BEYOND SPEEDUP - UTILIZING KV CACHE FOR SAMPLING AND REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

KV caches, typically used only to speed up autoregressive decoding, encode contextual information that can be reused for downstream tasks at no extra cost. We propose treating the KV cache as a lightweight representation, eliminating the need to recompute or store full hidden states. Despite being weaker than dedicated embeddings, KV-derived representations are shown to be sufficient for two key applications: **(i) Chain-of-Embedding**, where they achieve competitive or superior performance on Llama-3.1-8B-Instruct and Qwen2-7B-Instruct; and **(ii) Fast/Slow Thinking Switching**, where they enable adaptive reasoning on Qwen3-8B and DeepSeek-R1-Distil-Qwen-14B, reducing token generation by up to $5.7\times$ with minimal accuracy loss. Our findings establish KV caches as a free, effective substrate for sampling and reasoning, opening new directions for representation reuse in LLM inference.

1 INTRODUCTION

Large language models (LLMs) rely on key-value (KV) cache to accelerate autoregressive decoding by reusing past attention states, avoiding costly recomputation. This makes the KV cache indispensable for low-latency inference in production systems like vLLM (Kwon et al., 2023). However, its role is typically confined to this speedup. Beyond acceleration, the KV cache is seldom viewed as a reusable representation—with the notable exception of cache steering, a technique that modifies the cache's initial state to guide generation (Belitsky et al., 2025).

While the KV cache has been mostly confined to acceleration, hidden states have been widely exploited for *self-evaluation* (Wang et al., 2025b; Chen et al., 2024; Beigi et al., 2024; Zhang et al., 2025a) and for *adaptive reasoning and control* (Zhang et al., 2025b; Wang et al., 2025a; Yue et al., 2025). These methods, however, rely on storing full hidden states, which is costly in both memory and compute.

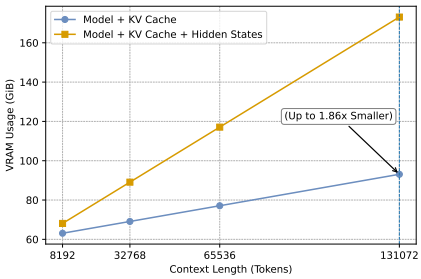

Figure 1: VRAM usage vs. context length for Qwen3-32B (QwenTeam, 2025), comparing Model+KV Cache vs. Model+KV Cache+Hidden States.

In this work, we investigate a simple but powerful question: **Can the KV cache do more than just accelerate decoding?** Since the KV cache is already computed and stored during inference, using it for downstream tasks incurs *no additional cost*. This is a major advantage over storing full hidden states, which is prohibitively expensive in terms of memory. As shown in Figure 1, the KV cache offers a significantly more compact and practical alternative for typical decoder-only models.

Though the KV cache is not explicitly trained as a general-purpose embedding—its sole objective is to support next-token prediction—we find it nonetheless encodes rich contextual information suitable for various downstream tasks. We explore this potential through two applications:

- **Chain-of-Embedding:** We repurpose the KV cache as a lightweight and readily available embedding. In experiments on Chain of Embedding (CoE) (Wang et al., 2025b)—a method for selecting optimal reasoning paths without external information—we show that

KV caches achieve classification performance comparable to or even surpassing that of using the model's hidden states.

- **Fast/Slow Thinking Switch:** We leverage the KV cache to implement an adaptive switching mechanism between fast, low-compute reasoning and slower, deliberate reasoning. By reusing KV cache, this approach achieves substantial efficiency gains with minimal performance loss.

Our contributions are fourfold:

1. We present the first systematic study of KV caches as reusable task representations, showing they can be repurposed at near-zero computational cost. In particular, we propose simple but effective aggregation techniques that make KV caches directly usable as embeddings.
2. Despite not being designed as general-purpose embeddings, we find that KV cache representations when processed with the proposed aggregation strategies, are competitively effective on certain classification tasks.
3. We propose **KV-CoE**, a variant of Chain-of-Embedding that reuses the KV cache already stored during decoding. KV-CoE achieves self-evaluation without extra activation storage, offering nearly zero memory overhead and seamless integration into existing inference frameworks.
4. We introduce **KVClassifier**, a fast/slow auto-thinking framework that reuses KV caches for adaptive reasoning with minimal overhead.

Our results suggest that KV caches are a versatile and low-cost foundation for sampling and reasoning, moving beyond their traditional role as a mere acceleration component to become a core resource for effective and efficient LLM-based inference.

