Exploring Multi-level Caching Techniques for Multimodal LLMs

Muzhe Wu [1] **Yuxuan Liu** [1] **Arnav Reddy** [1] **Nishant Dash** [1]

Abstract

Current Multimodal Large Language Models (MLLM) inference struggles with long-context inputs (e.g., video, sequences of visually similar images), resulting in significant computation and undesirable user experience (e.g., lack of proactiveness, repeated prompting and context capturing). While existing caching strategies such as KV caching improves efficiency related to decoding, it does not address the dominant costs of repeatedly re-encoding visually similar inputs across requests.

Beyond KV caching, additional semantic signals may further enhance inference efficiency. We introduce a Multi-Level Caching (MLC) system that combines varied lookup and inference reuse strategies. MLC performs hierarchical lookup across increasingly expensive similarity signals, ranging from exact text matching, semantic text similarity, to embedding-based retrieval over multimodal representations.

We implement MLC on top of the vLLM serving framework and show our system's performance in a vision-only context as well as in a multimodal question and answer setting. We present preliminary results showing the promises and drawbacks MLC can impose. Our source code is available at: https://github.com/arnavrUM/585-Vision-model-caching.

1. Introduction

Multimodal large language models (MLLMs) have rapidly emerged as a unifying interface for perception and language, enabling systems that answer questions about images, follow instructions grounded in visual context, and reason over long-form visual streams such as videos and interactive camera captures.

Recent model families that combine a vision encoder with an autoregressive language decoder have enabled visually grounded assistants that can follow instructions and answer questions about visual context (Alayrac et al., 2022; Li et al., 2023; Liu et al., 2023). As these models transition from offline benchmarks to real-world, multi-turn usage, inference efficiency becomes a first-order concern: users expect low-latency responses while repeatedly referencing the same scene, asking follow-up questions, or providing consecutive frames of a slowly changing environment such as seen in Figure 1.

Figure 1. The image on the left exhibits only minor differences compared to the image on the right. A comprehensive caching system should identify both images as similar and enable some reuse.

The compute cost of modern MLLMs is largely driven by Transformer-based decoding (Vaswani et al., 2017). Serving optimizations such as the key–value (KV) cache reduces repeated attention computation within an autoregressive generation by reusing past attention states. On the systems side, modern inference engines improve throughput and memory efficiency via optimized batching and KV memory management (Kwon et al., 2023), while kernel-level advances reduce the memory overhead of attention (Dao et al., 2022). These improvements are essential, but they largely target *within-request* efficiency. In contrast, many multimodal deployments exhibit substantial *cross-request* redundancy: successive queries often include the same image (or near-identical frames), and multi-turn prompts frequently reuse large portions of the context with only minor edits.

This paper is motivated by a simple observation: current MLLM inference pipelines repeatedly re-encode visually or semantically similar inputs, performing redundant computation that can dominate end-to-end latency. For example, consecutive images from the same scene commonly differ only in small regions, and users often ask multiple questions about the same image. Even when inputs are not exactly identical, they may be semantically equivalent (paraphrases) or close in a learned embedding space. Exact-match prompt caching can only exploit a narrow subset of these cases,

while KV caching alone does not address re-encoding of the visual context or reuse across separate requests.

We propose **Multi-Level Caching (MLC)**, a hierarchical caching framework that targets redundancy at multiple abstraction levels and execution stages of MLLM inference, and integrates directly with modern serving engines such as vLLM (Kwon et al., 2023). We organize caching into a hierarchy with the following layers:

- **L0.5: Exact Input Cache.** Fast normalized string matching (e.g., case/whitespace normalization) to identify duplicate queries with minimal lookup overhead.
- **L1: Text Semantics Cache.** Sentence-level embedding similarity to detect semantically equivalent queries, trading higher hit rate for additional embedding and retrieval cost.
- **L2: Multimodal Similarity Cache.** Retrieval over latent embeddings extracted from intermediate model layers (e.g., vision and prompt encoders) using approximate nearest-neighbor indexing to capture visually/semantically similar inputs (Johnson et al., 2019).

Given a new request, the pipeline first queries **L0.5 (Exact Text Cache)** by applying lightweight normalization (e.g., case and whitespace) and checking for a verbatim match. If **L0.5** misses, the system queries **L1 (Text Semantics)**, which embeds the textual prompt and retrieves the nearest cached prompt under cosine similarity. If the best match exceeds a threshold τ_{text}, the request is considered a hit at the text-semantic level. If **L1** also misses, the system falls back to **L2 (Multimodal Semantics)**, which performs similarity search over learned latent representations extracted from both the prompt encoder and the vision encoder. A hit occurs when the prompt and/or visual similarity exceeds their respective thresholds (e.g., τ_{prompt}, τ_{vision}).

This hierarchy is designed so that the cheaper, more reliable caches fire first, while more expensive similarity-based caches provide additional coverage when exact matches are absent.

