Empower Vision Applications with LoRA LMM

Liang Mi[1*†] Weijun Wang[2*] Wenming Tu[2†] Qingfeng He[2†] Rui Kong[2†] Xinyu Fang[2†] Yazhu Dong[2†]
Yikang Zhang[1] Yuanchun Li[2,3,4] Meng Li[1] Haipeng Dai[1] Guihai Chen[1] Yunxin Liu[2,3]
[1]State Key Laboratory for Novel Software Technology, Nanjing University
[2]Institute for AI Industry Research (AIR), Tsinghua University
[3]Shanghai AI Laboratory [4]Beijing Academy of Artificial Intelligence (BAAI)

Abstract

Large Multimodal Models (LMMs) have shown significant progress in various complex vision tasks with the solid linguistic and reasoning capacity inherited from large language models (LMMs). Low-rank adaptation (LoRA) offers a promising method to integrate external knowledge into LMMs, compensating for their limitations on domain-specific tasks. However, the existing LoRA model serving is excessively computationally expensive and causes extremely high latency. In this paper, we present an end-to-end solution that empowers diverse vision tasks and enriches vision applications with LoRA LMMs. Our system, VaLoRA, enables accurate and efficient vision tasks by 1) an accuracy-aware LoRA adapter generation approach that generates LoRA adapters rich in domain-specific knowledge to meet application-specific accuracy requirements, 2) an adaptive-tiling LoRA adapters batching operator that efficiently computes concurrent heterogeneous LoRA adapters, and 3) a flexible LoRA adapter orchestration mechanism that manages application requests and LoRA adapters to achieve the lowest average response latency. We prototype VaLoRA on five popular vision tasks on three LMMs. Experiment results reveal that VaLoRA improves 24-62% of the accuracy compared to the original LMMs and reduces 20-89% of the latency compared to the state-of-the-art LoRA model serving systems.

CCS Concepts

• **Computing methodologies** → **Machine learning**;

Keywords

Large language model, Machine learning system

EuroSys '25, March 30–April 3, 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1196-1/2025/03.
https://doi.org/10.1145/3689031.3717472

1 Introduction

Encouraged by the success of LLMs in NLP applications [7, 8, 10, 13, 56], Large Multimodal Models (LMMs) [11, 12, 100] have attracted great attention from both academia and industry. They enhance LLMs by perceiving and interpreting multimodal signals (*e.g.,* visual inputs [19, 60, 74]) and well accomplish many complex multi-modal tasks that prior models cannot. For example, GPT-4o [12] achieves leading accuracy on many multimodal tasks such as visual question answering [37]. Yet when applied to practical applications requiring domain-specific knowledge, LMMs often show suboptimal performance, similar to the early LLMs that experienced hallucinations [97].

Low-rank adaptation (LoRA) [30, 40] provides a promising way to integrate the external knowledge into LMM. It fine-tunes a small portion of model parameters, known as LoRA adapters, on domain-specific datasets to learn target knowledge, and freezes the base model to preserve its original capability (more in §2). LLMs often leverage retrieval-augmented generation (RAG) [52] to meet this goal. Unlike LoRA, which modifies model parameters, RAG appends the retrieved knowledge (*e.g.,* documents) onto requests (*i.e.,* input data) for accurate response. However, this data-augmented method is not appropriate for time-sensitive vision applications. Its retrieval process and appended long-context requests incur >10× response delay [44]. Conversely, LoRA merges fine-tuned adapters into LMMs at runtime and efficiently generates high-quality and consistent domain-specific responses without additional overhead.

Despite these advantages of LoRA, it introduces complex system challenges. Recent work [25, 69, 82], focusing on system optimization for linguistic applications with LoRA LLM, has made noticeable progress. Punica [25] and S-LoRA [69] propose *unmerged inference* to overcome the limitation of *merged inference* that can merge only one adapter at once. It computes multiple LoRA adapters in parallel while batching the shared base model computation across different requests, to boost system efficiency. dLoRA [82] further balances the throughput and latency by merged and unmerged inference mode switch (more in §2). However, these efforts fail to meet

*Liang Mi and Weijun Wang contributed equally to this work.
†Work was done while Liang, Wenming, Qingfeng, Rui, Xinyu, and Yazhu interned at the Institute for AI Industry Research (AIR), Tsinghua University.
Corresponding authors: Weijun Wang and Meng Li.

the high efficiency and diverse requirements of vision applications (more in §3.2).

In this paper, we answer the following research question: *Can we leverage LoRA LMMs to enrich vision applications while meeting their performance requirements?* We argue that yes, but need to tackle the following challenges.

First, many existing small models, trained on domain-specific datasets and used in current vision applications, outperform LMMs on target tasks. To keep accuracy, their knowledge must be integrated into LMM with LoRA adapters. However, simply training one LoRA adapter for each task is uneconomical, while fusing too much external knowledge (*e.g.,* multiple small models) into a single adapter causes inevitable accuracy degradation.

Second, a vision application often involves diverse external knowledge. When serving multiple vision applications, in particular, LoRA LMM is very likely asked to execute multiple LoRA adapters simultaneously. Although unmerged inference can compute heterogeneous adapters in parallel, they introduce excessive delays. Therefore, our LoRA LMM inference system must efficiently compute concurrent heterogeneous adapters with low latency.

Lastly, different vision applications have distinct performance requirements. Real-time video analytics application [92, 93] needs low latency, while visual retrieval [45] prefers high throughput. To meet these needs, carefully managing LoRA adapters and flexibly scheduling application requests is necessary. Typical LoRA adapter manager (*e.g.,* dLoRA) is not designed for vision applications, which incurs excessive overhead. Solely its inference mode switcher costs over half of LMM inference time.

This paper presents VaLoRA, an end-to-end LoRA LMM serving system, including LoRA adapter preparation (§4.2) and inference runtime (§4.3, §4.4), to empower vision applications. VaLoRA addresses the above challenges with the following techniques:

Accuracy-aware LoRA adapter generation. We propose a LoRA generator (§4.2) that prepares domain-specific LoRA adapters to generate accurate results on target tasks with external knowledge. Considering the complex accuracy variations of knowledge fusion (§3.2), LoRA adapters generation can be formulated as a constrained bin packing problem, that given external knowledge, *i.e.,* small models and domain-specific datasets, to generate the minimum number of LoRA adapters, ensuring the accuracy specified by vision applications. We design an accuracy-aware knowledge-fusion algorithm with a greedy heuristic to solve it. Additionally, we introduce vision task heads, incorporated as part of the LoRA adapter, enabling low-latency response for vision tasks.

Adaptive-tiling LoRA adapters batching. We propose a concurrent LoRA adapters batching method (§ 4.3), comprised of the Adaptive-Tiling Matrix Multiplication (ATMM) operator and its optimal tiling search algorithm, for efficient heterogeneous LoRA adapters computation. The offline search algorithm identifies the optimal tiling configurations for each possible input matrix shape, builds a hash table storing these input-optimal tiling pairs, and compiles their code implementations for standby. At runtime, ATMM adaptively selects the optimal tiling configuration in the hash table, according to the input shapes of both concurrent requests and invoked LoRA adapters, then executes the corresponding code implementation in an extreme efficiency.

Flexible LoRA adapters orchestration. For diverse requirements of vision applications, we propose an orchestrator (§4.4) that efficiently and flexibly orchestrates LoRA adapters at runtime. Two tools are developed to facilitate high efficiency. A switcher leverages ATMM and unified memory management to enable swift inference mode switch and LoRA adapters swap, and a mixture inference mode, deLoRA, mitigates the starvation. Using the above tools, we design an algorithm to dynamically switch between three inference modes, schedule requests, and manage LoRA adapters to satisfy the performance requirement of each application.

We summarize our key contributions as follows:

- To the best of our knowledge, we are the first to identify and solve the key problems in empowering vision applications with LoRA LMM.
- We prototype VaLoRA* that enables accurate, efficient, and flexible LoRA LMM serving for vision applications, which involves accuracy-aware LoRA adapter generation, adaptive-tiling LoRA adapters batching, and efficient and flexible LoRA adapters orchestration.
- We implement VaLoRA and conduct evaluations for five popular analytical tasks with real-world trace on three LMMs. Experimental results show that VaLoRA achieves 24-62% accuracy improvement compared to the original LMMs, and 20-89% latency reduction compared to the state-of-the-art methods.

This work does not raise any ethical issues.

2 Background

Vision applications in today exploit AI technology to process images or videos in RGB spaces [49, 67, 95]. In video analytics, for instance, multiple DNNs that are well-trained on domain-specific datasets separately take care of one target task and together serve the application well [46, 47, 91]. However, the limited capabilities of small models hinder the development of vision applications. Current applications yet stay on the simple combination of vision tasks such as image classification [92], vehicle counting [57], and target detection [31]. With the natural language interface inherited from LLM, LMM can enrich future vision applications. For example, serving by LMM, the police officer can find the right target when only given a text-described query such as "A

*Our code is available at https://github.com/mi150/VaLoRA

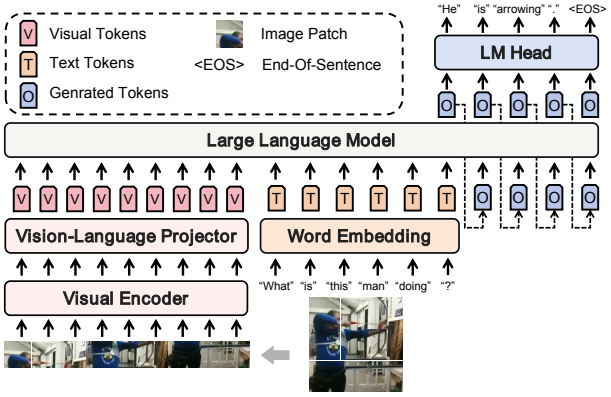
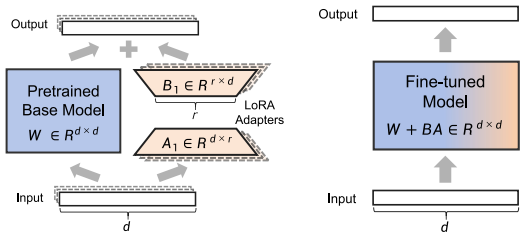


Figure 1. *Illustration of LMM inference. Qwen-VL-7B [20] generates the right action recognition answer to a piece of data from UCF-101 dataset [72] and the corresponding prompt.*

boy wearing a red sweater lost at the corner". Therefore, this paper tries to empower vision tasks with LoRA LMM and enrich future vision applications.

