TOWARDS ELASTIC, ROBUST AND PRIVACY-PRESERVING AI MODEL SERVING

CHEN JINYU

PhD

The Hong Kong Polytechnic University

2025

The Hong Kong Polytechnic University
Department of Computing

Towards Elastic, Robust and Privacy-Preserving AI Model Serving

CHEN Jinyu

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
April 2025

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

Signature: _____

Name of Student: CHEN Jinyu

Abstract

AI model serving has become a cornerstone of intelligent applications, transforming industries and enhancing daily life through AI-driven services. The emergence of foundation models, such as GPT and Vision Transformers, has revolutionized AI services across diverse domains. These models, with billions of parameters, exhibit remarkable generalization capabilities but introduce substantial computational and deployment challenges, underscoring the need for efficient serving strategies to enable real-world adoption. However, modern AI model serving systems face several critical challenges. First, the rapid expansion of model size and complexity results in significant inference overhead, necessitating extensive computational resources and memory bandwidth. Second, the dynamic and unpredictable nature of query loads in AI services leads to severe latency fluctuations and resource contention. Third, user requirements vary significantly in terms of accuracy and response time, demanding flexible serving solutions capable of adaptively balancing efficiency and quality. Additionally, privacy concerns arise when deploying AI models in edge environments, where user data cannot be directly transmitted to centralized servers. To address these challenges, this thesis investigates techniques to enhance elasticity, robustness, and privacy preservation in AI model serving.

First, we develop the first elastic serving system specifically designed for Transformer models. Unlike conventional approaches that pre-train multiple model variants of different sizes to accommodate diverse service requirements, which result in pro-

hibitive I/O delays and excessive training costs, we propose a lightweight token adaptation mechanism for elastic Transformer serving. This mechanism dynamically adds prompting tokens to enhance accuracy and reduces redundant tokens to accelerate inference, thereby enhancing system elasticity. To further improve serving throughput, our framework integrates an application-aware selective batching strategy and an online token adaptation algorithm, which dynamically adjusts the token allocation scheme in real time. Experimental results demonstrate that our method significantly enhances serving throughput while maintaining high accuracy.

Second, while token reduction techniques effectively accelerate inference by dynamically removing redundant tokens, they often introduce unpredictable accuracy degradation under varying reduction ratios, compromising service robustness. To address this challenge, we introduce Prodigy, an elastic and robust Transformer serving system based on token-reduction warm-up. The core idea is to pre-train multiple warmed-up models at different token reduction levels, leveraging the insight that fine-tuning with token reduction significantly enhances inference accuracy. Instead of fine-tuning models for every possible reduction setting, we develop a strategic fine-tuning planner and a model ensemble method that enable robust inference across a wide range of reduction ratios with high efficiency. These approaches substantially improve service quality while reducing the computational and storage costs for fine-tuning.

Third, to enable privacy-preserving optimization, we propose a fast multimodal edge inference framework with a selective feature distillation method. Our method selectively distills knowledge from a pre-trained model in the cloud by uploading only feature representations for public data selection, effectively preventing user data leakage. Additionally, we introduce a privacy-preserving feature clustering mechanism that transmits only prototype-based representations of local features, further enhancing security. To accommodate varying communication bandwidths, we design an adaptive feature compression module that efficiently reduces transmission costs. Experimental results demonstrate that the proposed framework ensures strong privacy

protection, optimizes resource utilization, and maintains high inference accuracy.

In summary, this thesis presents a set of innovative techniques to improve the elasticity, robustness, and privacy-preserving of AI model serving. Through extensive experiments and evaluations, we demonstrate that our proposed methods significantly enhance serving system performance across diverse real-world scenarios. These contributions pave the way for future advancements in scalable and advanced AI model deployment, ultimately fostering more intelligent, efficient, and trustworthy AI services for society.

Publications arising from the thesis

- [**TMC**] **Jinyu Chen**, Wenchao Xu, Yunfeng Fan, Haozhao Wang, Quan Chen, and Jing Li. "*Fast Multimodal Edge Inference via Selective Feature Distillation*", IEEE Transactions on Mobile Computing, 2025.

- [**INFOCOM**] **Jinyu Chen**, Wenchao Xu, Zicong Hong, Song Guo, Haozhao Wang, Jie Zhang and Deze Zeng. "*OTAS: An Elastic Transformer Serving System via Token Adaptation*", IEEE International Conference on Computer Communications, 2024.

- [**COMST**] Wenchao Xu, **Jinyu Chen**, Peirong Zheng, Xiaoquan Yi, Tianyi Tian, Wenhui Zhu, Quan Wan, Haozhao Wang, Yunfeng Fan, Qinliang Su, and Xuemin Shen. "*Deploying Foundation Model Powered Agent Services: A Survey*", IEEE Communications Surveys & Tutorials, 2025.

- [**ECCV**] Yunfeng Fan, Wenchao Xu, Haozhao Wang, Fushuo Huo, **Jinyu Chen**, and Song Guo, "*Overcome Modal Bias in Multi-modal Federated Learning via Balanced Modality Selection*", European Conference on Computer Vision, 2024.

Acknowledgments

This thesis is the result of years of dedication, persistence, and invaluable support from numerous individuals. I am profoundly grateful for the guidance, collaboration, and encouragement that I have received throughout this journey.

First and foremost, I would like to express my deepest gratitude to my supervisors, Assistant Professor Wenchao Xu, Professor Song Guo and Professor Bin Xiao, for their unwavering support and insightful guidance. Prof. Wenchao Xu is an exceptional mentor, whose profound expertise, strategic vision, and rigorous academic standards have greatly shaped my research direction and growth. His encouragement and constructive feedback have been instrumental in refining my ideas and advancing my work. I am also sincerely thankful to Prof. Song Guo for his valuable advice, patience, and continuous support during my research. His dedication and enthusiasm have made our collaboration both productive and enjoyable. I would also like to extend my sincere appreciation to Prof. Bin Xiao for his invaluable support, which have been instrumental in my academic development.

I am also immensely grateful to my collaborators, Dr. Jie Zhang, Dr. Zicong Hong, and Dr. Haozhao Wang, for their support, valuable discussions, and insightful contributions to my research. Their expertise and willingness to share knowledge have made our collaborations both productive and enjoyable. The stimulating discussions and joint efforts with them have greatly enriched my research work and broadened my academic perspective.

Furthermore, I am grateful to my friends who have made this academic journey more fulfilling with their camaraderie, encouragement, and intellectual exchanges. Their presence has provided both motivation and inspiration during my Ph.D. studies.

Finally, I would like to express my deepest gratitude to my parents. Their unwavering support, unconditional love, and immense patience have been my strongest pillars throughout this journey. Without their encouragement and understanding, this achievement would not have been possible.

This thesis marks the culmination of an important chapter in my life, and I am truly thankful to everyone who has played a part in making it possible.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis explores effective methods to enhance elasticity, robustness, and privacy-preserving for AI model serving systems. This is a crucial research problem as AI services continue to expand into critical domains (e.g., healthcare and finance), where the quality of services significantly impacts the potential and societal value of AI. However, existing model serving frameworks face significant challenges due to the significant inference overhead, dynamic query workloads, and diverse user requirements. Addressing these challenges is critical to ensuring scalable and high-performance AI services. In this thesis, we first introduce the research background and significance in section 1.1 and discuss the key challenges in section 1.2. Then, we outline our research framework in section 1.3 and summarize the main contributions in section 1.4. Finally, we provide an overview of the thesis organization in section 1.5.

1.1 Research Background and Significance

Deep learning models have catalyzed remarkable advancements in a broad spectrum of applications, ranging from computer vision and natural language processing to recommendation systems and autonomous driving [8, 44, 91, 10]. They derive their

strength from deep neural network architectures, which learn hierarchical feature representations from large datasets. As these networks continue to scale in both depth and breadth, they exhibit ever-improving performance and accuracy in tasks once deemed unattainable for traditional machine learning techniques. This transformation is further fueled by the abundance of computational resources and the proliferation of massive datasets, enabling deep learning models to reach unprecedented levels of performance [39, 27]. The impact of deep learning thus extends well beyond specialized research settings and has penetrated mainstream industries, influencing the design and deployment of modern data-driven solutions worldwide.

Building on the success of deep learning, foundation models have emerged as powerful, general-purpose models trained on diverse and extensive data sources [3]. These models, illustrated by large-scale vision and language models, are capable of adapting to a wide range of downstream tasks through fine-tuning or prompt learning [33]. Their broad applicability and strong performance across different domains underscore their significance as a key technological milestone in AI research. Foundation models serve as versatile building blocks, offering an efficient way to leverage shared representations and fostering a more unified approach to machine learning development [4, 18]. Furthermore, their potential for transfer learning and domain adaptation reduces the need to build specialized models from scratch, accelerating innovation across areas such as healthcare, finance, and robotics. Consequently, the rise of foundation models has prompted both academic and industrial sectors to rethink how AI technologies are conceived, refined, and deployed at scale.

As deep learning and foundation models mature, the concept of AI model serving has become increasingly critical. AI model serving involves the AI algorithms, system infrastructures, and best practices for deploying and managing trained models in production environments [88, 45]. Figure 1.1 illustrates a typical AI model serving system, where client devices from various AI applications send queries to a cloud-based server cluster. The server processes these queries using deployed AI models

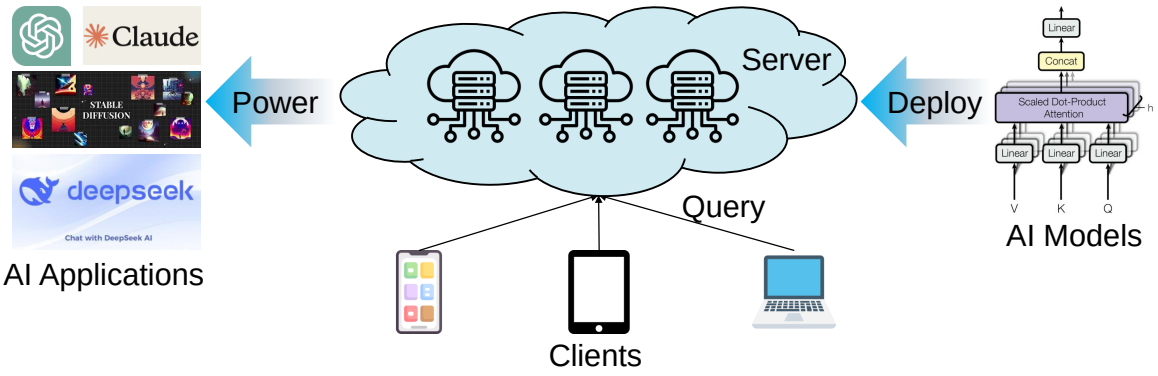


Figure 1.1: Illustration of the Serving System for AI Models. This figure illustrates how client devices in various AI applications send queries to a cloud-based server cluster, which processes them using deployed AI models and returns the results.

and returns the results, ensuring efficient and scalable AI inference. Effective serving mechanisms ensure that AI solutions are readily accessible, efficiently utilized, and seamlessly integrated into real-world workflows. Such systems must handle diverse workloads, meet rigorous latency and throughput requirements, and support continuous model updates. Beyond mere accessibility, AI model serving is key to the practical success of AI-enabled applications, as it underpins scalability, reliability, and user satisfaction. With the growing influence of AI across sectors, research on efficient model serving has gained prominent significance, driving efforts to optimize resource usage, maintain robust system performance, and uphold stringent data privacy standards [55]. By focusing on elasticity, robustness, and privacy-preserving optimization, this thesis addresses these critical aspects of AI model serving and contributes to the broader goal of making advanced AI models widely and effectively deployed in society.

1.2 Challenges of AI Model Serving

Despite significant advancements in AI model serving, several fundamental challenges remain, hindering the efficiency and scalability of AI systems. These challenges stem from the inherent complexity of modern AI models, fluctuating inference workloads,

and the diverse requirements of end users.

Significant Inference Overhead [87, 23, 56]. One of the primary hurdles in AI model serving is the substantial overhead associated with model inference, particularly for large-scale deep learning models. Modern neural networks, such as large language and vision models, require considerable amounts of compute and memory at inference time. These computational demands often necessitate high-end hardware, such as GPU clusters or specialized accelerators, which can be expensive to acquire and maintain. In addition, transferring large models and their associated data across different computational nodes introduces further latency and overhead, especially in distributed or cloud-based environments. Beyond the hardware requirements, the large memory footprints and power consumption of complex models can strain limited computational resources, making it challenging to deliver consistent performance. Even for relatively smaller models, the overhead can accumulate rapidly under high-traffic conditions, exacerbating the burden on already constrained infrastructure.

Dynamic Query Load [70, 90, 19]. Another significant challenge arises from the unpredictable and fluctuating nature of user requests. Real-world AI services often face sudden increases in query volume, putting pressure on the system and potentially leading to delays or degraded performance. These dynamic workloads make capacity planning difficult: provisioning too many resources during low-traffic times leads to inefficient usage and unnecessary cost, while under-provisioning during surges can lead to degraded performance, increased latency, and the possibility of dropped requests. Furthermore, different applications have very different request patterns, depending on factors like time zones, promotions, or viral content. To keep things running smoothly, it's important to monitor and manage these changing workloads carefully, preventing slowdowns that could affect users.

Diverse User Demand [89, 95, 46]. Complicating matters further is the wide range of user requirements and performance objectives. Some users prioritize highly accurate results, which often involve running heavier or more sophisticated models. Others

care more about minimal latency, especially for interactive applications like real-time data analytics or online recommendation engines. This diversity of demand forces AI service providers to balance competing goals: employing larger, more complex models for improved accuracy while still delivering responses quickly enough to maintain a positive user experience. Additionally, certain application domains, such as healthcare or finance, have strict requirements for reliability and correctness, whereas others, such as casual consumer applications, may tolerate slightly lower accuracy if it translates into faster response times. Accommodating these varied expectations without compromising service quality presents an ongoing challenge for AI model serving.

