Efficient Inference of Vision Instruction-Following Models with Elastic Cache

Zuyan Liu[1], Benlin Liu[2], Jiahui Wang[1], Yuhao Dong[1,5], Guangyi Chen[3,4], Yongming Rao[1,5], Ranjay Krishna[2,6], and Jiwen Lu[1†]

[1] Tsinghua University, [2] University of Washington, [3] Carnegie Mellon University, [4] Mohamed bin Zayed University of Artificial Intelligence, [5] Tencent, and [6] Allen Institute for AI

Abstract. In the field of instruction-following large vision-language models (LVLMs), the efficient deployment of these models faces challenges, notably due to the high memory demands of their key-value (KV) caches. Conventional cache management strategies for LLMs focus on cache eviction, which often fails to address the specific needs of multimodal instruction-following models. Recognizing this gap, in this paper, we introduce Elastic Cache, a novel approach that benefits from applying distinct acceleration methods for instruction encoding and output generation stages. We investigate the metrics of importance in different stages and propose an 'importance-driven cache merging' strategy to prune redundancy caches. Instead of discarding less important caches, our strategy identifies important key/value vectors as anchor points. Surrounding less important caches are then merged with these anchors, enhancing the preservation of contextual information in the KV caches while yielding an arbitrary acceleration ratio. For instruction encoding, we utilize the frequency to evaluate the importance of caches. Regarding output generation, we prioritize tokens based on their 'distance' with an offset, by which both the initial and most recent tokens are retained. Results on a range of LVLMs demonstrate that Elastic Cache not only boosts efficiency but also notably outperforms existing pruning methods in language generation across various tasks. Code is available at `https://github.com/liuzuyan/ElasticCache`

Keywords: Efficient Inference · Vision Instruction-Following Model

1 Introduction

ChatGPT [3,19] has rapidly gained popularity for its coherent and fluent responses. Its effectiveness stems from an instruction-following LLM [4, 21], which handles diverse tasks based on input instructions. The model can also integrate visual inputs, as seen in GPT-4V [20] and LLaVA [14], expanding its applications, including multimodal chatbots.

†Corresponding author.

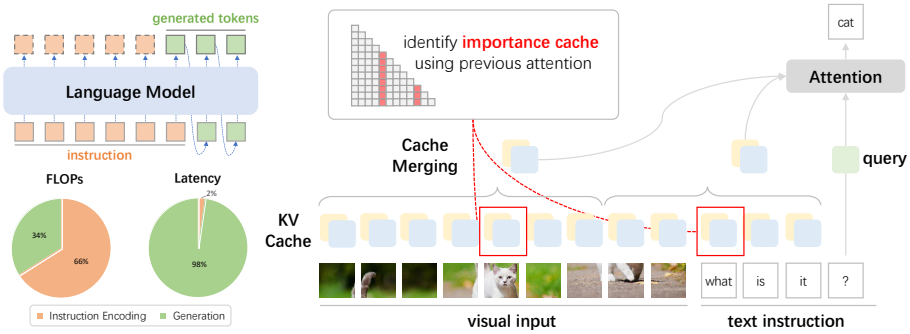


Fig. 1: The main idea of *Elastic Cache*. Instruction encoding accounts for most of the theoretical computation cost, while the actual latency is negligible (here we use generating 512 tokens based on a 1024-token instruction as an example). This underscores that it's not just model weights but also the KV cache used in output generation that can become a significant bottleneck. We propose *Elastic Cache* through a Cache Merging based on the importance scores of instruction tokens, complemented by a fixed-point elimination strategy in the output generation phase. Our designs yield significant inference acceleration while maintaining generation quality.

However, multimodal instruction-following models' high computational and memory demands pose a challenge. These demands are critical in dialogue systems, where real-time responsiveness is essential for user experience. Therefore, the need to enhance the efficiency of these models becomes increasingly evident, particularly when generating lengthy outputs, given that the complexity of this task is compounded by the quadratic computational demands of attention modules in transformers [28]. To address this, a KV Cache mechanism is used in generative inference, storing and reusing key/value vectors for prompt and output tokens to reduce redundant computations.

While effective, this widely used space-for-time strategy in KV cache management often leads to substantial GPU memory usage, sometimes exceeding the memory required for model weights. This can limit batch sizes and affect inference throughputs. Besides solutions like offloading KV cache to the CPU [1] or cache quantization [23] exist, recent studies [15, 36] explore pruning key/value vectors in the KV cache to reduce memory usage while maintaining language modeling performance. Such methods lower memory demands and improve computational efficiency, as they involve fewer vectors in attention calculations. Techniques like H_2O [36] and Scissorhands [15] utilize smaller KV caches. When input sequences surpass the reduced cache size, these methods prune less important tokens based on attention scores, enhancing both memory and computational efficiency.

These existing methods have two main shortcomings. Firstly, their time/memory efficiency can be enhanced. They currently only improve computational and storage efficiency when the sequence length surpasses the KV cache's maximum capacity, with the acceleration ratio largely tied to this capacity. Our goal is to boost efficiency for any sequence length, independent of the cache size, thus

improving efficiency even with a large cache for performance. Secondly, as stated in Fig. 2 and Tab. 4 these methods don't specifically address maintaining the model's capability to generate long, coherent outputs that follow instructions, particularly for multimodal instructions.