2 Related Work

Hidden–state self-evaluation. A growing line of work shows that internal activations encode reliable signals about answer correctness and hallucination risk. Wang et al. (2025b) propose *Chain-of-Embedding* (CoE), which models the trajectory of layerwise hidden states during inference and derives output-free correctness scores from the geometry of this path. Chen et al. (2024) (INSIDE) introduce *EigenScore*, computed from the eigenspectrum of hidden-state covariance, to assess semantic (in)consistency and detect hallucinations. Beigi et al. (2024) train a contrastive probe on *internal states* (attention, MLP activations) to produce well-calibrated confidence estimates across NLU/NLG tasks. Zhang et al. (2025a) further probes hidden states of reasoning models to predict whether a generated answer will be correct. All of these methods operate directly on hidden states or logits. **Our study**, by contrast, investigates whether *the KV cache alone*—which is already present at inference—suffices to support the same families of subtasks.

Adaptive fast/slow reasoning and dynamic control. To mitigate overthinking on easy inputs and underthinking on hard ones, recent work explores *adaptive* reasoning depth (Xing et al., 2025). Zhang et al. (2025b) quantify upper bounds of long- vs. no-thinking modes and propose *Adaptive Self-Recovery Reasoning* (ASRR), adding accuracy-aware length rewards to reduce unnecessary reasoning while allowing implicit recovery. PATS (Wang et al., 2025a) performs *process-level* switching via process reward models with beam search, enabling step-wise fast/slow adaptation with bad-step penalties. DOTS (Yue et al., 2025) views reasoning as a search over atomic actions and learn to select dynamic trajectories. These approaches typically require explicit chain-of-thought generation, external reward models, or re-decoding. **Our contribution** is orthogonal: we show that pooled *KV-cache* features can drive both one-shot (classification-style) and in-generation (generative-style) switching through simple control tokens, without storing hidden states or altering model architecture.

KV-cache interventions. While our work treats the KV cache as a read-only representation for evaluation and control, a concurrent line of research shows that it can also serve as a *control interface*. Belitsky et al. (2025) introduce *KV Cache Steering*, a one-shot intervention that adds layer-wise steering vectors—derived from contrastive CoT vs. non-CoT prompts—to the key and value

tensors after prefill, reliably inducing longer and more structured reasoning in small Language Models. Compared with activation steering, this method offers improved stability and negligible runtime overhead. **Our approach** is complementary: instead of modifying the cache, we *pool* it to derive difficulty-aware signals that gate slow reasoning.

3 BACKGROUND

3.1 TRANSFORMER, HIDDEN STATES, AND KV CACHE

Decoder-only transformers are the architectural foundation of modern large language models (LLMs) (Vaswani et al., 2017; Brown et al., 2020). During autoregressive generation, each transformer layer processes a new token to produce a contextual *hidden state*. A computational bottleneck arises because standard attention requires recomputing over all previous tokens at each step, resulting in $\mathcal{O}(T^2)$ complexity per step, where T is the sequence length. To mitigate this, the key–value (KV) cache (Dao et al., 2022) stores the attention keys and values for all past tokens at every layer. This allows the model to compute keys and values only for the new token and attend to the cached history, reducing the complexity to $\mathcal{O}(T)$ per step and enabling efficient long-sequence generation.

Formally, for layer l, we store

$$\text{KVCache}^{(l)} = \{K_{1:T}^{(l)}, V_{1:T}^{(l)}\},$$

where $K_{1:T}^{(l)}, V_{1:T}^{(l)} \in \mathbb{R}^{T \times H \times d_{\text{head}}}$ are the stacked key and value tensors across all attention heads H. The hidden state at step t is compuated via

$$h_t^{(l)} = \text{Attention}\Big(Q_t^{(l)}, K_{1:t}^{(l)}, V_{1:t}^{(l)}\Big).$$

Since $K_{1:t-1}^{(l)}$ and $V_{1:t-1}^{(l)}$ are already cached, only $K_t^{(l)}$ and $V_t^{(l)}$ need to be computed online.

3.2 MODERN LLM FRAMEWORKS AND KV CACHE MANAGEMENT

State-of-the-art LLM serving frameworks carefully manage KV caches to achieve high throughput, low latency, and efficient GPU memory utilization.

vLLM. vLLM (Kwon et al., 2023) introduces a *PagedAttention* mechanism that virtualizes the KV cache in a paging system similar to CPU virtual memory. This allows dynamic allocation and reuse of KV cache memory, significantly reducing fragmentation and enabling high-throughput serving for thousands of concurrent sequences. By paging KV blocks in and out efficiently, vLLM supports fine-grained preemption and scheduling, which is critical for production-scale inference.

Ollama. Ollama (Ollama Team, 2024) is a lightweight LLM deployment framework that emphasizes developer usability and local inference. It manages KV caches per session, allowing multi-turn conversations to reuse context efficiently without re-computation. Although less focused on extreme multi-tenant throughput compared to vLLM, Ollama provides a practical demonstration of KV cache persistence for interactive workloads.

Modern frameworks already manage the KV cache as a first-class resource, with sophisticated strategies for allocation, eviction, and sharing. This practice underscores our central claim:

"*Since the KV cache is an unavoidable byproduct of efficient inference, repurposing it for downstream tasks adds virtually no overhead.*"

4 OBSERVATION

4.1 CAN KV CACHES SERVE AS AN EMBEDDING SOURCE

The hidden states and attention projections stored in KV caches encode contextualized token representations, making them natural candidates for use as embeddings. While recent work has explored leveraging intermediate representations from LLMs as task-specific embeddings (Liu et al., 2024), we specifically investigate aggregating KV cache vectors into sentence-level representations.