2. Background and Motivation

2.1. Limits of Existing Caching Techniques

Traditional Transformer-based language decoders generate output tokens auto-regressively (Vaswani et al., 2017). Naively, attention computations are quadratic with respect to the sequence length making long-context inference prohibitively expensive. Key-Value (KV) caching aims to address this inefficiency by storing the key and value tensors produces at each attention layer for past tokens, allowing subsequent decoding steps to reuse these tensors rather than recompute them (Pope et al., 2022).

KV caching is now standard in modern inference engines such as vLLM (Kwon et al., 2023). By eliminating redundant attention computation within a single request, KV caching significantly reduces decoding latency and memory bandwidth usage (Pope et al., 2022).

Despite its effectiveness, KV caching is fundamentally limited to within-request reuse. Once a request completes, its cached KV states are typically discarded or reclaimed and not reused across requests.

Prefix caching extends on KV caching by enabling reuse of KVs across multiple requests (Gim et al., 2024). If a new request shares an identical prefix with a cached prompt, the corresponding KV states are injected into the model, allowing inference to resume from a later point in the sequence.

In multimodal models, KV caching and prefix caching fail to address a dominant source of inference cost: the visual encoding. In most MLLM architectures, images are first process by a vision encoder (e.g., a Vision Transformer) whose outputs are then projected into the language model's embedding space (Alayrac et al., 2022; Li et al., 2023; Liu et al., 2023). This computation occurs before auto-regressive decoding begins and therefore is not covered by the KV cache and prefix caching. As a result, even if two requests reference the same image, the vision encoder and multimodal fusion layers are re-executed for each request.

Other techniques explore storing whole responses rather than intermediate states. Exact prompt matching is where a cached response is returned when an identical prompt is provided similar to a traditional database or caching system. However, it is typically rare to see the exact same prompt between requests. This led to semantic caching for text based models which converts whole prompts to vectors and compares them to previous prompts using a similarity metric and if there is a match, the cached response is returned (Schroeder et al., 2025).

Existing research on semantic caching has been restricted to text-only contexts so we have explored this caching strategy in multimodal scenarios.

2.2. Unique Challenges of MLLMs

Caching for multimodal language models is accompanied with fundamental challenges which do not apply in text-only language models. Multimodal inputs combine high-dimensional visual representations alongside language, producing continuous representations (Dosovitskiy, 2020) as opposed to discrete textual representations. As a result, multimodal workloads are rarely exact and must be correctly grouped through some notion of similarity. These properties increase the complexity of caching for MLLM's, ultimately requiring careful consideration of the conditions under which reuse is acceptable.

Text-only language models operate over discrete token sequences (Vaswani et al., 2017; Brown et al., 2020) and thus, two prompts that share an identical token prefix will produce identical partial intermediate representations (assuming deterministic execution and identical model states). This makes exact prefix or prompt matching a viable method for caching. Vision encoders instead decompose images into patch embeddings, which are represented as high dimensional continuous vectors (Dosovitskiy, 2020; Radford et al., 2021). As a result, visually identical images may produce different embeddings due to floating point effects, variations in resolution or cropping, and many other slight differences. Exact matching as a method of caching becomes ineffective, so caching decisions must instead reason about similarity. However, similarity based caching will introduce approximation, uncertainty, and potential correctness trade-offs.

The expansion of an image into hundreds of patch tokens, through the process of visual encoding, significantly increases the compute and memory requirements compared to lightweight textual tokenization. Similarly, text prompts are typically short and inexpensive to encode whereas visual inputs incur a much higher upfront cost. Therefore, repeated processing of similar image frames will dominate latency, even when the associated textual portions of the query remain the same. Since the visual encoding stage of inference can often be the most expensive, caching opportunities for visual features or representations can be disproportionately valuable.

Semantic similarity can be useful in smoothing minor pixel variations due to changes in lighting, view angles, and sensor noise. However, in multimodal language models, semantic similarity is derived from the interaction between the visual and textual portions of an input. Textual input may focus which regions of an image are relevant, while visual input may supply additional context to under-specified textual components. Thus, caching strategies that only operate on a single modality will have limited effectiveness. Altogether, these considerations motivate semantic caching frameworks that operate across multiple pipeline stages (vision encoding, text encoding, multimodal fusion, decoding) in order to reduce the high inference costs associated with MLLM's.

3. Caching Hierarchy

We introduce a multi-level caching (MLC) framework which decomposes reuse into a hierarchy of cache levels. Each level operates over a different representation of the request and thus offers a different tradeoff between lookup overhead (latency), match precision (accuracy), and reuse coverage (hit rate). Figure 2 gives an overview.

3.1. Separation of Lookup and Inference Reuse

A core principle of MLC is the separation of *lookup* and *inference reuse*. Lookup specifies how cached entries are retrieved, whereas inference reuse specifies what computation is skipped once a cached entry is selected.

Lookup and inference reuse are intentionally decoupled. In practice, the same lookup level can support multiple reuse strategies depending on the execution backend and the fidelity guarantees required. This modularity supports continuous improvement (e.g., swapping in stronger embeddings or indices) without changing downstream reuse mechanisms.