To boost time and memory efficiency in multimodal instruction-following models for any sequence length, independent of cache budget, and maintain their multimodal instruction-following ability, we introduce a new KV cache management technique, *Elastic Cache*. The essence of our method is the use of distinct sparsification strategies during instruction encoding and output generation phases. This distinction allows our model to better adhere to multimodal instructions with a compressed KV cache, surpassing previous methods that uniformly pruned KV vectors in both phases. Additionally, our approach enhances efficiency for sequences of all lengths by reducing token storage in the cache earlier during instruction encoding rather than waiting for the cache to fill.

Specifically, during the instruction encoding phase, we globally apply sparsification to prune key/value vectors generated from all concatenated model inputs, including system prompts, user instructions, and chat history. Unlike previous cache compression methods that used eviction strategies to reduce the number of key/value vectors stored in the KV cache, we introduce a novel parameter-free cache merging approach to more effectively preserve context in a compact KV cache. In detail, we determine the importance of all key/value vectors from the instruction encoding phase and utilize the most crucial ones as anchor points. Subsequently, we merge all key/value vectors in the entire instruction sequence with their nearest anchor point. A layer-wise merging policy is adopted, with all attention heads in the same layer sharing anchor point positions, although the values of anchor points at the same position may vary across different heads. We can achieve an arbitrary acceleration ratio by controlling the proportion of anchors in the cache merging process. During the output generation phase, we dynamically manage the KV cache by adding new and removing older key/value vectors as tokens are generated. Unlike H_2O [36], our method employs a fixed-point elimination strategy at a tunable truncation point, balancing the cache between initial guidance and new content. This approach, akin to StreamingLLM [31], differs by retaining a length of initial vectors nearly equal to the input instruction length, better-preserving context to follow instructions.

Remarkably, our method is completely training-free, requiring no additional fine-tuning, and can be applied plug-and-play to any multimodal instruction-following model. This significantly saves the expenses associated with training extra-large models. Additionally, our cache update strategy incurs only negligible computational overhead during inference.

In our experimental analysis, we implemented the *Elastic Cache* method in visual instruction-following tasks, employing Perplexity (PPL) and ROUGE as our primary evaluation metrics to assess instruction-following capabilities. The results demonstrate *Elastic Cache*'s ability to significantly accelerate processing speeds without compromising on the quality of instruction following or generating lengthy outputs. It consistently outperforms both distance-based and frequency-

based cache strategies across various models and datasets. Notably, with the KV Cache Budget set to 0.2, *Elastic Cache* achieves a 78% increase in actual speed. This marked improvement in processing efficiency, along with maintained or even enhanced predictive accuracy, highlights the practicality and effectiveness of *Elastic Cache* in real-world scenarios, where time efficiency is as important as performance.

2 Related Work

Vision Instruction-Following Model After pre-training on trillions of tokens by predicting the next token, decoder-only large language models [3, 26, 27] demonstrate an astonishing understanding of language. To better enable language models to follow instructions and generalize zero-shot to new tasks [16,21,25,29,32, 33], these models are further instruction tuned on human-annotated 'instructional' data, which includes language instructional commands and desired outcomes. Recently, how to enable these blind language models to follow instructions containing visual signals has become a hot research topic. Compared to naively using image caption models for prompting to integrate visual signals [22, 34], works like LLaVA [14], MiniGPT-4 [37], Multimodal-GPT [7], InstructBLIP [5] use linear projection or perceivers [10] to integrate visual representations into the input of LLMs directly. This allows the model to follow multimodal instructions better. Our work focuses on improving the inference efficiency of multimodal instruction-following model [2, 14], especially in generating lengthy responses, which is particularly important for applications like multimodal chatbots.

Model Compression and KV Cache Management. Improving inference efficiency through model compression has always been a crucial topic for the practical application of deep learning models [8]. Like other models, LLM compression methods typically include quantization [12, 30], pruning [17, 24], distillation [9], etc. These methods are usually orthogonal and can be combined. However, past LLM compression methods have primarily focused on the computational overhead of model weights. Using a KV cache to speed up inference also results in a significant additional demand for memory during the inference stage, thereby affecting the throughput of the inference system. To address this, besides using quantization [23] or offloading [1], more research is focusing on reducing the number of KV vectors stored in the cache. Gist tokens [18] learn to compress input instruction tokens before storing them in the KV cache, but this method requires additional training, and its cache update strategy, which includes extra parameter computations, is costly. StreamingLLM [31] focuses on processing long sequences that exceed the cache's capacity limit, so it only can accelerate sequences with tens of thousands of tokens. H_2O [36] and Scissorhands [15] achieve greater efficiency improvements by lowering the cache capacity limit, but they also can't accelerate sequences of arbitrary lengths. Our method is committed to accelerating sequences of any length while maintaining the capability to follow multimodal instructions and generate lengthy responses. Moreover, we employ

a novel cache merging method for cache updates instead of the cache eviction approach used in previous works.

3 Method

3.1 Preliminaries: Instruction-Following Models

Instruction-following large vision-language models (LVLMs) are a cornerstone in natural language processing, especially when it comes to aligning with specific user intents and tasks. These models are invaluable in textual contexts for their adeptness at interpreting and executing instructions and demonstrate comparable value in multimodal settings. The core process of their inference involves two main stages: instruction encoding and output generation.