Large multimodal models (LMMs) aim to achieve stronger general intelligence via extending LLMs with multimodal inputs. Since more than 80% of human beings' perception and activities are mediated through vision [63], it is natural to start the exploration by equipping LLMs with "eyes". By introducing a visual receptor, comprised of a *visual encoder* (*e.g.,* ViT [65]) and a *vision-language projector* (*e.g.,* Q-former [53]), LMMs power LLMs with visual capacity. Fig. 1 illustrates the inference procedure of LMMs. Given an image input and its prompt, the visual encoder splits the image into small patches (*e.g.,* 14×14 pixels block) and extracts the visual features of each. The vision-language projector then converts patches' visual features into visual tokens and feeds into LLM [17, 28, 42, 77], together with the embedded text tokens from the prompt, to generate the answer. LLM maps the input into high-level features and predicts the probability distribution of the next token with the *language modeling (LM) head* in an autoregressive pattern [50, 89]. As the dotted arrows depicted in Fig. 1, it iteratively generates tokens with input tokens and previous output tokens once a time, until an end-of-sentence (<EOS>) token is emitted.

Low-rank adaptation (LoRA) [23, 30, 40] is a widely used parameter-efficient fine-tuning method [61] to integrate the external knowledge into large models. The LoRA adapter is a small number of trainable parameters to learn this knowledge, typically placed in attention layers [40]. The core of LoRA, as illustrated in Fig. 2(a), is to represent each weight update ΔW as a product of two matrices, A and B, with dimensions $r \times d$ and $d \times r$, respectively. Their ranks are much smaller than the base model weight W, with dimensions $d \times d$. This makes sense since the low intrinsic rank phenomenon of weight updates in large models [40]. When fine-tuning, LoRA adapters only update A and B while keeping W frozen. At inference, the computation of all LoRA adapters can be placed bypass, as shown in Fig. 2(a), and only adds the results onto the output. Or merge one multiplied matrix $B \times A$ (*i.e.,* ΔW) into base model W, which keeps computation overhead the same as the base model in Fig. 2(b).



(a) Unmerge mode. (b) Merge mode.

Figure 2. *LoRA model inference. (a) Unmerge mode supports computing multiple different LoRA adapters in a batch. A_1 and B_1 constitute LoRA adapter #1. (b) Merge mode supports no-extra-delay inference but only one adapter at once.*

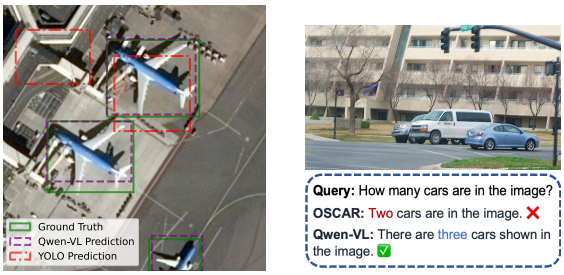
LoRA models serving system [25, 69, 82] targets to improve LoRA model inference efficiency. In unmerged inference, the core characteristic is that computing two small matrices of the adapter underutilizes GPU computing units, leading to unnecessary delay and resource waste. To tackle this, Punica [25] and S-LoRA [69] batch multiple heterogeneous adapters, as the cascade LoRA adapters illustrated in Fig. 2(a), and compute them within one single custom CUDA kernel. This method boosts the system throughput indeed but causes significant additional overhead (more in §3.2). dLoRA [82] merges the most accessed LoRA adapter into the base model and switches to unmerge if necessary. However, its mode switch causes an unacceptable time cost (more in §3.2). Besides, like Punica and S-LoRA, dLoRA's unmerged inference also fails to address the large number of extra computation overhead.

3 Motivation and Challenges

This section explores two questions: (1) What benefits can LoRA LMM bring to vision applications (§3.1)? (2) What challenges must be tackled when empowering vision applications with LoRA LMM (§3.2)?

3.1 Potential Benefits from LoRA LMM

LMMs offers state-of-the-art performance on many complex vision tasks. To demonstrate it, we take zero-shot grounding and visual question answering as examples, and conducted experiments on Aircraft [9] and VQAv2 [38] datasets with Qwen-VL-7B [20] (as the LMM), YOLO [33] and OSCAR [55] (as the baseline small models). Aircraft contains 103 remote sensing images that are not pre-trained on Qwen-VL and YOLO, being the zero-shot test; VQAv2 is the most popular visual question answering dataset which includes text and visual modalities, to test the multi-modal



(a) Zero-shot Grounding results on data #38 in Aircraft dataset [9].

(b) Visualization of data #133279 in Visual Question Answering [38].

Figure 3. *The potential of LMM. (a) To ground the airplanes in remote sensing view in zero-shot, LMM Qwen-VL, in general, delivers 67.2% accuracy* v.s. *the 18.3% of YOLO [33]. (b) In VQA, Qwen-VL yields 78.8% accuracy* v.s. *the 73.3% of OSCAR [55].*

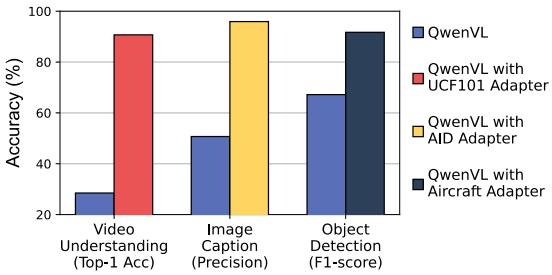


Figure 4. *LoRA adapters with domain-specific knowledge improve the Qwen-VL's accuracy on target tasks.*

ability. Fig. 3 shows that, with the solid linguistic and reasoning capabilities inherited from LLMs, Qwen-VL greatly outperforms small models. It delivers 48.9% higher F1-score in zero-shot grounding than YOLO. Fig. 3(a) visualizes the results on data #38, where Qwen-VL bounds more accurate boxes than YOLO. For multi-modal tasks, Qwen-VL achieves 78.8% accuracy, being 7.5% higher than OSCAR. Fig. 3(b) exemplifies a typical vehicle counting task in video analytics applications, only Qwen-VL generates the correct answer.

With external knowledge from LoRA adapters, LMM attains remarkable accuracy gain on domain-specific tasks. To investigate the accuracy improvement from external knowledge, we fine-tune three LoRA adapters for image classification, object detection, and video classification, respectively, on external datasets, AID [83], Aircraft [9], and UCF101 [73]. Fig. 4 shows the results. With fine-tuned LoRA adapters, Qwen-VL receives 45.2%, 24.5%, and 62.2% accuracy gains on three domain-specific task, respectively. Note that we only validated the potential gain without fully exploring advanced training techniques like data enhancement [94]. Nevertheless, based on current results, we believe these techniques could further improve accuracy in future work.

LoRA LMM enables more flexible serving. Today's vision applications are served by many domain-specific small models [49, 67]. When invoked, the specific model is loaded into GPU and swapped out to main memory after execution [21, 68]. This swapping method allows model execution in a large batch but incurs inevitable transmission costs. LoRA adapters usually have fewer parameters than small models (more in §4.4.1). Swapping them while keeping LMM in GPU provides a more flexible serving. In our test, swapping an adapter saves 97% delay than OSCAR, 15ms *v.s.* 520ms, and 86% of YOLO's 110ms.

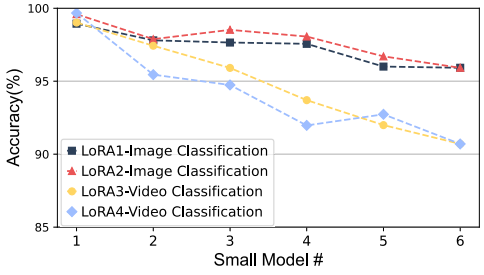


Figure 5. *Accuracy decreases when fusing knowledge from multiple domain-specific small models into one single LoRA. The trend varies regarding vision tasks.*

3.2 Challenges of Empowering Vision Applications with LoRA LMM

To empower diverse vision applications, the LoRA LMM system must offer accurate and efficient responses to the vision tasks involved and meet the distinct application-specified performance requirements. To this, we face three challenges.

C1: Limited capacity of LoRA adapter. Integrating external knowledge from existing small models or specific datasets into LMM is essential to generate accurate results. Parameter-efficient fine-tuning LoRA adapters show promise. However, it is challenging because the LoRA adapter has only limited capacity and varies on the vision tasks. Fig. 5 demonstrates this by fusing external knowledge from different numbers of small models on diverse tasks into a single LoRA adapter (experiment setup details in §6.1). Training a separate adapter for each small model consistently achieves high accuracy but results in significant adapter capacity waste, while fusing too many small models into one adapter incurs significant accuracy degradation. For example, the LoRA adapter that fuses six image classification models in Fig. 5 retains over 95% accuracy, while fusing six video classification models decreases remarkable accuracy.

C2: Inefficient concurrent requests batching. One vision application often involves multiple small models [31, 43]. Hence, serving multiple vision applications with LMM very likely leads to the simultaneous invocation of multiple heterogeneous LoRA adapters. However, current LoRA model inference systems struggle to process them efficiently, particularly under high concurrency. To demonstrate this, we measure three state-of-the-art systems. *Punica* [25] and *S-LoRA* [69] customize CUDA operator, respectively, to batch heterogeneous LoRA adapters computation in unmerge mode

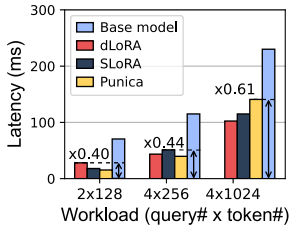


Figure 6. *Unmerged inference causes 27-140ms extra latency equivalent to 40-61% of base model inference time.*

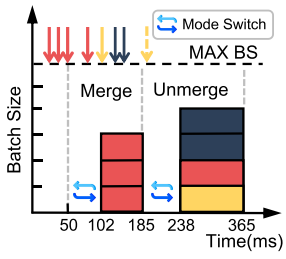


Figure 7. *Mode switch alone costs 53ms, occupying 64% of merged inference time of three 256-tokens requests.*

(more details in §4.3.1), while *dLoRA* [82] calls PyTorch operator `Einsum`† [3]. The experimental workload randomly generates 2-4 requests ranging from 128 to 1024 length of input tokens per second, and we repeat 1,000 times to measure the latency. For fairness, all experiments run on a server device equipped with NVIDIA A100 GPU [2] via PEFT [61] framework, and use Qwen-VL-7B [20] as the base model.