1.3 Thesis Framework

To address the challenges associated with advanced AI model serving, this thesis presents a series of systematic solutions aimed at improving the elasticity, robustness, and privacy-preserving capabilities of AI inference systems. As illustrated in Figure 1.2, our research is structured into three main parts, each tackling a specific dimension of AI model serving. In chapter 3, we focus on improving system elasticity by developing a dynamic token adaptation strategy for Transformer-based model serving. This method leverages token prompting to enhance accuracy and token reduction to accelerate inference. In chapter 4, we focus on enhancing system robustness by mitigating the accuracy degradation introduced by token reduction techniques. To mitigate this issue, we propose PRODIGY, an elastic and robust Transformer serving system that integrates a token-reduction warm-up strategy and a model ensemble method. In chapter 5, we focus on privacy-preserving AI model serving by developing a fast multimodal edge inference framework that enables secure and efficient inference on edge devices. Specifically, we introduce a selective feature distillation method that transmits only feature representations for public data selection and a personalized knowledge distillation method to obtain a lightweight yet effective model, optimizing

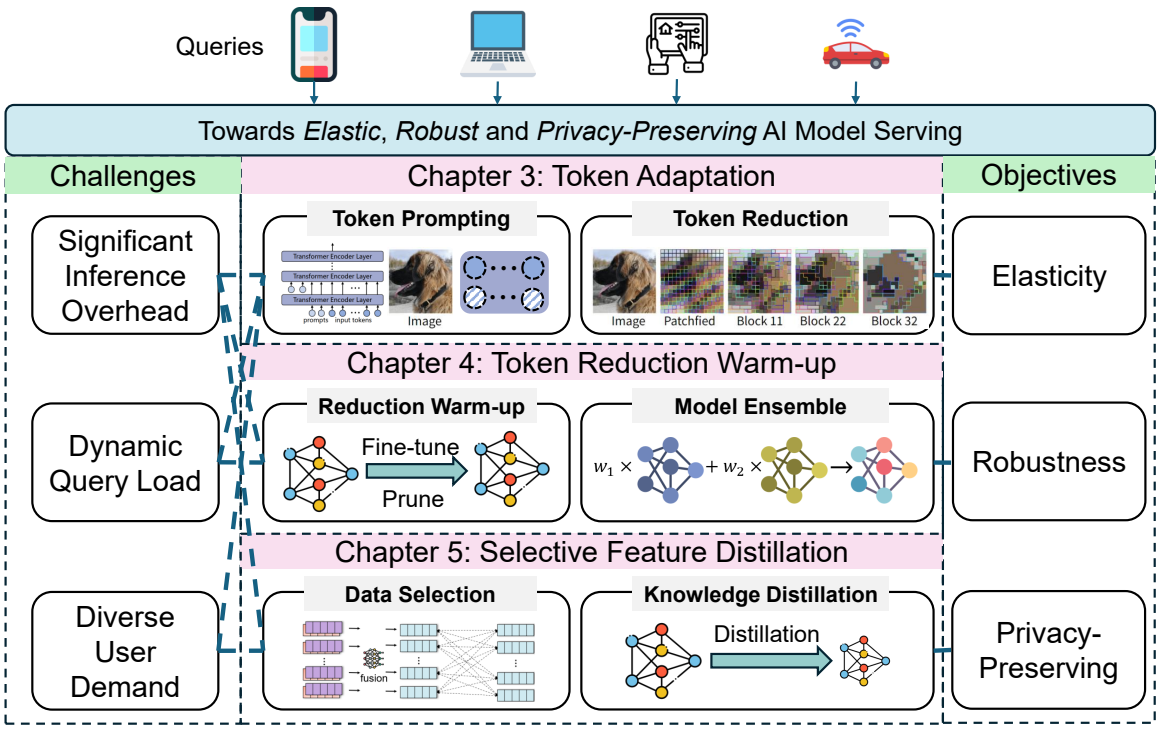


Figure 1.2: The theoretical framework for this thesis.

inference efficiency and accuracy.

1.4 Thesis Contributions

We briefly summarize our contributions below.

1. **Elastic Transformer Serving via Lightweight Token Adaptation.**

   In model serving, elasticity enables a system to dynamically adjust its inference processes in response to fluctuations in available hardware resources, incoming request rates, and user demands. To enhance this capability for Transformer-based models, we propose a novel token adaptation approach that dynamically adjusts the number of tokens to optimize both accuracy and efficiency. Our method introduces prompting tokens to enhance model performance while eliminating redundant tokens to accelerate inference. This enables an adap-

tive serving system that responds dynamically to fluctuating query loads and heterogeneous user requirements. Additionally, we introduce selective batching, which groups queries with similar service-level objectives into batches for inference, thereby improving system throughput. To effectively handle varying loads and diverse user demands, we develop an online token number allocation algorithm that balances accuracy gains with inference overhead, significantly enhancing serving efficiency while maintaining high prediction accuracy.

2. **Robust Transformer Serving with Token-Reduction Warm-up.**

   Although token reduction accelerates Transformer inference by eliminating redundant tokens, it often suffers from significant accuracy degradation as the reduction ratio increases, leading to undesirable robustness. To address this challenge, we propose PRODIGY, an elastic and robust serving system that leverages token-reduction warm-up to enhance inference accuracy. Motivated by the observation that fine-tuning with awareness of token reduction significantly improves performance, PRODIGY incorporates a warm-up module to strategically fine-tune a small subset of models at different token reduction levels, thus avoiding exhaustive training for every possible ratio. Additionally, we introduce a model ensemble technique to ensure effective generalization across arbitrary reduction settings. Our approach substantially enhances service robustness while mitigating the computational and storage overhead associated with fine-tuning.

3. **Privacy-Preserving Multimodal Inference via Selective Feature Distillation.**

   To enable privacy-preserving optimization and high-fidelity inference on resource-constrained edge devices, we propose a selective feature distillation framework tailored to multimodal data. In this framework, the client transmits only extracted features to the cloud server, where relevant public data is selected

Chapter	Existing Approaches	Their Limitations	Our Approach	Performance
Chapter 3	Model Adaptation	I/O Delay	Token Adaptation	+18.2% Service Utility
Chapter 4	Raw Model for Token Reduction	Poor Robustness	Token-adaptive Fine-tune	+27% Accuracy
Chapter 5	Distill a Small Model	Lake Generalization	Selective Feature Distillation	+2.12%∼6.48% Accuracy

Table 1.1: Comparison of our proposed methods with existing approaches across three technical chapters.

and used for knowledge distillation, thereby yielding a personalized lightweight model for edge deployment. We further incorporate a clustering-based prototype mechanism, which replaces raw features with cluster prototypes to protect user privacy. Additionally, to handle dynamic network conditions, we introduce an adaptive feature compression module that reduces communication costs while maintaining inference quality. Extensive experiments confirm that this approach ensures data confidentiality, optimizes resource utilization, and achieves competitive accuracy in real-world multimodal edge inference scenarios.

The comparison between existing approaches and our proposed methods across different chapters is summarized in Table 1.1.

1.5 Thesis Outline

The rest of this thesis consists of six chapters and is organized as follows. In chapter 2, we provide the background necessary to contextualize the subsequent chapters. Specifically, we survey related work on serving systems for AI models, introduce foundational concepts of AI models, and discuss the notion of token adaptation. In chapter 3, we present OTAS, an elastic Transformer serving system that harnesses

token prompting and token reduction to flexibly adjust token numbers based on query load and user requirements. In chapter 4, we propose PRODIGY, which extends token-reduction techniques by introducing a "warm-up" strategy to achieve robust inference accuracy across a wide range of token reduction ratios. In chapter 5, we focus on fast multimodal edge inference, in which a privacy-preserving and communication-efficient distillation framework is devised to support edge devices with limited resources. Finally, in chapter 6, we summarize the key contributions of this thesis and highlight promising future directions for advancing efficient, robust, and privacy-preserving AI model serving.

Chapter 2

Background and Literature Review

This chapter provides a comprehensive overview of the background and related works in AI model serving. We begin by introducing the fundamental principles of AI model serving and the critical role of efficient serving systems. Next, we review foundational AI model architectures, such as Convolutional Neural Networks (CNNs) and Transformers, along with emerging techniques in prompt learning and token reduction, which are increasingly relevant to optimizing AI model serving.

2.1 Preliminary for Serving System

The performance of a serving system plays a pivotal role in determining the overall quality of service for online AI applications. These applications typically face rapidly changing query loads and sudden bursts in request traffic, which demand systems that can adapt quickly and efficiently. Additionally, AI workloads often involve intensive inference operations that need to be executed under strict low-latency constraints, further highlighting the importance of an efficient serving system. To address these challenges, numerous research efforts have focused on developing and refining elastic computing strategies aimed at improving the efficiency and scalability of serving

systems. These existing research efforts can be broadly classified into two categories.

(1) *Model adaptation.* Model adaptation in serving systems aims to dynamically select and adjust AI models for inference tasks based on the current execution context, including available computational resources, request patterns, and application-specific requirements. By balancing trade-offs among performance, efficiency, and cost, this approach enhances the feasibility and adaptability of AI applications across different scenarios. While model adaptation brings substantial benefits, it also introduces several challenges. First, computational heterogeneity makes it difficult to optimize models across different environments. Edge devices operate under strict resource constraints, while cloud systems must manage cost and resource contention. Second, the operational environment is highly dynamic, requiring adaptive mechanisms that respond to fluctuations in network conditions, workload demands, and system constraints without excessive overhead. Third, achieving the right trade-off between inference speed, model accuracy, and cost remains complex, particularly in latency-sensitive or high-throughput applications.

To address these challenges, researchers have proposed model selectors that dynamically choose the most suitable model based on available resources, user inputs, and application requirements, as illustrated in Figure 2.1(a). These selectors optimize the trade-off between accuracy and latency, ensuring efficient AI model serving under varying conditions. INFaaS is an automated model selection serving system, which generates model variants optimized along different dimensions and automatically selects the most appropriate variant for each query based on performance, cost, and accuracy objectives [70]. The objective function optimizes for cost-efficient scaling: Minimize $\sum_{i,j} C_{ij}(\delta_{ij} + \lambda T_{\text{load},ij} \max(\delta_{ij}, 0))$, where C_{ij} is the hardware cost per second for model variant v_{ij}, $T_{\text{load},ij}$ represents the loading latency of the model variant, and λ is a tunable parameter balancing cost and response time. INFaaS employs a greedy heuristic that approximates the optimal scaling decision with sub-second response time. This heuristic estimates the current load capacity and selects between repli-

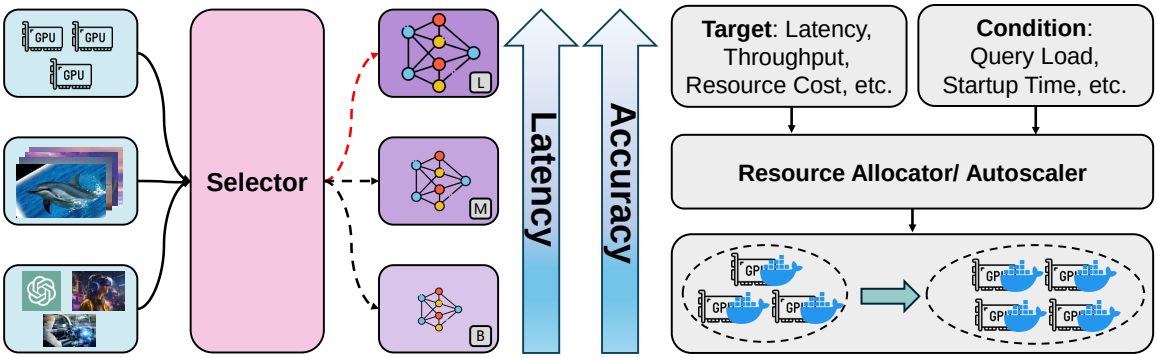


Figure 2.1: Two approaches to build an elastic serving system.

cating an existing model or upgrading/downgrading to a more suitable variant based on cost-effectiveness. EdgeAdaptor is designed to efficiently manage the trade-offs between accuracy, latency, and resource costs for edge-based Deep Neural Network (DNN) inference services. This framework addresses the problem by jointly optimizing application configuration, DNN model selection, and edge resource provisioning dynamically, in response to fluctuating demand and system conditions [95]. The optimization objective aims to minimize the holistic cost while satisfying latency and accuracy constraints. A two-step approach decomposes the long-term problem into single-shot fractional problems via regularization and rounds the solution using a randomized dependent scheme, ensuring feasible and efficient model selection. JellyBean is a system designed for optimizing and serving machine learning inference workflows across heterogeneous computing infrastructures [85]. For each ML operator within a workflow, JellyBean selects a model that meets the accuracy requirements at the lowest possible cost. This process considers the interaction between models to estimate the overall workflow's accuracy and utilizes a beam search algorithm to explore the space of possible model configurations efficiently.

(2) *Resource scheduling.* As shown in Figure 2.1(b), adaptive resource allocation is crucial for achieving an optimal balance between system performance and cost in serving systems. Resource management facilitates the optimal utilization of system

resources, including processing power, storage space, and network bandwidth. To meet dynamic workload demands, adaptive resource allocation strategies employ intelligent scheduling and autoscaling mechanisms that respond to real-time variations in query load, latency requirements, and resource availability.

Clipper uses model containers to encapsulate the model inference process in a Docker container [13]. It supports replicating these model containers across the cluster to increase the system throughput and utilize additional hardware accelerators for serving. Nexus adopts the squishy bin packing method to batch different types of tasks on the same GPU, enhancing resource efficiency by considering the latency requirements and execution costs of each task [71]. It also merges multiple tasks into the same GPU execution cycle as long as the latency constraints are not violated. InferLine utilizes a low-frequency planner and a high-frequency tuner to manage the machine learning pipeline effectively [12]. The low-frequency combinatorial planner finds the cost-optimal pipeline configuration under a latency requirement. The high-frequency auto-scaling tuner monitors the dynamic request arrival pattern and adjusts the number of replicas for each model. To cope with changes in query load, Cocktail designs a resource controller to manage CPU and GPU instances in a cost-optimized manner and a load balancer to allocate queries to appropriate instances [22]. It also proposes an autoscaler that leverages predictive models to predict future request loads and dynamically adjusts the number of instances in each model pool based on the importance weight of the models.

