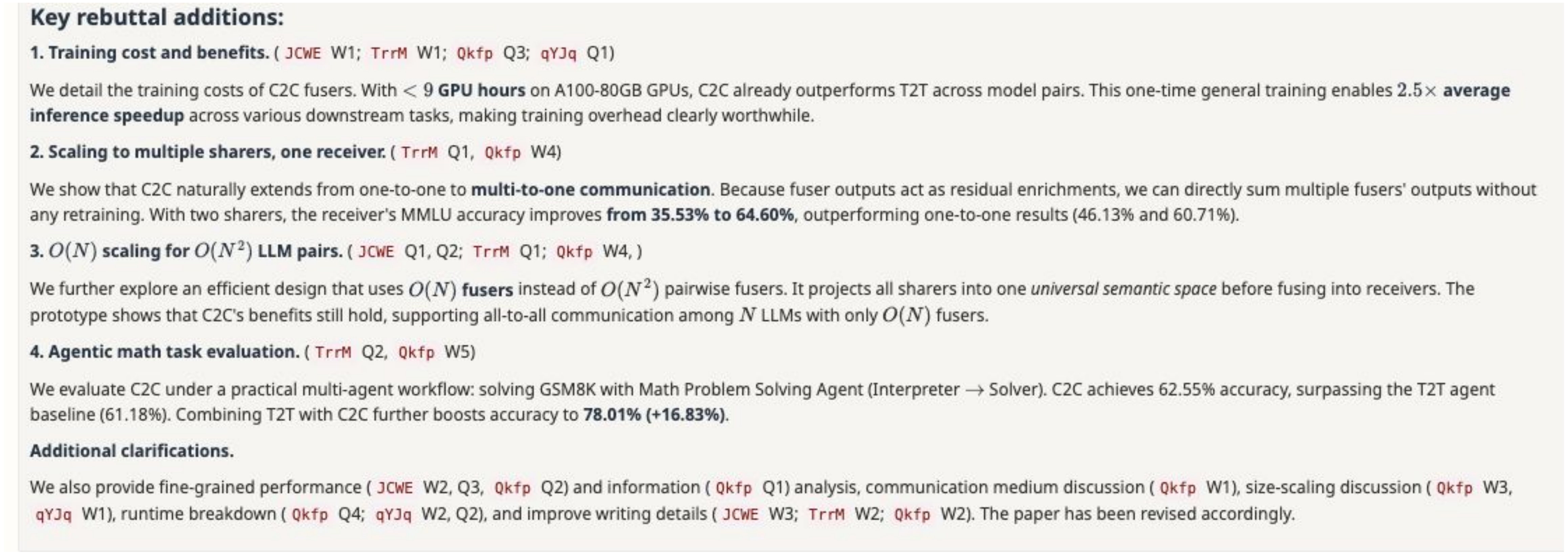
Takeaways

- Introduced a novel paradigm C2C demonstrating that LLMs can communicate directly via their internal representations rather than through textual outputs
- Consistent speedups and accuracy
- future work:
  - Privacy-aware cloud–edge collaboration
  - Multimodal integration

Discussions:

- They experimented with cache enrichment, but their design almost has nothing to do with that.
  - Why don't they use attention to fuse the sharer's KV into the receiver's KV?
- Does their effectiveness truly come from the diversity of the sharer's KV?
  - Or is it because of the training of extra parameters.
  - Should have an experiment of self-transfer as an ablation to understand what percent of improvement is coming from training.
- Need to be tested in multi-agent dialogue to see whether or not this C2C transfer is stable enough.

Takeaways



Key rebuttal additions:

1. Training cost and benefits. (`JCWE` W1; `TrrM` W1; `Qkfp` Q3; `qYJq` Q1)

We detail the training costs of C2C fusers. With < 9 **GPU hours** on A100-80GB GPUs, C2C already outperforms T2T across model pairs. This one-time general training enables $2.5\times$ **average inference speedup** across various downstream tasks, making training overhead clearly worthwhile.

2. Scaling to multiple sharers, one receiver. (`TrrM` Q1, `Qkfp` W4)

We show that C2C naturally extends from one-to-one to **multi-to-one communication**. Because fuser outputs act as residual enrichments, we can directly sum multiple fusers' outputs without any retraining. With two sharers, the receiver's MMLU accuracy improves **from 35.53% to 64.60%**, outperforming one-to-one results (46.13% and 60.71%).

