KV-Lingo: Learning KV-Cache Translators with Distillation

Valérie Castin

Work done during an internship at Apple MLR, Paris



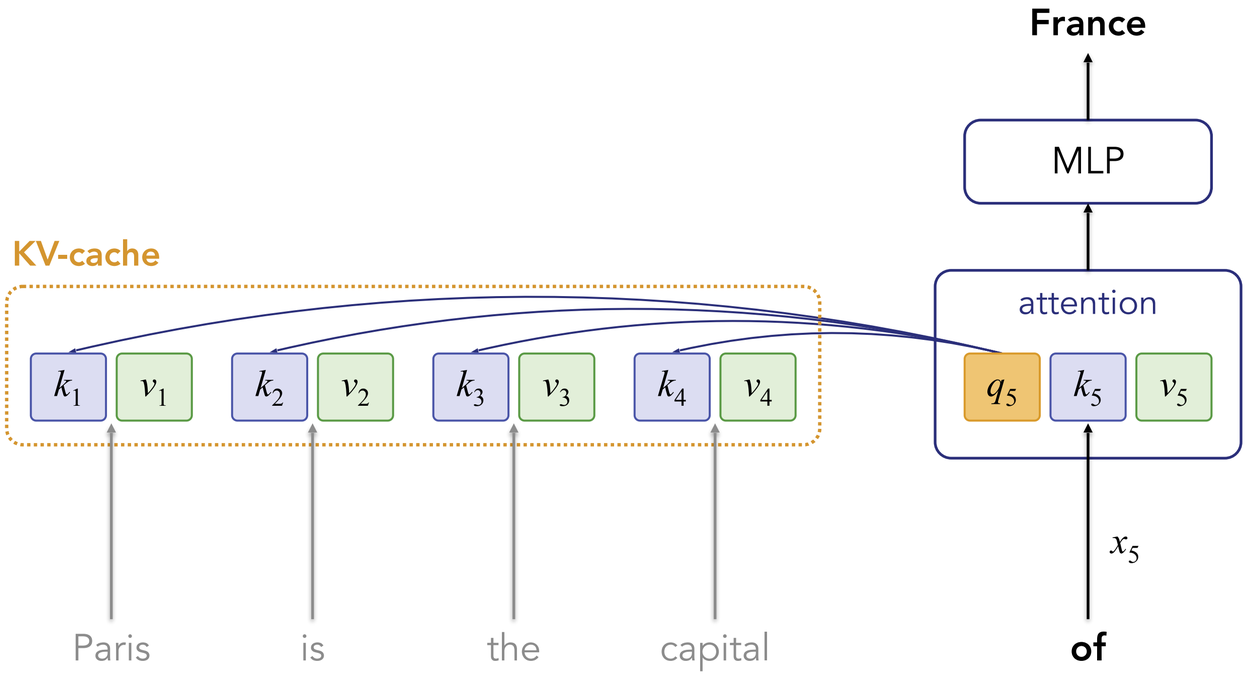




Keitaro Sakamoto Anastasiia Filippova João Monteiro Marco Cuturi Pierre Ablin

Linz, 5 November 2026

The KV-cache trades memory for speed

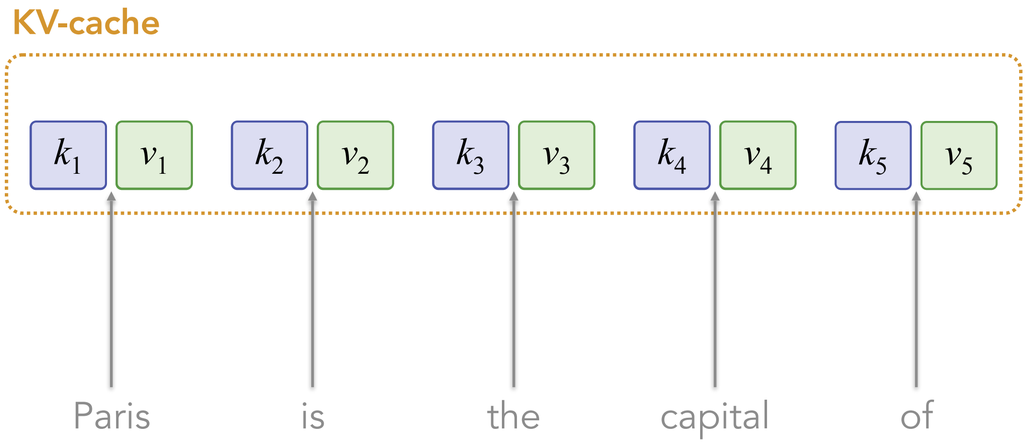


- Transformers interleave MLPs and attention blocks
- during decoding, attention looks at the keys and values of all previous tokens
- the cache avoids to recompute all past keys and values

One attention head:

$$o_n = \sum_{i=1}^{n} \text{softmax}_i\left(\frac{q_n^\top k_i}{\sqrt{d_h}}\right) v_i, \qquad q_n = W_Q x_n, \quad k_i = W_K x_i, \quad v_i = W_V x_i$$

The KV-cache trades memory for speed

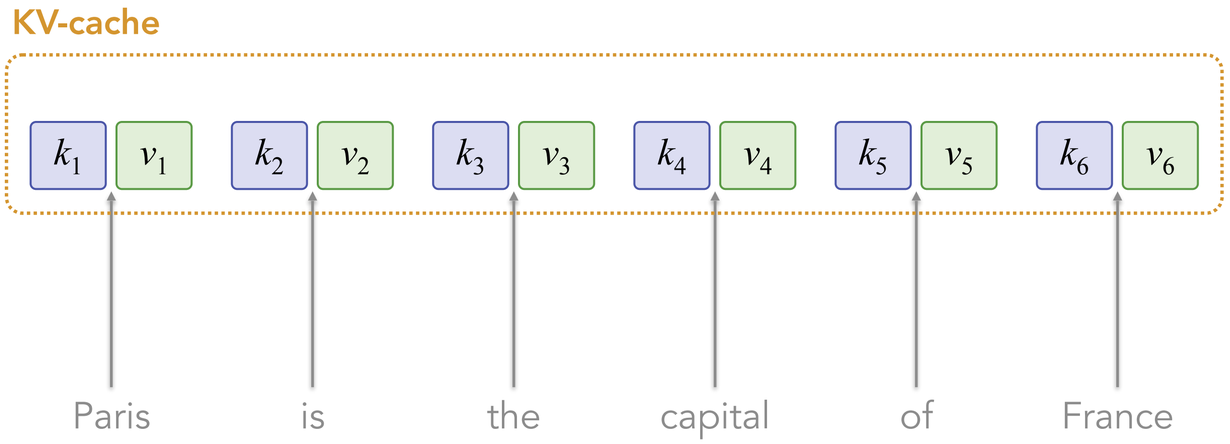


- Transformers interleave MLPs and attention blocks
- during decoding, attention looks at the keys and values of all previous tokens
- the cache avoids to recompute all past keys and values