Instruction Encoding. This is the foundational step in the operation of instruction-following models. During this phase, the model receives and interprets the system prompts, user-provided instructions, and chat history, which may include multimodal information. For autoregressive transformers, this involves processing the instruction inputs and encoding them into a series of tokens. Each token is then transformed into key and value vectors, which are vital for capturing the contextual nuances of the instruction. These vectors are stored in the KV cache, ensuring that the model has immediate access to the contextual information needed for generating accurate responses.

Output Generation. After the instruction has been encoded, the instruction-following model proceeds to the output generation phase. Here, the model incrementally builds its response, token by token. At each step of this process, the model references the KV cache to ensure that each new token generated is contextually aligned with the preceding ones. This incremental approach allows the model to maintain a coherent narrative or logical thread in its responses while staying true to the user's initial instructions.

The operation of the instruction-following model is challenged by the high memory demands of their KV caches. These caches store all encoded tokens' key and value vectors, which are crucial for contextual understanding during instruction encoding and output generation. As interactions progress, the cache size grows linearly, significantly straining memory resources in complex or lengthy tasks. To address this, developing effective cache eviction strategies is crucial for balancing the efficiency and accuracy in generating responses of instruction-following models.

3.2 Instruction following with Cache Merging

In this section, we introduce the *importance-driven cache merging* policy, serving as our core proposal for cache management. Its intuitive principle is finding the key/value vectors in the cache as the anchor point to aggregate the surrounding contextual information. Specifically, we mainly discuss two questions: 1) How

do we measure the importance, and 2) Why do we employ the proposed cache merging instead of conventional cache eviction.

Importance Metrics. We propose to measure the importance of the key/value vectors in the KV cache using their statistical data, motivated by the stationary nature of the statistical information throughout the inference process. This stability suggests that historical statistical data can be effectively used to forecast future cache requirements. By leveraging past usage patterns within the KV caches, we can develop policies to predict which vectors in the cache will be most pertinent in upcoming inference tasks, thereby optimizing cache management and resource allocation.

Building on our initial findings, we conducted a thorough investigation to identify the most crucial statistic for evaluating the importance of KV-caches. Our experiments span an array of statistics, encompassing both population types—identical, head-wise, and layer-wise populations—and statistical variables, including summation, moving average, maximum value, mean, and others. The comparative analysis of these different metrics was meticulously detailed in the ablation studies presented in Sec. 4.4. These investigations revealed that the layer-wise sum of attention scores yielded the most effective performance.

Cache Eviction v.s. Cache Merging. This policy should be strategically designed to maintain the integrity and efficiency of the KV cache system, ensuring that the most valuable cached vectors are retained and readily accessible for future inference processes while less critical caches are efficiently removed to optimize resource allocation and system performance.

Conventional cache management strategies, like those in StreamingLLM [31] and H_2O [36], typically employ a cache eviction policy that discards vectors less important for the generative inference process. However, such an approach, often rigid in nature, tends to overlook the potentially valuable information that might be contained within these deemed less important vectors. Our experimental findings reveal that this conventional approach does not perform optimally for multimodal instruction-following models.

This insight highlights the need for a more nuanced cache management strategy to discern and retain valuable data, even if it might appear less important initially. Thus, we propose Cache Merging, which merges the surrounding less-important cached vectors into the select anchor vector. We determine the importance of each token in the instruction sequence and use the key/value vectors of tokens with high importance as anchor points, dividing the instruction sequence into multiple buckets. Each bucket corresponds to the neighborhood around an anchor point. Then, we store the averaged results of all the key/value vectors corresponding to each bucket in the KV cache. By this merging policy, we reduce the number of key/value vectors in the KV cache and preserve the contextual information as much as possible.

3.3 One Sequence, Two Policies

In the context of multimodal instruction-following models, the model operates differently in the instruction encoding (perception, one-off feed-forward pass) and

output generation (generation, iterative output process) stages. This divergence catalyzes implementing varied acceleration policies tailored to each specific stage of inference. We call this strategy "*One Sequence, Two Policies*".

Instruction Encoding. During the instruction encoding stage, we adopt our *importance-driven cache merging* policy, which unfolds in two primary steps: 1) selection of important key/value vectors with the layer-wise sum of attention scores as the metric and 2) division and merging of vectors based on selected anchors.

Suppose the model has L layers in total, each with K heads. Let's consider the $\boldsymbol{j}$-th attention head in the $\boldsymbol{i}$-th layer, denoted as $\text{Attn}_{i,j}$. The input to $\text{Attn}_{i,j}$ is a sequence of T tokens $\{x_1^{i,j}, \ldots, x_T^{i,j}\}$. For each token in the sequence, we first obtain the query, key, and value vectors using linear projection. For a specific token $x_t^{i,j}$, its query, key, and value vectors are denoted as $q_t^{i,j}$, $k_t^{i,j}$, and $v_t^{i,j}$, respectively. For the attention head $\text{Attn}_{i,j}$, we can derive a causal attention matrix A of size $T \times T$, which is a lower triangular matrix. For simplicity, we omit the superscripts on the attention matrix here. This matrix can be written as:

$$A_{m,n} = \begin{cases} \frac{\exp(\langle q_m^{i,j}, k_n^{i,j} \rangle)}{\sum_{m' < m} \exp(\langle q_m^{i,j}, k_{m'}^{i,j} \rangle)} & \text{if } m \geq n \\ 0 & \text{if } m < n \end{cases} \tag{1}$$

where $\langle q_m^{i,j}, k_n^{i,j} \rangle$ represents the dot product of $q_m^{i,j}$ and $k_n^{i,j}$.