Notation. Let the cache contain entries $\{(x_i, y_i, s_i)\}_{i=1}^{N}$, where x_i is a previously seen input, y_i is its response, and s_i denotes optional reusable inference state (e.g., KV tensors). Given a new request x, each lookup level ℓ defines a representation map $\phi_\ell(\cdot)$, a similarity function $\sigma_\ell(\cdot,\cdot)$, and a threshold τ_ℓ. The best match at level ℓ is

$$\hat{i}_\ell(x) = \arg\max_{i\in\{1,\dots,N\}} \sigma_\ell\big(\phi_\ell(x), \phi_\ell(x_i)\big), \tag{1}$$

and a hit occurs when

$$H_\ell(x) = \Big[\ \sigma_\ell\big(\phi_\ell(x), \phi_\ell(x_{\hat{i}_\ell(x)})\big) \geq \tau_\ell\ \Big]. \tag{2}$$

MLC uses a first-hit policy, selecting the smallest ℓ for which $H_\ell(x) = 1$.

3.2. Multi-Level Lookup Techniques

MLC explores three tiers of lookup that escalate from strict to approximate matching. Earlier levels emphasize precision and minimal overhead; later levels prioritize recall by utilizing semantic similarity in richer representation spaces.

3.2.1. Exact Input Match (L0.5)

L0.5 is an exact-text cache designed to capture repeated prompts with near-zero lookup cost. Incoming prompts are first normalized and then queried against an in-memory hash map. A hit occurs only under exact match, i.e., without approximation.

L0.5 targets exact repeats that commonly arise in user interactions (e.g., retries and template reuse), where a hash lookup is negligible relative to model inference. However, L0.5 is sensitive to small textual variations (e.g., paraphrasing), which can reduce recall.

3.2.2. Text Semantics (L1)

L1 generalizes beyond exact matching by retrieving cached requests based on semantic similarity in a text embedding space. For each request, we compute a representation of

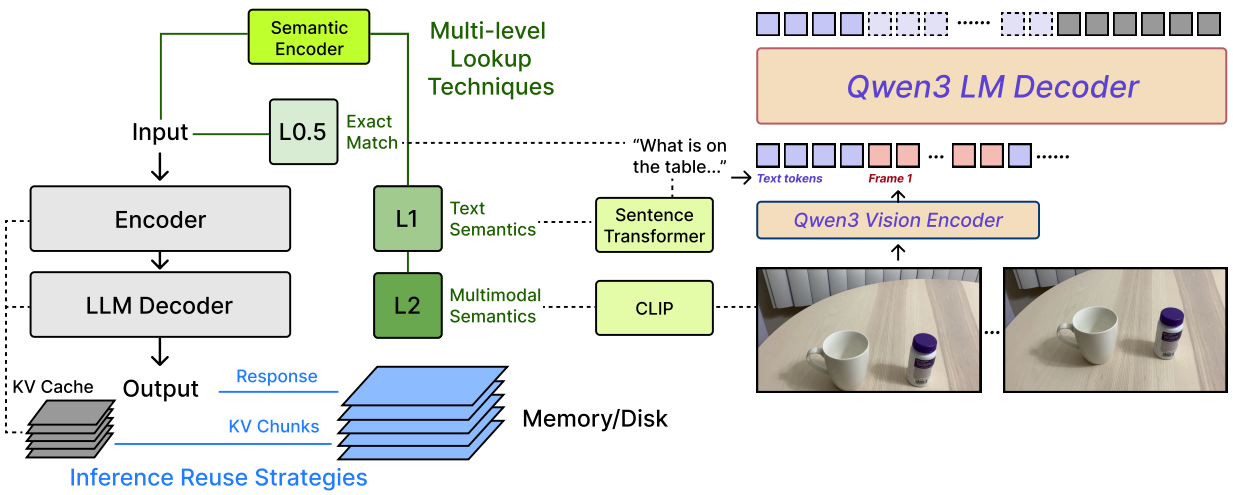

Figure 2. Left: Abstract overview of multi-level caching (MLC), illustrating three lookup layers (L0.5: exact input match, L1: text semantics, L2: multimodal semantics) and two inference reuse strategies (response reuse and KV-chunk caching). Right: Instantiation of MLC in the Qwen3 architecture, shown through a scene description example.

the textual input and retrieve the nearest cached entry under cosine similarity, applying a configurable threshold to gate reuse. This level improves recall by recognizing paraphrases and near-duplicates that exact matching misses, while retaining a tunable precision–recall tradeoff through the similarity threshold. However, because L1 is driven primarily by the linguistic content, it may fail to capture reuse opportunities that arise from repeated or similar visual contexts.