Fig. 6 plots the results. The bars of dLoRA, S-LoRA, and Punica denote their extra latency than the merged inference, *e.g.*, $latency_{dLoRA} - latency_{merge}$. The bar of the base model denotes its time cost under the same workload. Unmerged inference yields up to 140ms additional latency when serving four 1024-token requests. This unnecessary waste is sufficient for the base model to perform 4x256 inference, with resources to spare! The reason for such a high cost is two-fold. 1) Inherently, additional overhead stems from two additional matrix multiplications and one additional matrix addition per layer as depicted in Fig. 2(a); meanwhile, these computations run in parallel with the base model computations, each layer requires additional CUDA kernel context operations at each layer. 2) Upon concrete implementations, all three LoRA adapters batching operators fail to fully utilize computing units in GPU, due to the significant padding or ill-considered matrix tiling (more in §4.3.1).

C3: Inflexible LoRA adapters orchestration. To cope with the distinct performance requirements specified by vision applications, an orchestration that can carefully manage LoRA adapters and flexibly schedule application requests is necessary. We believe the inference mode switch like dLoRA is promising, yet it falls significantly short of efficiency. Its mode switch yields unacceptable overhead. Fig. 7 illustrates a real scheduling state and mode switch latency for two consecutive inference slots of dLoRA. In this case, dLoRA serves 8 requests, each with an input length of 256, in a first-come-first-service manner. In the first slot, dLoRA serves requests 1-3 in merge mode using the same LoRA adapter, and the heterogeneous requests 4-7 are processed in an unmerged mode in the following slot. A mode switch delay of over 53 ms makes the last request (the dotted arrow in Fig.7) have to wait 165ms until the next inference slot begins. This significant cost stems from 1) unnecessary memory copy of LoRA matrices due to dLoRA's inefficient memory management, and 2) the substantial overhead of LoRA matrices ΔW computation, by matrix multiplication $A \times B$, then added (merge) or subtracted (unmerge) ΔW onto or from the base model, by invoking `torch.addmm`, per layer. Conceivably, if the mode switch can be reduced to <10ms (as this paper achieved in §4.4.1), the average response time of Fig.7 case can save 45ms, with the last request only need to wait <80ms.

†Einsum uses Einstein summation convention [32], describing tensor index operations in a concise string form, for efficient LoRA adapter batching.

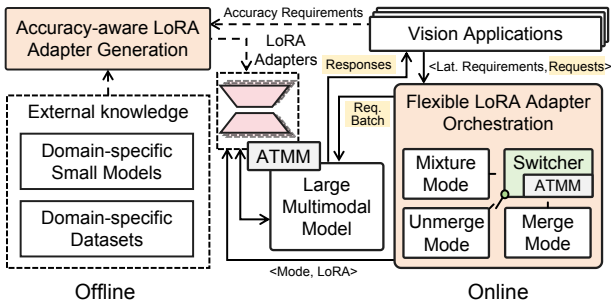


Figure 8. *VaLoRA overview.*

4 VaLoRA Design

VaLoRA is an end-to-end system that empowers diverse vision tasks and enriches vision applications with LoRA LMM by addressing the above challenges. We first provide an overview of VaLoRA's operation, then describe three core techniques it leverages.

4.1 System Overview

VaLoRA includes two phases. During the offline phase, the accuracy-aware LoRA adapter generation approach takes the external knowledge, from the existing domain-specific small models or datasets, as well as the accuracy requirements specified by vision applications (as the dotted arrows plotted in Fig.8), to generate the minimum number of LoRA adapters (§4.2). The generated LoRA adapters are rich in domain-specific knowledge that can output accurate responses to tasks involved in vision applications.

During the online phase, the flexible LoRA adapter orchestration ingests the requests from vision applications (as the solid arrow plotted in Fig.8), organizes them into batches, chooses the inference mode, and orchestrates their corresponding LoRA adapters, to minimize the average response latency while guaranteeing each vision application's latency constraint (§4.4). Each request batch is delivered to the corresponding adapters and LMM and inferred in the chosen mode. The LoRA adapter batching and inference mode switcher are implemented with ATMM, the adaptive-tilling matrix multiplication operator (§4.3), achieving high efficiency.

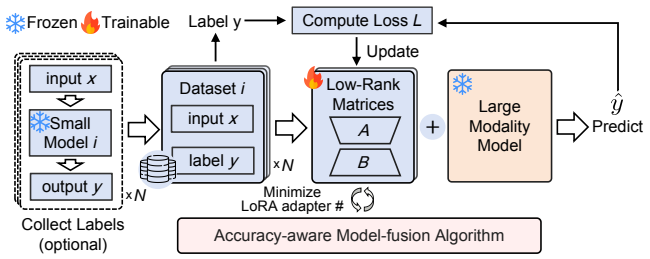


Figure 9. *Accuracy-aware LoRA generation integrates the domain-specific knowledge into LoRA adapters with the proposed accuracy-aware knowledge-fusion algorithm.*

4.2 Accuracy-aware LoRA Adapter Generation

To offer accurate results on domain-specific tasks, we propose the accuracy-aware LoRA adapter generation consisting of an accuracy-aware knowledge-fusion algorithm and the vision task head.

4.2.1 Accuracy-aware knowledge-fusion algorithm. To make it easy to manage at runtime, we aim to integrate external knowledge into the fewest LoRA adapters without violating the accuracy requirements of any vision tasks. To this end, the training method must account for the limited capacity of the LoRA adapter and the complex accuracy variations arising from the knowledge fusion.

This problem is highly challenging. Suppose we have an oracle who knows the accuracy of a LoRA adapter that fused arbitrary knowledge combinations in advance. Then, the problem can be formulated as a *constrained bin-packing problem* [36], where the objective is to pack the knowledge into the minimum number of LoRA adapter bins, ensuring each adapter maintains each vision task's accuracy beyond the requirement. However, this relaxed variant is NP-hard, and unfortunately, such an oracle does not exist.

To solve the original problem, we propose a simple and easy-to-implement heuristic algorithm, the *accuracy-aware knowledge-fusion algorithm*, to determine which knowledge fuses into one LoRA adapter. It first collects the dataset, as illustrated in Fig. 9, by executing representative data on every existing domain-specific small model; if applications provide the datasets, we directly use them. After that, the training process is a standard supervised learning pipeline that computes the cross-entropy loss, $L = CE(y, \hat{y})$, for parameter update. The knowledge fusion employs greedy and accuracy-aware heuristics. It begins with a random dataset and sequentially uses each dataset to train the LoRA adapter until its accuracy on a specific task falls below the required threshold. If this occurs, the adapter's weights are rolled back, and a new adapter is initialized to learn from the most recent dataset. The worst case of such a method may generate one LoRA adapter for each dataset, but in our practical experiments, every LoRA adapter fuses 4 domains of knowledge (datasets) on average.

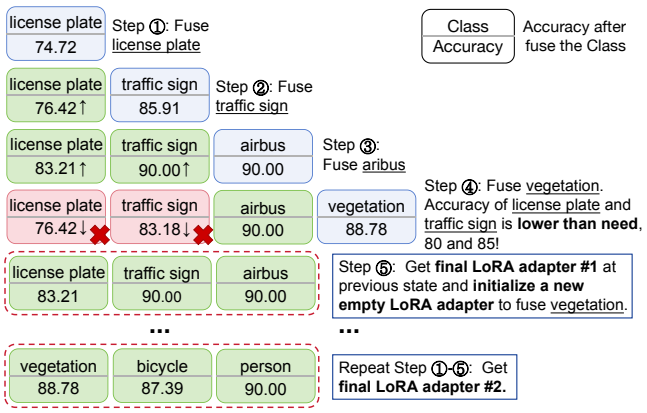


Figure 10. *An example of fusing the knowledge from six object detection models, each detects one class of object, with the accuracy-aware knowledge-fusion algorithm.*

Fig. 10 illustrates an example of integrating the knowledge of six object detection models, each detecting one class of target, into LoRA adapters. When fusing the vegetation-detect model (Step ④), the accuracy of LoRA adapter 1 fails on the license plate, needs above 80%, and traffic sign detection, 85%. Hence, the algorithm returns the LoRA adapter 1 of the previous state and initializes LoRA adapter 2 (Step ⑤). As the vegetation, bicycle, and person datasets are fused into LoRA adapter 2 without accuracy violation, we get the second LoRA adapter (Step ⑥). As LMM has powerful learning abilities, the training procedure costs only 25 minutes in this example. We leave the exploration of this to the future.

4.2.2 Vision task head. To reduce the inference latency, we design the vision task head. It is designed as a trainable linear layer as a part of the LoRA adapter, as shown in Fig. 11, to predict task-specific results based on the output features of LMM. Vision task heads can be flexibly customized to various vision tasks during the LoRA adapter training[‡], *e.g.*, action recognition head in Fig. 11, provided the fusing knowledge is from the same task type. Fig. 11 compares doing action recognition with the original language modeling (LM) head and the vision task head. By replacing LM head with the vision task head, LMM saves 4 inference rounds, around 180ms time cost. The reason to do so is that the outputs of a large portion of vision tasks are a limited discrete set of candidate options, such as the number of vehicle counts [57], classes of action recognition [92], and binary query for a specific target on image or video [88].

We retain the LM head for vision applications that need the natural language interface. For example, when video query applications ask for "A boy wearing a red sweater lost at the corner" and specify the person detection, VaLoRA will invoke the corresponding LoRA adapter containing a

[‡]As the vision task head is served as a part of the LoRA adapter, it is included in the cost of training a LoRA.