Based on this attention matrix, we can calculate the importance score of each token. For the n-th token, we can define its importance value as obtained from the $\text{Attn}^{i,j}$ module, denoted by $I_n^{i,j} = \sum_m A_{m,n}^{i,j}$. Then, by averaging the importance values obtained from all attention heads in the same layer, the importance value of the n-th token in the i-th layer can be expressed as $I_n^i = \frac{1}{K} \sum_j \sum_m A_{m,n}^{i,j}$. Consequently, we obtain independent importance scores for each layer, implying that our pruning strategy is layer-wise.

Given a predefined retention ratio γ, we select the top $N_I = \gamma \times T$ tokens in I_n, with the indices $\{t_k \mid k = 1, 2, \ldots, N_I\}$ in ascending order. Taking them as anchor points, we can have N_I buckets as

$$B_k = \begin{cases} \{0, ..., \left\lfloor \frac{t_1 + t_2}{2} \right\rfloor\}, & k = 1 \\ \{\left\lfloor \frac{t_{k-1} + t_k}{2} \right\rfloor + 1, \ldots, \left\lfloor \frac{t_k + t_{k+1}}{2} \right\rfloor\}, & 1 < k < N_I \\ \{\left\lfloor \frac{t_{N_I - 1} + t_{N_I}}{2} \right\rfloor, ..., T\}, & k = N_I \end{cases}, \tag{2}$$

where $\lfloor \cdot \rfloor$ denotes the floor function. The bucket division varies across different layers while remaining consistent for all attention heads within each layer. For each attention head, we average all its key/value vectors as KV_k corresponding to each bucket B_k and store them in the KV cache:

$$\text{KV}_k = \frac{1}{|B_k|} \sum_{t \in B_k} kv_t. \tag{3}$$

Notably, each attention head has its own unique KV_k; again, we omit the superscript for simplicity.

Output Generation. The output generation stage functions iteratively, updating the KV cache at each iteration. It necessitates that the cache management policy be adaptable for continuous increment. The conventional methods, exemplified by H_2O [36], adhere to the policy in the instruction encoding stage that accumulates attention scores and prioritizes cached vectors with higher scores. However, this approach appears less effective in the context of new token generation. Compared to the already cached vectors, the newly generated tokens lack an inherent advantage under this scoring system. As the generation progresses, the likelihood of newly generated tokens being retained in the cache diminishes incrementally, making this method less sensible for dynamically evolving caches.

To address this issue, we introduce a *fixed-point elimination* strategy, in light of the assumption that the initialized instruction guidance and closely related generation carry greater importance, whereas the earlier generated content holds less significance. Specifically, we employ a queue-based system where, upon generating a new token and adding its corresponding key/value vectors, we systematically remove earlier cached vectors from a fixed truncation point in the queue. Consider a current KV cache denoted as $\{\mathrm{KV}_k | k = 1, 2, \ldots, N_C\}$ where N_C represents the current length of the cache. We define a fixed truncation location N_{tl}. Upon adding a new token to the cache, we calculate the current token retention ratio and compare it with a predefined threshold. If the ratio is lower than the threshold, the token is retained. Otherwise, the token at the truncation location N_{tl} is removed, leading to an updated cache represented as: $\{\mathrm{KV}_k | k = 1, 2, \ldots, N_{tl} - 1, N_{tl} + 1, \ldots, N_C + 1\}$

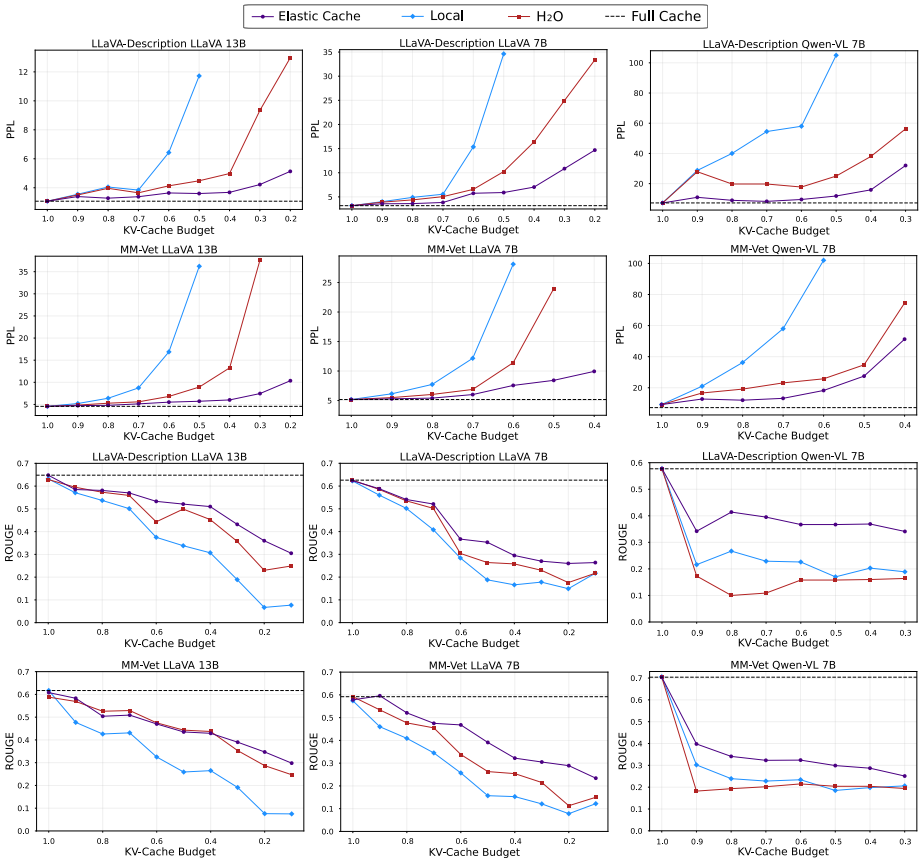
Experimentally, selecting a fixed position N_{tl} that is arbitrarily close to N_I has often yielded impressive results. To approach this more systematically, this position can be regarded as a hyperparameter, which allows for tuning through model selection techniques.