3.2.3. Multimodal Semantics (L2)

L2 extends semantic reuse to a broader embedding space that incorporates richer signals from the multimodal request, enabling reuse based on similarity in the underlying scene or visual context in addition to the prompt text. Intuitively, L2 can recover reuse when the same (or highly similar) image content recurs across requests, even if the wording changes, and can avoid incorrect reuse when prompts are textually similar but grounded in different images. As a result, L2 is better suited to realistic scene-description workloads where redundancy often comes from repeated visual frames and shared environments rather than repeated phrasing alone.

3.3. Inference Reuse Strategies

Given a retrieved cache entry, MLC applies one of two reuse strategies. Both strategies are driven by the same multi-level lookup pipeline; they differ in *what* is reused and therefore in the amount of computation avoided and the coupling to the inference backend.

3.3.1. Response Reuse

Response reuse performs result-level reuse: upon a cache hit, the system returns the previously generated response $y_{\hat{i}(x)}$, bypassing the full model forward pass:

$$\hat{y}(x) = y_{\hat{i}(x)}. \tag{3}$$

This strategy yields large latency reductions and is backend-agnostic (it only requires stored responses). Because it substitutes an entire output, response reuse is most appropriate when the lookup stage provides high-confidence matches (e.g., exact match or conservative similarity thresholds).

3.3.2. KV Chunk Reuse

KV chunk reuse is a compute-level reuse strategy enabled by the transformer KV cache. In autoregressive decoding, each layer stores attention keys and values for previously processed tokens, allowing future decoding steps to attend to the prefix without recomputing those states.

KV chunk reuse extends this mechanism across requests. Upon a cache hit, we retrieve KV tensors (or a chunk aligned to a shared prefix/span) from the matched entry and inject them into the decoding engine. Let $s_{\hat{i}(x)}$ denote the retrieved KV chunk and $\mathrm{Decode}(\cdot;\, s)$ denote decoding conditioned on an injected state s. Then KV chunk reuse can be expressed as

$$\hat{y}(x) = \mathrm{Decode}\big(x;\, s_{\hat{i}(x)}\big), \tag{4}$$

which avoids recomputing attention for the reused portion and reduces prefill and early decoding costs.

KV chunk reuse aims to eliminate repeated computation

Encoder	τ	Acc.	Hit Rate	Time (s)	Speedup
No Cache Baseline	–	0.8897	–	44.12	1.00
ResNet-18	0.875	**0.7488**	**0.7806**	**41.35**	**1.07**
	0.900	**0.7953**	**0.6495**	**40.81**	**1.08**
	0.925	0.8223	0.4338	45.38	0.97
	0.950	0.8762	0.1826	48.11	0.92
MobileNet V2	0.875	0.8125	0.4632	46.33	0.95
	0.900	0.8468	0.3088	47.69	0.93
	0.925	0.8811	0.1544	49.57	0.89
	0.950	0.8873	0.0527	50.95	0.87
EfficientNet B0	0.875	0.8370	0.4056	47.75	0.92
	0.900	0.8762	0.2623	48.94	0.90
	0.925	0.8836	0.1422	50.38	0.88
	0.950	0.8873	0.0748	51.39	0.86

Table 1. Experiment 1: Multimodal Semantics Cache (L2) with three encoders and four similarity thresholds τ. Bold rows highlight trials that achieve speedup (> 1.0) but incur accuracy degradation relative to the baseline.

while preserving the model's inference flow, which can be preferable when response substitution is brittle. In this project, we explored integrating KV chunk reuse with the vLLM serving framework. However, fully enabling KV injection remains an open systems challenge, as the current vLLM execution path does not expose a stable interface for externally supplying KV states for arbitrary requests. Our evaluation therefore focuses on response reuse, and we discuss KV injection support as future work in Section 6.

4. Evaluation

We designed two experiments to evaluate the effectiveness and generalizability of MLC.

4.1. Experiment 1: Image Classification

This preliminary experiment explores response caching in the simplest form, where the system returns previously computed output directly when an incoming request is deemed sufficiently similar to a cached one. The specific task we based this experiment on was a 8-class image classification task. We chose image classification because it provides a clean and minimal setting to illustrate the core idea of response caching, as a cached response (a class label) is cheap to store and the correctness cost of a false hit is immediately observable as an accuracy drop.

Task and model. We fine-tuned a Vision Transformer (ViT) classifier for rice image classification task (Malviya, 2024). Given an input image x, the model outputs a predicted class label $\hat{y} = f_{\text{ViT}}(x)$. We treat each prediction as the "response" to be cached and reused.

Cache Construction. For each processed image x, we compute an embedding $e(x)$ using a lightweight encoder (ResNet-18, MobileNetV2, or EfficientNet-B0) (He et al., 2015; Sandler et al., 2018; Tan & Le, 2019). We store tuples $(e(x), \hat{y})$ in an in-memory index. At inference time, for a new query image x', we compute $e(x')$ and retrieve the nearest cached embedding under cosine similarity. If the best match exceeds a similarity threshold τ, we return the cached label instead of running the ViT forward pass:

$$\hat{y}(x') = \begin{cases} \hat{y}(x) & \text{if } \max_{x \in \mathcal{C}} \cos(e(x'), e(x)) \geq \tau, \\ f_{\text{ViT}}(x') & \text{otherwise.} \end{cases}$$

Here $\mathcal{C}$ denotes the set of cached entries. We report *hit rate* as the fraction of queries that satisfy the threshold and are served by the cache. *Speedup* is computed as baseline time/cache time (larger is better).