Recent advancements in serverless LLM systems have focused on optimizing GPU resource allocation, model loading efficiency, and inference migration to enhance LLM serving performance. SpotServe is a serverless LLM system that adjusts the GPU instances and updates the parallelism strategy flexibly [57]. When a new parallel configuration needs to be switched, SpotServe maps the available GPU instances to this logical device grid. It uses a bipartite graph matching algorithm to orchestrate model layers to hardware devices, thus maximizing the reusable model parameters and

key-value caches. ServerlessLLM is another serverless LLM system that introduces a loading-optimized checkpoint format and a multi-tier loading system to expedite model initialization [19]. It supports live migration of ongoing inferences, allowing the system to reassign tasks to different servers with minimal disruption. A locality-aware scheduler evaluates the status of each server in a cluster and effectively schedules model startup time to capitalize on local checkpoint placement.

Limitation of existing works. Despite the above efforts and benefits, model scaling presents significant challenges when applied to large Transformer models, primarily due to the constraints of GPU memory capacity. Unlike traditional deep learning models, which can often be dynamically loaded and switched based on demand, large-scale Transformer models contain billions of parameters, making it infeasible to store multiple model variants within the limited memory of a single GPU. This constraint forces frequent model swapping between disk and memory, leading to substantial I/O overhead and prolonged loading times. As a result, the latency introduced by model switching can become a major bottleneck, rendering these approaches impractical for real-time AI applications.

Additionally, while resource scheduling techniques optimize inference efficiency, they also introduce significant complexity. Coordinating hardware resources and dynamically adjusting parallel execution strategies is challenging, especially in large-scale distributed systems. Moreover, resource switching incurs overhead due to context migration and cold start delays, where transferring execution states across resources and initializing new instances can introduce latency overhead.

In this thesis, we explore an alternative approach for the elastic model serving through token adaptation. Unlike conventional model scaling strategies that rely on preloading multiple model variants, chapter 3 dynamically adjusts the number of processed tokens per query, enabling fine-grained control over inference latency and accuracy.

2.2 Preliminary for AI Models

AI models have evolved significantly over the past decades, driven by advancements in deep learning. Neural networks have become the foundation of modern AI, enabling breakthroughs in fields such as computer vision, natural language processing, and speech recognition. Among these, convolutional neural networks (CNNs) and Transformer models represent two dominant architectures for processing structured and unstructured data.

CNNs have been the cornerstone of numerous state-of-the-art solutions in AI tasks such as image classification, object detection, and semantic segmentation [43, 74, 28]. By leveraging convolution operations with local receptive fields and weight sharing, CNNs excel at extracting hierarchical features from images. These capabilities allow CNNs to learn increasingly complex representations in deeper layers, thus achieving high performance across various benchmarks. Nevertheless, CNNs sometimes require careful architectural designs (e.g., dilated convolutions, larger receptive fields, specialized pooling) to capture long-range dependencies effectively. As the need to handle broader contexts and large-scale datasets has grown, new model families, particularly the Transformer, have gained increasing popularity in both research and industry.

Transformer models, known for their strong performance across a range of vision and language tasks, have become the primary backbone for neural networks [18, 16, 66]. These models are distinguished by their capacity to capture long-range dependencies within sequential data and their scalability, reaching millions or billions of parameters [4, 15]. The current trend in AI has shifted towards using large-scale pre-trained Transformer models as foundational models [52, 25].

We demonstrate our approach using the Vision Transformer (ViT) model, as depicted in Figure 2.2 [18, 77]. In a ViT model, an image is divided into fixed-size patches, which are then converted into embeddings via a linear projection. These image embeddings are combined with positional embeddings to encode spatial infor-

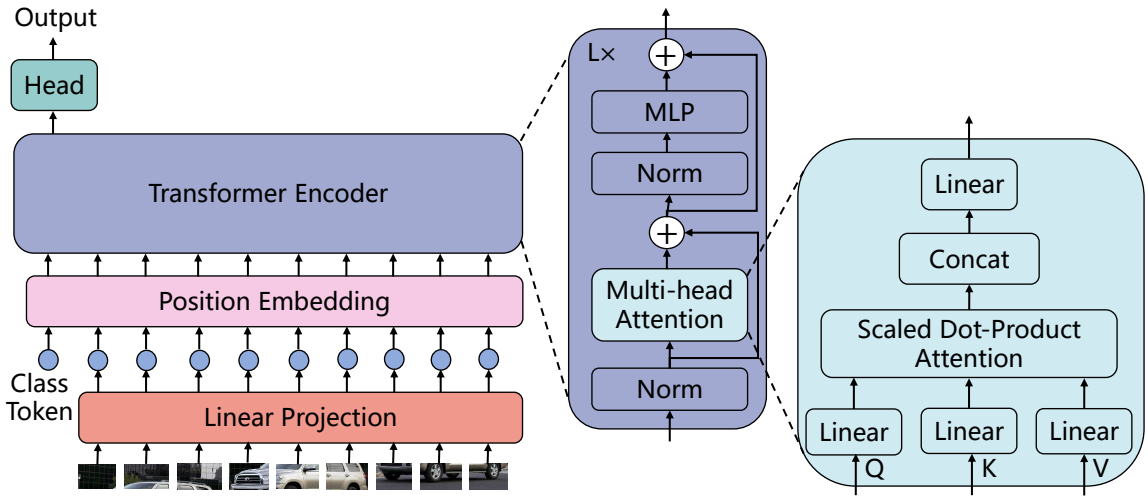


Figure 2.2: The vision Transformer model.

mation, forming a sequence of tokens that is passed to the Transformer encoder. The Transformer encoder consists of multiple stacked attention blocks, each comprising normalization, multi-head attention, and a multi-layer perceptron.

Multi-head attention enables the model to capture features from multiple representation subspaces, with each subspace referred to as an attention head i. In the attention mechanism, we use a query Q, a key K, and a value V, where $Q, K, V \in \mathbb{R}^{n \times d_{\text{model}}}$, with n as the token sequence length and d_{model} as the feature dimension. As shown in Equation (2.1), Q, K, and V are first projected into a lower-dimensional space using the projection matrices W_i^Q, W_i^K, and W_i^V. The attention mechanism then models interactions among tokens to capture semantic features. Specifically, in Equation (2.2), we compute the attention weight between the query Q_i and key K_i, which is then applied to V_i to generate a new representation.

$$\text{head}_i = \text{Attn}(QW_i^Q, KW_i^K, VW_i^V). \tag{2.1}$$

$$\text{Attn}(Q_i, K_i, V_i) = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{d_k}})V_i. \tag{2.2}$$

$Q_i, K_i \in \mathbb{R}^{n \times d_k}$, $V_i \in \mathbb{R}^{n \times d_v}$, and d_k, d_v are the feature dimensions. The outputs of the attention module are concatenated and forwarded to the following modules.

The Transformer model can be pre-trained on large-scale datasets to gain extensive knowledge across diverse domains. Rather than training models from scratch, researchers increasingly leverage large-scale pre-trained Transformers for a variety of tasks, a strategy that has proven effective for AI development [3]. However, the large size of these models results in high inference latency, making it difficult to handle sudden increases in query volume. Thus, designing an elastic computation method for Transformer serving is essential, especially in resource-limited settings.

2.3 Preliminary for Token Adaptation

In this subsection, we introduce two methods for adjusting the number of tokens.

Prompt Learning. Prompt learning has recently emerged as a powerful strategy for leveraging large-scale foundation models to perform a wide range of tasks. Traditionally, prompts were manually designed by engineers to elicit desired outputs from these models [4, 67]. This manual process often involves domain expertise and trial-and-error, as prompts must be carefully tailored to match the requirements of different tasks. However, manual prompt crafting can be time-consuming and may not easily generalize across tasks or datasets.

To address these challenges, researchers have introduced learnable soft prompts, which are tunable parameters optimized for specific datasets or tasks. Unlike manually crafted prompts, these soft prompts are trained with frozen model parameters, reducing the human effort involved in prompt design. For instance, CoOp designs learnable context vectors for vision-language models like CLIP, enabling efficient adaptation to various multi-modal tasks [98, 65]. VPT introduces an efficient method for adapting large-scale Vision Transformer models to downstream tasks. It integrates a small set of learnable parameters, known as visual prompts, into the input space while keeping the pre-trained model's backbone frozen [38].

These advancements in prompt learning demonstrate the potential of optimizing input representations rather than modifying the underlying model, making them particularly relevant for efficient adaptation in various AI applications. Similarly, in serving systems, increasing the token number can enhance accuracy by providing richer context, effectively leveraging the model's pre-trained knowledge. In chapter 3, we use prompting tokens to enhance the accuracy of a serving system.

Token Reduction. By analyzing the inference process, researchers find that the significant computational cost of Transformer models primarily comes at the self-attention mechanism [75]. Processing long sequences or large batches of data can become computationally intensive because the number of tokens increases dramatically. Token reduction, encompassing token pruning and token merging, is an advanced technique in the field of AI aimed at enhancing the efficiency and performance of large Transformer models [26]. The core idea behind token reduction is to shorten the input data that a model processes, thereby reducing computational overhead and potentially improving the model's ability to focus on the most important information. A key research topic in token reduction is to identify redundant or similar tokens in the input and remove or merge them without negatively affecting model performance.

A number of token reduction methods have been developed to accelerate the inference of the Transformer. *1) Token pruning.* There are only a limited number of words or image patches that contribute to the prediction of final results, and a lot of redundant tokens can be regarded as noisy information. Therefore, token pruning selectively removes tokens from the input sequence during the inference process of a Transformer model [94, 68, 47]. The core idea is to identify and retain only the most informative tokens for the task, thereby reducing the sequence length and the computational load. There are two common approaches to indicate the significance of tokens that can guide the pruning process. The first method is *training a prediction module*, which takes tokens as input and outputs the importance score for each token. This module is constructed with a two-layer linear network and trained with soft masking

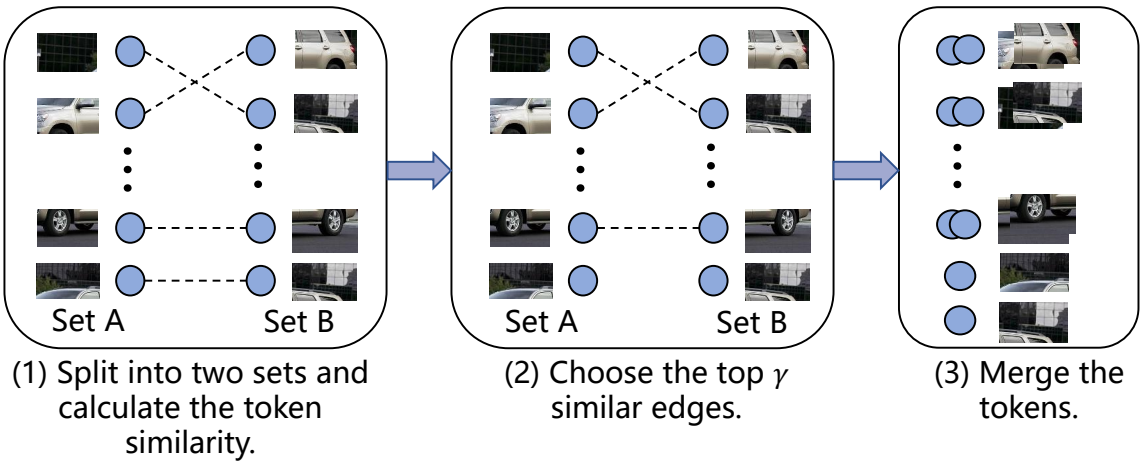


Figure 2.3: The token merging mechanism [2]. We first calculate the token similarity based on the attention keys, keep the top r token pairs and merge them.

of tokens. During inference, the k tokens with the lowest importance scores will be removed to reduce the token number. The second method is directly *collecting the attention weight* to prioritize tokens. The attention mechanism calculates the interdependence among tokens, and the attention weight reflects the relationship between them. Tokens with lower attention values contribute less to the output of other tokens, therefore indicating their smaller significance. Therefore, the attention weight serves as a valuable metric for evaluating the token importance.

2) Token Merging. Token merging, on the other hand, combines adjacent or similar tokens into single tokens, effectively reducing the sequence length without a substantial loss of information [93, 50, 96]. This technique is particularly useful for large Transformer models, where processing extensive sequences of tokens can be computationally intensive. The motivation behind token merging is the presence of numerous similar patches in images and videos, which exhibit similar functionalities within the Transformer model. One of the most well-known methods is TOME, which demonstrates the effectiveness of token merging [2]. As shown in Figure 2.3, the input tokens are initially divided into two sets, with tokens in set A selecting the most similar token in set B using cosine similarity of features. The top k edges, representing the highest

similarity of tokens, are retained. Finally, these tokens are merged using a weighted average approach to reduce the length of the input.

These token reduction techniques leads to a significant decrease in computational overhead, enabling faster inference while maintaining model accuracy. Moreover, by strategically merging tokens with minimal information loss, token merging is particularly suitable for real-time and resource-constrained applications. In chapter 3, we leverage this technique to optimize the serving system's throughput, demonstrating its practical benefits in latency-sensitive scenarios.