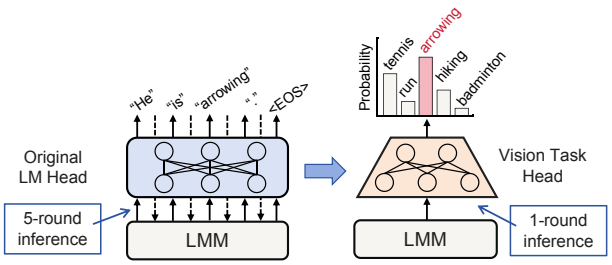


Figure 11. *Vision task head. Replacing the original language modeling head in LMM with vision task head reduces 4 rounds of LMM inference on the action recognition task of Fig. 1.*

Tiling Configuration	Input ① (256×4096, 4096×32)	Input ② (8192×4096, 4096×128)
Punica [25] (16, 64, 64, 16, 16, 64)	0.087ms Low SM Utilization	0.19ms Freq. Global Mem. Access
Config ① (64, 32, 32, 32, 32, 32)	**0.07ms**	0.12ms Freq. Global Mem. Access
Config ② (64, 64, 64, 32, 64, 64)	0.13ms Low SM Utilization	**0.10ms**

Table 1. *Adaptive tiling saves remarkable computing time compared to static-tiling Punica. Configuration (a,b,c,d,e,f) indicates the thread block tile a×b, b×c, and warp tile d×e, e×f. We omit thread tile to save space. Input (u×v, s×t) indicates input matrix shapes. The text under latency explains why this configuration performs poorly under corresponding input.*

detection head for efficient response[§]. By using the LoRA adapters batching operator, ATMM, in the next section, these different vision task heads can be executed concurrently, as well as in parallel with the LM head.

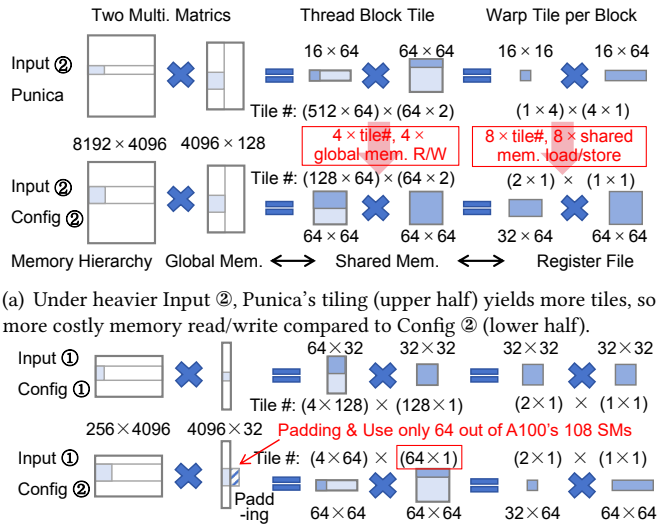
4.3 Adaptive-tiling LoRA Adapters Batching

Serving vision applications with LMM very likely invokes heterogeneous LoRA adapters concurrently. To compute them in high performance, we propose an Adaptive-Tiling Matrix Multiplication operator, ATMM, enabling efficient unmerged inference, inference mode switching (§4.4.1), and mixture inference mode (§4.4.2).

4.3.1 ATMM: Adaptive-tiling matrix multiplication operator. Directly batching heterogeneous adapters computation upon standard kernels, as batched GEMM (General Matrix Multiplication) in dLoRA [82], is feasible, but yields excessive latency and hardware underutilization. This stems from the significant padding arising from the heterogeneity of application request lengths and LoRA adapter ranks. Hence, a customized kernel is necessary.

To motivate our design, we first analyze two existing customized kernels from S-LoRA [69] and Punica [25], respectively. S-LoRA's kernel utilizes *tiling* technique to avoid the

[§]Some work, *e.g.*, task automation [56, 80] and dynamic LoRA [34], automatically identify adapters from queries. They are orthogonal to VaLoRA.



(a) Under heavier Input ②, Punica's tiling (upper half) yields more tiles, so more costly memory read/write compared to Config ② (lower half).

(b) But under lighter Input ①, Config ② (lower half) cannot deliver good performance because it underutilizes SM since fewer block number.

Figure 12. *Paired comparison of tilling configurations in Table 1. Following tiling configurations, two multiplied matrices are divided into thread block tiles and further warp tiles. The data in each tile is transferred to their corresponding memory.*

significant padding. Unlike batched GEMM, which pads heterogeneous input matrices to a uniform shape, it splits them into fine-grained blocks and computes the output for each block in parallel on CUDA cores. Punica's kernel also employs the tiling technique and further enhances efficiency by leveraging the well-developed CUTLASS [15] library and higher-performance Tensor cores [16]. However, as the motivational study in §3.2, both kernels fail to achieve satisfactory efficiency. The root cause is their static tiling configurations, which are inadequate to handle diverse input shapes, resulting in underutilized computational resources.

Our key observation is that the computational efficiency varies significantly with different tiling configurations. We conduct an experiment with two input shapes and three tiling configurations and present the results in Table 1. Under Input ① and Input ②, three tiling configurations yield the most 1.9× latency difference. This stems from the inappropriate tiling against input shape and hardware architecture, as well as between tiling levels. Fig. 12 compares two configuration pairs in Table 1. In Fig.12(a), Punica's smaller-tile configuration produces more thread block tiles and warp tiles. Compared to Config ②, it results in more frequent accesses to global and shared memory, so more launching data transfer times incur Punica's 1.9× latency. However, larger tile does not always benefit. In Fig.12(b), Config ②'s large tiling incurs Streaming Multiprocessor (SM) under-utilization. It can only utilize 64 of 108 SMs in A100, as each thread block tile can only fetch one SM, far less than Config ①. In sum, we need dynamic tiling to achieve efficiency.

We propose ATMM, an adaptive-tiling matrix multiplication CUDA kernel that fully utilizes computational resources,

achieving efficient heterogeneous LoRA adapters batching. It has two key design choices. (1) Adaptive tiling against input shapes. Given two input matrices, ATMM retrieves the optimal tiling configuration from the *hash table* (established in §4.3.2), and following this configuration, divides the matrix into *thread block tiles*, then further into *warp tiles*[¶]. (2) Pipeline data loading and computing. After tiling, ATMM transfers the data of each tile to their corresponding memories, the correspondence as shown in Fig. 12(a), and executes computation on CUDA/Tensor cores with the corresponding executable kernel (pre-compiled in §4.3.2). To hide data loading latency, ATMM allocates double the space in shared memory and register file for each tile: one for the current tile's computation and one for prefetching the next tile's data. Note that such double buffering refers to the use of dual buffers in shared memory, an on-chip cache (*e.g.,* 20MB in A100) independent from GPU memory (*e.g.,* 80G in A100). It is essentially a cache usage strategy without incurring global memory overhead. This double buffering is feasible due to each level's uniform tile shape, enabling efficient location of the next tile.

4.3.2 Profile-based optimal tiling search. Keeping the benefit of adaptive tiling in mind, we aim to search out the optimal tiling configuration for every different input and switch among them at runtime. This can be done offline, as it is only affected by input shapes rather than their numerical values. Some prior work [71, 85, 101] though studied the optimal tiling for matrix multiplication, it is still challenging as the complicated GPU thread parallelism mechanism and memory access patterns, and the large search space of input shapes.

To tackle this challenge, we approach the search as a black-box problem and propose a profile-based searching algorithm. It profiles the execution time for all possible ATMM input matrix shapes with the help of CUTLASS Profiler [15], records the optimal tiling configurations in a hash table, and compiles corresponding executable kernels. We use the following expert knowledge to reduce the search space. (1) From the hardware perspective, architectural characteristics restrict the feasible input shapes, *e.g.,* limited memory of each hierarchy, and tiling configurations, *e.g.,* every dimension of tile at least 16 and must be powers of two. (2) From the perspective of input data, the model dimensions of LMMs restrict the input matrix shape changing at large steps, *e.g.,* 4096 in Qwen-VL. By doing so, the search space can be reduced up to 20 times. For example, the total search space of Qwen-VL on A100 is 50,000 configurations, calculated by 288 configurations (A100's 36 common thread block shapes × 4 warp configurations × 2 instruction shapes) according to CUTLASS documentation [15] and Qwen-VL model's 2048 maximum context length, reduced down to 3,000. Appx. A shows the detailed algorithm. With our algorithm, searching all optimal configurations for vision tasks with Qwen-VL on A100 GPU only costs <30 minutes.

[¶]Each warp can be divided into thread tiles in <16,8,16> or <16,8,8> shape.

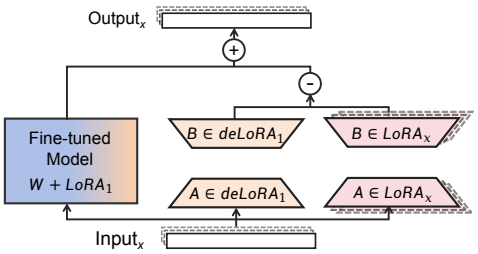


Figure 13. *Mixture mode allows simultaneous execution of unmerged and merged modes to alleviate starvation.*

4.4 Flexible LoRA Adapters Orchestration

To meet distinct performance requirements of vision applications, we propose an orchestrator to schedule requests, manage LoRA adapters, and switch inference modes, to enable efficient and flexible LoRA LLM inference runtime. We first implement two tools with ATMM, a swift mode switcher and a mixture inference mode, to facilitate the orchestrator.

4.4.1 Swift inference mode switch. As discussed in §3.2, prior systems (*e.g.,* dLoRA [82]) introduce excessive extra latency during mode switch. To reduce these overheads, one common method is pre-computing LoRA matrices (*i.e.,*, matrix ΔW, $B \times A$) for the entire base model and storing them in main memory, then swapping into GPU when needed. However, such a mass of data, *e.g.,* ~3GB per LoRA of QWen-VL-7B[‖], incurs very high delay, ~1s each, on swapping. Conversely, our swift mode switcher computes LoRA matrices at runtime and stores adapters, *i.e.,* A and B, only 43MB each, on GPU. It has two core designs. (1) Eliminate unnecessary memory copy with contiguous memory allocation. Our switcher pre-allocates contiguous memory for weight matrices, avoiding the memory copy in tensor reshape, enables efficient in-place LoRA matrices un-/merge. (2) Compute all-layer LoRA matrices and un-/merge them in one shot. With ATMM, the switcher can efficiently compute LoRA matrices of the entire model and add/subtract all of them onto/from the base mode weights in one shot. By doing so, our mode switch costs only <10ms, which speeds up dLoRA >5×.