Experimental Setup. We evaluate (i) a no-cache inference baseline that runs the ViT on every input, and (ii) an inference pipeline augmented with the L2 Multimodal Semantics. We tested four similarity thresholds $\tau \in \{0.875, 0.90, 0.925, 0.95\}$ and three embedding encoders on a dataset of 816 rice images. For each configuration, we measure end-to-end runtime over the dataset, along with classification accuracy and cache hit rate (Table 1).

Results and Analysis. Table 1 shows a clear accuracy-hit rate trade-off. Lower thresholds produce higher hit rates (e.g., up to 0.78 for ResNet-18 at τ=0.875) but substantially reduce accuracy, indicating frequent false hits where the nearest neighbor corresponds to a different label. Raising τ reduces the hit rate sharply (e.g., ResNet-18 drops from 0.65 at τ=0.90 to 0.18 at τ=0.95), and accuracy approaches the no-cache baseline, but this diminishes the purpose of caching.

In terms of latency, caching only yields speedups in limited conditions. The only configurations that outperform the baseline are ResNet-18 at τ=0.875 and τ=0.90 (speedup $1.07\times$ and $1.08\times$), but both come with significant accuracy tradeoff (bold rows in Table 1). For higher thresholds, overall runtime is worse than the baseline (speedup < 1), which suggests that the overhead of computing embeddings and performing similarity search can offset (or exceed) the saved ViT forward passes once cache hits become rare.

These findings expose a key limitation of the caching mechanism in Experiment 1: the unit of reuse is the final output, so any approximate match can directly return an incorrect label. To improve latency, the cache must raise its hit rate, often by lowering the similarity threshold. However, this increases false hits and hurts accuracy. Raising the threshold reduces false hits, but hit rates drop and the augmented caching overhead can outweigh the saved ViT forward passes. This accuracy–latency tension motivates reuse strategies that cache and reuse intermediate computation (e.g., KV chunks) rather than final responses, which we explore in Experiment 2.

We also note that Experiment 1 evaluates L2 Multimodal Semantics on a dataset comprised of independent rice images, rather than sequences of related images where near-duplicates are common. In real multimodal assistant workloads, consecutive frames from a camera stream often exhibit far higher redundancy, and user interactions commonly involve repeated or slightly edited prompts grounded in similar visual contexts. This suggests that cross-request reuse opportunities may be larger in streaming or multi-turn settings than in our image-classification setup. Motivated by this, we did experiment 2 which introduces a multiple layer caching pipeline designed for such redundant multimodal workloads (Experiment 2 in Section 4.2).

4.2. Experiment 2: Scene Description

Experiment 2 evaluates MLC under a more realistic and challenging workload, i.e., scene description, which involves both textual and visual inputs and requires more nuanced semantic reuse.

Implementation. We test two series of models, Qwen3-VL-Instruct and InternVL3_5-Instruct, using a Hugging Face Transformers inference backend. MLC is implemented as a hierarchical cache with all proposed lookup levels and response reuse. For text semantics caching (L1), we encode chunk text using Sentence-Transformers `sentence-transformers/all-MiniLM-L6-v2` and index embeddings with FAISS `IndexFlatIP`. For multimodal semantics caching (L2), we extract prompt embeddings using the same `all-MiniLM-L6-v2` encoder (384-d) and image embeddings using a CLIP-style Sentence-Transformer (`clip-ViT-B-32`, 512-d), again indexed with FAISS `IndexFlatIP`. We ran experiments on a server with one RTX 6000 Ada GPU.

Dataset. We evaluate MLC on the GQA dataset, featuring paired images and natural language questions that require fine-grained scene understanding. GQA dataset reflects our target workflow of repeated multimodal queries over shared visual contexts. We sample 1024 datapoints from the validation split for controlled evaluation. We observed that the GQA benchmark's answer matching criteria were overly restrictive, failing to recognize equivalent responses. To address this limitation, we re-evaluated the question-answer pairs using GPT-o3-mini as an alternative grader.

Technique Comparison Results. Table 2 compares different multi-level caching (MLC) techniques across similarity thresholds using Qwen-VL-2B and InternVL-3.5-4B. Overall, all caching techniques reduce inference latency relative to the no-cache baseline, confirming the effectiveness of MLC for accelerating multimodal inference. Exact-text caching achieves modest speedup while preserving accuracy (improvement might not be systematic), despite a low hit rate (7.8%). This indicates that even sparse exact matches can yield safe latency gains without compromising output quality.