Chapter 3

OTAS: An Elastic Transformer Serving System via Token Adaptation

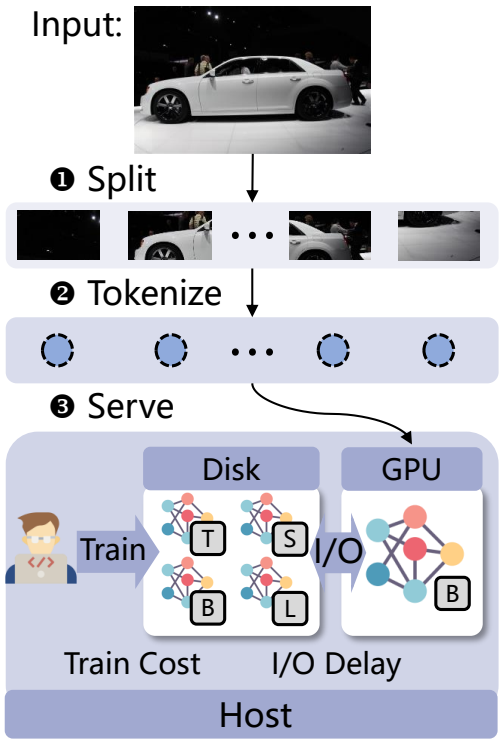
Transformer model empowered architectures have become a pillar of cloud services that keeps reshaping our society. However, the dynamic query loads and heterogeneous user requirements severely challenge current Transformer serving systems, which rely on pre-training multiple variants of a foundation model, i.e., with different sizes, to accommodate varying service demands. Unfortunately, such a mechanism is unsuitable for large Transformer models due to the additional training costs and excessive I/O delay. In this chapter, we introduce OTAS, the first elastic serving system specially tailored for Transformer models by exploring lightweight token management. We develop a novel idea called *token adaptation* that adds prompting tokens to improve accuracy and removes redundant tokens to accelerate inference. To cope with fluctuating query loads and diverse user requests, we enhance OTAS with application-aware selective batching and online token adaptation. OTAS first batches incoming queries with similar service-level objectives to improve the ingress

throughput. Then, to strike a tradeoff between the overhead of token increment and the potentials for accuracy improvement, OTAS adaptively adjusts the token execution strategy by solving an optimization problem. We implement and evaluate a prototype of OTAS with multiple datasets, which show that OTAS improves the system utility by at least 18.2%.

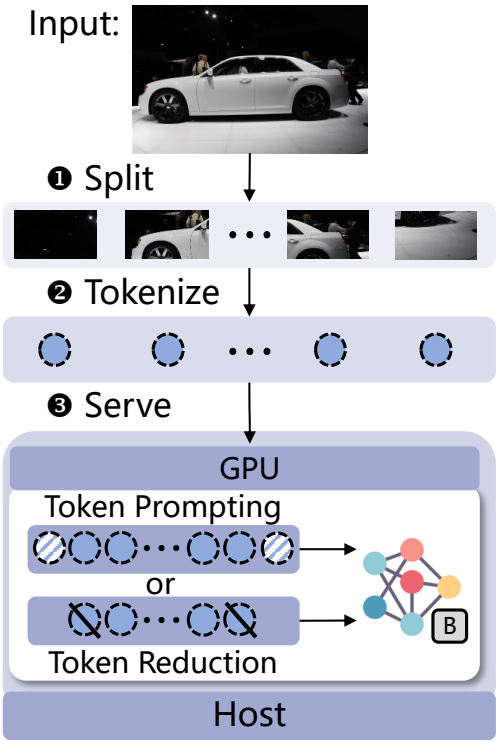
3.1 Introduction

It is of vital importance for the cloud to effectively serve the machine learning models for AI applications, which can substantially affect the quality of user experience and the accompanied economic profits. For example, Facebook has 1.82 billion daily active users and issues tens of trillions of model inference queries per day, which necessitates fundamental re-designs for facilitating the model optimization and the serving efficiency [32].

Recent advances in self-supervised pre-training techniques have boosted the development of large Transformer models while imposing a substantial burden on the model serving. These pre-trained Transformer models have dramatically revolutionized our lives and brought remarkable potential to our society. For example, large pre-trained models like GPT-3 have spawned a host of applications, such as Copilot [21] and ChatGPT [62]. In particular, ChatGPT has more than 100 million active users and received 176 million visits in April 2023 [7]. Despite the explosion of such diverse applications, the resource-intensive nature of Transformer models, coupled with dynamic query loads and heterogeneous user requirements have exacerbated the challenges associated with Transformer serving, making it extremely challenging to accommodate various service demands. In this context, the implementation of an elastic serving system that can adapt the serving process for improving service quality emerges as a promising solution.



(a) *Model Adaptation.* The service provider prepares multiple versions of Transformers, e.g., tiny (T), small (S), base (B) and large (L) models, and switches them during runtime.



(b) *Token Adaptation.* We realize elastic Transformer serving through lightweight token adaptation that adds or reduces execution tokens according to service characteristics.

Figure 3.1: Comparison between model adaptation and token adaption.

As shown in Figure 3.1(a), a common approach to realize an elastic serving platform, called *model adaptation*, is to pre-train multiple model variants and dynamically select one to accommodate the variations in query load [85, 79, 23, 70]. Unfortunately, it is unsuitable for large Transformer models because loading such kind of model to GPU may creates prohibitive I/O overhead [72]. This scheme also leads to additional training costs, such as high monetary costs and time overhead. Moreover, the size of different Transformer models varies significantly, and it is hard to prepare different fine-grained model versions.

In this chapter, we delve into the inherent characteristic of Transformer models and develop a novel idea called *token adaptation* for elastic model serving. As shown in

Figure 3.1(b), a token is a basic unit of text, code, or a patch of an image [58]. The Transformer models process these tokens with attention mechanism [77], which calculates the similarity (i.e., attention weight) between the query and key, and projects the attention value with the similarity weight to get a new representation. One of the key properties of attention is its ability to support token sequences of varying lengths. A line of works have shown that input tokens have a large influence on the model performance, such as the accuracy and running time. For example, identifying and removing redundant or unnecessary tokens, such as those representing the background, can expedite inference with little accuracy drop [2]. On the other side, recent studies also demonstrate that adding prompting tokens that contain specific semantic features of the object and interact with input tokens can help produce more accurate results [38]. Motivated by the above findings, we seek to explore a novel design space about token adaptation that dynamically adjusts the execution tokens of the Transformer model to improve service quality with negligible training and I/O costs, i.e., serving more important queries.

Despite the promising potential, designing token adaptation for a serving system is non-trivial and faces the following challenges: (1) *Diverse user requests.* The queries in a batch can vary significantly in query content, task, utility reward, and latency requirements. The allocation of token numbers needs to balance the demands of different requests. (2) *Fluctuating query load.* In real scenarios, the query load is fluctuating and bursty. The system should accommodate different numbers of requests with a short reaction time. (3) *Model design.* None of the existing Transformer models can support fine-grained token management. A unified Transformer model needs to process prompt parameters from various tasks.

Therefore, in this chapter, we design an elastic Transformer serving **S**ystem via **O**nline **T**oken **A**daptation, named OTAS. We first introduce the modules of the system and present the pipeline to process the incoming queries. Then, we propose a unified Transformer model that can flexibly adjust the execution schemes with token

prompting and token reduction. Moreover, we present an adaptive batching algorithm to group queries with similar service-level objectives, such as latency requirement and utility value, to improve throughput. To cope with fluctuating query loads and diverse user requirements, we formulate a utility maximization problem with user latency constraints to adaptively adjust the token execution strategy. Then, we design an efficient dynamic programming algorithm to derive the token execution plan for a batch.

We summarize the contributions as follows.

- We explore a novel design space for Transformer serving, called token adaptation, allowing for flexible manipulation of the token execution plan for Transformers. It reveals a new trade-off space between the potential for improving accuracy and the cost of inference latency.

- We present OTAS, an elastic Transformer serving system via an online token allocation algorithm. We design a batching algorithm that groups similar queries for execution to improve throughput. Besides, we adaptively allocate the token number for a batch to cope with the dynamic query load and diverse user requirements.

- We implement a prototype system of OTAS. The experimental results show that OTAS improves the serving utility by at least 18.2% and serves more requests.

This chapter is organized as follows. Section 3.2 provides the motivation of this chapter. In section 3.3, we introduce the system design, model design and our methodology, including *Adaptive Batching* and *Online Token Adaptation*. Section 3.4 further refines our designs, incorporating *Iterative Prompt Learning* and *Layer-wise Token Merging*. Sections 3.5 and 3.6 show the implementation and experimental results. We conclude this chapter with section 3.7.

3.2 Motivation

As discussed in section 2.2, Transformer models, known for their strong performance across a range of vision and language tasks, have become the primary backbone for neural networks [18, 66]. These models are distinguished by their capacity to capture long-range dependencies within sequential data and their scalability, reaching millions or billions of parameters [4, 15]. The current trend in AI has shifted towards using large-scale pre-trained Transformer models as foundation models [52, 25]. However, the large size of these models results in high inference latency, making it difficult to handle sudden increases in query volume. Thus, designing an elastic computation method for Transformer serving is essential, especially in resource-limited settings.

A common-used method for elastic serving is *model adaptation* that pre-trains multiple versions of models and dynamically loads an appropriate one during runtime. However, it is infeasible for Transformers due to the exorbitant training costs and large I/O delay. In this chapter, we explore a novel design for elastic Transformer serving by utilizing the inherent characteristic of attention: its ability to support token sequences of varying token lengths. Specifically, we propose *token adaptation* that improves accuracy by token prompting and accelerates inference by token reduction. In this section, we will delve into the attention mechanism and illustrate how to enlarge and compress the attention space for elastic serving in section 3.2.2 and section 3.2.3.

3.2.1 Delving into the Attention Mechanism

In this section, we delve into the attention mechanism from the perspective of token interactions, aiming to inspire a new design for elastic Transformer serving. The basic calculation unit of the Transformer is a token. As shown in the left part of Figure 3.2, we view the attention mechanism as a form of token retrieval and similarity-aware

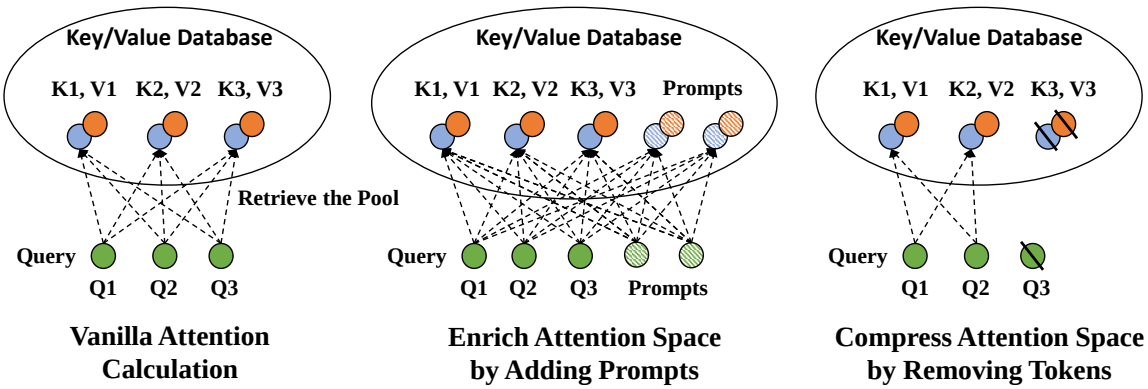
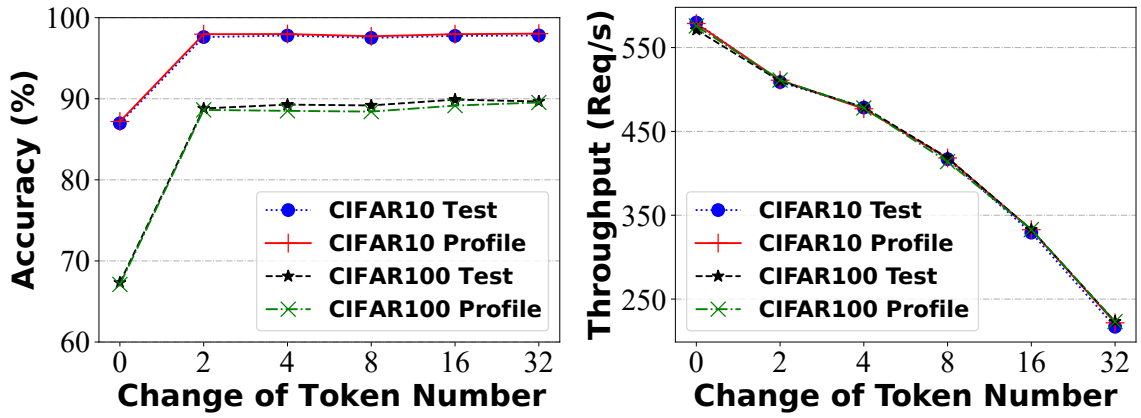


Figure 3.2: The attention mechanism, which can be regarded as the token retrieval and similarity-aware token combination. A query vector is compared against all key vectors to compute a set of similarity scores. These scores, normalized into a probability distribution, serve as weights to compute a weighted sum of the corresponding value vectors, producing a context-aware output. Therefore, we can enrich the attention interaction process by adding or removing tokens.

token combination, where a key/value database is involved. The calculation of attention weight (i.e., $\text{softmax}(\frac{Q_i K_i^T}{\sqrt{d_k}})$) serves as a form of token retrieval, where a token in the query is matched with key tokens in the database based on cosine similarity values. After the attention weight is calculated, it is applied to the value tokens in the database to obtain a new representation for the query. An item in the database is a pair of key/value tokens. Based on the above analysis, an interesting question is raised: *Can we enrich or compress the queries and database space to improve the elasticity of a Transformer model while maintaining the model weight fixed?*

Fortunately, the answer to this question is yes, thanks to a key characteristic of Transformers: their ability to support variable token length. In Figure 3.2, we introduce a novel concept called *token adaptation* that employs fine-grained token management in the Transformer model to adjust the token number for inference. Specifically, we incorporate prompt tokens into the input, enriching both the query and the database with external information, resulting in enhanced performance. Besides, we remove some useless tokens to expedite the attention calculation, thus reducing the inference latency. This flexible adaptation scheme for Transformer serving represents a



(a) Accuracy Comparison. (b) Throughput Comparison.

Figure 3.3: Accuracy and throughput comparison with different token numbers.

new trade-off space between the opportunity for improving accuracy and the cost of longer inference latency. We will delve into the characteristics of token prompting and token reduction and provide a detailed explanation of how to adjust tokens during online serving.