To evaluate their quality as an embedding source, we construct embeddings by concatenating keys and values at every layer, then averaging across token positions, attention heads, and layers before applying ℓ_2 normalization. We benchmark these KV-derived embeddings on the Massive Text Embedding Benchmark (MTEB) (Muennighoff et al., 2023) against a strong, dedicated embedding model (gemini-embedding-001).

Dataset	Llama-3.1-8B-Instruct KV Cache	Gemini-Embedding-001
AmazonCounterfactualClassification	0.3530	0.8820
DBpediaClassification	0.5937	0.9476
FinancialPhrasebankClassification	0.6254	0.8864
TweetTopicSingleClassification	0.3714	0.7111

Table 1: Performance of KV cache-based embeddings vs. a dedicated embedding model on selected MTEB classification tasks. Despite being significantly weaker than trained embeddings, KV-derived embeddings still capture meaningful semantics.

As shown in Table 1, KV-derived embeddings significantly underperform their dedicated counterpart across all datasets, confirming they are *not perfect general-purpose embeddings*. This gap stems from three factors: (i) KV representations are optimized for causal language modeling, not contrastive learning, leading to poor isotropy; (ii) they are inherently token- and position-specific, requiring heuristic pooling for sentence-level use; and (iii) their projection into a lower-dimensional head space ($d_{\text{head}} \ll d_{\text{model}}$) reduces their discriminative power.

Despite these limitations, the results show that KV caches encode substantial semantic information—enough to be competitive on certain classification tasks. This finding motivates our exploration of reusing the KV cache for *Chain-of-Embedding* and *Fast/Slow Thinking Switch*, where global embedding quality is less critical than local, relative separability between candidates.

4.2 Why KV Caches Are Sufficient for Chain-of-Embedding and Fast/Slow Thinking Switch?

While KV caches are suboptimal for general-purpose embedding tasks—as they are trained solely for next-token prediction, are position-specific, and reside in a reduced-dimensional space—we contend they are nevertheless **sufficient** for *Chain-of-Embedding* and *Fast/Slow Thinking Switch*. Their poor performance on global semantic similarity tasks (e.g., 0.35 vs. 0.88 accuracy on AmazonCounterfactualClassification in our MTEB results) is less critical for these applications, which rely on different criteria.

Relative Separation via Margin. Let $\mathcal{X}$ denote the embedding space and $f : \mathcal{X} \to \mathcal{Y}$ a classifier. For a pair of classes (y_i, y_j), the empirical margin on embedding $x \in \mathcal{X}$ is defined as

$$\gamma(x) = f_{y_i}(x) - f_{y_j}(x).$$

While contrastive learning aims to learn embeddings by maximizing $\mathbb{E}[\gamma(x)]$ globally. In contrast, our target applications (Chain-of-Embedding and Fast/Slow Thinking Switch) only require that for a *restricted candidate set* $\mathcal{C} \subset \mathcal{Y}$ (e.g., a small label space or a few candidate continuations), the margin satisfies

$$\min_{x \in \mathcal{C}} \gamma(x) > 0.$$

Thus, even with a noisy or anisotropic KV embedding space, preserving correct relative ordering within $\mathcal{C}$ is sufficient, explaining their competitive performance on tasks with limited labels.

Contextual Conditioning. The pooled KV embedding $e = \text{Pool}(K, V)$ in an instruction-tuned model can be expressed as $e = g(x, \iota)$, where x is the input and ι is the instruction. Unlike general-purpose embeddings, this representation is contextually conditioned on the task, which inherently shapes the relevant decision boundaries. This conditioning reduces the need for a globally isotropic embedding space.

Efficiency Constraint. Let C_{hidden} and C_{KV} denote the memory cost of storing hidden states and reusing the KV cache, respectively, with $C_{\text{hidden}} \gg C_{\text{KV}} \approx 0$. The expected utility of reusing the KV cache is

$$U = \text{Acc}_{\text{KV}} - \lambda C_{\text{KV}} \approx \text{Acc}_{\text{KV}},$$

while storing hidden states incurs

$$U' = \text{Acc}_{\text{hidden}} - \lambda C_{\text{hidden}},$$

for a resource trade-off parameter $\lambda > 0$. In latency-sensitive regimes where λ is large, $U > U'$ holds even when $\text{Acc}_{\text{KV}} < \text{Acc}_{\text{hidden}}$.