Technique	τ_t	τ_v	Qwen			InternVL		
			Latency	Acc.	Hit Rate	Latency	Acc.	Hit Rate
No Cache Baseline	–	–	0.606	0.313	–	0.190	0.295	–
Exact-text	–	–	0.562	0.331	0.078	0.210	0.299	0.078
Semantic	0.6	–	0.056	0.025	0.927	0.018	0.304	0.927
	0.7	–	0.159	0.050	0.786	0.047	0.047	0.786
	0.8	–	0.330	0.112	0.506	0.108	0.118	0.506
	0.9	–	0.493	0.214	0.208	0.179	0.152	0.208
Embedding	0.7	0.7	0.120	0.306	0.821	0.033	0.236	0.853
	0.7	0.8	0.173	0.085	0.745	0.045	0.289	0.804
	0.7	0.9	0.185	0.114	0.732	0.047	0.105	0.793
	0.8	0.7	0.170	0.124	0.719	0.059	0.119	0.740
	0.8	0.8	0.300	0.129	0.523	0.095	0.129	0.591
	0.8	0.9	0.344	0.203	0.460	0.108	0.143	0.543
	0.9	0.7	0.223	0.232	0.643	0.081	0.175	0.647
	0.9	0.8	0.388	0.153	0.354	0.147	0.197	0.370
	0.9	0.9	0.473	0.253	0.228	0.175	0.164	0.255

Table 2. Ablated multi-level lookup technique comparison showing different tradeoffs between latency, accuracy and hit rate at different thresholds (τ_t: prompt similarity; τ_v: visual similarity).

Both semantic and embedding-based lookup exhibit consistent and predictable tradeoff patterns as thresholds vary (Figure 3). Lower thresholds result in higher hit rates and substantial latency reductions, but incur significant accuracy degradation due to aggressive reuse. This homogeneous trend holds across both models and lookup types. This suggest that it is critical to select thresholds that balance between performance gains and satisfying output quality.

Model Comparison Results. We next examine how the effectiveness of MLC varies across models of different sizes. As larger models incur higher inference costs yet produces more accurate and stable outputs, it is interesting to see how the benefit of response caching on latency reduction can be amplified. For this experiment, we enable all lookup techniques and fix all similarity thresholds at 0.8, informed by the previous ablation study as a balanced tradeoff between latency reduction and accuracy preservation.

As shown in Table 3, MLC consistently reduces inference latency across all evaluated models, yielding speedups ranging from $1.75\times$ to $2.48\times$. Notably, larger models benefit more from caching: Qwen3-VL-8B achieves a $2.48\times$ speedup, compared to $1.75\times$ for Qwen3-VL-2B, reflecting stronger amortization effects when bypassing expensive forward passes. A similar pattern is observed for InternVL models, where speedups reach maximum for InternVL-3.5-8B.

Accuracy degradation introduced by MLC varies systematically with model capacity. Larger models experience smaller relative accuracy drops (e.g., 22.8% for Qwen3-VL-8B) compared to their smaller counterparts, suggesting