3. $O(N)$ scaling for $O(N^2)$ LLM pairs. (`JCWE` Q1, Q2; `TrrM` Q1; `Qkfp` W4,)

We further explore an efficient design that uses $O(N)$ **fusers** instead of $O(N^2)$ pairwise fusers. It projects all sharers into one *universal semantic space* before fusing into receivers. The prototype shows that C2C's benefits still hold, supporting all-to-all communication among N LLMs with only $O(N)$ fusers.

4. Agentic math task evaluation. (`TrrM` Q2, `Qkfp` W5)

We evaluate C2C under a practical multi-agent workflow: solving GSM8K with Math Problem Solving Agent (Interpreter $\rightarrow$ Solver). C2C achieves 62.55% accuracy, surpassing the T2T agent baseline (61.18%). Combining T2T with C2C further boosts accuracy to **78.01% (+16.83%)**.

Additional clarifications.

We also provide fine-grained performance (`JCWE` W2, Q3, `Qkfp` Q2) and information (`Qkfp` Q1) analysis, communication medium discussion (`Qkfp` W1), size-scaling discussion (`Qkfp` W3, `qYJq` W1), runtime breakdown (`Qkfp` Q4; `qYJq` W2, Q2), and improve writing details (`JCWE` W3; `TrrM` W2; `Qkfp` W2). The paper has been revised accordingly.

Linear Representation Transferability Hypothesis: Leveraging Small Models to Steer Large Models

Femi Bello, Anubrata Das, Fanzhi Zeng, Fangcong Yin, Liu Leqi

“

When two LLMs share tokenizer and training data, their hidden representations are related by a simple affine map.

We can use this to transfer steering vectors from a small model to a large one

Why care about representation similarity?

- Models with similar architectures/data
  - Do they learn similar internal representations?
- Can we use small models to cheaply understand or control large models?
- Interpretability?

Model Steering as a testbed

- Steering: add a vector to hidden states $\rightarrow$ change behavior
  - e.g., more/less refusal, talk about dogs, speak French.
- Contrastive Activation Addition (CAA): difference of average activations on positive vs negative datasets $\rightarrow$ steering vector.
- Idea: if representation spaces of two models are linearly related, we should be able to learn steering on a 2B model and transfer to 9B.

Linear Representation Transferability Hypothesis

- Is there an **affine map** between hidden states of small and large models that preserves the semantics of steering directions?

- Define an affine map:

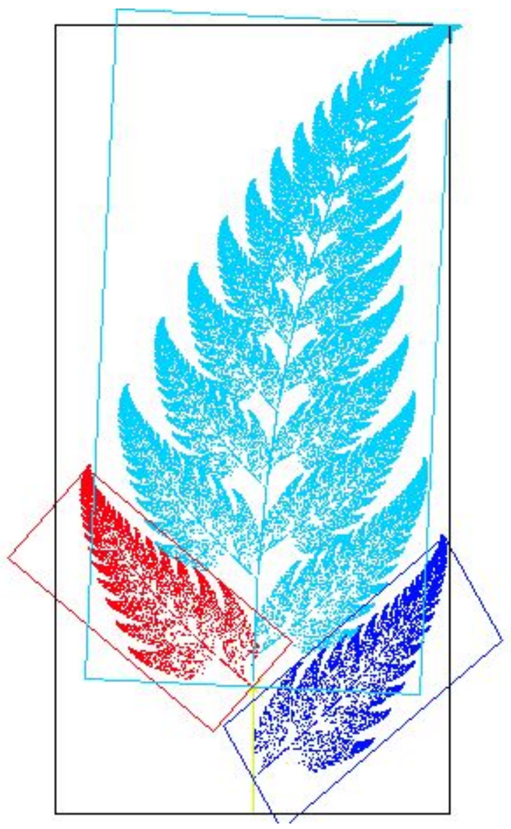
$$A : \mathbb{R}^{d_S} \rightarrow \mathbb{R}^{d_T}$$

"Affine" = linear + bias

- with bias *p* such that:

$$h^T_{l_T}(x) \approx A(h^S_{l_S}(x)) + p$$

"Approximate" – they don't claim perfect equality.