One attention head:

$$o_n = \sum_{i=1}^{n} \text{softmax}_i\left(\frac{q_n^\top k_i}{\sqrt{d_h}}\right) v_i, \qquad \begin{aligned} q_n &= W_Q x_n, \\ k_i &= W_K x_i, \quad v_i = W_V x_i \end{aligned}$$

The KV-cache trades memory for speed

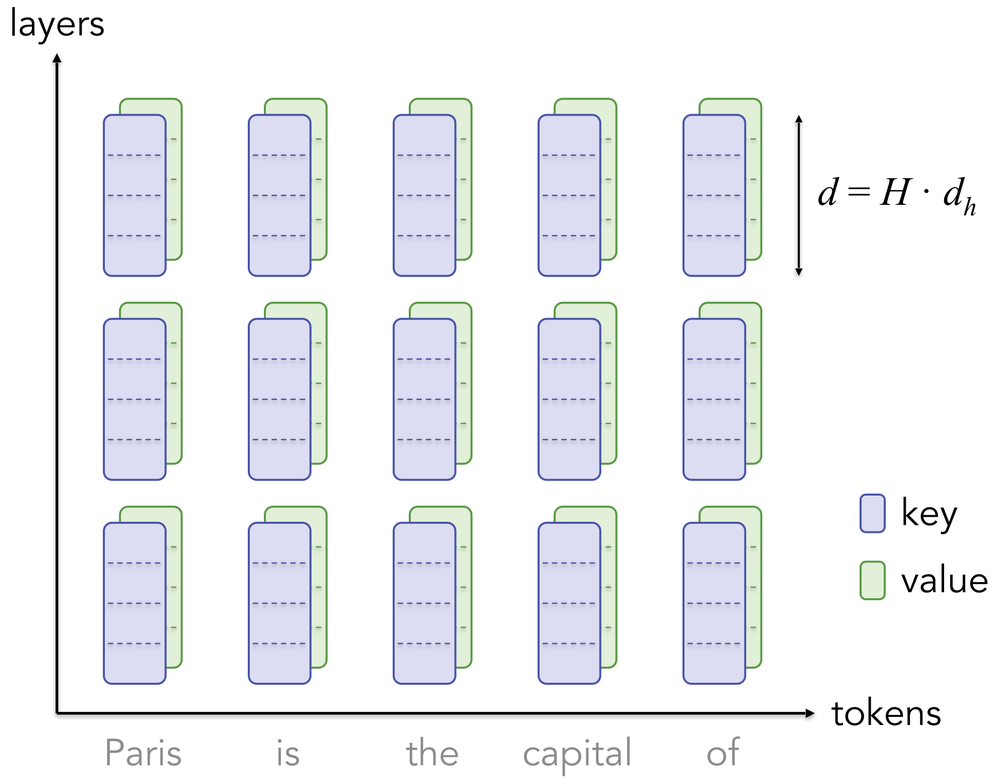


- Transformers interleave MLPs and attention blocks
- during decoding, attention looks at the keys and values of all previous tokens
- the cache avoids to recompute all past keys and values

One attention head:

$$o_n = \sum_{i=1}^{n} \text{softmax}_i\left(\frac{q_n^\top k_i}{\sqrt{d_h}}\right) v_i, \qquad \begin{aligned} q_n &= W_Q x_n, \\ k_i &= W_K x_i, \quad v_i = W_V x_i \end{aligned}$$

The KV-cache trades memory for speed



$$C = (K^\ell, V^\ell)_{\ell=1}^{N}, \qquad K^\ell, V^\ell \in \mathbb{R}^{d \times n}, \qquad d = H \cdot d_h$$

- one key and one value vector per token and layer (stacking the heads)
- big memory footprint: for Qwen3-4B, 55k tokens → 4B values (as big as the model)

Prefill: compute the cache of a context provided to the model, all n tokens in parallel $\mathcal{O}(n^2)$