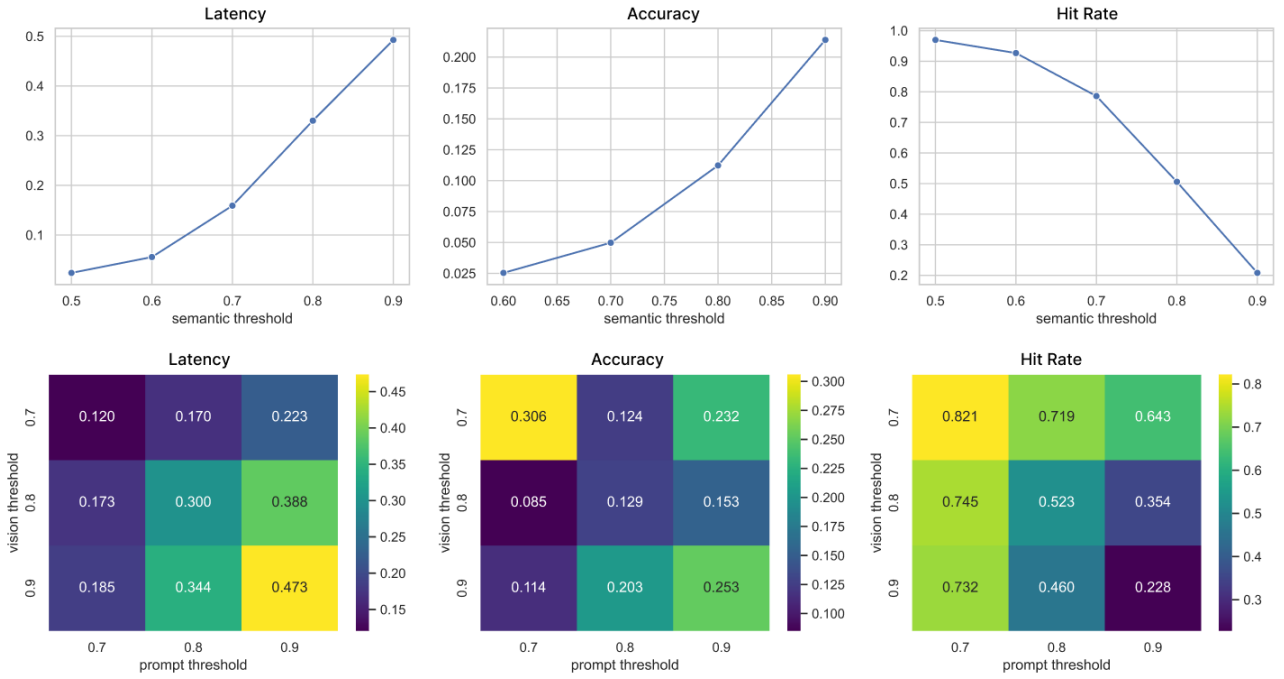

Figure 3. Plots of latency, accuracy, and cache hit rate across different similarity thresholds for semantic and embedding caching techniques on Qwen; InternVL exhibits similar trends.

Model	w/o MLC		w/ MLC			Speedup	Acc. Drop
	Latency	Acc.	Latency	Acc.	Hit Rate		
Qwen3-VL 2B	0.606	0.313	0.347	0.205	0.516	1.75×	34.5%
Qwen3-VL 4B	0.415	0.355	0.208	0.240	0.516	1.99×	32.4%
Qwen3-VL 8B	0.908	0.412	0.366	0.318	0.516	2.48×	22.8%
InternVL 3.5 2B	0.221	0.324	0.095	0.225	0.547	2.33×	30.6%
InternVL 3.5 4B	0.529	0.354	0.251	0.209	0.547	2.11×	40.9%
InternVL 3.5 8B	0.191	0.295	0.080	0.213	0.547	2.39×	27.8%

Table 3. End-to-end performance comparison with and without multi-level caching across different multimodal models. Metrics report average latency, accuracy, and cache hit rate.

that higher-capacity models are more robust to approximate reuse.

Together, these results suggest that MLC scales favorably with model size, offering increasing latency benefits and more favorable accuracy–latency tradeoffs for larger multimodal models. This trend highlights MLC's practical value as a systems-level optimization for deploying high-capacity models under tight latency constraints.

5. Related Works

Compared to text-only language models that operate over one-dimensional, discrete token sequences, inference in vision transformers and multimodal large language models (MLLMs) is substantially more computationally demanding. Visual inputs are encoded as high-dimensional, continuous patch embeddings, often expanding a single image or frame into hundreds of tokens. This expansion significantly increases both attention computation and memory footprint, making inference latency and hardware utilization critical bottlenecks in practical deployments.

Inference frameworks for large language models. A growing body of work has focused on optimizing inference for large language models through specialized systems frameworks. vLLM (Kwon et al., 2023) introduces paged attention, enabling efficient KV-cache management and dynamic batching to improve throughput and memory utilization on GPUs. Similar frameworks, such as FasterTransformer (NVIDIA, 2025) and TensorRT-LLM (NVIDIA, 2023), optimize kernel fusion, parallelism, and memory layouts for text-based inference. However, these systems are primarily designed around discrete token streams and autoregressive decoding, and extending them to multimodal settings introduces additional challenges due to the size, heterogeneity, and continuous nature of visual representations.

Low-level visual token reuse. At the lowest level of the inference stack, several approaches exploit redundancy in visual inputs by caching or reusing static vision tokens. For

example, techniques such as VLA-Cache (Xu et al., 2025) identify unchanged or redundant regions across consecutive frames and reuse their corresponding visual features, reducing redundant computation in video or streaming settings. These methods are effective when temporal locality is strong, but their applicability diminishes when inputs vary semantically rather than pixel-wise.

Intermediate activation and KV caching. At the intermediate representation level, prior work has explored caching and compressing activations to reduce memory and computation overhead. VL-Cache (Tu et al., 2024), for instance, applies modality-aware KV cache compression to vision–language models, selectively retaining or compressing visual keys and values to accelerate inference while preserving output quality. These approaches align closely with recent advances in KV-cache management for LLMs, but primarily focus on GPU-resident memory and per-layer optimizations.

Contextual and prefix-based caching. At a higher semantic level, multi-turn and prefix caching techniques aim to avoid recomputing shared context across inference requests. Methods such as MPIC (Zhao et al., 2025) cache cross-modal interaction histories or shared prompt prefixes, enabling efficient reuse in conversational or interactive scenarios. While effective for reducing redundant computation across turns, these techniques typically assume strong contextual continuity and do not explicitly reason about semantic similarity between distinct visual inputs.

Limitations of prior caching approaches. Despite these advances, most existing work adopts a narrow view of caching, focusing either on GPU-resident memory or on exploiting short-term temporal locality. Relatively little attention has been paid to holistic caching strategies that leverage hierarchical memory systems (e.g., DRAM, Disk) or that reason about semantic and spatial similarity in visual inputs. Moreover, response-level caching—where final model outputs are reused to bypass the entire inference pipeline—remains largely underexplored in vision-based transformers, despite its potential to amortize the high cost of vision encoding and enable near-instantaneous responses in repeated or semantically similar scenarios.