4 Experiments

In this section, we conduct extensive experiments to illustrate the effectiveness and efficiency of *Elastic Cache*. We use two mainstream LVLMs, LLaVA-1.5 [13, 14] and Qwen-VL [2] as our backbone and adopt the *Elastic Cache* on instruction-following chat generation datasets. As no benchmark exists on the efficient inference of the vision instruction tuning field, we design precise and general evaluation metrics to validate our method. We subsequently conduct experiments on inference speed, ablations on our method design, and real-world analysis.

4.1 Experimental Settings.

Baselines. To evaluate the effectiveness of our proposed caching mechanism for input instruction handling, we establish two state-of-the-art baseline methods for comparison: Heavy-Hitter Oracle [36] and StreamingLLM [31], where we term as H_2O and Local in our experiments, respectively. The techniques proposed by H_2O and StreamingLLM can be summarized as Frequency Cache and Local



(a) LLaVA-1.5 13B (b) LLaVA-1.5 7B (c) Qwen-VL 7B

Fig. 2: Results on visual instruct-following tasks. We evaluate *Elastic Cache* together with baselines on PPL (lower better) and ROUGE (higher better) metrics. We conduct LLaVA-1.5 of different sizes (a),(b) and Qwen-VL-7B(c) for visual tasks. Our *Elastic Cache* outperforms baselines consistently.

Cache methods. Frequency Cache method records how often each key-value pair is accessed in response to input instructions. When the need arises to free up cache space to adhere to the pre-defined memory budget, this method selects the least frequently accessed key-value pairs for eviction. The Frequency Cache method continuously updates the frequency counts throughout the generation process. The Local Cache method employs a spatial heuristic for cache eviction. When the cache reaches capacity, the Local Cache method identifies and removes the most distant key-value pairs from the current prediction.

Evaluation Metrics. In the evaluation period of our method, we employ two metrics to assess performance rigorously: perplexity (PPL) and the ROUGE score [11]. PPL calculates the expositional value of the cross-entropy loss between

the predicted next token and the ground truth. The ROUGE score measures the longest common subsequence (LCS) between the generated text and a set of reference texts. We use the F1-score to evaluate our methods. These metrics are chosen for their ability to capture distinct aspects of model performance, with perplexity reflecting the model's uncertainty in predicting the next token and the ROUGE score indicating the quality of text generation in terms of overlap with reference sequences. The detailed algorithms of the evaluation metrics are stated in Supplementary Materials.

Datasets. *Elastic Cache* is a training-free approach during the inference phase. To evaluate the text generation capabilities of the key-value (KV) cache strategy on long and high-quality text samples, we randomly integrate a subset of 100 detailed description instructions from the training set of LLaVA-1.5 [13] dataset, where we name as LLaVA-Description. Additionally, we also select the MM-Vet [35] dataset which covers a diverse range of tasks, enabling us to evaluate the model's multimodal understanding and generation performance comprehensively. Note that to prevent data leakage and ensure the integrity of our assessment, we redo the instruction tuning period of the LLaVA-1.5 models and exclude the aforementioned 100 detailed description instructions for the evaluation employing the KV cache strategy in our experiments.

Implementation Details. We mainly apply our proposed *Elastic Cache* on three instruct-tuning VLMs. We choose LLaVA-7B/13B and Qwen-VL-7B as the visual instruct-tuning model. In our implementation of *Elastic Cache*, we protect the first and the most recent token following [31]. Following the analysis results in Fig. 3, we fixed the recent distance during the generation period at the length of 25 caches. It is noteworthy that for the reference texts required for the ROUGE evaluation, we use the fully-cached model to generate the reference text due to the ROUGE evaluation method. Notably, as the generation results are different during multiple experiments when the temperature is larger than zero, when the KV-Cache budget is set to 1.0, we use the ROUGE score of two-generation results, so the ROUGE score is smaller than 1.0. In other cases, we set the temperature to 0 to ensure the reproducibility of the results.

4.2 Main Results

Results on Visual Instruct-Following. Our experiments focus on visual instruction-following, utilizing a subset of the LLaVA dataset specifically curated for detailed description analysis. This subset comprises 100 instances of image description tasks, with each instance including an instruction and a corresponding answer generated by GPT-4 [19]. For our evaluation metrics, we employ Perplexity (PPL), where a lower score indicates better model performance, and the F1 variant of the ROUGE-L score, where a higher score reflects greater quality. We leverage the LLaVA-1.5/13B, LLaVA-1.5/7B, and Qwen-VL-7B as our backbone architectures. Compared to the baseline strategies, namely Local cache and H_2O cache, our proposed *Elastic Cache* demonstrates superior performance across both metrics over a spectrum of KV-Cache Budgets ranging from 1.0 to 0.2. Notably, in

Fig. 2b, at a KV-Cache Budget of 0.5, *Elastic Cache* surpasses the H_2O cache by a margin of 4.34 in PPL and 0.089 in ROUGE-L F1 score on LLaVA-Description. Compared to the Local cache, *Elastic Cache* shows an improvement of 28.72 in PPL and 0.165 in ROUGE-L F1 score. This enhancement in performance is likely attributable to the fact that the *Elastic Cache*'s dynamic pruning strategy is more adept at retaining image-relevant knowledge, in contrast to the fixed strategy of most recent pruning, which can overlook critical visual information.