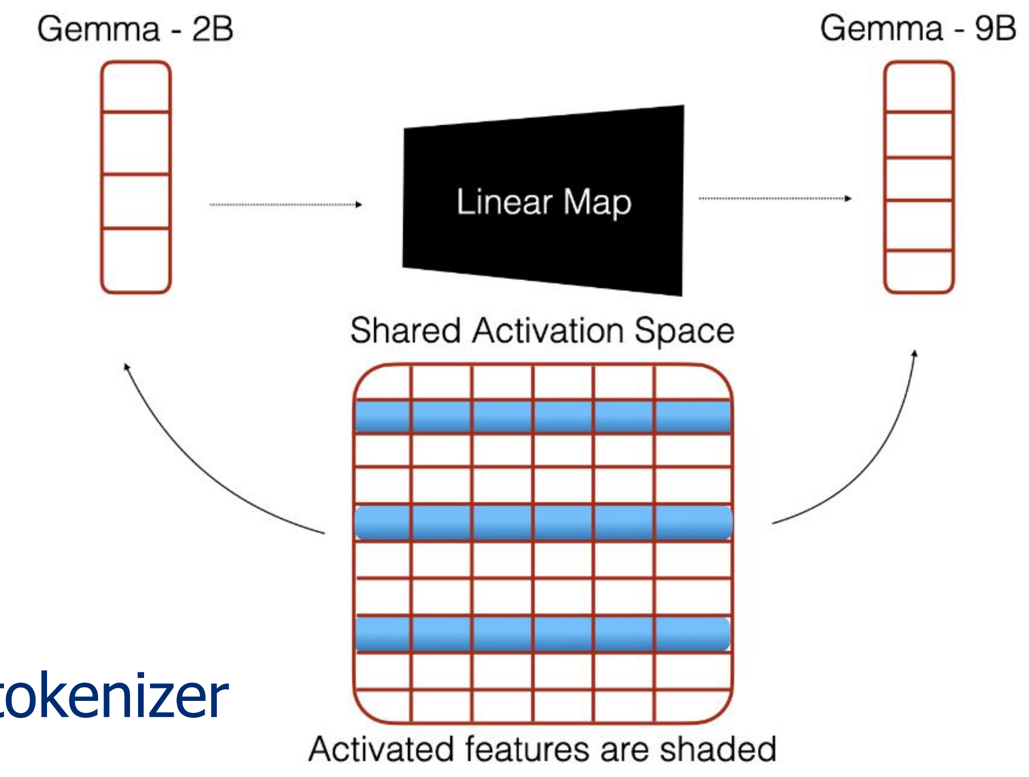


Affinity: a transformation that preserves lines and parallelism, but not necessarily Euclidean distances and angles

A Universal Representation Space?

Conceptually :

- There exists a universal feature space with basis vectors V_U
- Each model's hidden states are linear combinations of shared features, but projected into different subspaces.



Intuitively :

- Both models see the same text distribution and tokenizer
- They must learn syntax, basic semantics, etc
  - suggests shared "universal" features

Formalization: features and projections

- Suppose we have a universal feature matrix

$$W_U \in \mathbb{R}^{n \times D}$$

- For a model (either source or target), have hidden state

$$h(x) = W^\top c(x) + b$$

  - *c(x)* is the sparse coefficients (which features fires)
  - *W* is model-specific feature matrix derived from W_U
  - *b* is bias
- Different models have feature matrices that are linear transforms of the universal feature matrix

A Linear Map?

- Assume:
  - Both models activate the same universal features with the same coefficients c(x)
  - Their feature matrices are linear transforms of shared basis
- We have:

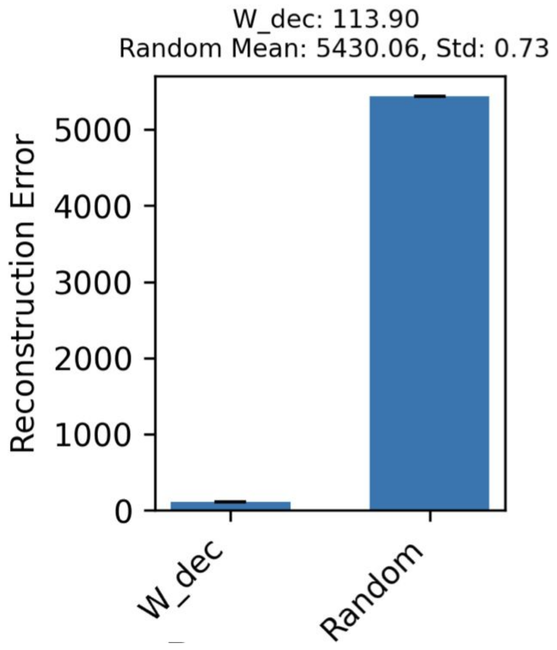
$$h^T_{l_T}(x) = A h^S_{l_S}(x) + p$$

$$A = P_T^\top (P_S^\top)^\dagger$$
$$p = b^T_{l_T} - A b^S_{l_S}$$

- If both models are just different projections of the same underlying feature space, then a linear map must exist between their hidden states.

Empirical Support

- They use Gemma-2B and Gemma-9B sparse autoencoders (SAE
- Decoder matrices of SAEs are interpreted as feature vectors
- Experiment:
  - Least-squares: $M_c = \arg\min_M \|W_T - W_S M\|_2$
  - Projects: $W_T^c = W_S M_c$
  - Measure reconstruction error: $\|W_T^c - W_T\|_F$
- The reconstruction error is small compared to random, suggesting the 2B and 9B feature spaces are close to linearly related



Training

- Training:

$$\min_{A,p} \mathbb{E}_{x\sim D} \| A h^S_{l_S}(x) + p - h^T_{l_T}(x) \|_2^2$$

- Dataset: The Pile, LMSYS-chat-1M
- Choose the layer of similar relative depth (layer 20 in Gemma 2B and 9B)

Application / Testbed / Downstream Task

Steering Large Models with Small Models

Steps:

- Train the linear mapping between hidden states (or another variation: coefficient vectors to hidden states)
- Identify steering vectors in the smaller model
- Map them to the larger model using the learned linear mapping
- Evaluate the steering behavior

Find steering vectors: Contrastive Activation Addition (CAA)

- Given two datasets $(\mathcal{D}_p, \mathcal{D}_n)$ of positive and negative prompts, we find the steering vector for that behavior on layer ℓ via

$$\boldsymbol{v}_l = \frac{1}{|\mathcal{D}_p|} \sum_{\boldsymbol{x} \in \mathcal{D}_p} \boldsymbol{h}_l(\boldsymbol{x}) - \frac{1}{|\mathcal{D}_n|} \sum_{\boldsymbol{x}' \in \mathcal{D}_n} \boldsymbol{h}_l(\boldsymbol{x}').$$

$\boldsymbol{h}_l(\boldsymbol{x})$ denotes the activation of the model in layer ℓ at the last token of input $\boldsymbol{x}$.

Steer the Model

Remember we have the affine map:

$$h_{l_T}^{T}(x) \approx A(h_{l_S}^{S}(x)) + p$$

- Map to another model

$$\tilde{\boldsymbol{v}}_{l_T}^{T} = \boldsymbol{A}\boldsymbol{v}_{l_S}^{S} + \mathbf{p}$$

- Intervention at all token positions
  - Apply a Normal Steering Vector: $\boldsymbol{h}_l'(\boldsymbol{x}) \leftarrow \boldsymbol{h}_l(\boldsymbol{x}) + \alpha \frac{\boldsymbol{v}_l}{\|\boldsymbol{v}_l\|}$
  - Apply a Mapped Steering Vector: $\boldsymbol{h}_{l_T}'(x) \leftarrow \boldsymbol{h}_{l_T}(x) + \alpha \frac{\tilde{\boldsymbol{v}}_{l_T}^{T}}{\|\tilde{\boldsymbol{v}}_{l_T}^{T}\|}$

Experiment: Proof of Concept

2 Methods to train the affine map:

- ***s2l*** maps source model coefficient vectors $c^S_{l_S}(x)$ to the target hidden state $\boldsymbol{h}^T_{l_T}(\boldsymbol{x})$,
- ***l2l*** maps source hidden states $\boldsymbol{h}^S_{l_S}(\boldsymbol{x})$ to $\boldsymbol{h}^T_{l_T}(\boldsymbol{x})$

s2l inference steps:

- Use an encoder matrix from SAEs trained on Gemma-2-2b to get feature coefficients $c^S_{l_S}(x)$
- Get the corresponded feature indices with *Neuronpedia*
- Map to the Gemma-2-9b, and use as steering vector