6. Conclusion

This work explored Multi-Level Caching (MLC) as a practical framework for reducing redundant computation in MLLM inference. MLC stores prior responses or intermediate KV chunks and serves future requests by consulting a hierarchical sequence of cache layers, from cheap, high-precision matching (L0.5) progressively to more expensive semantic and multimodal similarity lookup (L1, L2). We conducted two experiments to iterate and evaluate our design. Experiment 1 highlights the basic tradeoffs of response caching: a higher hit rate (achieved by lowering the similarity threshold) can reduce accuracy, and lookup overhead can offset latency gains when hits are rare. We then ran experiment 2 to evaluate MLC on more realistic multimodal workloads and with intermediate caching layers. We showed that with our solution, we can achieve upwards of $2.48\times$ improvement in latency albeit with an accuracy drop. We also observed the latency benefit grows with model scale.

Overall, for applications that have many similar requests, Multi-Level Caching can substantially reduce inference latency.

7. Future Work

While our MLC system aims to be comprehensive, we have identified some areas for further exploration:

- **Enable KV Chunk Caching:** KV-chunk caching can reduce redundant computation while preserving the model's inference path, potentially offering a better latency–accuracy tradeoff than response caching. However, vLLM does not currently support user-defined KV injection, since KV state is tightly coupled with internal sequence metadata, memory block allocation, and attention scheduling. Enabling this likely requires non-trivial changes to vLLM's cache management and execution interfaces.

- **Explore more modalities:** Our implementation focuses on text and images. A more comprehensive system should incorporate audio and video, where redundancy patterns (e.g., temporally adjacent frames or repeated audio segments) may yield larger caching benefits and require modality-specific similarity metrics and thresholds.

- **Evaluate with more datasets**: For each experiment, we evaluated our strategy with one dataset. Further work would emphasize the MLC strategy's effectiveness across different use cases and workloads.

- **Identify Real-World Scenarios**: We found that MLC can reduce latency, but at the cost of lower accuracy due to approximate matches. To make this tradeoff worthwhile in practice, future work should identify concrete scenarios where faster responses clearly matter more than occasional quality loss. For example, multi-turn troubleshooting over a mostly unchanged scene, or streaming camera assistants where consecutive frames and prompts are naturally redundant. Establishing when and why approximate caching is acceptable will be critical for practical adoption.

References

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., Ring, D., Subramanian, S., Tuyls, J., van den Oord, A., Vinyals, O., Walker, L., and Wray, J. Flamingo: A visual language model for few-shot learning. In *Advances in Neural Information Processing Systems (NeurIPS 2022)*, 2022.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS 2022)*, 2022.

Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Gim, I., Chen, G., seob Lee, S., Sarda, N., Khandelwal, A., and Zhong, L. Prompt cache: Modular attention reuse for low-latency inference, 2024. URL https://arxiv.org/abs/2311.04934.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 2019.

Kwon, W., Li, J., Zhuang, H., Chen, S., Pan, J., Wang, J., Zhang, C., and Stoica, I. vLLM: Easy, fast, and cheap LLM serving with pagedattention. arXiv preprint, 2023. arXiv:2309.06180.

Li, J., Li, D., Savarese, S., and Hoi, S. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*. PMLR, 2023.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual instruction tuning. arXiv preprint, 2023. arXiv:2304.08485.

Malviya, S. Rice disease classification dataset, Dec 2024. URL https://huggingface.co/datasets/Subh775/Rice-Disease-Classification-Dataset.

NVIDIA. Tensorrt-llm. https://github.com/NVIDIA/TensorRT-LLM, 2023. Open-source library for optimized large language model inference.

NVIDIA. Fastertransformer. https://github.com/NVIDIA/FasterTransformer, 2025. Highly optimized transformer inference library.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference, 2022. URL https://arxiv.org/abs/2211.05102.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, 2018.

Schroeder, L. G., Desai, A., Cuadron, A., Chu, K., Liu, S., Zhao, M., Krusche, S., Kemper, A., Stoica, I., Zaharia, M., and Gonzalez, J. E. vcache: Verified semantic prompt caching, 2025. URL https://arxiv.org/abs/2502.03771.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 6105–6114, 2019.

Tu, D., Vashchilenko, D., Lu, Y., and Xu, P. Vl-cache: Sparsity and modality-aware kv cache compression for vision-language model inference acceleration, 2024. URL https://arxiv.org/abs/2410.23317.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS 2017)*, 2017.

Xu, S., Wang, Y., Xia, C., Zhu, D., Huang, T., and Xu, C. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching, 2025. URL https://arxiv.org/abs/2502.02175.

Zhao, S., Hu, J., Huang, R., Zheng, J., and Chen, G. Mpic: Position-independent multimodal context caching system for efficient mllm serving, 2025. URL https://arxiv.org/abs/2502.01960.