3.2.2 Enlarge the Attention Space by Token Prompting

An effective way to expand the attention space is by incorporating pre-trained tokens through prompt learning, a parameter-efficient fine-tuning technique for pre-trained Transformers [17]. Prompts are prepended to the input tokens, and the combined token sequence is processed by the Transformer layer to perform multi-head attention. These prompts consist of one-dimensional vectors with gradients, which help guide the foundational model toward generating more accurate results. The calculation process is shown in Eq. (3.1), where the token lengths in attention operations are extended by adding prompt P at each layer. Several studies have explored the use of learnable soft prompts that better align with the data distribution of specific local datasets [52, 31, 38]. These prompts are initialized randomly and trained via stochastic gradient descent. During inference, these well-trained prompts can be directly prepended to

the input. Prompt learning is a post-training approach, as it leaves the original model parameters unchanged, and tokens can be seamlessly added or removed in real time.

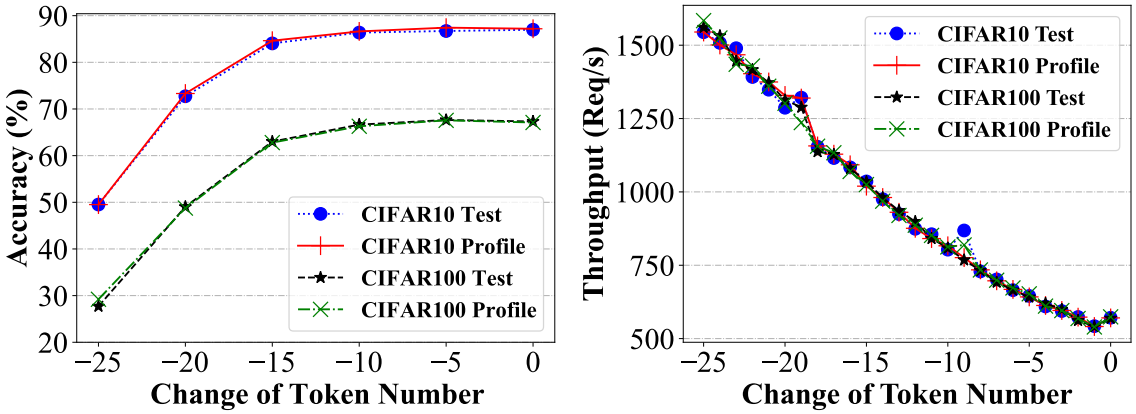
$$\text{head}_i = \text{Attn}\Big(\text{concat}(P,Q)W_i^Q, \text{concat}(P,K)W_i^K, \text{concat}(P,V)W_i^V\Big). \tag{3.1}$$

Before applying prompts for token adaptation, it is crucial to examine the characteristics of prompt learning, particularly regarding accuracy and throughput. To this end, we trained prompts on a pre-trained ViT model using the CIFAR10 and CIFAR100 datasets [42], setting the number of prompts to 2, 4, 8, 16, 32. We used a subset consisting of 1/5 of the training data as a profiling set and assessed performance on both the profiling and test sets. As shown in Figure 3.3(a), accuracy rises sharply with the addition of two prompts per layer, then stabilizes with minimal improvement as the number of prompts increases further. Prompt learning demonstrates more substantial benefits for complex tasks, such as CIFAR100. In Figure 3.3(b), we observe the throughput results on an NVIDIA GeForce RTX 4080. Increasing the number of prompts leads to a decline in throughput, dropping from 580 requests per second to 220 requests per second, alongside an increase in inference latency per sample.

3.2.3 Compress the Attention Space by Token Reduction

An effective approach to compressing the attention space involves reducing redundant or unnecessary tokens through token reduction methods. This technique accelerates inference in Transformer models by shortening sequence length with minimal impact on accuracy. Token merging is among the most widely used methods [2, 80]. In cases where redundant semantic information appears across different parts of sentences or images, similar tokens can be combined into a single token that encapsulates the shared information, which is then utilized in subsequent layers.

ToMe is a leading token merging method [2], which incorporates a token merging



(a) Accuracy Comparison. (b) Throughput Comparison.

Figure 3.4: Accuracy and throughput comparison when we remove different numbers of tokens.

module after the multi-head attention layer. This module utilizes key similarity to assess token distances. Initially, tokens are divided into two sets, where tokens in set A identify their closest match in set B based on similarity. Given that γ tokens are merged per layer, the next step retains only the γ edges with the highest similarity scores. Finally, these similar tokens are combined through a weighted average and then reassembled into a new sequence.

We first discover the effectiveness of token merging using ToMe [2]. The merging parameter is adjusted from -25 to 0. As illustrated in Figure 3.4(a), reducing the token numbers initially causes a modest decline in accuracy. However, when the token number drops below -15, accuracy decreases sharply to 50% and 28% for CIFAR10 and CIFAR100, respectively. Figure 3.4(b) demonstrates the impact on throughput, showing a gradual reduction from 1500 Req/s to 500 Req/s. This suggests that token merging can substantially enhance throughput and decrease inference latency. Furthermore, the merging algorithm adds an overhead of under 20ms, making it highly efficient and suitable for practical deployment.

While token prompting and token merging offer distinct advantages, selecting an appropriate prompting number or merging ratio remains challenging due to varying

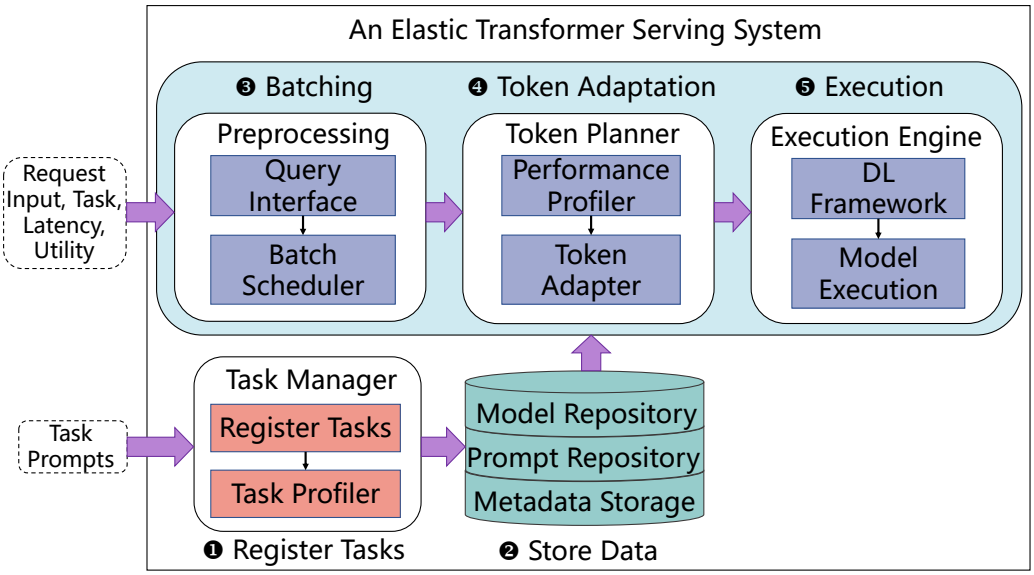


Figure 3.5: The framework of OTAS, which can assign a query to a batch and allocate the token number automatically.

request inputs, query loads, and computational resources. To maximize the benefits of token adaptation, it is essential to identify the optimal token number for the Transformer during the serving process. This approach enables a balance between accuracy and latency, adapted to the specific request load, task requirements, and available hardware resources.

3.3 OTAS: Online Token Adaptation System

Based on the above analysis, we propose OTAS, a unified framework that achieves online token adaptation for Transformer serving. Our framework can autonomously change the token number during the serving period, which allows for efficient and adaptive serving of Transformer models.

3.3.1 System Design

We first introduce a new serving framework for elastic Transformer inference named OTAS. The components of OTAS are shown in Figure 3.5. The framework consists of two main workflows: task register and query processing.

To register a new task, the developer should submit the prompts with the required token numbers, and the prompt parameters are stored in the repository. The task profiler calculates the accuracies and inference latencies for different token numbers and batch sizes on the target device. The profiling data is stored in the metadata storage for future use.

The system is designed to handle incoming queries with varying arrival times, inputs, tasks, utilities, and latency requirements. When a query is received, it is added to a batch using a batch scheduler. The batching strategy is described in section 3.3.3. The resulting batch may contain queries from different tasks with varying utilities and latency requirements. The batch is then stored in a batch queue and awaits execution. The performance profiler is used to predict the accuracy and inference time for different token number settings. The token adapter module uses the profiling data to allocate token numbers for the batches, which is illustrated in section 3.3.4. Finally, the execution engine is responsible for sequentially executing the batches with a Transformer model.

3.3.2 Model Design

To support flexible Transformer inference, we propose a unified model that supports token prompting and reduction. As shown in Figure 3.6, the prompting module is added before the normalization, and the merging module is added before the MLP. We define the token number change per layer as γ, where $\gamma > 0$ means adding the prompting tokens and $\gamma < 0$ means removing some useless tokens.

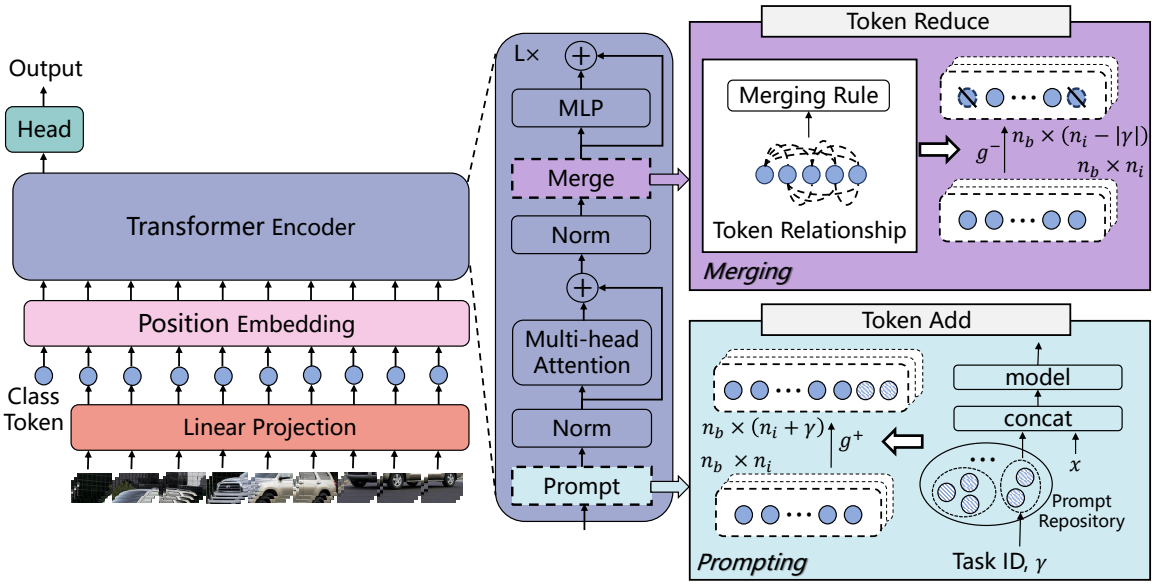
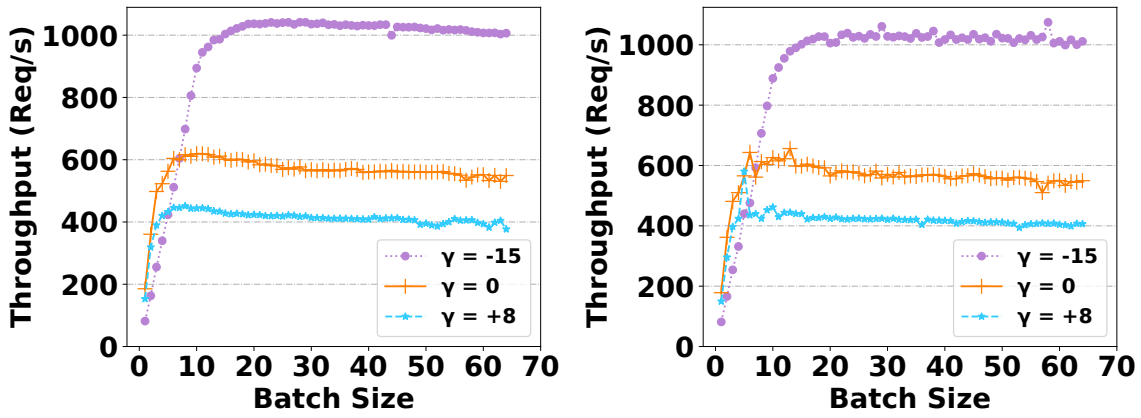


Figure 3.6: A unified Transformer model that incorporates token prompting and token merging.

The prompting tokens are trained offline and stored in the prompt repository. A token pair is associated with a task and a prompt number, which serves as its index. We first initialize the prompt repository randomly and train each token pair separately. We acquire a token pair from the prompt repository at every training epoch and concatenate them with the input tokens. If the batch size is n_b and the input token length is n_i, the token shape becomes $n_b \times (n_i + \gamma)$ after prompting. The concatenated tokens can be forwarded to the next module. During inference, the model uses the well-trained prompt parameters in the repository directly. The added prompting tokens can inspire the multi-head attention to generate a better result. Regarding token merging, the model directly processes the input tokens. Given the token similarity obtained from multi-head attention and a merging rule, the merging module can reduce the token shape from $n_b \times n_i$ to $n_b \times (n_i - |\gamma|)$.

We insert these two modules at each layer. To simplify the design, we assume the model can perform either token prompting or token reduction during inference. Different tasks also have specific head parameters, and the model forwards the sample



(a) Throughput comparison on CIFAR10. (b) Throughput comparison on CIFAR100.

Figure 3.7: Throughput comparison of different batch sizes.

to its corresponding head to obtain an appropriate prediction probability.

3.3.3 Adaptive Batching

Batching queries for inference can improve the system's throughput by making full use of the computational capabilities of the device and reducing the costs associated with model initialization and data communication [13, 14]. Figure 3.7 illustrates the advantages of batching, where we evaluate the throughput with batch sizes ranging from 1 to 64. The throughput shows a rapid increase as the batch size is increased. For example, when $\gamma = -15$, the throughput increases from 100 Req/s to 1000 Req/s and converges when the batch size is 20. Therefore, batching can significantly improve throughput and enable more efficient query processing.