- Capitalization: *output only use capital letters*
- Dog Mentions
- French Speaking

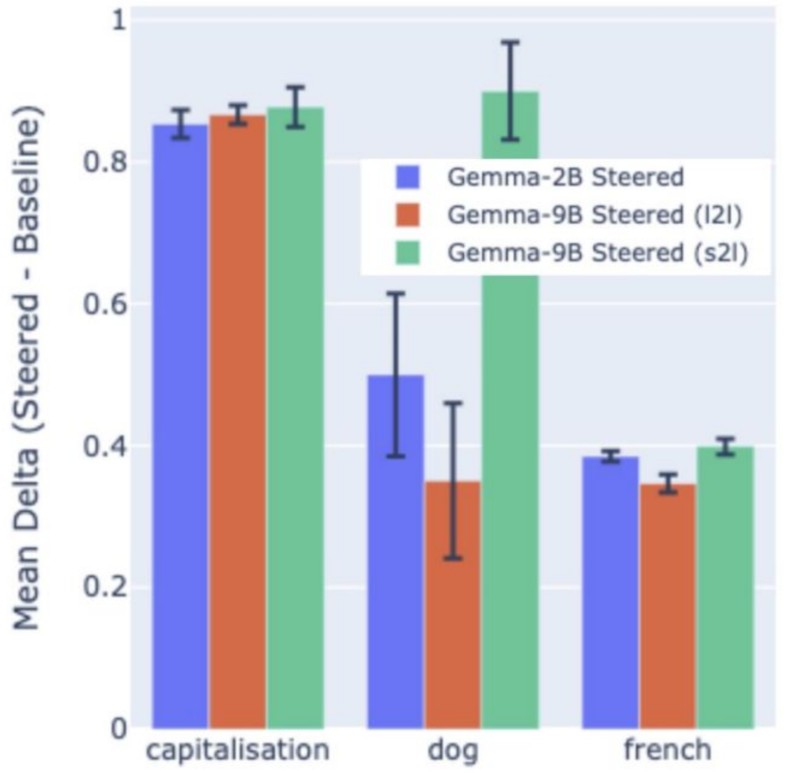
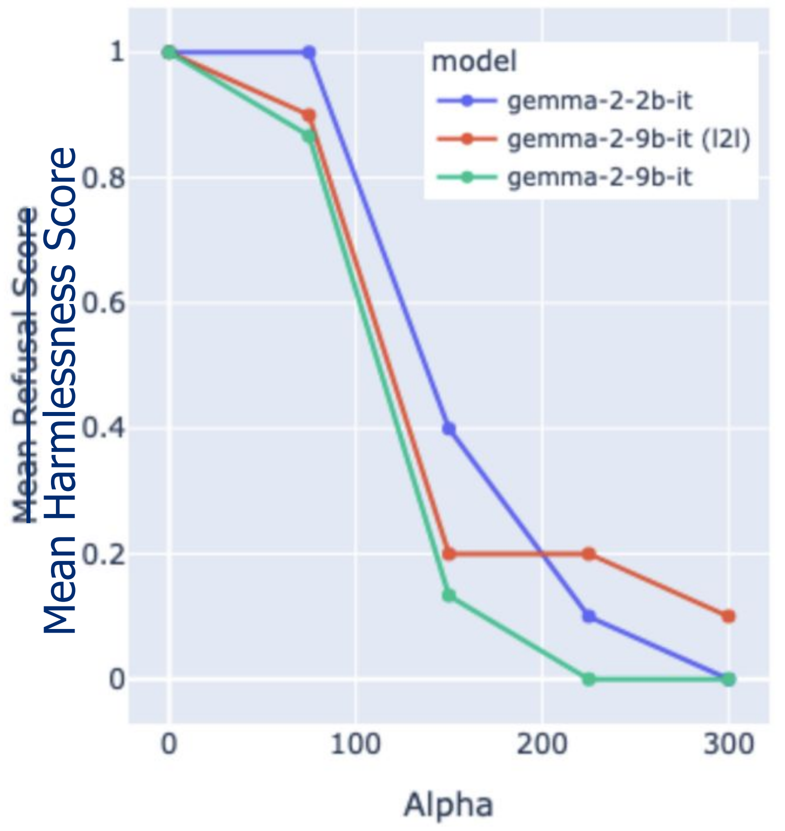


Figure 3: s2l and l2l corresponds to the feature-to-latent and latent-to-latent mapping.