Local Decision Adequacy. Both target applications require only *local* discrimination—such as ranking a few candidates or deciding whether to engage in deeper reasoning—rather than globally faithful embeddings. Formally, if ϵ is the probability of local ranking error under KV embeddings, and $\epsilon \ll$ the baseline task error rate, the final performance degradation is bounded by

$$\left| R_{\text{task}}^{\text{KV}} - R_{\text{task}}^{\text{ideal}} \right| \leq \epsilon,$$

where R is task risk. Our experiments confirm that ϵ remains sufficiently small for KV embeddings to maintain competitive performance..

5 Chain of Embedding with KV Cache

5.1 Background

LLMs exhibit emergent reasoning capabilities, though their internal decision processes remain largely opaque. To address this, Wang et al. (2025b) introduce *Chain-of-Embedding* (CoE), a method that probes the model's latent space by tracking the evolution of sentence-level representations across layers. Formally, for an LLM $\mathcal{M}$ with L layers, let $h_l^{(t)}$ denote the hidden representation of token t at layer l. The sentence-level representation at layer l is obtained by by averaging over the sequence length T:

$$s_l = \frac{1}{T} \sum_{t=1}^{T} h_l^{(t)}, \quad l = 0, 1, \ldots, L. \tag{1}$$

The CoE trajectory is then defined as the sequence of these layer-wise representations:

$$\text{CoE} = \{s_0, s_1, \ldots, s_L\}. \tag{2}$$

CoE characterizes this trajectory by measuring both magnitude and directional changes of embeddings between consecutive layers:

$$\Delta r_l = \|s_{l+1} - s_l\|_2, \text{ and } \Delta\theta_l = \arccos\left(\frac{s_{l+1} \cdot s_l}{\|s_{l+1}\| \|s_l\|} \right). \tag{3}$$

These features are aggregated into self-evaluation scores. For instance, the real-space combination (CoE-R) is

$$\text{CoE-R} = \frac{1}{L-1} \sum_{l=0}^{L-1} \left(\alpha \Delta r_l + \beta \Delta\theta_l \right), \tag{4}$$

where α, β are weighting coefficients. A more robust complex-space variant (CoE-C) treats each $(\Delta r_l, \Delta\theta_l)$ pair as a complex number $z_l = \Delta r_l + i\Delta\theta_l$ and computes the magnitude of their average:

$$\text{CoE-C} = \left| \frac{1}{L-1} \sum_{l=0}^{L-1} z_l \right|. \tag{5}$$

CoE has demonstrated strong discriminative power in distinguishing correct from incorrect model generations, achieving state-of-the-art performance on self-evaluation benchmarks.

5.2 Methodology

Our key innovation is to adapt the CoE framework to use the KV cache, eliminating its primary computational overhead. While vanilla CoE constructs trajectories from hidden states $h_l^{(t)}$, requiring expensive activation storage or re-computation, we instead leverage the key-value tensors $K^{(l,t)}, V^{(l,t)}$ that are already maintained by autoregressive decoders. This modification preserves the CoE analytical framework while rendering it virtually cost-free.

Embedding Construction. For each token t and layer l, we start with the cached key-value tensors $K^{(l,t)}, V^{(l,t)} \in \mathbb{R}^{H\times d}$. We flatten the head and key/value dimensions and average across layers to produce a compact per-token embedding:

$$e_t = \frac{1}{L}\sum_{l=1}^{L} \text{flatten}\big(K^{(l,t)}, V^{(l,t)}\big) \in \mathbb{R}^{H\cdot d}. \tag{6}$$

The resulting token-wise trajectory is defined as:

$$\text{KV-CoE} = \{e_1, e_2, \ldots, e_T\}, \tag{7}$$

which directly parallels the structure of vanilla CoE but operates along the token dimension.

Trajectory Characterization. We characterize this trajectory using the established CoE metrics, simply substituting the token index t for the layer index l:

$$\Delta r_t = \|e_{t+1} - e_t\|_2, \text{ and } \Delta\theta_t = \arccos\Big(\tfrac{e_{t+1}\cdot e_t}{\|e_{t+1}\|_2\,\|e_t\|_2}\Big), \tag{8}$$

$$\text{KV-CoE-R} = \tfrac{1}{T-1}\sum_{t=1}^{T-1}\big(\alpha\,\Delta r_t + \beta\,\Delta\theta_t\big), \text{ and } \text{KV-CoE-C} = \Big|\tfrac{1}{T-1}\sum_{t=1}^{T-1}(\Delta r_t + i\,\Delta\theta_t)\Big|. \tag{9}$$

These formulations maintain the analytical rigor of CoE-R and CoE-C with minimal conceptual alteration.

Contributions and Advantages. As illustrated in Figure 2, which compares vanilla CoE and our **KV-CoE**, our method offers two main advantages:

1. **No extra activation cost.** Since the KV cache is already computed and stored during standard autoregressive decoding, reusing it for trajectory analysis incurs virtually no additional activation cost. The required reductions are computationally negligible compared to a full forward pass, resulting in $\Delta M \approx 0$ extra memory and minimal FLOPs.
2. **Deployment-friendly.** The approach works directly with standard inference stacks (e.g., `past_key_values` in Transformers or vLLM). It requires no architectural changes, re-forwarding, or activation hooks, making it immediately deployable in production LLM serving systems.