While batching has clear benefits, one challenge is to design a batching strategy that effectively groups similar requests together. To achieve this, we batch incoming queries based on their similar arrival patterns and service-level objectives, such as latency constraint and utility. We use the notation r to represent a request, where s_r, l_r, d_r, and u_r represent the request's arrival time, latency requirement, finish deadline, and utility, respectively, such that $d_r = s_r + l_r$.

Algorithm 1: Batching Algorithm: Adding a Query into a Batch.

Input: Batch Queue B, Query r;
Output: Batch Queue B;

```
for b ∈ [N_B, 1] do
    if s_b + δ < s_r then                          // Arrival time;
        break;
    else if |B_b| >= ϵ then                        // Batch size;
        continue;
    else if ||d_b − d_r|| > η then                 // Finish time;
        continue;
    else if ||u_b − u_r|| > μ then                 // Utility;
        continue;
    B_b.add(r);                                    // Find a batch b;
    return B;
B.add({r});                                        // Create a new batch;
return B;
```

The grouped queries are stored in the batch queue B. We denote the b-th batch as B_b. The arrival time of a batch b is defined as the earliest arrival time among its requests, i.e., $s_b = \min\{s_r\}, r \in B_b$. Similarly, the finish deadline of a batch b is defined as the earliest required finish time among its requests, i.e., $d_b = \min\{d_r\}, r \in B_b$.

The batching algorithm is described in Algorithm 1, which assigns a query to the current batches or initializes a new batch. The key idea of the algorithm is constructing a batch with constraints on batch size, arrival time, utility and deadline. Specifically, the algorithm ensures that the waiting time of the first request in a batch is less than δ, the batch size is smaller than a pre-defined threshold ϵ, and the deadline difference between the batch and the query r is not larger than a threshold η. We use u_b to represent the utility of the first arrival query in a batch b, and restrict the utility value for subsequent incoming query r to be close to the value of u_b with a threshold μ. These constraints ensure that queries with similar arrival patterns and service-level objectives can be processed together, which is beneficial for token adaptation. If a batch that meets the constraints for the incoming query is found, the query is added

to that batch b (Line 1∼9). Otherwise, a new batch is created for the query and added to the batch queue.

3.3.4 Online Token Adaptation

After constructing the batch queue, the next step is to assign token adaptation schemes for batches. In this section, we present an optimization problem for token adaptation and propose a dynamic programming algorithm to obtain the solution.

Problem Formulation. We define the token adaptation scheme for a batch b as γ_b, where $\gamma_b < 0$ indicates reducing the token number, $\gamma_b > 0$ indicates adding some prompting tokens, and $\gamma_b = 0$ indicates making the inference with the vanilla Transformer model. γ is a discrete value that can be selected from a pre-defined list. If the serving system successfully provides an accurate result for request r under the latency requirement, the system can be rewarded with utility u_r, such as the money. We use $\alpha_r \in \{0, 1\}$ to represent whether it successfully serves query r. The required memory of batch b and the available GPU memory are denoted as M_b and M_{GPU}.

The optimization problem is defined in Eq. (3.2), where the goal is to allocate the token change number γ to maximize the overall utility for all requests. Constraint (3.2a) ensures that all requests can be completed within their respective deadlines, where $t_r^{(q)}$ and $t_r^{(p)}$ are the queuing time and processing time. Constraint (3.2b) ensures that the batches are executed sequentially. Constraint (3.2c) imposes a memory restriction, as larger batch sizes and prompt numbers may increase the memory demand.

$$\max_{\gamma_b} \sum_{b\in[1,N_B]} \sum_{r\in B_b} u_r \cdot \alpha_r; \tag{3.2}$$

$$s.t. \quad s_r + t_r^{(q)} + t_r^{(p)} < d_r, \forall r \in B_b; \tag{3.2a}$$

$$s_r + t_r^{(q)} + t_r^{(p)} < s_{r'} + t_{r'}^{(q)}, \forall r \in B_b \text{ and } \forall r' \in B_{b+1}; \tag{3.2b}$$

$$M_b < M_{\text{GPU}}. \tag{3.2c}$$

The above problem formulation considers both the query load and request characteristics. If the batch queue has a high volume of queries, we should pick a smaller γ to reduce the queuing and processing time and serve more requests. Conversely, we can increase the value of γ to derive an accurate result and earn more utilities. Then, we analyze the NP-hard property of problem (3.2).

Theorem 1. *The problem* (3.2) *is an NP-hard problem.*

Proof. The token adaptation problem is an NP-hard problem because it can be reduced from another NP-hard problem–Weighted Interval Scheduling Problem (WISP) [40]. Given a set of intervals with a weight, the objective of WISP is to select some intervals that can maximize the sum of the weights while the selected intervals are pairwise disjoint. We can transform our problem to the WISP. We consider each batch as an interval with a weight equal to its utility. Our goal is to efficiently process the batches so that the sum of utilities is maximized. Our problem is more difficult than WISP because we also need to adjust the running time for the picked intervals with different γ values. □

Algorithm Design. Due to the NP-hardness of the above problem, we propose an efficient dynamic programming algorithm to derive the solution in Algorithm 2, which takes the batch queue B, current time T, the available γ list and the estimated arriving rate q as inputs and outputs the updated batch queue with allocated token number γ. The key idea is to find the largest utility value for a batch b with a γ_b through iterative traversal.

We begin by sorting the batches according to their required deadlines. If the size of the batch queue is less than a threshold β or the serving system is in the initial stage, we allocate the token number based on the query load with Algorithm 3. This is because the dynamic programming algorithm works well when there are sufficient batches to make a long-term schedule. Algorithm 3 allocates the token number γ by comparing the incoming request rate q and the throughput of different γ values.

Algorithm 2: Autonomous Token Adaptation Algorithm

Input: Batch Queue B, Clock Time T, Selection List $L^{(\gamma)}$, Request Rate q;

Output: Batch Queue B;

Sort(B) according to d_b;
if $N_B \leq \beta$ *or initial_stage=True* **then**
    B = Manually_Allocate($B, T, L^{(\gamma)}, q$) ;
    return B;
Initialize $dp \in \mathbb{R}^{(N_B+1)\times(N_\gamma+1)}$ by 0, $S \in \mathbb{R}^{(N_B+1)\times(N_\gamma+1)}$ by 1, $C \in \mathbb{R}^{(N_B+1)\times(N_\gamma+1)}$ by T, $J \in \mathbb{R}^{(N_B+1)\times(N_\gamma+1)}$ by 0;
for $b \in [1, N_B]$ **do**
    for $l_b \in [0, N_\gamma]$ **do**
        for $l_{b-1} \in [0, N_\gamma]$ **do**
            if $dp[b-1, l_{b-1}] == -\infty$ **then**
                continue;
            if $l_b == 0$ **then**
                if $dp[b-1, l_{b-1}] > dp[b, l_b]$ **then**
                    $dp[b, l_b] = dp[b-1, l_{b-1}]$; $S[b, l_b] = l_{b-1}$; $C[b, l_b] = C[b-1, l_{b-1}]$; $J[b, l_b] = 1$;
            else
                $\gamma_b = L^{(\gamma)}[l_b]$;
                $\hat{t}_r^{(p)}, \hat{U}_b$ = Profile(B_b, γ_b);
                if $C[b-1, l_{b-1}] + \hat{t}_b^{(p)} < d_b$ **then**
                    $u = dp[b-1, l_{b-1}] + \hat{U}_b$; $J[b, l_b] = 1$;
                    if $u > dp[b, l_b]$ **then**
                        $dp[b, l_b] = u$; $S[b, l_b] = l_{b-1}$; $C[b, l_b] = C[b-1, l_{b-1}] + \hat{t}_b^{(p)}$;
        if $l_b > 0$ *and* $J[b, l_b] == 0$ **then**
            $dp[b, l_b] = -\infty$; $C[b, l_b] = +\infty$;
$l = \arg\max dp[N_B]$; $B_{N_B}.\gamma = L^{(\gamma)}[l]$;
for $b = N_B - 1$ *to 1* **do**
    $l = S[b+1, l]$; $B_b.\gamma = L^{(\gamma)}[l]$;
return B;

Algorithm 3: Manually_Allocate: Allocate γ According To The Arriving Rate.

Input: Batch Queue B, Clock Time T, Selection List $L^{(\gamma)}$, Incoming Request Rate q;
Output: Batch Queue B;

```
1  γ = f(q);
2  for b ∈ [1, N_B] do
3      t̂_r^(p) = Profile(B_b, γ);
4      if T + t̂_r^(p) ≥ d_b then
5          B_b.γ ← min(L^(γ));
6      else if Ū_b > κ then
7          B_b.γ ← max(L^(γ));
8      else
9          B_b.γ ← γ;
10     t̂_r^(p) = Profile(B_b, B_b.γ);
11     T = T + t̂_r^(p);
12 return B;
```

We calculate the arriving rate q within the previous inference window and apply a function f to map q to a suitable value of γ (Line 1). f can be profiled offline according to the throughput of different γ values. Then, we adjust the selection of γ according to the query characteristics. We predict the execution time for a batch b. If the estimated completion time exceeds the deadline, we set the token number as the minimum value to meet the latency constraints (Line 3∼5). If the average utility $\overline{U_b}$ is larger than a threshold κ, we set the token number as the maximum value to prioritize the critical queries. Finally, we estimate the execution time and update current time T.

Algorithm 2 utilizes four auxiliary arrays of size $(N_B + 1) \times (N_\gamma + 1)$ to implement dynamic programming, where N_B and N_γ are the sizes of batch queue and the number of available γ values. Specifically, *dp* records the accumulated utilities, S records the previous γ selection scheme, and C records the clock time after executing batch b with γ. The array J indicates whether executing b with γ satisfies the deadline requirement. For each batch in the batch queue, we iteratively assign a value of γ

from the list $L^{(\gamma)}$ to batch b using the index l_b (Line 9∼11). If batch $b-1$ cannot be executed with γ indexed with l_{b-1}, we continue to the next iteration of the loop (Line 12∼13). When the value of l_b is 0, it indicates that batch b is not executed, and we directly find a larger utility value from batch $b-1$ and assign it to batch b (Line 14∼19).

When executing γ_b for batch b, we first estimate inference time and utility through profiling (Line 22). If the completion time is smaller than the required deadline, we calculate the overall utility and set the execution plan as 1 (Line 23∼25). If the utility is larger than the previous values, we update the matrixes. If there is no feasible execution plan for batch b with γ_b, we set the dp value as $-\infty$ and the clock time as $+\infty$ (Line 30∼32).

Once we have calculated the utility values and their corresponding choices, we can derive the solution by backtracking. We first determine the value of γ for the N_B-th batch based on the highest dp value. For each batch, we obtain the index of γ according to the value of $S[b+1, \gamma]$. Finally, we return the updated batch queue B.

We estimate the execution time and utility for a batch b with the profiling data. We profile the accuracy and sample-level inference latency for all tasks and store them in the metadata storage. To estimate the inference time for the current batch, we first count the number of samples for each task, and then multiply the sample number by the corresponding profiling inference time to obtain the execution time for that task. We then sum up the calculation results for all tasks to obtain the predicted inference time of a batch. To calculate the overall utility, we compute the product of the accuracy with a selected γ and the utility of each query in the batch. Then, we sum up the product result of all queries to obtain the total utility of a batch. During profiling, we ensure that all the running processes adhere to the memory constraints of Eq. (3.2c).

3.4 Design Refinement

In this section, we enhance OTAS with two additional designs, including iterative prompt learning and layer-wise token merging.

3.4.1 Iterative Prompt Learning

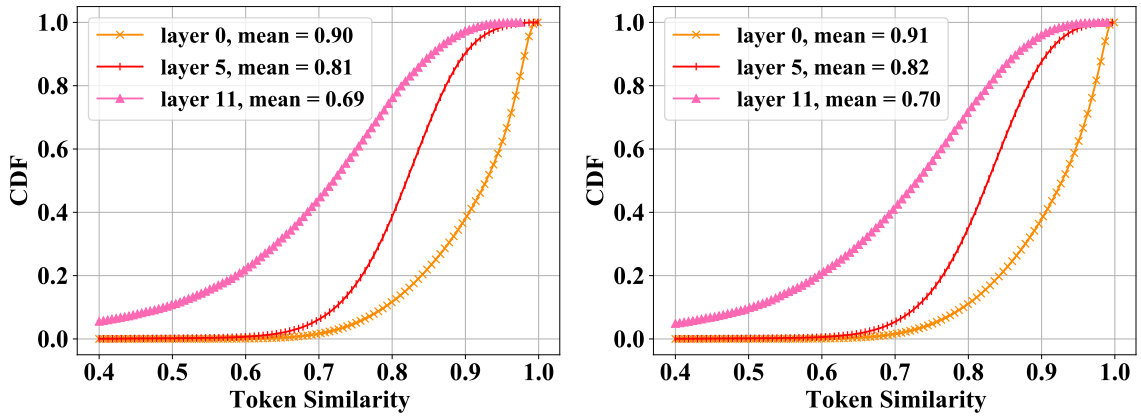
According to Figure 3.3(a), there is a significant increase in accuracy as the value of γ increases from 0 to 2; however, beyond this point, even with a large increase in the value of γ, the impact on accuracy remains minimal. For instance, in the CIFAR100 dataset, when the value of γ exceeds 2, the accuracy stabilizes at approximately 90%, suggesting significant potential for improving the prompt learning algorithm. The primary limitation of the current approach lies in the restricted learning capacity of each prompt, which hinders further accuracy enhancements.

We propose an iterative training strategy to fully optimize the prompts. The core idea is to gradually increase the number of prompting tokens during training, allowing each token to be trained independently. This approach ensures that each token reaches its full potential for performance improvement. As the iterations progress, a small set of tokens is trained first, with each iteration generating new tokens based on the previous ones. For instance, training begins with two prompting tokens on a downstream dataset; the parameters learned from this step are then used to initialize the first two tokens when training four tokens. The subsequent training phase focuses on iteratively optimizing the remaining two tokens. By leveraging the parameters learned in earlier iterations, each stage builds on the knowledge of its predecessors, ensuring a more targeted and effective optimization of the prompting tokens.