Decode: generate the next tokens $\mathcal{O}(n)$

Outline

I - Motivation for KV-cache translation

II - KV-Lingo

III - Experiments

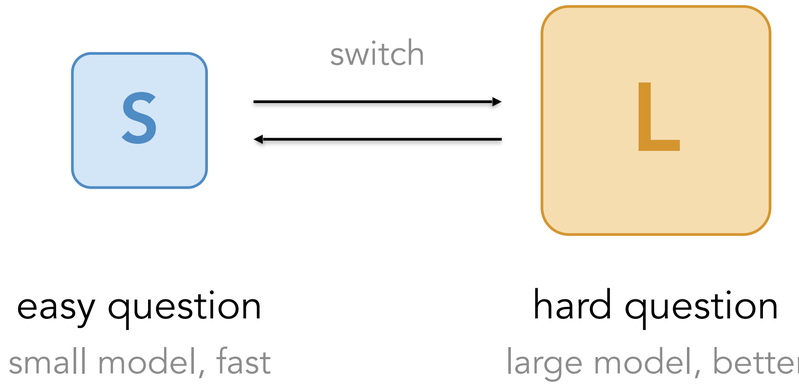
I - Motivation

The KV-cache is model-specific

Example task: model switching

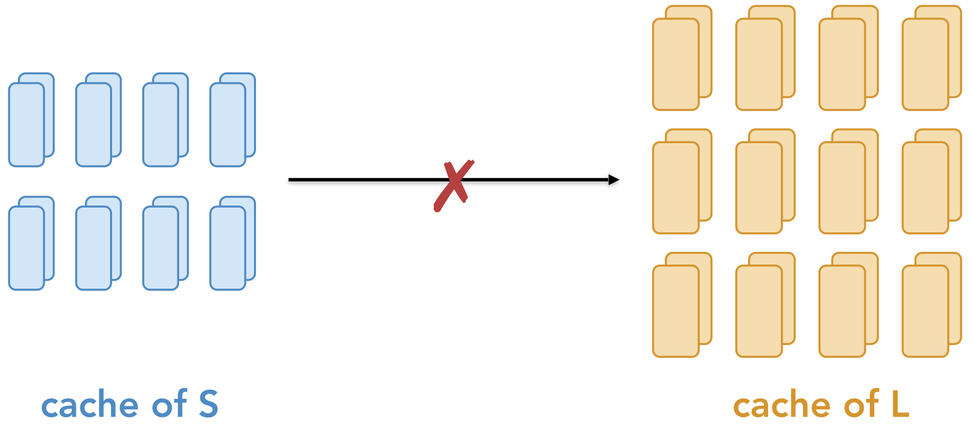
Q1, A1, Q2, A2, Q3 ⋮ A3

switch



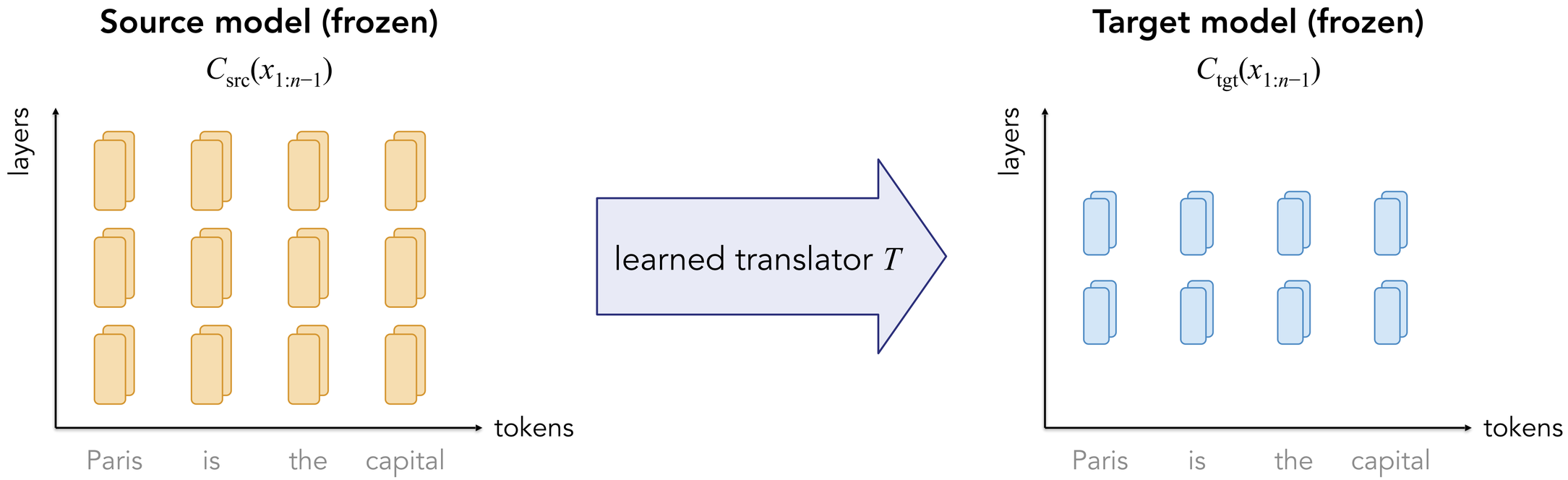
The cache does not transfer:

- possibly incompatible shapes
- even a same shape hand-off degrades generation [Liu et al., 2024]



Today: **re-prefill** the whole prefix at every switch → **time-to-first-token grows with the context length ($\mathcal{O}(n^2)$)**

Goal: train a KV-cache translator

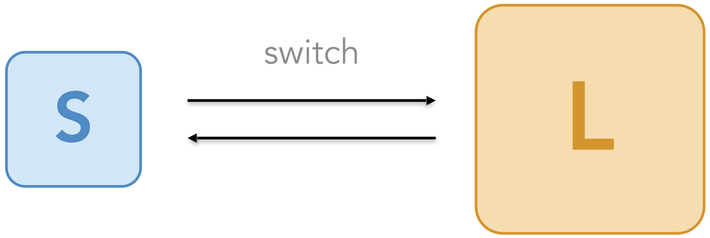


Goal: the target model predicts from $T(C_{\text{src}})$ what it would predict from its own cache C_{tgt}

$$p_{\text{tgt}}\big(\cdot \mid T(C_{\text{src}}(x_{1:n-1})),\, x_n\big) \;\approx\; p_{\text{tgt}}\big(\cdot \mid C_{\text{tgt}}(x_{1:n-1}),\, x_n\big)$$

Use cases of cache translation

Model switching

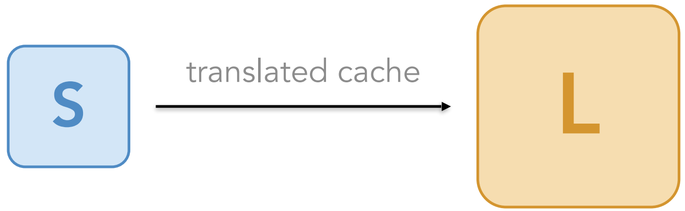


easy question
small model, fast

hard question
large model, better

- route models according to the complexity of the task (eg. L plans, S executes)