Comparisons under GPT-4V Evaluation. We further conduct experiments incorporating GPT-4V [20] evaluation to demonstrate the effectiveness of our *Elastic Cache* mechanism from a different perspective. In detail, we first collect 200 generations with the KV-Cache ratio 1.0 (the original generation procedure with full cache) for each method as our baseline. Then, we perform each method with a different pruned ratio to ask the same questions and forward the image-question pair and the corresponding answers to GPT-4V. For each method, GPT-4V is asked to determine whether the generated text is better or worse than the corresponding baseline. We follow the prompt proposed in FastGen [6] and modify it to customize it for image-question pairs. As shown Tab. 1, our method illustrates superior robustness as the pruned ratio grows. H_2O also demonstrates its robustness, but the lower win rate evaluated by GPT-4V indicates the unsatisfied capability compared with our method. On the contrary, Local exhibits competitive results at a low pruned ratio. However, it encounters a disastrous performance drop when the pruned ratio grows to 0.3, which further reveals the splendid ability of our method to accommodate both effectiveness and robustness.

Table 1: Win-rate comparison using GPT-4V API. We primarily compare the win rate at different pruned ratios with existing work under GPT-4V. *Elastic Cache* still obtains 38% win-rate with a pruned ratio of 0.3, surpassing previous work by a large margin.

Method	Budget=0.1	Budget=0.2	Budget=0.3
Elastic Cache	**47.54%**	**46.63%**	**37.56%**
H_2O [36]	38.55%	35.26%	30.26%
Local [31]	46.37%	35.29%	10.10%

4.3 Inference Speed

We evaluate the inference speed of our novel *Elastic Cache* mechanism, implemented within the LLaVA-1.5/13B framework, to demonstrate its practical efficiency. We conduct this evaluation under two distinct experimental configurations: the first with an input comprising 1024 prompt tokens followed by the generation of 512 tokens, and the second with a reduced input of 624 prompt tokens, and generate 256 tokens. The latter setup represents the minimal prompt length, including image patches and system prompts that can be processed. The inference tests are performed on a single NVIDIA A100 GPU, with batch size adjusted to maximize the available memory, thereby reflecting a realistic usage scenario optimized for both efficiency and throughput. Results are shown in Tab. 2 Our results indicate that when pruning 80% of the KV-Cache, the *Elastic*

Table 2: Evaluations on inference latency and throughput. We set the KV-Cache budget as 0.2 on the LLaVA-1.5/13B backbone. *Elastic Cache* can lead to a maximum of 77.9% actual speed up and avoid out-of-memory in limiting cases.

Batch Size	Model Size	Token Length	Latency (s)		Throughput (token/s)	
			Elastic Cache	Full Cache	Elastic Cache	Full Cache
8	13B	1024+512	$20.2_{+(33.8\%)}$	30.5	$202.8_{+(52.6\%)}$	132.9
16	13B	624+256	$11.8_{+(34.1\%)}$	17.9	$347.1_{+(51.7\%)}$	228.8
16	7B	1024+512	$17.2_{+(43.8\%)}$	30.6	$476.3_{+(77.9\%)}$	267.7
48	7B	624+256	$13.6_{+N/A}$	OOM	$903.5_{+N/A}$	OOM

Table 3: Ablation studies on the components of *Elastic Cache*. We conduct four ablation studies on the key design of *Elastic Cache*. The experiments are conducted on the LLaVA-1.5/13B backbone with the KV-Cache budget set as 0.5. We use PPL metric for the LLaVA detailed description dataset for evaluation.

Discard Position	PPL
Most Recent	3.93
Frequency	3.75
Fixed-point	**3.60**

(a) Ablation on discard position.

Merging Strategy	PPL
Clustering	3.61
Cache Eviction	3.68
Cache Merging	**3.60**

(b) Ablation on merging strategy.

Attn. Procedure	PPL
Shared	3.73
Head-wise	3.75
Layer-wise	**3.60**

(c) Ablation on attention procedure.

Importance Metric	PPL
Moving Average	8.43
Mean	8.70
Sum	**3.60**

(d) Ablation on importance metric.

Cache method can achieve an actual speed acceleration of up to 78% compared to full cache baselines, while also reducing the memory footprint during inference.

4.4 Ablation Experiments

We perform complete and detailed ablation experiments to evaluate the effects of each component in *Elastic Cache* in Tab. 3.

Discard Position. The strategy during the output generation period contributes to maintaining the KV-Cache ratio with more generated caches. We explore how to discard older caches to improve the efficiency during output generations. Compared with leaving the most recent caches while discarding the farthest caches and continually taking the frequency policy during generation, we find that fixing a specific discard position can achieve better performance.