4.4.2 Mixture inference mode. Compared to the unmerged mode, merged inference supports only one LoRA adapter at once, which results in the starvation of requests from other vision tasks. To alleviate starvation, we propose a novel inference mode, deLoRA, to enable the simultaneous execution of merged and unmerged inference. As shown in Fig.13, LoRA_1 handles the requests in merged mode, while other LoRA adapters, LoRA_x, process their requests in unmerged mode. To maintain the consistent results of LoRA_x's request, we introduce the deLoRA branch to prevent contamination from LoRA_1. The weight of deLoRA is the same as

[‖]Layer#×ΔW×precise, 32×4096×4096×FP16, ΔW's shape is same as LMM.

Algorithm 1 Scheduling Policy

Input: Request R={r_1, ..., r_n}, LoRA adapter L={l_1, ..., l_n}, Infer mode M
Output: The batch of requests to be executed B_{next}

1: **function** SCHEDULING(R, M, L)
2: $R_{starve} = [R_i.credit > \theta \ for \ R_i \ in \ R]$
3: $len = MaxBS - |R_{starve}|$
4: $R_{merge} = argmax_{l_i \in L} \{r_i \in R | r_i.lora == l_i\}$
5: **if** $|R_{starve}|/MaxBS \leq 0.5$ and $|R_{merge}|/MaxBS > 0.5$ **then**
6: **if** $|R_{starve}| == 0$ **then**
7: $M = Merge$ and MODESWITCH(M, $R_{merge}.lora$)
8: $B_{next} = R_{merge}$ [: $MaxBS$]
9: **else**
10: $M = Mix$ and MODESWITCH(M, $R_{merge}.lora$)
11: $B_{next} = R_{starve} + (R_{merge} - R_{starve})$ [: len]
12: INITDELORA(B_{next})
13: **else**
14: $M = Unmerge$ and MODESWITCH(M,)
15: $B_{next} = R_{starve} + (R - R_{starve})$ [: len]
16: **return** B_{next}

that of the merged LoRA. Based on the distributive property of matrix multiplication, the correctness can be verified as follows.

$$\begin{aligned}\mathbf{output_x} &= \mathbf{input_x} \times (\mathbf{W_{merge}} - \mathbf{W_{deLoRA_1}} + \mathbf{W_{LoRA_x}}) \\ &= \mathbf{input_x} \times (\mathbf{W_{base}} + \mathbf{W_{LoRA_x}}),\end{aligned}$$

in which $\mathbf{W_{merge}} = \mathbf{W_{base}} + \mathbf{W_{LoRA_1}}$, $\mathbf{W_{LoRA_1}} = \mathbf{W_{deLoRA_1}}$, and $\mathbf{W}$ means the weight of the base model, deLoRA, and LoRA adapter; $\mathbf{input_x}$ and $\mathbf{output_x}$ are the input and output of the request of LoRA_x. Overhead of computing W_{deLoRA_1} is exactly the same as $LoRA_1$, and subtracting it introduces very few overhead.

Mixture inference mode shows two advantages. 1) It does not incur mode-switching costs from merged to unmerged mode. 2) It costs less extra computation than unmerged inference when there are more requests for the merged LoRA adapter than others.

4.4.3 Scheduling policy. To minimize the average response latency and meet each request's latency constraint, our orchestrator must carefully orchestrate requests, adapters, and inference modes. Our policy follows a greedy heuristic which includes two principles. (1) Executing in merged mode whenever possible, as it produces the fastest response and without extra overhead. (2) When starvation occurs, switch to mixture first, then unmerge mode, in order of the switching cost and extra computation. Alg.1 shows the pseudo-code. To alleviate starvation, it assigns each request a credit, indicating its waiting time adds the execution time in current mode and the mode switch latency (line #2), and sets a tolerance threshold θ as the mixture mode condition. When the request workload meets the criteria for switching to merge mode, the algorithm switches the mode to merge (line #5-8). When the number of starving requests exceeds θ, Alg.1 processes them with mixture mode immediately (line #9-12). When it further exceeds half the maximum batch size, it switches to unmerge mode (line #13-15).

One may think that a static inference mode by one-shot search can work well without switching. However, it is hardly that in practice. Most applications experience dynamic workloads, making it difficult to define an optimal execution mode or order. For example, the video analytics application serves multiple users. The workload changes when new registered tasks (specified stream, tasks/adapter, accuracy) arrive for video analytics application, and the workload of LoRA adapters changes along with the multi-round VQA for visual retrieval application. Detailed results can be found in §6.3.

5 Implementation

We implement VaLoRA upon several tools, including Pytorch (v2.0.1) [6], Triton (v2.0.1) [75], CUTLASS (v3.5.1)[15], and vLLM (v0.3.0) [51], with ~7.1K LOC. Most of the code is implemented in Python (v3.9) except the ATMM in CUDA. We use vLLM, especially the LightLLM [1] version, to build VaLoRA because of its advanced features, such as PagedAttention, iteration-level scheduling, and token-based memory management. The three key techniques in VaLoRA are implemented as follows. (1) We implement the accuracy-aware LoRA adapter generation for popular LLMs, including Qwen-VL [20] and LLaVA [59] series, based on transformers [81] and PEFT [61] library. (2) We implement ATMM using CUDA C++ based on CUTLASS. Its hash table, which stores optimal input-tiling pairs, is implemented with a 128-bit unsigned integer as a key to map the input shapes. Since CUTLASS operators cannot be dynamically compiled, we created a Python interface to bind and package the code implementation of optimal tiling configurations with Pybind11 [41] and setuptools [5]. (3) We integrate ATMM into vLLM to support unmerged inference, mixture inference, and swift inference mode switch. To manage the complicated adapters and requests in unmerged and mixture mode, we transform the LoRA type of each request into a one-hot vector and build a request-type mapping matrix of the current batch. For efficient memory management, we use the unified memory management [69] for KV cache and adapters. VaLoRA streams text and images in requests asynchronously via RPyC [4]. After receiving a request, VaLoRA identifies its LoRA adapter, dispatches it to the adapter, then generates a response and returns it.

LoRA adapter swap. Considering it may generate an amount of LoRA adapters, we leverage the main memory. To reduce delay, we only store A and B, not ΔW, and swap them asynchronously between the GPU and host memory, and compute ΔW with ATMM at runtime.

Prefix caching. The same images may be accessed multiple times in some applications, *e.g.,* the multi-round visual question-answering [38]. We implement a prefix caching, based on CacheBlend [87] and SGLang [98], to reuse the same images' KV cache, avoiding redundant storage.

6 Evaluation

We evaluate VaLoRA with two vision applications involving five tasks on three LMMs. The key takeaways are:

• VaLoRA decreases 20-89% end-to-end latency compared to state-of-the-art serving systems and achieves comparable accuracy with small models on specific domains. (§6.2)

• Accuracy-aware LoRA Adapter Generation brings remarkable accuracy and throughput benefits; Adaptive-tiling LoRA Adapters Batching and Flexible LoRA Adapters Orchestration boost great latency and throughput. (§6.3)

• VaLoRA shows strong stability to diverse workloads and LoRA adapter numbers, as well as scalability to multiple GPU resources. (§6.4)

6.1 Experimental Setup

Vision applications and datasets. We select two distinct types of visual applications: visual retrieval and video analytics, to evaluate the performance of VaLoRA. Visual retrieval aims to analyze images and respond to queries. It involves visual question-answering, image caption, and specific-target detection tasks when needed by queries. We evaluate visual retrieval on SharedGPT-4V [26] and RefCOCO [48, 90] datasets. Video analytics ingests and analyzes each RGB frame from the video, then outputs results of fixed vision tasks, including object detection and video understanding like prior work [79, 92]. Object detection locates and identifies objects on each video frame on YODA [84] and Cityscapes [29]. Video understanding recognizes actions on consecutive video frames on UCF101 [73].

Testbed and workload. We evaluated VaLoRA on a server equipped with one NVIDIA A100 80GB GPU and Intel Xeon Platinum 8358 CPU, with 128 GB of host memory. The workload of visual retrieval is from production traces from the Microsoft Azure LLM inference trace 2023 [14]. As the high volume of requests in this workload trace exceeds current hardware capabilities, like prior work [82], we randomly sample requests at varying rates in a round-robin approach. Like prior work [31, 79], the video analytics workload ingests one video chunk per second, 30 frames each, each stream. Our scope is primarily focused on optimizing single-GPU, single-instance LMM services. If not specifically mentioned, the experiments are conducted on one instance of an A100 as reported above.

Metrics. Accuracy metrics follow the standard metrics. Visual retrieval is measured by vqa-score [37], object detection is evaluated by average F1-score and video understanding is evaluated via Top-1 accuracy. To the system metrics, we evaluate the end-to-end throughput, *i.e.,* requests per second, and the average token latency, *i.e.,* the sum of each request's end-to-end latency divided by the total number of tokens.

Models. We choose the widely-used open-sourced LMMs, Qwen-VL [20] and LLaVA-1.5 series [59]. We use various LLaVA models with different sizes, including LLaVA-v1.5-7B and LLaVA-v1.5-13B. The details of these models are shown in Table 2. The rank of LoRA adapters is set to 64. We also use five small models, VisionMamba [102], YOLO [33], OSCAR [55] VideoMAE [76], and UNINEXT [86] that yield state-of-the-art performance for accuracy comparison on corresponding datasets.

Model	Vision Encoder	Size	Layer #	Dimension
Qwen-VL-7B	Openclip-ViT (1.9B)	18GB	32	4096
LLaVA1.5-7B	CLIP-ViT (0.3B)	13GB	32	4096
LLaVA-1.5-13B	CLIP-ViT (0.3B)	24GB	40	5120

Table 2. *Model configurations.*

Baselines. To show the superiority brought by VaLoRA, we compare it with three different baselines:

• **S-LoRA [69]** only serves in unmerge mode and employs its customized CUDA operator to batch concurrent heterogeneous LoRA computation.

• **Punica [25]** also serves in unmerge mode only, like S-LoRA, but employs its own operator.

• **dLoRA [82]** dynamically switches between unmerged and merged mode based on workload and invokes PyTorch operator `Einsum` to implement unmerged inference.

6.2 End-to-End Performance

This section reports the E2E performance of VaLoRA on multiple LMMs and applications.