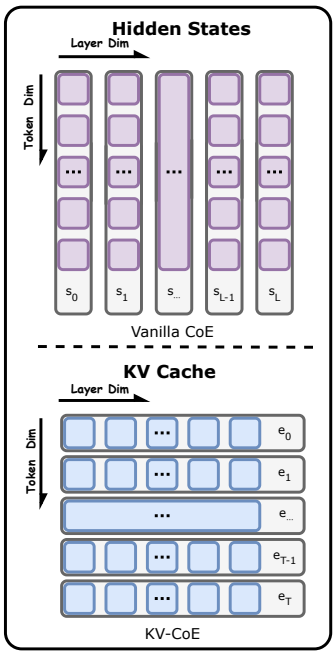

Figure 2: Comparison between vanilla CoE (top) and **KV-CoE** (bottom). Vanilla CoE aggregates hidden states across the token dimension to construct a representation for each layer, whereas KV-CoE aggregates KV Cache across the layer dimension to construct a representation for each token.

5.3 Experimental Results

We evaluate **KV-CoE** on two reasoning benchmarks from the original CoE work: MATH (Hendrycks et al., 2021) for multi-step arithmetic and TheoremQA (Chen et al., 2023) for theorem proving. Experiments are conducted on two popular instruction-tuned models: Llama-3.1-8B-Instruct (LlamaTeam, 2024) and Qwen2-7B-Instruct (QwenTeam, 2024).

We construct embeddings directly from the KV cache by extracting value vectors at every layer, concatenating across attention heads, and averaging over layers to obtain one embedding per token, all without storing hidden states. This reuse of cached information introduces negligible VRAM overhead and incurs minimal FLOPs consumption compared to vanilla CoE.

Model	Method	MATH		TheoremQA	
		AUROC ↑	FPR95 ↓	AUROC ↑	FPR95 ↓
Llama-3.1-8B-Instruct	MaxProb	59.16	87.96	45.41	98.60
	PPL	60.82	86.42	46.45	97.82
	Entropy	62.74	84.14	47.37	97.82
	CoE-R† (Llama3-8B)	72.54	75.61	63.12	89.83
	CoE-C† (Llama3-8B)	73.08	79.60	55.85	90.14
	KV-CoE-R (ours)	**64.36**	**63.82**	74.74	62.93
	KV-CoE-C (ours)	64.13	67.42	**74.93**	**62.46**
Qwen2-7B-Instruct	MaxProb	12.40	99.34	4.92	99.87
	PPL	12.43	99.50	5.11	100.00
	Entropy	16.19	99.42	5.28	99.87
	CoE-R	75.75	65.95	66.68	85.84
	CoE-C	76.68	64.48	62.70	87.42
	KV-CoE-R (ours)	76.92	49.83	**88.87**	**54.30**
	KV-CoE-C (ours)	**84.12**	**44.82**	83.27	58.35

Table 2: Self-evaluation results on reasoning tasks. KV-CoE consistently improves AUROC and reduces FPR95 relative to MaxProb, PPL, and Entropy. Bold indicates the best value per model–dataset pair except CoE baselines. CoE-R and CoE-C results are taken from the original CoE paper (Wang et al., 2025b). †These baseline results are reported on Llama3-8B-Instruct, while our experiments use the updated Llama3.1-8B-Instruct, so the numbers may not perfectly align.

Analysis. As shown in Table 2, KV-CoE substantially outperforms baselines such as MaxProb, PPL, and Entropy on both MATH and TheoremQA. This demonstrates that the Chain-of-Embedding approach retains its strong discriminative power even when using KV cache-derived trajectories instead of hidden states. The token-level evolution captured by the KV cache provides a rich signal for identifying correct reasoning paths, especially in multi-step problems, all while adding negligible overhead since the cache is inherently available.

6 Fast/Slow Thinking Switching with KV Cache

6.1 Background

Large Reasoning Models (LRMs) can operate in two modes: *fast thinking*, which produces short, direct answers, and *slow thinking*, which generates detailed, step-by-step reasoning chains (Yao et al., 2023; Lightman et al., 2024). Although slow thinking enhances reliability on complex tasks, it incurs substantial computational overhead by producing significantly more tokens. For example, on GSM8K (Cobbe et al., 2021), Qwen3-32B (QwenTeam, 2025) slow thinking yields a marginal improvement in accuracy (0.95 vs. 0.94) while generating nearly four times the tokens, drastically increasing latency and cost as shown in Figure 3. This inefficiency motivates **adaptive reasoning**, where slow thinking is triggered selectively based on problem difficulty.

6.2 Methodology

We propose a method for adaptive reasoning that selects between fast and slow thinking on a per-instance basis to minimize unnecessary computation while maintaining accuracy. Our approach leverages the **KV cache** from the prompt encoding phase to make this decision, eliminating the need for additional forward passes.