Merging Strategy. Compared with simply evicting the chosen caches, we find that merging the unused caches with validated caches can achieve better performance. We also compare our recent merging method with the clustering algorithm, which clusters the unused cache to 10 cluster centers based on the key of caches. Though the clustering method involves extra cache cost, the overall performance is poorer than the simple recent merging method.

Attention Procedure. As we obtain the attention score across the layer and head dimensions, we are curious about whether applying the dynamic results across such dimensions contributes to better performance. We can conclude from

Table 4: Generations on image recognition question. We fix the KV-Cache Budget as 0.5. Local and H_2O cache pruning methods fail to generate rational results under such experimental settings while *Elastic Cache* maintains the generation ability with a detailed and correct description of the image.

	**User:** What's happening in the scene? **Local:** The image shows a plate of assorted doughnuts on a table, The doughnuts are arranged in a plate, with a total of 10 doughnuts on a table, showcasing a total of 10 doughnuts on a table ... **H_2O:** The image shows a park scene with a doughnut on a table. **Elastic Cache:** The image features **a large pile of assorted donuts**, including **glazed and chocolate** donuts, arranged in a visually appealing display. ... making it an **attractive sight** for anyone passing by.
	**User:** What's happening in the image? **Local:** The squirrel, a squirrel, a small gray squirrel, which appears to be a grey squirrel, is perched on a tree branch, sitting on a tree branch. **H_2O:** The squirrel is holding on to a tree branch. **Elastic Cache:** The squirrel is **sitting on a tree branch**, holding onto it while **eating**. The tree is surrounded by a forest setting, and the squirrel is **enjoying its meal**.

the results that the layerwise strategy performs better than leaving an identical strategy across layers. However, including the dynamic nature in the multi-head dimension will lead to an accuracy drop.

Importance Metric. The way of calculating the importance metric of each cache based on the attention score is a crucial problem to explore. We try the three most simple strategies to statistic the attention score. By using the moving average, mean, and sum value of the attention score, we observe that simply sum up the attention score outperforms another method to a large extent.

4.5 Analysis

Generation. In the empirical analysis of our method's performance in real-world text generation scenarios, we rigorously evaluate the robustness of our caching strategy under constrained conditions in Tab. 4. We set the Key-Value (KV) Cache Budget to 0.5. Our method demonstrates remarkable resilience in the face

of significant cache limitations. Notably, when 50% of the KV cache is evicted, our approach continues attending to image details and producing coherent and rational outcomes. In contrast, as the cache budget decreases, the Local and H_2O strategies suffer a conspicuous performance degradation. With the KV Cache budget set to 0.5, the Local strategy begins generating repetitive, looping text and fails to respond. Similarly, the H_2O strategy yields significantly shorter responses, suggesting an inability to sustain longer, more intricate narratives when deprived of less frequently accessed cache content.

Effects of the fixed-point elimination. We explore the efficacy of a fixed-point elimination strategy within the caching mechanism, seeking to determine the optimal position for fixation to enhance performance. Intrigued by the potential impact of different fixed points on the overall system effectiveness, we conduct a series of tests across various positions in Fig. 3. The empirical evidence suggests that anchoring the fixed point within the middle section of the cache sequence yields superior results. Guided by these findings, we strategically fix the position at the 25 most recent caches based on the experimental results.

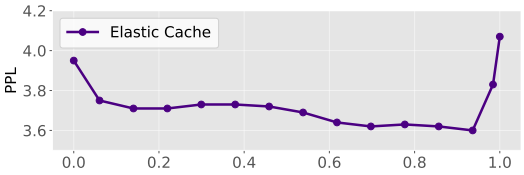


Fig. 3: Effects of the fixed-point elimination. We observe that fixing the elimination at the middle of the KV-Cache of the instruction attention score leads to better performance.

5 Conclusion, Limitations and Societal Impact

In this paper, we have proposed *Elastic Cache*, an innovative framework designed to significantly enhance the efficiency of inference processes in widely utilized instruction-following models. Our novel approach leverages the importance-driven cache merging strategy, which utilizes attention scores as a measure of importance to optimize cache utilization. We further dissect the instruction-following paradigm into two distinct components: instruction encoding and output generation, applying the most effective strategy independently. The experimental results are compelling. *Elastic Cache* not only surpasses existing baselines but also demonstrates robust generation capabilities coupled with remarkable speed improvements. We hope our work can open a new path for the following work to explore a better strategy and efficient inference of vision large models.

One limitation of our approach could be that the reliance on attention scores for cache optimization may not always align with the most computationally efficient caching strategy. A potential negative social impact is that the increased efficiency and speed of *Elastic Cache* might accelerate the deployment of AI systems in surveillance applications.

Acknowledgements

This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFB280690, and in part by the National Natural Science Foundation of China under Grant 62321005, Grant 62336004, and Grant 62125603.