System performance. VaLoRA achieves notably lower average token latency than dLoRA, Punica, and SLoRA regardless of vision applications and LMMs. The first row in Fig. 14 shows their performance for visual retrieval on three LMMs. Across three LMMs, VaLoRA reduces 72%, 50%, and 20% average token latency, compared to dLoRA, Punica, and S-LoRA, respectively. To Punica and S-LoRA, this acceleration is obvious. They only work in unmerge mode, which ignores the merge-friendly workload pattern, *e.g.,* 60% of requests asking for the same LoRA adapter. dLoRA, though, takes the preference of both inference modes into account, its high mode switching cost and inefficient `Einsum` operator in unmerged inference incurs this 20% drop; conversely, VaLoRA's ATMM operator enables swift switch and efficient unmerge and mixture mode (more experiments in §6.3). Moreover, along with the increasing requests per second, the inflection points of most serving systems occur at 6, which can be eliminated if equipped with more GPUs (more in §6.4).

VaLoRA delivers the best service on video analytics than other systems, as shown in the second row of Fig. 14. It makes 89%, 83%, and 71% of average token latency reduction than dLoRA, Punica, and S-LoRA, respectively. This benefit arises from the vision task head. It effectively eliminates the multi-rounds inference in the autoregressive manner of LLMs (more experiments in §6.3.1).

Compare each column in Fig. 14. VaLoRA produces more remarkable benefits than other serving systems on the video

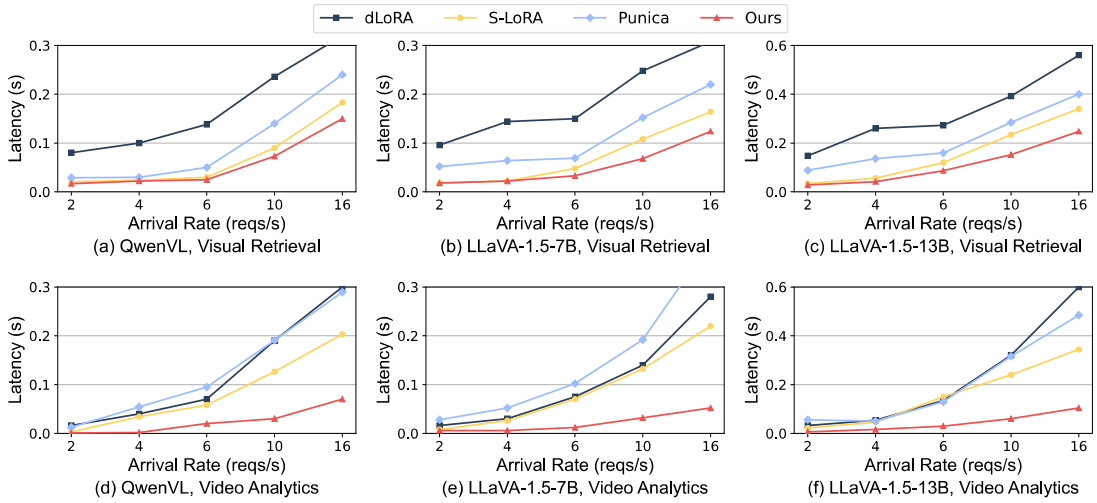


Figure 14. *Average token latency comparison over various serving systems on two vision applications and three LMMs.*

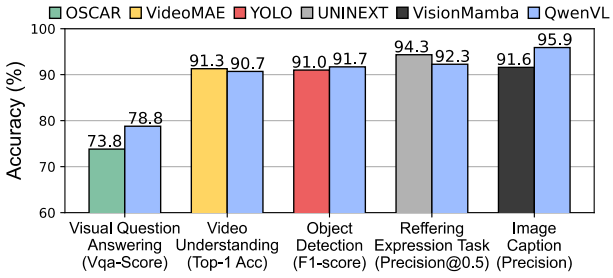


Figure 15. *Accuracy comparison between small models and LMM model across five different vision tasks.*

analytics application. This difference stems from the different distribution of input and output token lengths in two vision applications. Unlike visual retrieval, video analytics typically has fewer output tokens and more input tokens, *e.g.,* each video understanding request has 6×256 input and 5-10 output tokens, while VQA has 256 and 200+. Since input tokens can be batched computation in the prefill stage, they cost much less time (<1ms per token) than decode-stage output tokens (30-50ms per token). Furthermore, the extra latency of unmerged inference increases along with longer input length (See Fig. 17); thus, longer-input video analytics can obtain more benefits from VaLoRA with mode switch.

Accuracy performance. VaLoRA achieves performance close to or surpassing the SOTA accuracy of domain-specific small models across various tasks. We report the results of Qwen-VL with fine-tuned LoRA adapters for different vision tasks as shown in Fig. 15. We conduct training on A100 80GB for five distinct tasks and test against corresponding SOTA small models. For instance, VaLoRA achieves a 4.3-5% accuracy improvement in Visual QA and image captioning tasks. Additionally, for tasks where small models typically excel, such as object detection and video understanding, VaLoRA's fine-tuned LoRA adapters improve Qwen-VL 24.5-62.2% accuracy, achieving competitive accuracy in these domains.

6.3 Comprehensive Component-wise Analysis

We provide an in-depth performance analysis of individual system components. If not mentioned, all results are tested on the Qwen-VL model with 10 requests per second.

6.3.1 Accuracy-aware LoRA Adapter Generation helps VaLoRA achieve great throughput improvement while keeping high accuracy. The accuracy gain has been discussed above; we only analyze the throughput gain from the vision task head. Especially for video analytics tasks, the vision task head employed by VaLoRA significantly reduces the rounds of autoregressive decodes, thereby greatly enhancing system performance. As illustrated in Fig. 16, VaLoRA achieves a 41-63% reduction in latency compared to the original language modeling head. This gain is attributed to the video analytics head's contribution to minimizing the prompt length and requiring only one inference round. In video understanding tasks, VaLoRA equipped with the video analytics head can match the accuracy of certain small models and handle 3-4 video streams in real time.

6.3.2 Adaptive-tiling LoRA Adapters Batching gives the most efficient and stable matrix multiplication by ATMM among all comparisons. As shown in Fig. 17, by testing over 100 rounds after 10 warm-ups on large amounts of diverse inputs, ATMM achieves the lowest average latency across different batch sizes, speeds up 2.7×, 2.3×, and 3.4× of S-LoRA, Punica, and dLoRA, respectively. On stability, the statistical results plotted in Fig. 18 show that ATMM delivers the most robust performance, which reduce the latency fluctuation by 3×, 2×, and 2× compared to S-LoRA, Punica, and dLoRA. These benefits stem from the profile-based optimal tiling search at the offline phase. When the batch size exceeds 1024, for instance, ATMM adaptively adjusts to a larger tile shape, fully utilizing hardware resources, while other operators suffer from static tiling.

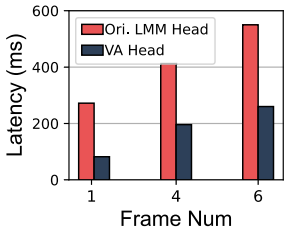


Figure 16. *Latency comparison between original LMM head and video analytics head.*

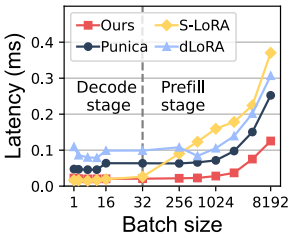


Figure 17. *Latency comparison of different operators across different token batch sizes.*

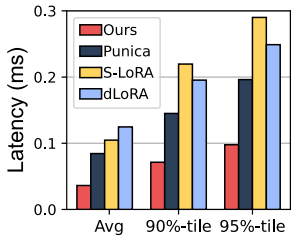


Figure 18. *Performance of different operators at average, 90-tile, and 95-tile .*

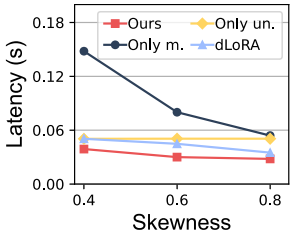


Figure 19. *Performance of different schedulers under different skewness.*

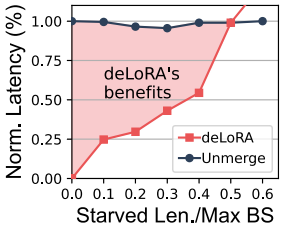


Figure 20. *Latency gain of mixture mode.*

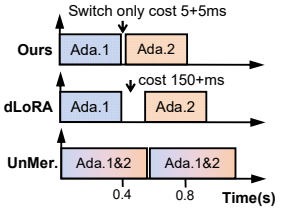


Figure 21. *Benefits from swift mode switch.*

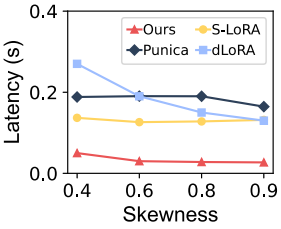


Figure 22. *Impact of the request skewness.*

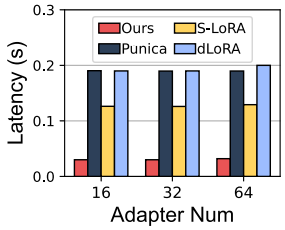


Figure 23. *Impact of the adapter number.*

At the decode stage with small input matrix shapes, the left part in Fig. 17, ATMM maintains high efficiency by adapting smaller tile shapes. It delivers comparable latency to S-LoRA, outperforming dLoRA and Punica by 4.5× and 2.6×. dLoRA suffers from the large context-switching overhead of repeated kernel calls by `Einsum`, while Punica results in a low core utilization due to its mismatched tile shapes. Benefiting from the adaptive tiling, ATMM spends only 5ms to compute and un-/merge all-layer LoRA matrices.

6.3.3 Flexible LoRA Adapters Orchestration dynamically selects and switches inference modes with our swift switcher and deLoRA operation offering the best service among comparisons. As shown in Fig. 19, VaLoRA outperforms merge only, unmerge only, and dLoRA by 33%, 59%, and 21% of latency under different skewness, respectively. The skewness indicates the proportion of the most required LoRA adapter. Merge only processes requests invoking the same LoRA adapter, leading to underutilized resources and small batch sizes, while unmerge only introduces significant extra computation. dLoRA shows benefits only in highly skewed workloads as the poor performance of its unmerged inference operator `Einsum`. Our scheduling policy performs the best because it fully utilizes low-latency merge mode, and invents mixture mode to eliminate some mode switch.

deLora significantly reduces the latency compared to unmerged inference as plotted in Fig. 20. Its early execution for the starved requests saves an average of 62% computation overhead when the starved requests' number is lower than 50% of max batch size. On the other hand, the swift inference mode switcher contributes a lot, too. Supported by ATMM, it yields 1.2× and 1.4× speed up compared to dLoRA and unmerge in Fig. 21 case that infers with two LoRA adapters.