Prefill acceleration



prefill

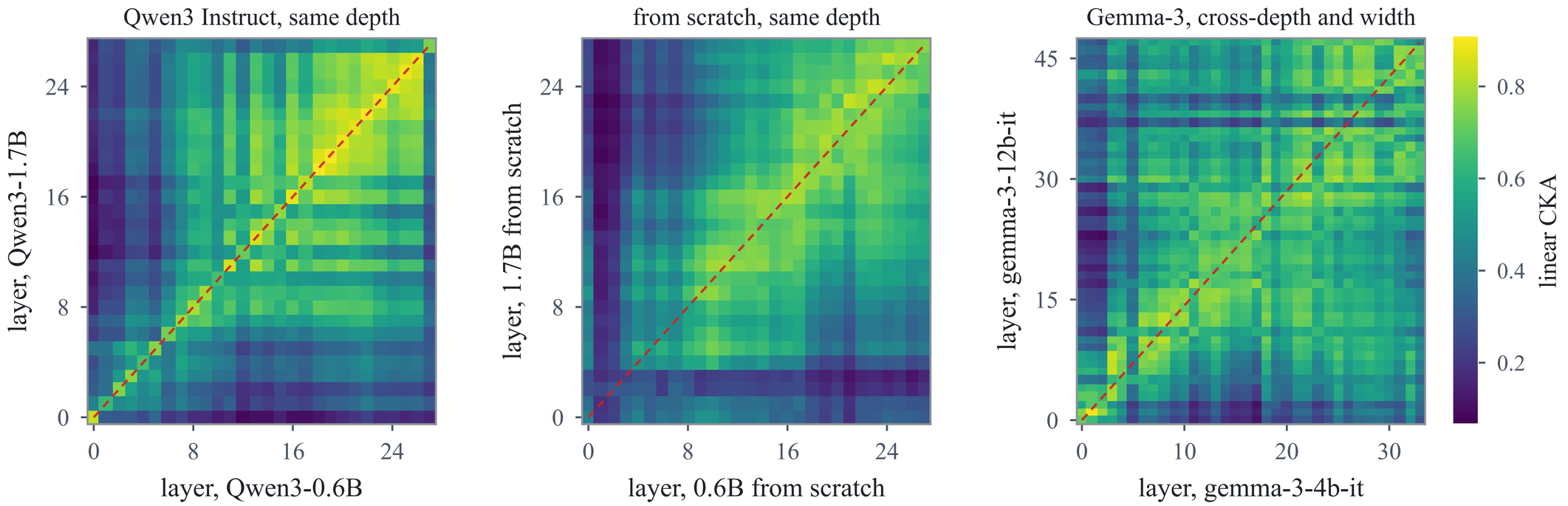
decode

- **Small → large dense model:** S prefills the prompt, L decodes from the translated cache
- **Dense → MoE:** a MoE decodes as fast as its active size, but its prefill loads most experts

II - KV-Lingo

KV-caches are surprisingly linearly related

Layer correspondence: linear CKA between every layer of a large model and every layer of a small one

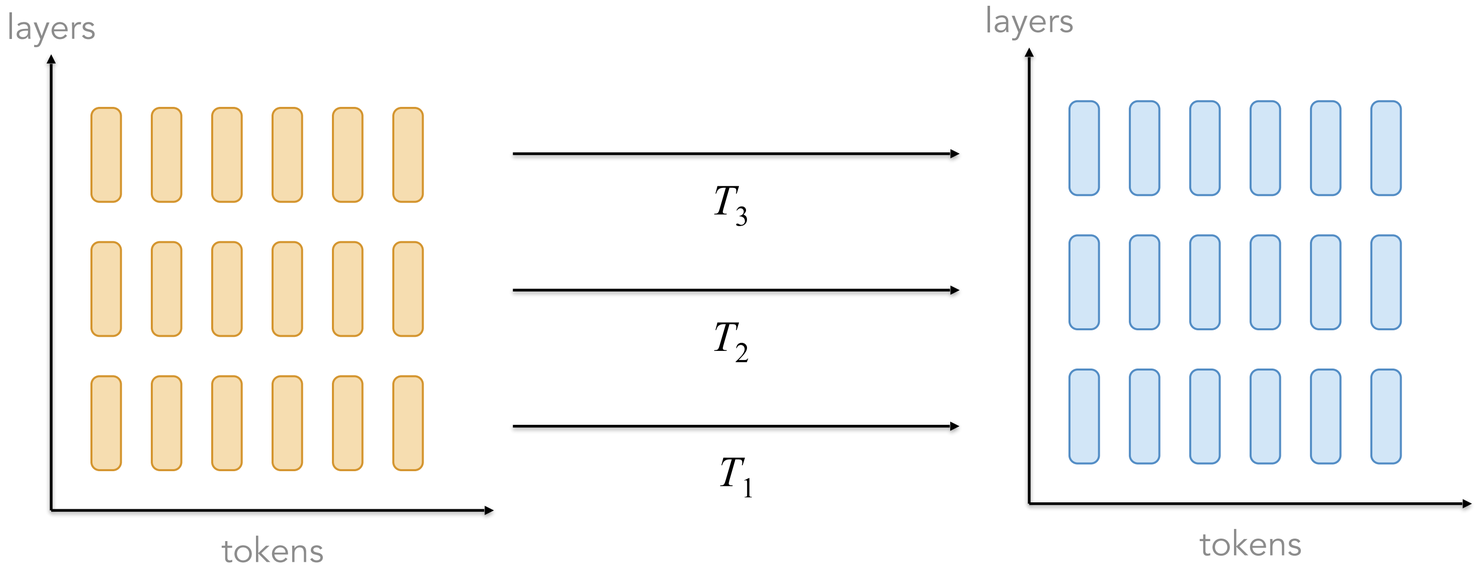


$$\mathrm{CKA}(X, Y) = \frac{\|X^\top Y\|_F^2}{\|X^\top X\|_F \, \|Y^\top Y\|_F}$$

→ the dependency of two cache layers of similar relative depth is almost linear!