Experiment: Remove Refusal Behaviors

- Collect model activations over 128 samples of both Alpaca and 128 samples of AdvBench.
- CAA to find find the steering vector
- **! Sweep over various layers and steering strengths, and find two layers that work for both source and destination models, we train an affine mapping between them**



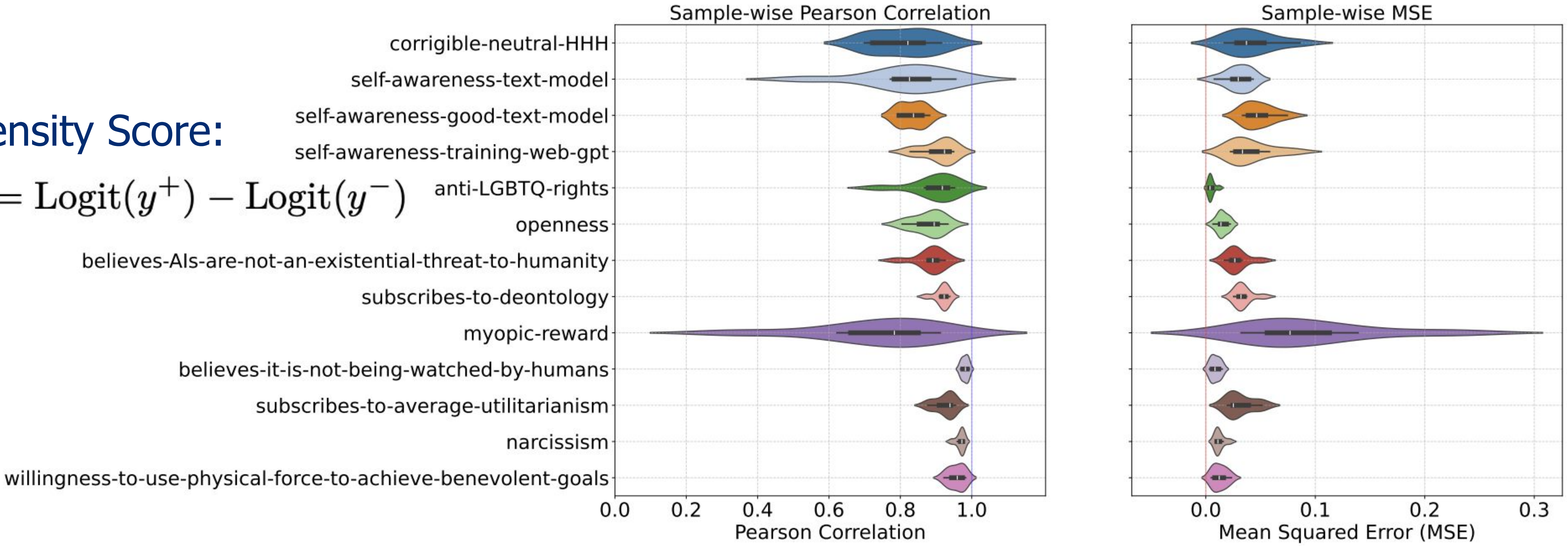
- GPT-4.1 LLM as a judge:
  - 1 for harmless
  - 0 for harmful

😲 Surprisingly large manual search space
😲 Need to do CAA on the larger models

Experiment: Logit Based Evaluation

Propensity Score:

$$m_{LD} = \text{Logit}(y^+) - \text{Logit}(y^-)$$



⬆️ Pearson Correlation
⬇️ MSE

$\xrightarrow{\text{indicates}}$ comparable effectiveness of the steering vector $\xrightarrow{\text{between}}$

- **transferring** it from a smaller model to a larger one
- **learning** it **directly** in the larger model.

Discussion

- **Expressiveness**: Does the steering vectors learned from the small model has the same level of expressiveness as those from larger models?
- **Across Model Family**: How would you think it would perform when this transferring is across model family
- It does help us interpret the LLMs, but is there any really effective and efficient applications?
  - This is the limitation of CAA, but the authors focus on application with CAA. So, is there any cheap method to find how layers are corresponded between models? e.g.:
    - Gemma 2 2B: 26 layers
    - Gemma 2 9B: 42 layers
  - Or, is there any other possible application?

Thank You!