References

1. Aminabadi, R.Y., Rajbhandari, S., Awan, A.A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al.: Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (2022)
2. Bai, J., Bai, S., Yang, S., Wang, S., Tan, S., Wang, P., Lin, J., Zhou, C., Zhou, J.: Qwen-vl: A frontier large vision-language model with versatile abilities. ArXiv:abs/2308.12966 (2023)
3. 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 pp. 33, 1877–1901 (2020)
4. Chiang, W.L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J.E., et al.: Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. See https://vicuna. lmsys. org (accessed 14 April 2023) (2023)
5. Dai, W., Li, J., Li, D., Tiong, A.M.H., Zhao, J., Wang, W., Li, B., Fung, P., Hoi, S.: Instructblip: Towards general-purpose vision-language models with instruction tuning (2023)
6. Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., Gao, J.: Model tells you what to discard: Adaptive kv cache compression for llms. arXiv preprint arXiv:2310.01801 (2023)
7. Gong, T., Lyu, C., Zhang, S., Wang, Y., Zheng, M., Zhao, Q., Liu, K., Zhang, W., Luo, P., Chen, K.: Multimodal-gpt: A vision and language model for dialogue with humans. arXiv preprint arXiv:2305.04790 (2023)
8. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
9. Hsieh, C.Y., Li, C.L., Yeh, C.K., Nakhost, H., Fujii, Y., Ratner, A., Krishna, R., Lee, C.Y., Pfister, T.: Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. arXiv preprint arXiv:2305.02301 (2023)
10. Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., Carreira, J.: Perceiver: General perception with iterative attention. In: International conference on machine learning. pp. 4651–4664. PMLR (2021)
11. Lin, C.Y.: Rouge: A package for automatic evaluation of summaries. In: Text summarization branches out. pp. 74–81 (2004)
12. Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., Han, S.: Awq: Activation-aware weight quantization for llm compression and acceleration. arXiv preprint arXiv:2306.00978 (2023)

13. Liu, H., Li, C., Li, Y., Lee, Y.J.: Improved baselines with visual instruction tuning. ArXiv:abs/2310.03744 (2023)
14. Liu, H., Li, C., Wu, Q., Lee, Y.J.: Visual instruction tuning. arXiv preprint arXiv:2304.08485 (2023)
15. Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., Shrivastava, A.: Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. arXiv preprint arXiv:2305.17118 (2023)
16. Liu, Z., Dong, Y., Rao, Y., Zhou, J., Lu, J.: Chain-of-spot: Interactive reasoning improves large vision-language models. arXiv preprint arXiv:2403.12966 (2024)
17. Ma, X., Fang, G., Wang, X.: Llm-pruner: On the structural pruning of large language models. arXiv preprint arXiv:2305.11627 (2023)
18. Mu, J., Li, X.L., Goodman, N.: Learning to compress prompts with gist tokens. arXiv preprint arXiv:2304.08467 (2023)
19. OpenAI: Gpt-4 technical report. ArXiv:abs/2303.08774 (2023), https://arxiv.org/abs/2303.08774
20. OpenAI: Gpt-4v(ision) system card. OpenAI Blog (2023), https://cdn.openai.com/papers/GPTV_System_Card.pdf
21. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al.: Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems pp. 35, 27730–27744 (2022)
22. Shen, Y., Song, K., Tan, X., Li, D., Lu, W., Zhuang, Y.: Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. arXiv preprint arXiv:2303.17580 (2023)
23. Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Fu, D.Y., Xie, Z., Chen, B., Barrett, C., Gonzalez, J.E., et al.: High-throughput generative inference of large language models with a single gpu. arXiv preprint arXiv:2303.06865 (2023)
24. Sun, M., Liu, Z., Bair, A., Kolter, J.Z.: A simple and effective pruning approach for large language models. arXiv preprint arXiv:2306.11695 (2023)
25. Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., Hashimoto, T.B.: Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca (2023)
26. Team, X.L.: Xwin-lm (9 2023), https://github.com/Xwin-LM/Xwin-LM
27. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
28. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 5998–6008 (2017)
29. Wei, J., Bosma, M., Zhao, V.Y., Guu, K., Yu, A.W., Lester, B., Du, N., Dai, A.M., Le, Q.V.: Finetuned language models are zero-shot learners. arXiv preprint arXiv:2109.01652 (2021)
30. Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., Han, S.: Smoothquant: Accurate and efficient post-training quantization for large language models. In: International Conference on Machine Learning. pp. 38087–38099. PMLR (2023)
31. Xiao, G., Tian, Y., Chen, B., Han, S., Lewis, M.: Efficient streaming language models with attention sinks. arXiv preprint arXiv:2309.17453 (2023)
32. Yang, J., Dong, Y., Liu, S., Li, B., Wang, Z., Jiang, C., Tan, H., Kang, J., Zhang, Y., Zhou, K., et al.: Octopus: Embodied vision-language programmer from environmental feedback. arXiv preprint arXiv:2310.08588 (2023)

33. Yang, J., Dong, Y., Liu, S., Li, B., Wang, Z., Kang, J., Jiang, C., Tan, H., Zhang, Y., Zhou, K., Liu, Z.: Learning embodied vision-language programming from instruction, exploration, and environmental feedback (2024), https://openreview.net/forum?id=VUA9LSmC2r
34. Yang, Z., Li, L., Wang, J., Lin, K., Azarnasab, E., Ahmed, F., Liu, Z., Liu, C., Zeng, M., Wang, L.: Mm-react: Prompting chatgpt for multimodal reasoning and action. arXiv preprint arXiv:2303.11381 (2023)
35. Yu, W., Yang, Z., Li, L., Wang, J., Lin, K., Liu, Z., Wang, X., Wang, L.: Mm-vet: Evaluating large multimodal models for integrated capabilities. ArXiv:2308.02490 (2023)
36. Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al.: H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. arXiv preprint arXiv:2306.14048 (2023)
37. Zhu, D., Chen, J., Shen, X., Li, X., Elhoseiny, M.: Minigpt-4: Enhancing vision-language understanding with advanced large language models. arXiv preprint arXiv:2304.10592 (2023)