6.4 Stability and Scalability

VaLoRA demonstrates great stability and scalability.

Impacts of different skewness of requests. VaLoRA achieves the best average token latency compared to other systems under diverse skewness. Fig. 22 shows that VaLoRA achieves a reduction in average token latency by 76-81%, 72-83%, and 63-76% compared to dLoRA, Punica, and S-LoRA under four different skewness conditions. This superiority arises from the VaLoRA's timely mode switch and proper requests and adapters orchestration. With the swift switcher and mixture mode, it responds to workload changes fast.

Impacts of different number of LoRA adapters. VaLoRA maintains the best and most stable performance when the number of LoRA adapters increases. As shown in Fig. 23, it suffers the minimal impact, which benefits from VaLoRA's efficient memory management. VaLoRA's pre-allocated contiguous memory reduces unnecessary memory copy or movement and memory fragmentation. In addition, when the number increases to need LoRA to swap, VaLoRA's strategy swaps adapter and computes matrix at runtime, and the asynchronous swap contributes to the low latency. With the highly optimized ATMM kernel, the LoRA matrix swapping keeps high stability compared to the matrix multiplication via batched GEMM in dLoRA.

Scales to multiple GPUs. VaLoRA demonstrates excellent scalability and can significantly enhance the overall system throughput by multiple GPUs. As shown in Tab. 3, on servers equipped with 1, 2, and 4 A100 GPUs, the total system throughput can reach 6.07, 11.48, and 23.97 requests per second, respectively. In future work, we can further improve

GPU Num.	TPT (req/s)
1	6.07
2	11.48
4	23.97

Table 3. *Scales to multiple GPUs.*

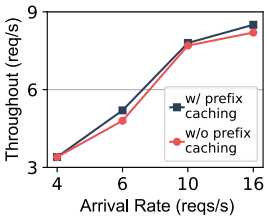


Figure 24. *Perf. breakdown of prefix caching.*

system performance in multi-GPU scenarios by incorporating inter-GPU scheduling like dLoRA [82] and support larger LMM like InternVL2-76B [35].

Impacts of prefix caching. VaLoRA maintains stable performance when removing prefix caching. As shown in Fig. 24, VaLoRA loses less than 4% of throughput after removing prefix caching. In VaLoRA, prefix caching is only a minor implementation supporting efficient multi-round VQA in visual retrieval applications.

7 Disscussion and Limitations

Limited flexibility of vision task head. The vision task head is deployed for one specific task and thus limits the flexibility. For this reason, VaLoRA keeps the language modeling head for visual retrieval applications. Even though, the vision task head, as a part of LoRA, facilitates efficient runtime by batching execution with ATMM. We leave improving the flexibility to future work.

Knowledge fusion order. The knowledge fusion order affects the final LoRA model quality, as illustrated in Fig. 5. Seeking the best order of training data is also an important but unsolved problem in the machine learning community, *e.g.,* curriculum learning and catastrophic forgetting problems in LLMs. In the future, we will borrow the idea from the ML community for knowledge fusion.

Knowledge pre-clustering. Cluster knowledge before fine-tuning LoRA adapters can also affect the final quality. VaLoRA clusters the data of the same vision task and keeps one LoRA adapter to serve one type of task. We leave combining VaLoRA with other data clustering methods to future work.

8 Related Works

Large model serving systems recently leveraged system optimization techniques to improve LLM's inference efficiency. With paged thinking, vLLM [50] proposes a PagedAttention operator cooperating with its block-based KV cache to minimize GPU memory fragmentation. From advanced batching mechanisms, Orca [89] introduces iteration-level scheduling to continuously batch requests of varying lengths, and DeepSpeed-FastGen [39] further improves it with a dynamic split-fuse strategy. With a distributed architecture, FlexGen [70] employs offloading to enhance LLM serving throughput, while Mooncack [64] features a KVCache-centric architecture separating the prefill and decoding clusters. With LLM inference characteristics, SpecInfer [62] utilizes speculative decoding to reduce latency, while SARATHI [18] schedules requests by piggybacking decodes and chunked prefills.

S-LoRA [69], Punica [25], dLoRA [82] are the only three systems that also serve multiple LoRA LLMs by batching requests destined for different adapters. Their shortcomings are deeply analyzed in this paper. An earlier study, PetS [99], also considers the scenario of serving multiple parameter-efficient DNN models, but it does not consider serving autoregressive LLMs and the unique system characteristics of LoRA adapters. Compared to them, VaLoRA provides a more efficient serving runtime and, as an end-to-end system, includes LoRA adapter generation.

Parameter-efficient fine-tuning (PEFT) [40, 54] is developed to adapt large pre-trained models to specific tasks or domains. By adjusting a few layers or parameters, PEFT retains core pre-trained knowledge while efficiently learning task-specific nuances [58]. As a verification system, VaLoRA's LoRA adapter generation adopts a heuristic algorithm. It can be easily replaced by many advanced PEFT techniques [96], and we leave this in future work.

Retrieval-augmented Generation (RAG) [22, 44, 52] enhances LLMs by incorporating relevant knowledge from external databases, enabling comparable performance to fine-tuned LLMs [24]. Some work [66, 78] suggest iterative retrieval throughout generation for higher quality. VaLoRA's system performance greatly outperforms RAG because RAG's costly vector search [27] and long-context prompt.

9 Conclusion

In this paper, we first explore the utilization of LMMs as foundation models for vision applications to achieve high serving efficiency and user-friendly nature language interface. To achieve this, we propose VaLoRA, an end-to-end system that adapts LMMs for domain-specific visual tasks with LoRA adapters and efficiently manages them at runtime to enrich vision applications. Across two typical vision applications, we show that VaLoRA enables the effective utilization of a single LMM to achieve superior performance and generalization in multiple visual tasks. While VaLoRA by no means is the final answer, we hope it serves as a stepping stone towards poly-basic design for future vision applications and demonstrates the potential of adapting LMM for visual tasks.

10 Acknowledgment

We would like to thank the anonymous reviewers and our shepherd Prof. Lai Fan, for their valuable guidance. Weijun Wang and Meng Li are the corresponding authors of this paper. This work is supported by Carbon Neutrality and Energy System Transformation (CNEST) Program, NSFC (No.62402280, No.62272261, No.62272223), CPSF (No.20 24M761683),

Shuimu Tsinghua Scholar Program (No.2023SM 201), Tsinghua University (AIR)-AsiaInfo Technologies (China), Inc. Joint Research Center for 6G Network and Intelligent Computing, Wuxi Research Institute of Applied Technologies, Tsinghua University under Grant 20242001120, the New Generation Information Technology Innovation Project 2023 (2023IT196), the Fundamental Research Funds for the Central Universities under Grant No. 2024300349, the Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing University, the Jiangsu High-level Innovation and Entrepreneurship (Shuangchuang) Program, and Beijing Academy of Artificial Intelligence (BAAI).

References

[1] Lightllm. https://github.com/ModelTC/lightllm.
[2] NVIDIA A100. https://www.nvidia.com/en-us/data-center/a100/.
[3] Pytorch torch.einsum. https://pytorch.org/docs/stable/generated/torch.einsum.html.
[4] rpyc. https://github.com/tomerfiliba-org/rpyc.
[5] setuptools. https://github.com/pypa/setuptools.
[6] Pytorch: Official website of the python platform. https://pytorch.org/, 2021.
[7] 12 Practical Large Language Model (LLM) Applications - Techopedia. https://www.techopedia.com/12-practical-large-language-model-llm-applications, (Accessed on 09/18/2024).
[8] 7 top large language model use cases and applications. https://www.projectpro.io/article/large-language-model-use-cases-and-applications/887, (Accessed on 09/18/2024).
[9] Airbus Aircraft dataset. https://www.kaggle.com/datasets/airbusgeo/airbus-aircrafts-sample-dataset/data, (Accessed on 09/18/2024).
[10] Applications of large language models - indata labs. https://indatalabs.com/blog/large-language-model-apps, (Accessed on 09/18/2024).
[11] Claude3.5-Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, (Accessed on 09/18/2024).
[12] OpenAI GPT-4o. https://openai.com/index/hello-gpt-4o/, (Accessed on 09/18/2024).
[13] Real-world use cases for large language models (llms). https://cellstrat.medium.com/real-world-use-cases-for-large-language-models-llms-d71c3a577bf2, (Accessed on 09/18/2024).
[14] Azure LLM inference trace 2023. https://github.com/Azure/AzurePublicDataset/blob/master/AzureLLMInferenceDataset2023.md, (Accessed on 10/07/2024).
[15] NVIDIA CUTLASS 3.5.1. https://github.com/NVIDIA/cutlass, (Accessed on 10/07/2024).
[16] NVIDIA Tensor Core. https://www.nvidia.cn/data-center/tensor-cores/, (Accessed on 10/07/2024).
[17] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
[18] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.
[19] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. 2023.
[20] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 2023.
[21] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. PipeSwitch: Fast pipelined context switching for deep learning applications. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 499–514, 2020.
[22] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning (ICML)*, pages 2206–2240, 2022.
[23] Arnav Chavan, Zhuang Liu, Deepak Gupta, Eric Xing, and Zhiqiang Shen. One-for-all: Generalized lora for parameter-efficient fine-tuning. *arXiv preprint arXiv:2306.07967*, 2023.
[24] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2024.
[25] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.
[26] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. Sharegpt4v: Improving large multi-modal models with better captions, 2023.
[27] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.
[28] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.
[29] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
[30] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
[31] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. Server-Driven Video Streaming for Deep Learning Inference. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 557–570. ACM, 2020.
[32] Albert Einstein. *The General Theory of Relativity*, pages 54–75. Springer Netherlands, Dordrecht, 1922.
[33] Glenn Jocher et. al. ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support, October 2021.
[34] Wenfeng Feng, Chuzhan Hao, Yuewei Zhang, Yu Han, and Hao Wang. Mixture-of-loras: An efficient multitask tuning for large language models. *arXiv preprint arXiv:2403.03432*, 2024.
[35] Zhangwei Gao, Zhe Chen, Erfei Cui, Yiming Ren, Weiyun Wang, Jinguo Zhu, Hao Tian, Shenglong Ye, Junjun He, Xizhou Zhu, Lewei Lu, Tong Lu, Yu Qiao, Jifeng Dai, and Wenhai Wang. Mini-internvl: A flexible-transfer pocket multimodal model with 5 *arXiv preprint arXiv:2410.16261*, 2024.
[36] Michael R Garey and David S Johnson. Approximation algorithms for bin packing problems: A survey. In *Analysis and design of algorithms*

in combinatorial optimization, pages 147–172. Springer, 1981.