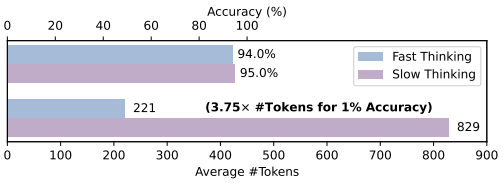

Figure 3: Comparison of efficiency and effectiveness of fast vs. slow thinking on GSM8K using Qwen3-32B. Slow thinking achieves slightly higher accuracy but at a much higher token cost.

Key Idea. Instead of predicting a binary "slow or fast" mode, we estimate a continuous difficulty score $d \in [0, 100]$ from the pooled KV cache representation:

$$d = f_\theta\Big(\text{Pool}\Big(KV_{1:T}^{(1:L)}\Big)\Big),$$

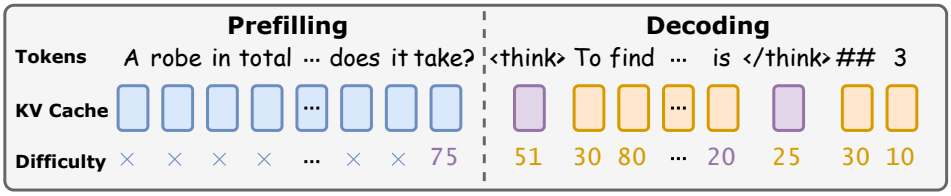

Figure 4: KVClassifier: special tokens are dynamically inserted to perform thinking-mode switching based on KV-derived difficulty scores.

where $\text{Pool}(\cdot)$ aggregates keys and values across layers, heads, and token positions via mean pooling, and $f_\theta(\cdot)$ is a lightweight MLP classifier. This score determines whether to engage slow thinking.

Switching Mechanism. We control the reasoning mode by injecting special control tokens (`<think>` and `</think>`) during decoding:

- **Initial Decision:** Before generation starts, d is compared to a predefined threshold τ:
  - If $d > \tau$, prepend `<think>` to trigger slow thinking.
  - Otherwise, proceed with fast thinking.
- **Dynamic Adjustment During Decoding:** During generation, the difficulty score is recomputed from the updated KV cache at predefined checkpoints:
  - If $d < \tau_{\text{fast}}$ during slow thinking, append `</think>` to switch back to fast mode.
  - If $d > \tau_{\text{slow}}$ during fast mode, inject `<think>` to re-engage slow thinking and continue decoding with step-by-step reasoning.

This approach enables a fine-grained, difficulty-aware control over reasoning depth. Since the KV cache is already available from prompt encoding, both initial and ongoing difficulty assessments add negligible overhead. This significantly reduces token generation and latency for easy queries while allocating more resources to challenging problems. The overall workflow is illustrated in Figure 4.

Training Data Construction. To train the difficulty estimator $f_\theta(\cdot)$, we construct supervision signals from public reasoning datasets (training splits of GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021)). For each training question, we generate two candidate solutions using the base model: a *fast thinking* response (no chain-of-thought) and a *slow thinking* response (with chain-of-thought). We then extract the final answers and compare them against the ground truth.

Based on the outcomes, we assign a difficulty label $d \in \{0, 25, 75, 100\}$ reflecting the required reasoning depth:

- $d = 0$ (Very easy): Fast-thinking answer is correct and short (< 128 tokens).
- $d = 25$ (Moderate): Fast-thinking answer is correct but long (≥ 128 tokens).
- $d = 75$ (Hard): Fast-thinking answer is wrong, Slow-thinking answer is correct.
- $d = 100$ (Very hard): Both answers are incorrect.

This labeling scheme creates a natural difficulty progression, enabling $f_\theta(\cdot)$ to learn a smooth score that correlates with both correctness and reasoning effort. The token-length criterion distinguishes trivial questions from those needing lengthy outputs without explicit reasoning. The trained estimator provides the continuous difficulty score needed for our adaptive switching mechanism.

6.3 Experimental Results

Setup. We evaluate our KV-cache-based fast/slow thinking mechanism on two reasoning benchmarks: GSM8K evaluation split (Cobbe et al., 2021) and MATH500 (OpenAI / HuggingFaceH4 / Vals AI, 2025). Our experiments compare two switching strategies:

- **One-step switch (KV-Classification):** This strategy makes a single decision at generation start based on the predicted difficulty score, committing to either slow or fast thinking for the entire decoding process. It functions as a *classification-style* controller.