The result is presented in Table 3.1. We apply iterative prompt learning with four tokens on three datasets—CIFAR10, CIFAR100, and Eurosat. As demonstrated, iterative training enhances the effectiveness of token prompting and is beneficial for our

Dataset \ Scheme	Train from scratch	Iterative training
CIFAR10	97.79%	98.08%
CIFAR100	89.28%	89.61%
EuroSAT	97.10%	97.83%

Table 3.1: Accuracy comparison of different prompt learning schemes when prompt number is 4.



(a) CIFAR10 dataset. (b) CIFAR100 dataset.

Figure 3.8: The token similarities at different layers.

serving system. However, the performance improvements do not scale proportionally with the increase in token numbers. This result indicates that the performance is approaching the limitations of the prompt tokens and the current model architecture.

3.4.2 Layer-wise Token Merging

According to Figure 3.4(a), the accuracy of token merging has a noticeable drop when the value of γ is smaller than -15. A high merging ratio has a significantly negative influence on accuracy. It is because many semantic tokens at the deeper layer are removed. To enhance performance, we propose a layer-wise merging scheme that differentiates the number of merged tokens across different layers.

The performance of token merging is influenced by the similarity of tokens, so we begin by analyzing token similarity at different Transformer layers. We set the merging number to -20 and calculate the cosine similarities of tokens at layers 0, 5, and 11 [2]. The Cumulative Distribution Function (CDF) plots of these cosine similarities are presented in Figure 3.8. The CDF plot illustrates the percentage of data points that fall below a certain value along the x-axis [82]. As shown in Figure 3.8, token similarity decreases progressively with the depth of the network. At layer 0, the average similarity is approximately 0.90, but by layer 11, this value drops to around 0.70. This trend suggests that there are substantial redundancies in the shallow layers. As the Transformer model processes the input features, it generates high-level semantic features at deeper layers, leading to greater dissimilarity between tokens. Consequently, it is important to preserve the semantic tokens at deeper layers.

Motivated by this observation, we propose a layer-wise token merging scheme that adjusts the merging number for different layers. Our method is based on an offline profiling strategy, which evaluates the accuracy of various merging candidates on a validation dataset and selects the optimal configuration. To illustrate, we take the search process for a γ value of -20 as an example. We design four merging strategies: (1) merging 20 tokens at each layer; (2) merging 25 tokens at the first two layers and no merging at the last four layers; (3) merging 23 tokens at the first three layers and no merging at the last four layers; and (4) merging 22 tokens at the first four layers and no merging at the last four layers. We then profile the accuracy of these four strategies using a validation set and select the one that demonstrates the best performance.

The accuracy and throughput comparison of four schemes is shown in Table 3.2. The results demonstrate that the layer-wise merging design improves average accuracy by 6% and also marginally increases throughput. This improvement is attributed to the layer-wise token merging method, which optimizes the distribution of tokens across layers. Based on the evaluation results, the fourth merging scheme, which achieves

Dataset \ Scheme	1	2	3	4
CIFAR10	72.05% 1316	76.16% 1376	76.72% 1335	77.13% 1355
CIFAR100	47.27% 1336	51.52% 1387	51.68% 1362	52.68% 1338
EuroSAT	70.93% 1328	82.81% 1377	83.77% 1394	85.08% 1384

Table 3.2: Accuracy and throughput (Req/s) comparison of different merging schemes.

Data Structure	**Interface**
TransformerModel	forward (input, prompt, γ)
TaskModel	get_params (params)
ServeModel	forward (input, task_list, taskmodel_list, γ)
Batch	add_a_query (query) get_profile (γ) get_query4inference (curtime)

Table 3.3: The data structure of OTAS and related methods.

the highest accuracy, can be deployed for online serving.

3.5 Implementation

OTAS Description. We provide four data structures and corresponding interfaces to implement the OTAS, summarized in Table 3.3. TransformerModel is a Transformer model class that comprises token prompting and token reduction modules. This model is loaded with pre-trained weights. TaskModel stores all parameters for a task, such as the prompts and classification head. ServeModel serves as the base model for the front-end surface. Its forward method accepts a batch of inputs, the corresponding input tasks, the parameter list of TaskModel and the γ value as input and returns the inference result. The Batch class is responsible for adding a query to

the batch, providing profiling results and returning a batch of queries within latency constraints for inference.

Implementation Tools. We implement the OTAS based on the PetS [99]. We use Python to process the incoming queries and implement the batching and token adaptation algorithms. We use PyTorch to define the neural networks, including TransformerModel, TaskModel and ServeModel. We build the Transformer model with timm library [81] and insert two modules to add and remove the processing tokens at each layer. We implement the prompt learning and token reduction methods according to VPT [38] and ToMe [2].

User Interface. The system enables users to make a query and register tasks with two interfaces. The Make_Query interface processes a query that comprises an image sample and various attributes, such as the task ID, latency requirement and utility. Then, the query can be assigned to a batch with Algorithm 1. The Register_Task interface saves the task parameters in the task model list and the corresponding latency and utility values in the task data list.

3.6 Experiment

Setup. We use the ViT-Base model pre-trained on ImageNet 21K as the foundation model, which contains 12 Transformer layers. The head number of attention is 12, and the feature dimension is 768. The patch size of the images is 16×16. We use three datasets, including CIFAR10, CIFAR100 [42] and EuroSAT [29], and 1/5 of the training data was randomly selected as the profiling set. We define the γ selection list as {-20, -15, -10, -5, 0, 2, 4, 8} and adjust it according to the query rate. The values of δ, ϵ, η and μ in Algorithm 1 are set as 0.5s, 64, 0.5s and 0.8 respectively. We set the value of β as 5 and define the initial_stage as the first 2 seconds of the service. The value of κ in Algorithm 3 is 0.8. According to Figure 3.3 and Figure 3.4, the f

γ	8	4	2	0
q (Req/s)	1∼279	280∼319	320∼348	350∼379
γ	-5	-10	-15	-20
q (Req/s)	380∼449	450∼519	520∼999	>1000

Table 3.4: The projection function from arriving rate to γ.

function is defined in Table 3.4.

We train the prompts for tasks offline. The training batch size, epochs and learning rate are set as 32, 50 and 0.002.

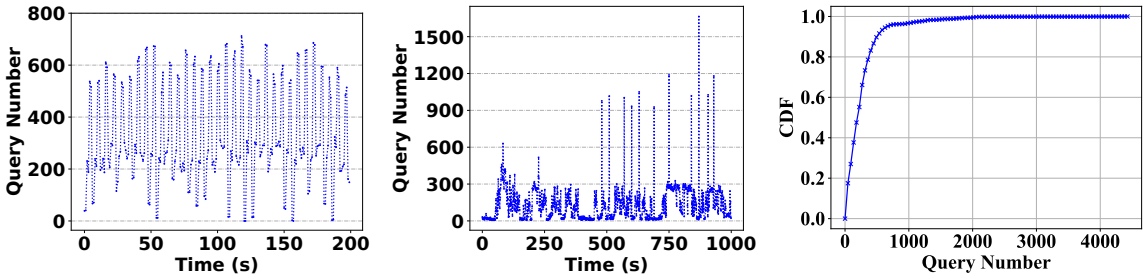
We evaluate OTAS on an NVIDIA GeForce RTX 4080 (12th Gen Intel(R) Core(TM) i9-12900K CPU) machine.

Baseline. We compare OTAS with PetS [99] and INFaaS [70]. PetS is a unified framework for serving Transformers with parameter-efficient methods and optimizes task-specific and task-shared operators. We remain token unchanged for PetS and perform inference with a shared foundation model and task-specific heads. INFaaS is a model adaptation method that selects an appropriate model according to the query load. We set the candidate model list as ViT-Small, ViT-Base and ViT-Large. We also compare OTAS with ToMe [2] and VPT [38] that uses fixed merging or prompting number. ToMe is an inference acceleration technique based on token merging, and we experiment with merging numbers of -20, -15, -10, -5. VPT is a visual prompt learning method used for finetuning pre-trained models, and we conduct experiments with prompt numbers of 2, 4, 8. We implement all evaluated policies in our OTAS prototype to ensure fairness.

Workloads. We evaluate the algorithms using both synthetic query traces and real-world production traces. For synthetic workloads, we generate query traces with varying load patterns. Query arrival times are randomly generated following a Poisson distribution, consistent with previous work [70, 24, 69]. Each query type is randomly

Query Type	Task	Latency	Utility
1	CIFAR10	0.6s	0.3
2	CIFAR10	1s	0.01
3	CIFAR100	0.6s	1
4	CIFAR100	1s	0.2
5	EuroSAT	0.6s	0.3
6	EuroSAT	1s	0.01

Table 3.5: The latency and utility of queries.



(a) The query trace on the synthetic dataset in the first 200 seconds.

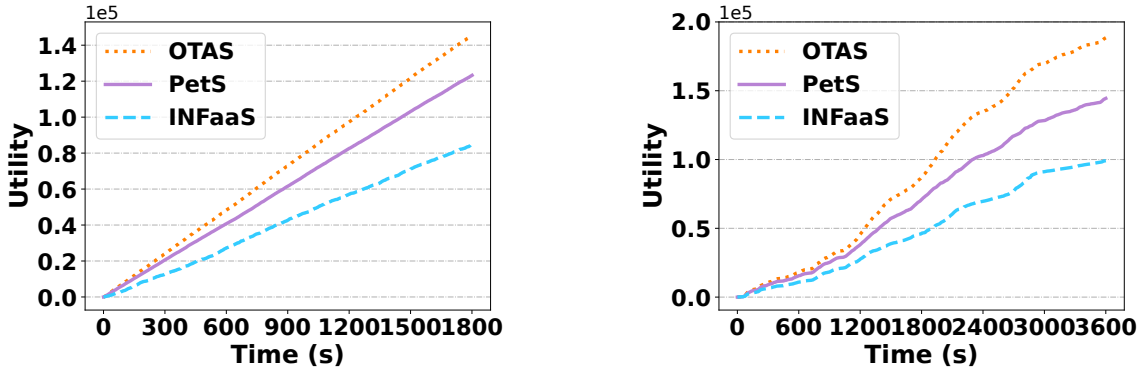
(b) The query trace on the MAF dataset in the first 1000 seconds.

(c) The CDF plot of query number per second on the MAF dataset.

Figure 3.9: The query trace on two datasets.

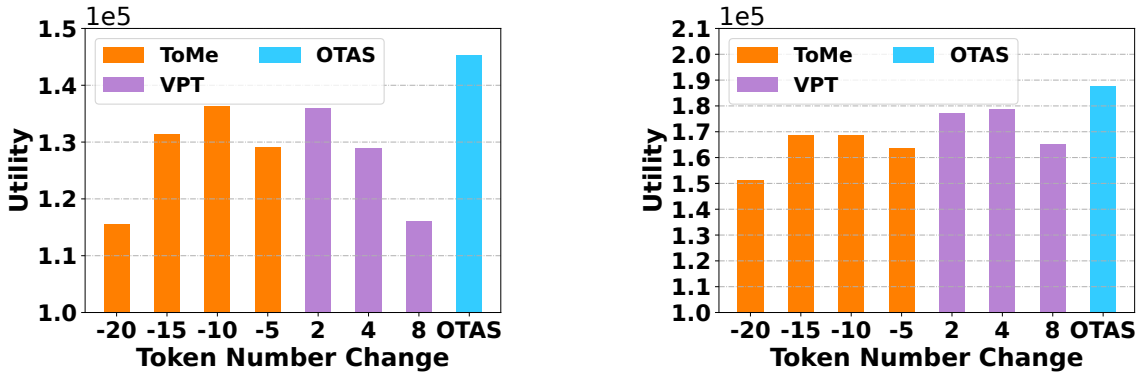
selected from those listed in Table 3.5. The experiments run over a 30-minute serving period, processing over 63,000 queries in total. Figure 3.9(a) shows the query rate per second during the first 200 seconds of the experiment, with fluctuations between 200 requests per second (Req/s) and 700 Req/s.

For real-world workloads, we use publicly available traces from Microsoft, collected from Azure Functions in 2021 (MAF) [92, 90, 48]. We select a 120-hour trace for our experiments. To simulate a more challenging scenario, requests originally recorded over two-minute intervals are aggregated into one-second intervals. Figure 3.9(b) shows the number of queries per second during the first 1000 seconds, illustrating the difficulties elastic serving systems face when handling sudden load surges. Additionally, the cumulative distribution function (CDF) of the query rate is displayed in Figure 3.9(c). Over 60% of the time, the query rate remains below 300 Req/s.



(a) Synthetic dataset. (b) MAF dataset.

Figure 3.10: The utility comparison of different system designs.



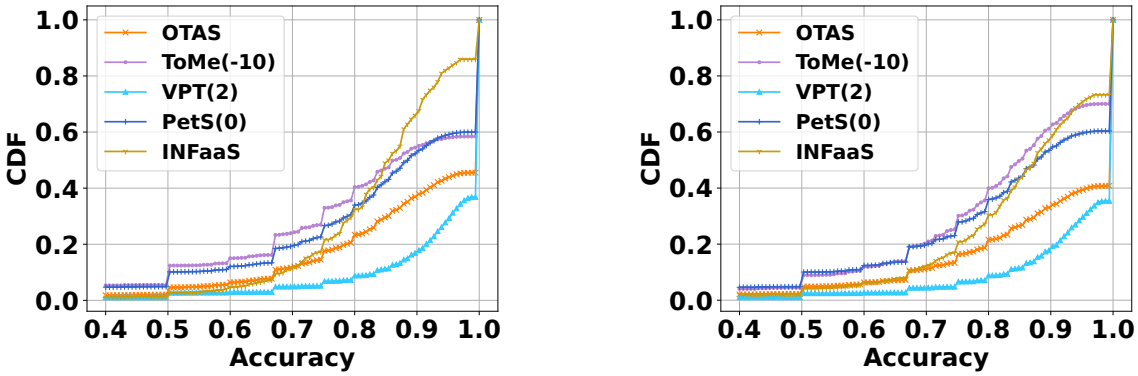
(a) Synthetic dataset. (b) MAF dataset.

Figure 3.11: The utility comparison of different token number.

However, there are instances where the rate exceeds 600 Req/s, with peaks reaching beyond 4000 Req/s.