The architecture of KV-Lingo: the same-shape case

One linear map per layer, the same for every token, keys and values separately

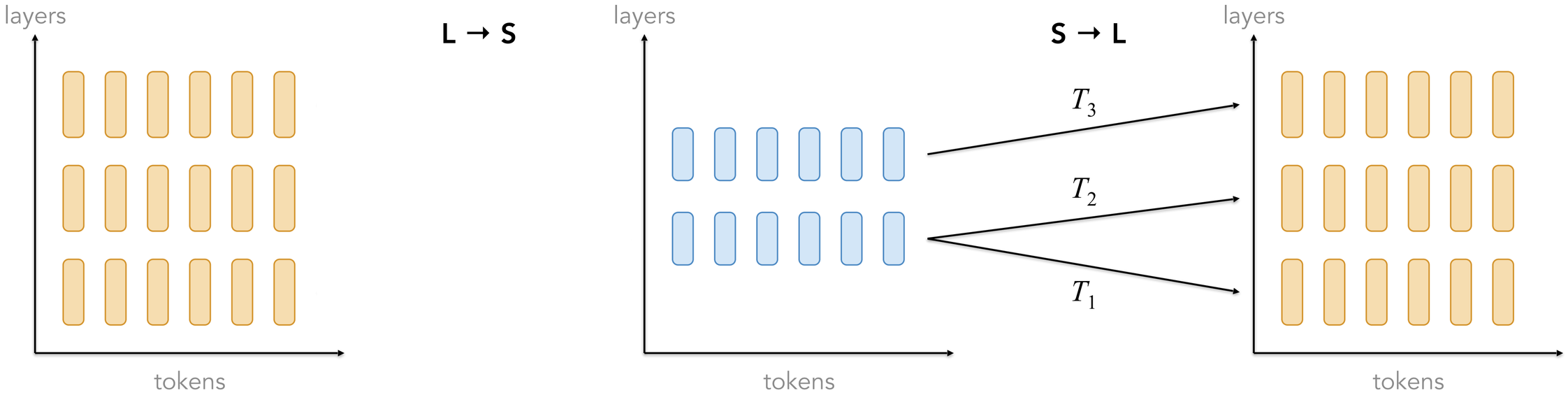


$$\hat{K}_i^\ell = T_K^\ell K_{\text{src},i}^\ell, \qquad \hat{V}_i^\ell = T_V^\ell V_{\text{src},i}^\ell, \qquad i = 1, \ldots, n-1$$

$$T_K^\ell, T_V^\ell \in \mathbb{R}^{d_{\text{tgt}} \times d_{\text{src}}}$$

The architecture of KV-Lingo: the cross-layer case

One linear map per target layer, keys and values separately, E_ℓ := set of source layers for target layer ℓ (band of layers at same relative depth as ℓ), with a fixed number of source layers $\nu = \sharp E_\ell$



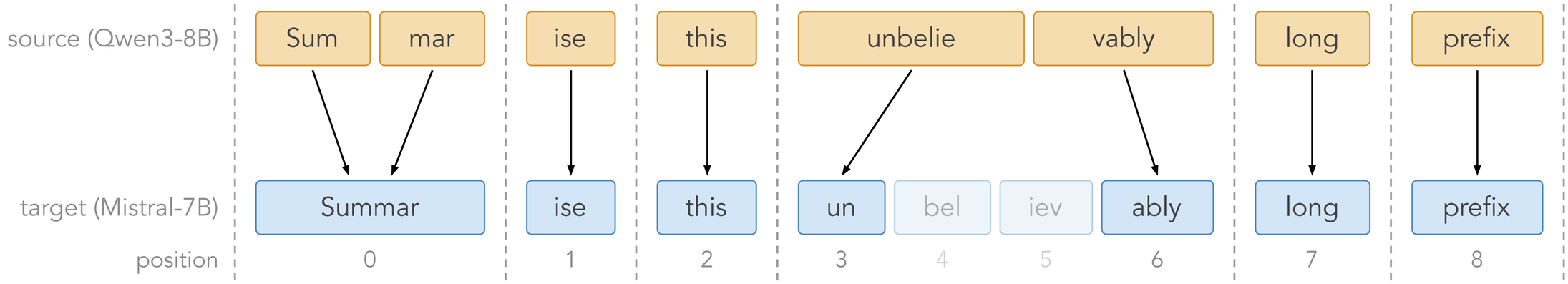
$$\widehat{K}_i^\ell = \sum_{\ell' \in E_\ell} T_K^{\ell' \to \ell} K_{\mathrm{src},i}^{\ell'}, \qquad \widehat{V}_i^\ell = \sum_{\ell' \in E_\ell} T_V^{\ell' \to \ell} V_{\mathrm{src},i}^{\ell'}, \qquad i = 1, \ldots, n-1$$

$$T_K^{\ell' \to \ell},\ T_V^{\ell' \to \ell} \in \mathbb{R}^{d_{\mathrm{tgt}} \times d_{\mathrm{src}}}$$

Layer assignment: relative depth assignment

The architecture of KV-Lingo: the cross-tokenizer case

Different tokenizers: the source and the target caches have different numbers of tokens



- **Minimal groups**: cut wherever both tokenizers end a token at the same character
- **Inside a group**: spread the p source tokens evenly over the q target tokens

KV-Lingo is trained with distillation

❄️ source and target frozen 🔥 only T is trained

Training data: Nemotron-SFT-Instruction-Following-Chat-v2, one dataset for all evaluations

User turn + Assistant turn + … + User turn | switch | Assistant turn

prefix $x_{1:n}$ (10 to 16k tokens) | continuation $y_{1:m}$ (≤ 2048 tokens)

Stage 1: closed-form initialisation by cache reconstruction

$$T_K^\ell = \arg\min_M \ \mathbb{E}_{x_{1:n}} \left\| M K_{\text{src}}^\ell(x_{1:n}) - K_{\text{tgt}}^\ell(x_{1:n}) \right\|_F^2 \qquad \ell = 1, \ldots, L$$

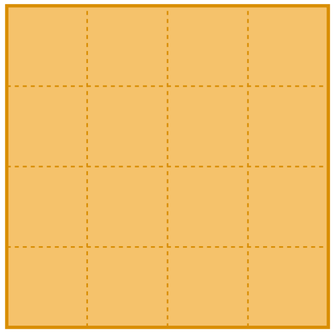
→ **not enough: hurts reasoning & long-context generation**

Stage 2: self-distillation on the continuation (Adam, starting from stage 1)

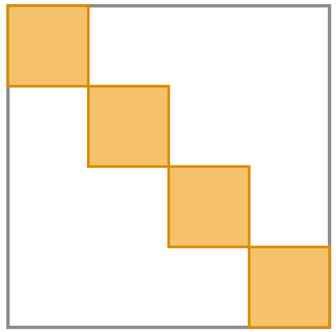
$$\mathcal{L}(T) = \mathbb{E}_{(x_{1:n}, y_{1:m})} \frac{1}{m} \sum_{j=1}^{m} \mathrm{KL}\Big(p_{\text{tgt}}\big(\cdot \mid C_{\text{tgt}}(x_{1:n}), y_{<j}\big) \,\Big\|\, p_{\text{tgt}}\big(\cdot \mid T(C_{\text{src}}(x_{1:n})), y_{<j}\big) \Big)$$