Dataset	Method	DeepSeek-R1-14B	Qwen3-8B
GSM8K	Fast Thinking	0.845 / 218	0.904 / 211
	Reasoning	0.847 / 432	0.933 / 1632
	KV-Classification	0.845 / 218 -49.5%	0.914 / 554 -66.1%
	KV-Generative	0.835 / 242 -44.0%	0.892 / 273 -83.3%
MATH500	Fast Thinking	0.536 / 540	0.568 / 616
	Reasoning	0.590 / 1839	0.610 / 4150
	KV-Classification	0.578 / 1506 -18.1%	0.604 / 3963 -4.5%
	KV-Generative	0.566 / 657 -64.3%	0.578 / 727 -82.5%

Table 3: Comparison of accuracy and average token usage for fast thinking, full reasoning, and our KV-cache-based switching methods on GSM8K and MATH500. For each KV-based method, we report the result from best hyper-parameter configuration identified in Appendix B.

- **Two-step switch (KV-Generative):** This method performs an initial mode selection and continuously monitors difficulty during decoding. If difficulty drops below τ_{fast} during slow thinking, it appends `</think>` to terminate reasoning early; if difficulty exceeds τ_{slow} during fast thinking, it injects `<think>` to engage slow thinking mid-generation. This implements a *generative-style* controller that dynamically adjusts reasoning depth.

We deploy both strategies on two open-weight models: DeepSeek-R1-14B (DeepSeek-AI, 2025) and Qwen3-8B (QwenTeam, 2025), evaluating their ability to selectively trigger slow thinking and reduce unnecessary token generation.

We construct representations by concatenating key and value tensors across all heads, summing over selected token positions, and averaging across selected layers without normalization, then feed the result into a two-layer MLP (hidden dimension 512, ReLU activation) for difficulty prediction. This design directly reuses the KV cache available during decoding, introduces negligible VRAM or FLOPs overhead, and functions as a modular component that can be seamlessly integrated into existing inference pipelines without modifications to the base model.

Analysis. As shown in Table 3, our KV-cache-based switching approach achieves an effective balance between accuracy and efficiency. For instance, on MATH500 using Qwen3-8B, two-step generative switching reduces average token count from 4,150 (full reasoning) to 727 ($5.7\times$ reduction) with only a minimal 3.2% accuracy drop. The one-step classification strategy is more conservative, using more tokens but achieving near-full-reasoning accuracy (0.604 vs. 0.610). Similar trends are observed on GSM8K, where KV-cache-based switching maintains high accuracy (up to 0.914) while significantly reducing token consumption compared to full reasoning. These results demonstrate that difficulty scores derived from the KV cache generalize well across tasks and models, enabling efficient and effective adaptive reasoning with negligible overhead.

7 Conclusion

This work repurposes the KV cache, moving beyond its conventional role in decoding acceleration to unlock its potential as a versatile, cost-free representation. We demonstrate that although not designed as general-purpose embeddings, KV caches encode rich contextual information that can be effectively utilized for downstream tasks without incurring additional computational overhead. Our experiments establish two practical applications: (i) **Chain-of-Embedding**, where KV-derived embeddings match or surpass the performance of hidden-state embeddings, and (ii) **Fast/Slow Thinking Switching**, which uses KV-cache-based difficulty scores to enable adaptive reasoning-reducing token usage by up to $5.7\times$ with minimal accuracy loss. These findings position the KV cache as a deployment-friendly substrate for advanced inference techniques, opening new avenues for reusing inference-time artifacts to improve efficiency and controllability in LLMs.

REFERENCES

Mohammad Beigi, Ying Shen, Runing Yang, Zihao Lin, Qifan Wang, Ankith Mohan, Jianfeng He, Ming Jin, Chang-Tien Lu, and Lifu Huang. InternalInspector i^2: Robust confidence estimation in LLMs through internal states. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 12847–12865, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.751. URL https://aclanthology.org/2024.findings-emnlp.751/.

Max Belitsky, Dawid J. Kopiczko, Michael Dorkenwald, M. Jehanzeb Mirza, Cees G. M. Snoek, and Yuki M. Asano. Kv cache steering for inducing reasoning in small language models, 2025. URL https://arxiv.org/abs/2507.08799.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Chao Chen, Kai Liu, Ze Chen, Yi Gu, Yue Wu, Mingyuan Tao, Zhihang Fu, and Jieping Ye. INSIDE: llms' internal states retain the power of hallucination detection. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Zj12nzlQbz.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A theorem-driven question answering dataset. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7889–7901, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.489. URL https://aclanthology.org/2023.emnlp-main.489.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Chun Liu, Hongguang Zhang, Kainan Zhao, Xinghai Ju, and Lin Yang. LLMEmbed: Rethinking lightweight LLM's genuine function in text classification. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7994–8004, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.433. URL https://aclanthology.org/2024.acl-long.433/.

AI@Meta LlamaTeam. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. MTEB: Massive text embedding benchmark. In Andreas Vlachos and Isabelle Augenstein (eds.), *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 2014–2037, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.eacl-main.148. URL https://aclanthology.org/2023.eacl-main.148.

Ollama Team. Ollama: Open llm deployment made simple. https://ollama.ai, 2024.