[37] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6904–6913, 2017.

[38] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[39] Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, et al. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*, 2024.

[40] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[41] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 — seamless operability between c++11 and python, 2016. https://github.com/pybind/pybind11.

[42] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[43] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 253–266, August 2018.

[44] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*, 2024.

[45] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *VLDB Endowment*, 13(4), 2020.

[46] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10(11):1586–1597, August 2017.

[47] Daniel Kang, Francisco Romero, Peter D Bailis, Christos Kozyrakis, and Matei Zaharia. Viva: An end-to-end system for interactive video analytics. In *CIDR*, 2022.

[48] Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. Referitgame: Referring to objects in photographs of natural scenes. In *ACL conference on empirical methods in natural language processing (EMNLP)*, pages 787–798, 2014.

[49] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali, Yuanchao Shu, Mohammad Alizadeh, and Victor Bahl. RECL: Responsive resource-efficient continuous learning for video analytics. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 917–932, 2023.

[50] 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 *USENIX Symposium on Operating Systems Principles (OSDI)*, pages 611–626, 2023.

[51] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[52] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[53] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning (ICML)*, pages 19730–19742. PMLR, 2023.

[54] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of Annual Meeting of the Association for Computational Linguistics and the Joint Conference on Natural Language Processing*, pages 4582–4597, 2021.

[55] Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, et al. Oscar: Object-semantics aligned pre-training for vision-language tasks. In *Springer European Conference Computer Vision (ECCV)*, 2020.

[56] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 2024.

[57] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 359–376, 2020.

[58] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:1950–1965, 2022.

[59] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning, 2023.

[60] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, 2024.

[61] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

[62] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 932–949, 2024.

[63] Thomas Politzer. Vision is our dominant sense. https://www.brainline.org/article/vision-our-dominant-sense, (Accessed on 09/18/2024).

[64] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *arXiv preprint arxiv:2407.00079*, 2024.

[65] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[66] Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.

[67] Bhardwaj Romil, Xia Zhengxu, Ananthanarayanan Ganesh, Jiang Junchen, Shu Yuanchao, Karianakis Nikolaos, Hsieh Kevin, Bahl Paramvir, and Stoica Ion. Ekya: Continuous learning of video analytics models on edge compute servers. In *USENIX Symposium on*

Networked Systems Design and Implementation (NSDI), 2022.

[68] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. Nexus: A gpu cluster engine for accelerating dnn-based video analysis. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 322–337, 2019.

[69] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. S-lora: Serving thousands of concurrent lora adapters. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

[70] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning (ICML)*, pages 31094–31116, 2023.

[71] Yining Shi, Zhi Yang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Ziming Miao, Yuxiao Guo, Fan Yang, and Lidong Zhou. Welder: Scheduling deep learning memory access via tile-graph. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23)*, July 2023.

[72] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[73] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012.

[74] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[75] Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. MAPL 2019, page 10–19, New York, NY, USA, 2019. Association for Computing Machinery.

[76] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training, 2022.

[77] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[78] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

[79] Weijun Wang, Liang Mi, Shaowei Cen, Haipeng Dai, Yuanchun Li, Xiaoming Fu, and Yunxin Liu. Region-based content enhancement for efficient video analytics at the edge. *arXiv preprint arXiv:2407.16990*, 2024.

[80] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. In *Proceedings of the ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 543–557, 2024.

[81] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[82] Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 911–927, 2024.

[83] Gui-Song Xia, Jingwen Hu, Fan Hu, Baoguang Shi, Xiang Bai, Yanfei Zhong, Liangpei Zhang, and Xiaoqiang Lu. Aid: A benchmark data set for performance evaluation of aerial scene classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(7):3965–3981, July 2017.

[84] Zhujun Xiao, Zhengxu Xia, Haitao Zheng, Ben Y Zhao, and Junchen Jiang. Towards performance clarity of edge video analytics. In *IEEE/ACM Symposium on Edge Computing (SEC)*, pages 148–164, 2021.

[85] Jiarong Xing, Leyuan Wang, Shang Zhang, Jack Chen, Ang Chen, and Yibo Zhu. Bolt: Bridging the gap between auto-tuners and hardware-native performance. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 204–216, 2022.

[86] Bin Yan, Yi Jiang, Jiannan Wu, Dong Wang, Ping Luo, Zehuan Yuan, and Huchuan Lu. Universal instance perception as object discovery and retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15325–15336, 2023.

[87] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving with cached knowledge fusion. *arXiv preprint arXiv:2405.16444*, 2024.

[88] Juheon Yi, Sunghyun Choi, and Youngki Lee. EagleEye: wearable camera-based person identification in crowded urban spaces. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 1–14, April 2020.

[89] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 521–538, 2022.

[90] Licheng Yu, Patrick Poirson, Shan Yang, Alexander C Berg, and Tamara L Berg. Modeling context in referring expressions. In *European Conference on Computer Vision (ECCV)*, pages 69–85. Springer, 2016.

[91] Shan Yu, Zhenting Zhu, Yu Chen, Hanchen Xu, Pengzhan Zhao, Yang Wang, Arthi Padmanabhan, Hugo Latapie, and Harry Xu. Vqpy: An object-oriented approach to modern video analytics. In *Machine Learning and Systems (MLSys)*, 2024.

[92] Mu Yuan, Lan Zhang, Xuanke You, and Xiang-Yang Li. Packetgame: Multi-stream packet gating for concurrent video inference at scale. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2023.

[93] Tingting Yuan, Liang Mi, Weijun Wang, Haipeng Dai, and Xiaoming Fu. Accdecoder: Accelerated decoding for neural-enhanced video analytics. In *IEEE Conference on Computer Communications (INFOCOM)*, 2023.

[94] Sangdoo Yun, Seong Joon Oh, Byeongho Heo, Dongyoon Han, and Jinhyung Kim. Videomix: Rethinking data augmentation for video classification, 2020.

[95] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A. Lee. AWStream: adaptive wide-area streaming analytics. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 236–252. ACM, 2018.

[96] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations (ICLR)*, 2023.

[97] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*, 2023.

[98] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Efficiently programming large language models using

sglang. *arXiv preprint arXiv:2312.07104*, 2023.

[99] Zhe Zhou, Xuechao Wei, Jiejing Zhang, and Guangyu Sun. Pet: A unified framework for Parameter-Efficient transformers serving. In *USENIX Annual Technical Conference (ATC)*, pages 489–504, 2022.

[100] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.

[101] Hongyu Zhu, Ruofan Wu, Yijia Diao, Shanbin Ke, Haoyu Li, Chen Zhang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Wei Cui, Fan Yang, Mao Yang, Lidong Zhou, Asaf Cidon, and Gennady Pekhimenko. ROLLER: Fast and efficient tensor compilation for deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 233–248, Carlsbad, CA, July 2022. USENIX Association.

[102] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model, 2024.

A Profile-based Optimal Tiling Search Algorithm

Algorithm 2 Tiling Search(model, hardware)

Input: model, hardware
Output: Optimal configuration for each input shape

```
function TilingSearch(model, hardware)
    min, max=GetRange(model, hardware)      ▷ Limit the input size by limited memory
    configlist=GetConfig(harware)           ▷ Limit the tiling step s.t. the arch characteristics

    for shape in range(min, max, 32) do     ▷ Reduce the search step size based on the vision task characteristics
        for rank in ranklist do
            for tbtile in configlist[0] do
                for warptile in configlist[1] do
                    Profile(shape, tbtile, warptile)
                    UpdateBestConfig( )
```

B Prompt Template of Vision Applications

We report the prompt template for two vision applications, visual retrieval and video analytics, as shown in Fig. 25. Visual retrieval includes three tasks: referring expression task, visual question answering, and image captioning. The black text represents the prompt, and the blue text shows the response. Video analytics involves two tasks: object detection and video understanding. Object detection uses a similar prompt as the referring expression task. Video understanding provides multiple image frames as input, followed by an instruction to analyze the actions depicted in the sequence.

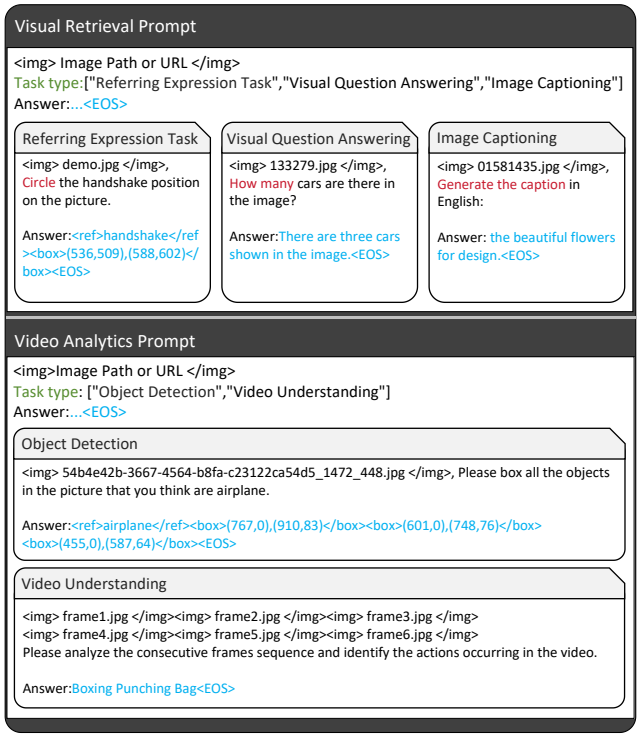


Figure 25. *The prompt template of vision applications.*