Variants of KV-Lingo

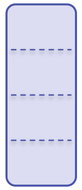
Head mixing vs head-wise



head-mixing

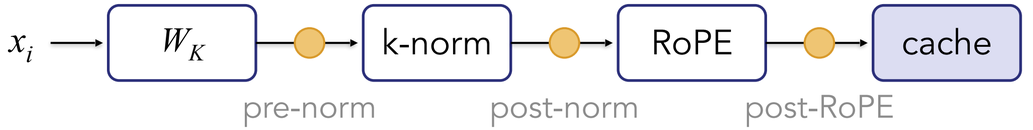


head-wise



- **Head-mixing**: unconstrained (target width × source width) matrix
- **Head-wise**: target head h reads only source head h: H× fewer parameters, faster, less expressive

Choosing the capture point



Pre-norm

- translated keys are then normalised and rotated by the target
- extrapolates beyond the training length

Post-RoPE

- matches pre-norm on training prefix lengths
- collapses for OOD prefix lengths
- allows to merge translation in the query (if headwise & layerwise)

$$\langle q, T_K K_{\text{src},i} \rangle = \langle T_K^\top q, K_{\text{src},i} \rangle$$

III - Experiments

Experimental setup

Models

- Qwen3 (0.6B/1.7B/4B/8B)
- Gemma-3 (4B/12B), Gemma-4 (E2B/E4B)
- Qwen3-4B → Qwen3-30B-A3B (dense to MoE)
- Qwen3-8B ↔ Mistral-7B (cross-tokenizer)
- MLA-Qwen3 (4B/8B), converted with CARE

Training

- **one translator per model pair**, trained on one multi-turn chat dataset (Nemotron)
- prefixes of 10 to 16k tokens

Evaluation

- **knowledge & reasoning:** MMLU-Pro, ARC-C, GSM8K
- **instruction following:** IFEval
- **context-grounded QA:** RepLiQA, CoQA, XQuAD
- **multi-turn chat:** MT-Bench-101
- **long context:** RULER, LongMemEval, LongBench-v2

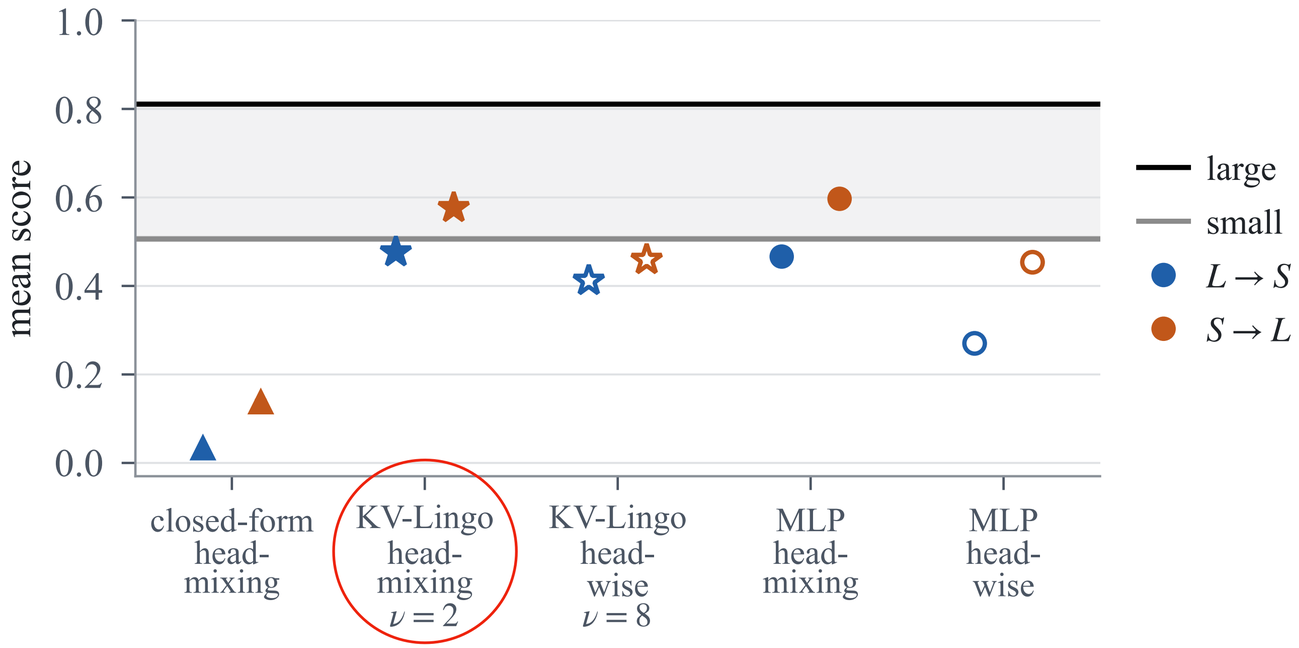
1. prefill the prompt with source model
2. translate the cache
3. decode with the target model

KV-Lingo matches MLP scores, and is close to native decoding

Baselines

- full reprefill (target model score)
- head-wise closed-form linear map [Heo et al., 2026]
- head-wise MLP [Chen et al., 2026], head mixing MLP