OpenAI / HuggingFaceH4 / Vals AI. Math-500: A 500-problem subset of the math benchmark. HuggingFace / Vals AI Benchmark / Datasets, 2025. URL https://huggingface.co/datasets/HuggingFaceH4/MATH-500. Derived from "Measuring Mathematical Problem Solving With the MATH Dataset"; subset of 500 test problems.

Alibaba Group QwenTeam. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

Alibaba Group QwenTeam. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Yi Wang, Junxiao Liu, Shimao Zhang, Jiajun Chen, and Shujian Huang. Pats: Process-level adaptive thinking mode switching, 2025a. URL https://arxiv.org/abs/2505.19250.

Yiming Wang, Pei Zhang, Baosong Yang, Derek Wong, and Rui Wang. Latent space chain-of-embedding enables output-free llm self-evaluation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Representation Learning*, volume 2025, pp. 70938–70970, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/b0b1cfc8ede53f452cabf8b9cf4eef76-Paper-Conference.pdf.

Zeyu Xing, Xing Li, Huiling Zhen, Xianzhi Yu, Mingxuan Yuan, and Sinno Jialin Pan. Large reasoning models know how to think efficiently. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*, 2025. URL https://openreview.net/forum?id=pLKDeGm2t1.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Murong Yue, Wenlin Yao, Haitao Mi, Dian Yu, Ziyu Yao, and Dong Yu. Dots: Learning to reason dynamically in llms via optimal reasoning trajectories search. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Representation Learning*, volume 2025, pp. 37976–37997, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5e5d6f9ac33ba9349ba7b2be9f21bad9-Paper-Conference.pdf.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. In *Second Conference on Language Modeling*, 2025a. URL https://openreview.net/forum?id=O6I0Av7683.

Xiaoyun Zhang, Jingqing Ruan, Xing Ma, Yawen Zhu, Haodong Zhao, Hao Li, Jiansong Chen, Ke Zeng, and Xunliang Cai. When to continue thinking: Adaptive thinking mode switching for efficient reasoning, 2025b. URL https://arxiv.org/abs/2505.15400.

A THE USE OF LARGE LANGUAGE MODELS (LLMS)

We acknowledge the use of a Large Language Model (LLM) to support the preparation of this manuscript. The LLM was employed exclusively for editorial purposes, such as refining the clarity of exposition, improving grammar and readability, and polishing the overall presentation. At times, it was also used to suggest alternative phrasings for technical descriptions in order to make the arguments more accessible.

Importantly, the LLM did not contribute to the conceptual development, methodology, or experimental design of this work. All ideas, including the proposal to treat the KV cache as a reusable representation, the development of KV-CoE for output-free self-evaluation, and the design of KV-based adaptive Fast/Slow Thinking Switching for token-efficient reasoning, were conceived and implemented solely by the authors. The LLM was not used to generate research results, proofs, or derivations.

All scientific claims, analyses, and conclusions presented in this paper remain the full responsibility of the authors. Any text initially produced with LLM assistance was carefully reviewed, revised, and verified prior to inclusion.

B HYPER-PARAMETER SELECTION FOR KVCLASSIFIER

To better understand the effect of hyper-parameters on KV-based classification, we conduct a systematic study by varying the number of layers pooled and the number of tokens selected from the end of the sequence. Importantly, we fix the *total amount of KV data* to be approximately constant across configurations (256 token $\times$ layer units). This ensures a fair comparison: for example, selecting 8 layers with 32 tokens, 4 layers with 64 tokens, or 2 layers with 128 tokens all yield the same KV budget.

Model	Dataset	Method	8L, Len=32	4L, Len=64	2L, Len=128
DeepSeek-14B	GSM8K	KV-Classification	0.845 / 218	0.845 / 218	**0.845** / 218
		KV-Generative	**0.835** / 242	0.825 / 232	0.805 / 217
	MATH500	KV-Classification	0.536 / 540	0.550 / 905	**0.578** / 1506
		KV-Generative	0.538 / 524	0.550 / 544	**0.566** / 657
Qwen3-8B	GSM8K	KV-Classification	0.904 / 211	0.904 / 217	**0.914** / 554
		KV-Generative	**0.892** / 273	0.886 / 276	0.881 / 257
	MATH500	KV-Classification	0.570 / 736	0.598 / 3673	**0.604** / 3963
		KV-Generative	**0.578** / 727	0.524 / 933	0.550 / 837

Table 4: Hyper-parameter selection results for KV-Classification and KV-Generative. Values are reported as Accuracy / #Tokens. Best accuracy for each dataset–method pair is in bold.

Table 4 summarizes the results on GSM8K and MATH500 for both DeepSeek-R1-14B and Qwen3-8B, under KV-Classification and KV-Generative settings. We observe that while performance varies slightly with the allocation of layer vs. token depth, the overall trends are consistent: (i) accuracy remains competitive across different allocations, and (ii) increasing token coverage (e.g., 2L $\times$ 128) tends to favor more complex datasets such as MATH500, whereas shallow but wider layer coverage (e.g., 8L $\times$ 32) can suffice for GSM8K.