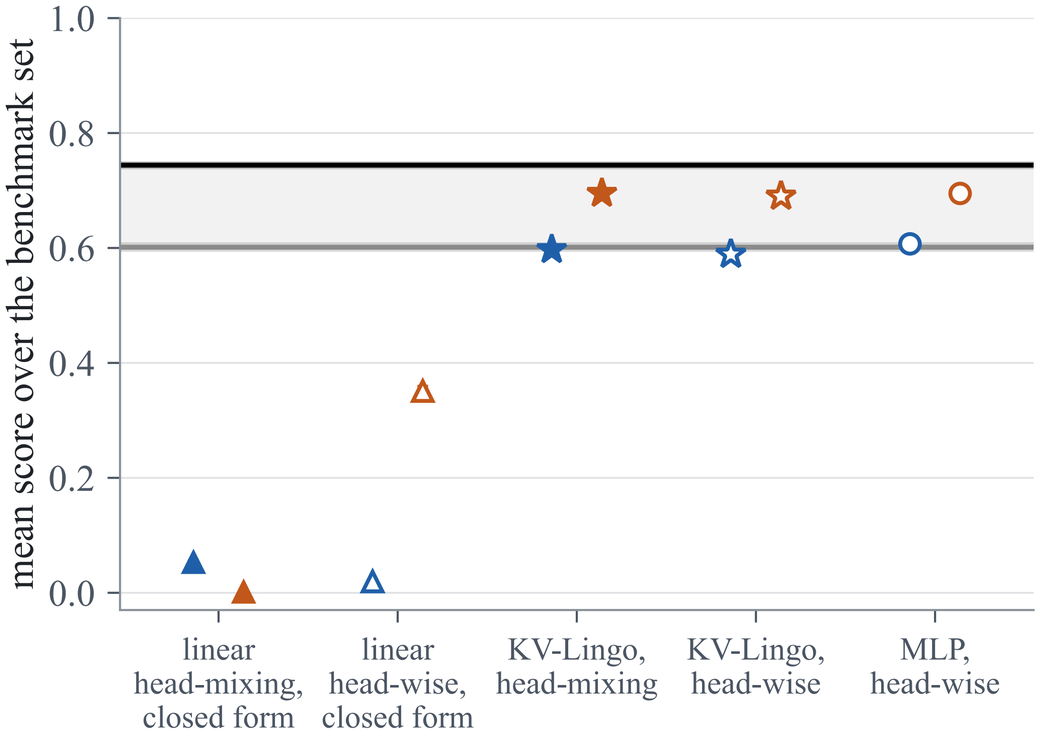
Reasoning-off, Qwen3 8B / 0.6B



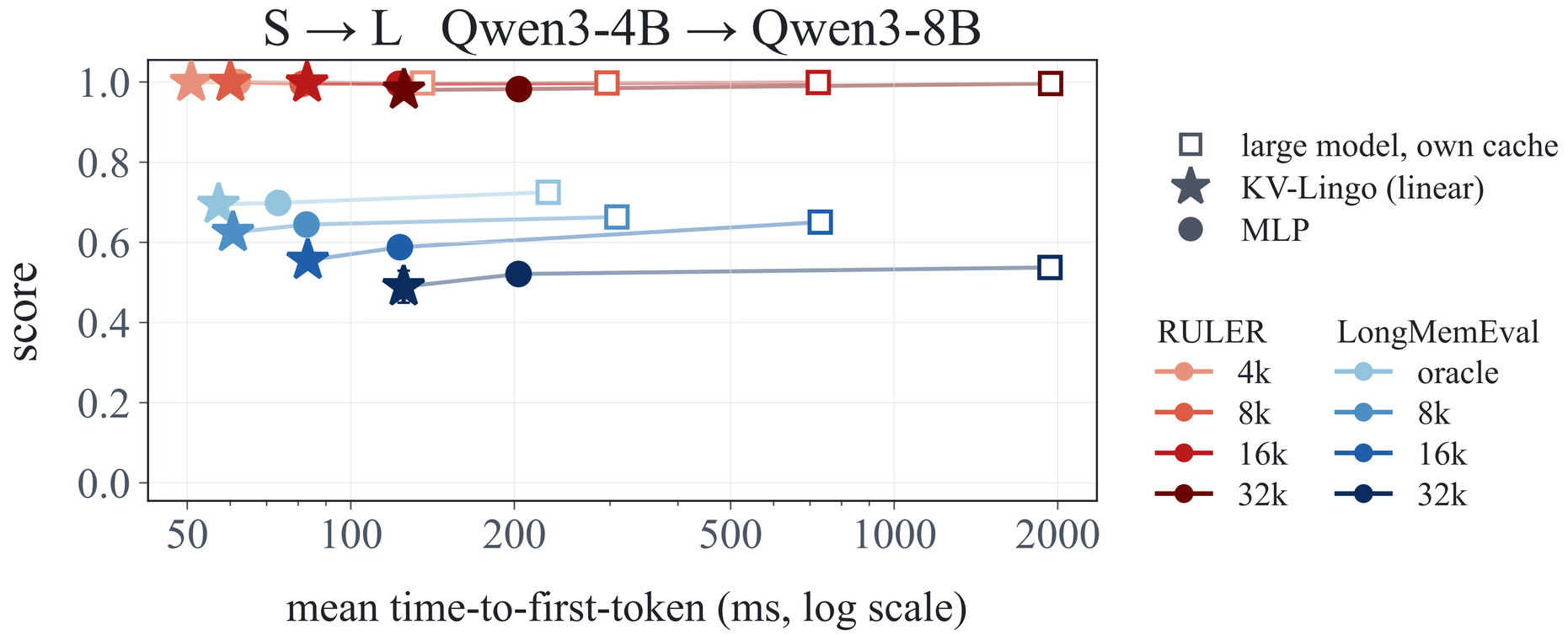
faster than the MLP

ν: number of source layers per target layer

Reasoning-on, Qwen3 1.7B / 0.6B

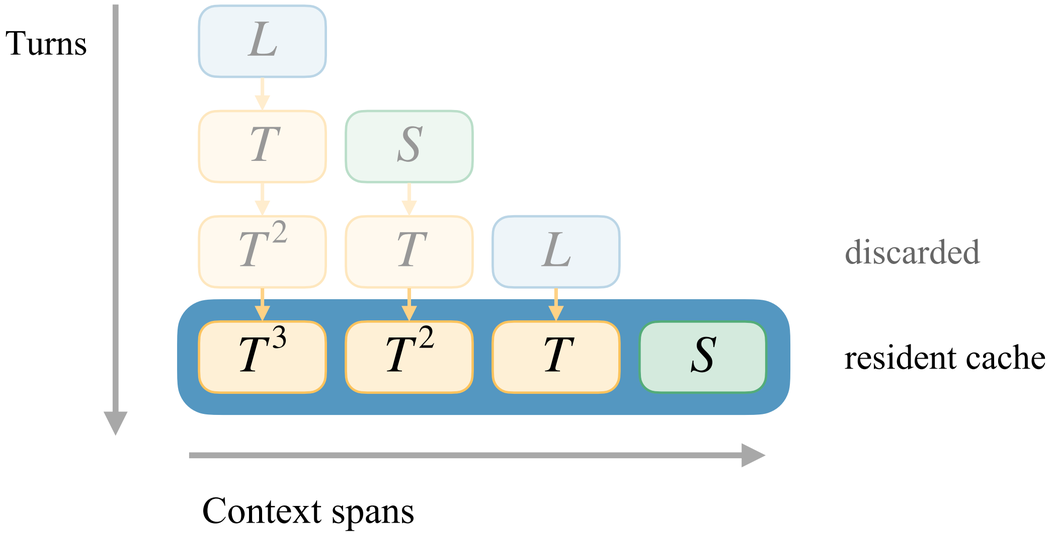


Long context: KV-Lingo significantly accelerates the switch

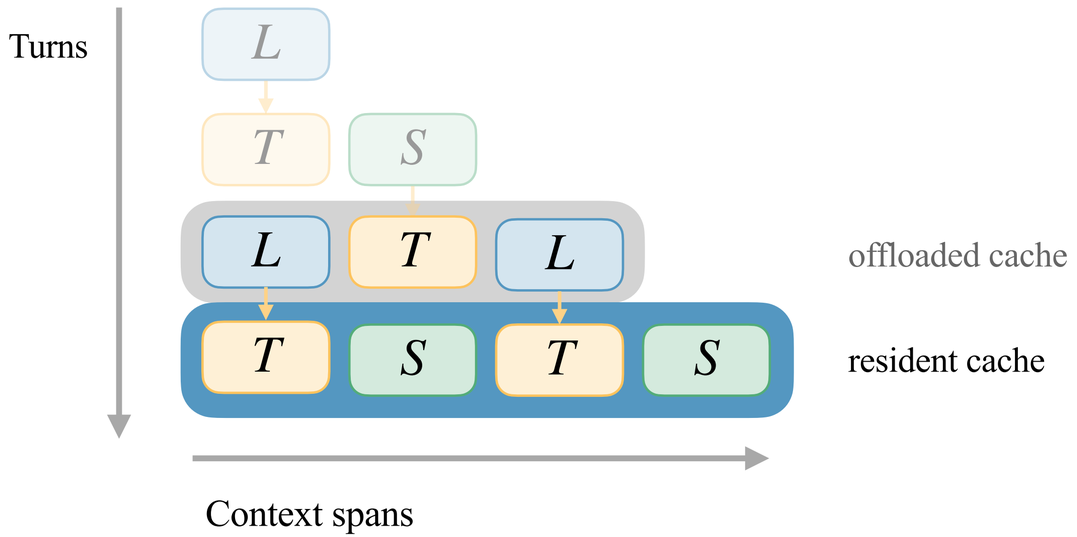


KV-Lingo is robust to several switches in one conversation

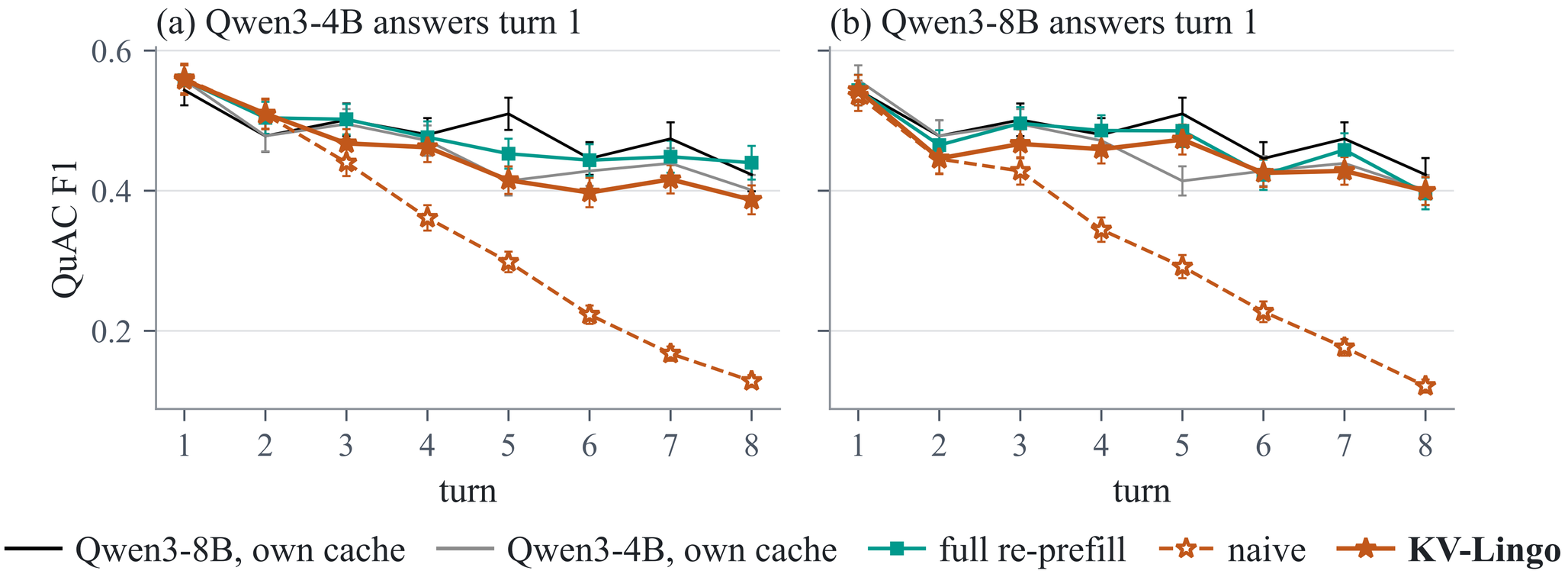
- **Naive**: translate the whole prefix at every switch → translation errors compound



- **KV-Lingo**: each model keeps its own cache line and only translates the tokens generated since the last switch → every span is translated at most once



KV-Lingo is robust to several switches in one conversation



Conclusion

- **KV-Lingo**: token-wise linear maps, closed-form initialisation + self-distillation
- **Downstream performance close to native** across sizes, families, tokenizers, MLA and MoE
- Up to **29× faster time to first token** after a switch
- Robust to **repeated switches**

Perspectives:

- Train models jointly for compatible caches?
- Theoretical analysis of cache structure?
- Translate and compress at the same time?

Paper link

Thank you!

References:

[Castin et al., 2026] KV-Lingo: Learning KV-cache translators with distillation, V. Castin et al., *Arxiv preprint*, 2026

[Liu et al., 2024] DroidSpeak: KV cache sharing for cross-LLM communication and multi-LLM serving

[Heo et al., 2026] Cross-model KV cache transfer in LLM families: a closed-form linear mapping for prefill reuse

[Chen et al., 2026] See what I see, know what I think: dense latent communication across heterogeneous agents

[Kornblith et al., 2019] Similarity of neural network representations revisited