3.6.1 Main Results

The overall utility. If the system returns an accurate result for a query within the latency constraint, it earns the utility value associated with that query. Figure 3.10 shows the cumulative utility values for three system designs on the synthetic dataset. OTAS achieves approximately 1.45×10^5 in utilities, resulting in a utility improvement



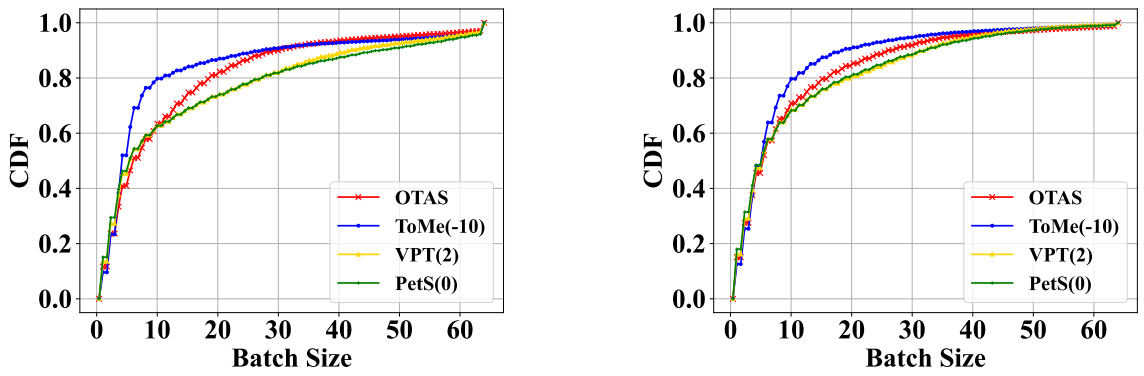
(a) Synthetic dataset. (b) MAF dataset.

Figure 3.12: The CDF plot of accuracies for served batches.

of 18.77% and 72.93% over the other designs. INFaaS performs the lowest, primarily due to high I/O latency when switching models. The total utility for the MAF dataset is displayed in Figure 3.11(b), where OTAS demonstrates a utility gain of up to 96.5%.

Figure 3.11 presents a utility comparison using a fixed token number. OTAS outperforms both ToMe and VPT, as it dynamically adjusts the token strategy based on query load. The ToMe method with a merging number of -10 and the VPT method with a prompting number of 2 achieve the next-best performance, benefiting from their capacity to either speed up inference or enhance accuracy. Conversely, the ToMe method with a merging number of -20 and the VPT method with a prompting number of 8 yield the lowest utility due to their limited prediction accuracy or high inference latency.

The accuracies of batches. Figure 3.12(a) shows the CDF of accuracies for batches served on the synthetic dataset. Higher accuracy values generally reflect greater algorithmic precision. The VPT method, using a prompting number of 2, achieves the highest accuracy due to the use of well-trained prompting tokens. In comparison, the ToMe method shows slightly lower accuracy, likely due to the reduction in token number. OTAS dynamically selects an optimal execution scheme, effectively balancing



(a) Synthetic dataset. (b) MAF dataset.

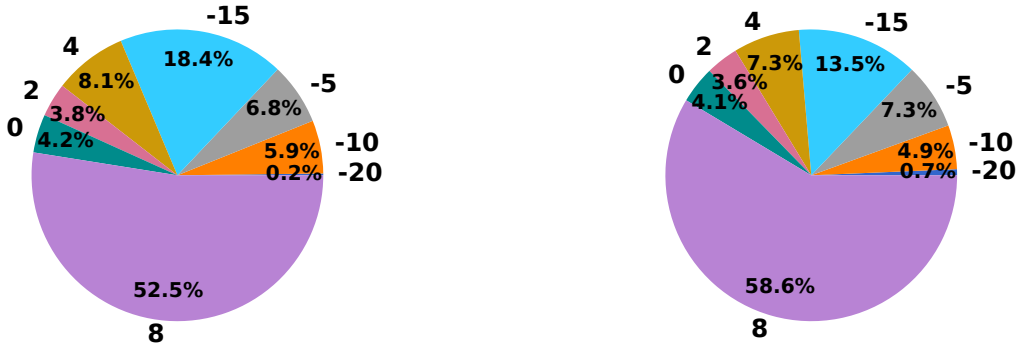
Figure 3.13: The CDF plot of the batch size.

accuracy and latency. The average accuracy of OTAS exceeds 90%, demonstrating its ability to deliver accurate results for served queries.

As illustrated in Figure 3.12(b), the batch accuracy on the MAF dataset closely resembles the pattern observed with the synthetic dataset. The accuracy curve shows a sharp rise as it nears 1, largely due to the substantial number of batches achieving a perfect accuracy score of 1.

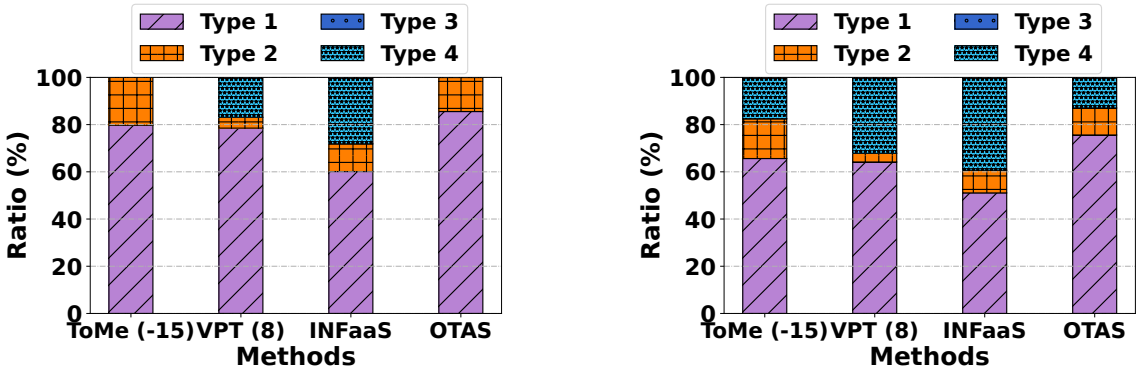
The batch size. Batching enhances throughput, and batch size significantly impacts execution efficiency. Figure 3.13 visualizes batch sizes across different methods. The ToMe method has the smallest batch sizes because token merging allows for rapid batch execution, resulting in shorter query wait times for batch formation. OTAS maintains a moderate batch size, averaging around 10. In contrast, VPT and PetS have larger batch sizes, which can increase the risk of exceeding latency constraints when too many queries are grouped together.

The γ selection. OTAS adapts the token number γ based on incoming load and query characteristics. The distribution of γ selections is shown in Figure 3.14. For the synthetic trace, OTAS most frequently selects $\gamma = 4$, as the request load remains relatively steady for much of the serving period. Another common choice is $\gamma = -15$, which helps reduce inference time while maintaining nearly the same accuracy.



(a) Synthetic dataset. (b) MAF dataset.

Figure 3.14: The γ selection of OTAS.

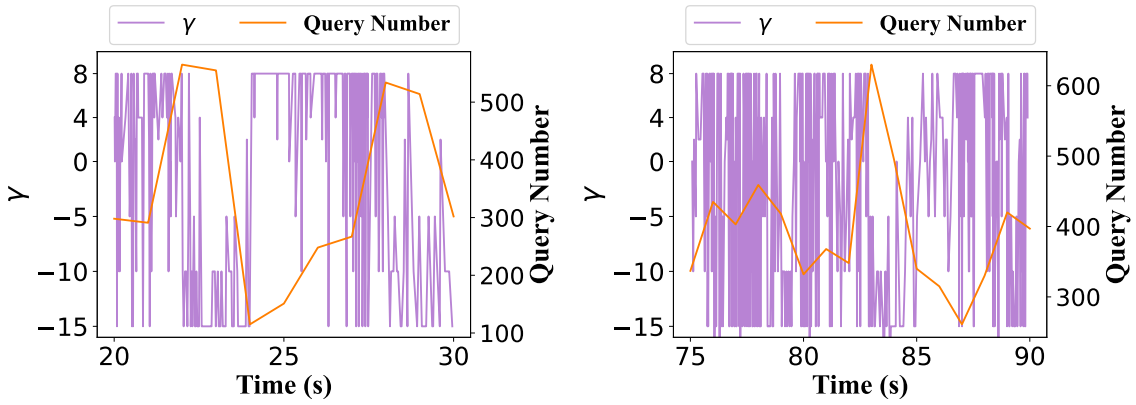


(a) Synthetic dataset. (b) Azure dataset.

Figure 3.15: The ratio of execution information of different queries.

In the MAF trace, a larger proportion of batches were processed with a prompting number of 4, reflecting lower query loads. During peak periods, however, OTAS adjusts γ to -15 to handle the increased query volume more efficiently.

The execution type of a query. Queries can yield different processing outcomes, which fall into four distinct categories. Type 1 includes queries that achieve accurate results while meeting latency requirements. Type 2 covers queries that meet latency constraints but yield incorrect results. Type 3 consists of queries that produce inference results but do not meet latency deadlines. Lastly, Type 4 comprises queries that cannot meet latency constraints before execution and are consequently evicted from



(a) Synthetic dataset. (b) MAF dataset.

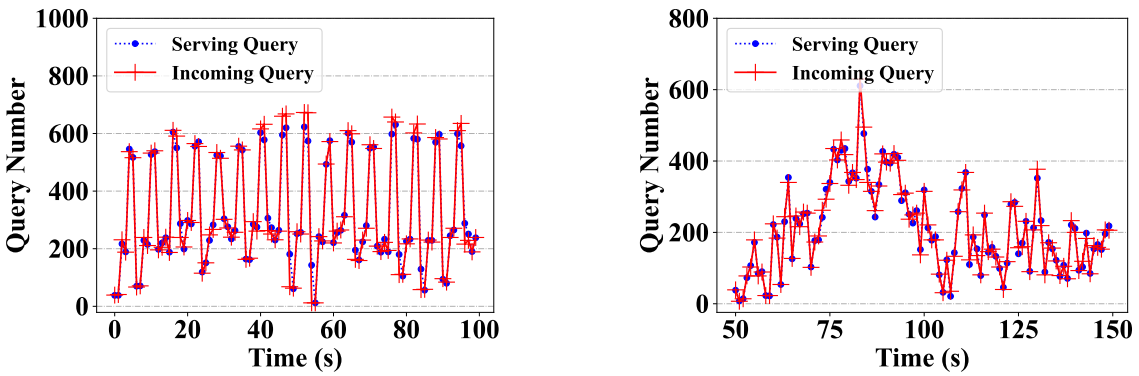
Figure 3.16: The γ selection trace and the query number trace.

the system.

Figure 3.15(a) illustrates the execution ratios of various query types on the synthetic dataset. As shown, OTAS successfully serves 85.35% of queries (Type 1), ensuring that all served queries meet latency requirements. In comparison, ToMe serves fewer queries due to its lower prediction accuracy, while VPT has longer inference times, resulting in a higher eviction rate. Thus, OTAS achieves a better balance between accuracy and latency, leading to a higher overall query serving rate.

Figure 3.15(b) shows the execution ratios of different query types on the MAF dataset. Due to the presence of highly bursty loads, the proportion of evicted queries (Type 4) increases, highlighting limitations in computational resources. Among the methods, OTAS achieves the highest request serving rate, with a success rate of 75.62%. In contrast, the ToMe method mispredicts 15.5% of requests, negatively impacting service quality. VPT's success rate is lower, at 72.93%, largely due to increased inference latency from token prompting, resulting in 23.11% of requests missing their deadlines and being evicted. Overall, OTAS demonstrates greater adaptability in handling bursty query loads.

The γ selection trace. We collect the γ selection trace and the query number trace



(a) Synthetic dataset. (b) MAF dataset.

Figure 3.17: The incoming query number and serving query number per second.

from a selected period and show them in Figure 3.16. If the query number is large (i.e., bursty incoming queries), the algorithm should choose a smaller γ value to accelerate inference and serve more queries. On the other hand, if the query number per second is small (i.e., fewer incoming requests), the algorithm should choose a larger γ value to enhance the prediction result. Figure 3.16 shows this token adaptation tendency, and it represents that our algorithm can deal with different query loads well. For synthetic traces, OTAS selects the token reduction scheme when the query number per second is larger than 500 Req/s and the token prompting scheme when the query number per second is smaller than 300 Req/s. The γ selection on the MAF trace is more sensitive to changes in load, resulting in fluctuations between 4 and -15.

The serving query number. The incoming query number and serving query number per second within a selected period are presented in Figure 3.17. The serving number of OTAS can approach the number of incoming queries, which means it has a desirable serving throughput.

The waiting and inference time. The waiting time of served queries and the inference time of batches is demonstrated in Figure 3.18. In most cases, the waiting time and running time are less than 100ms, indicating that the queries can be processed quickly.

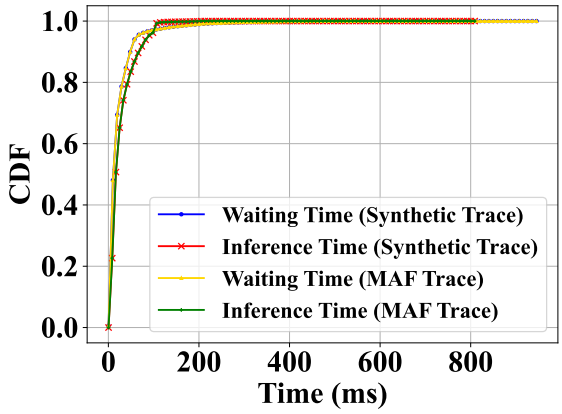


Figure 3.18: The CDF plot of waiting time and inference time.

3.7 Chapter Summary

We present OTAS, an elastic serving system for large Transformers based on an innovative idea of token adaptation. We implement a prototype that supports dynamically allocating the token number for a batch and running the Transformer model in flexible way. The results show that OTAS can improve the utility by at least 18.2% on both simulated and real-world production traces from Azure compared with other state-of-the-art methods. The observed performance improvement is achieved because OTAS can identify an optimal balance between the overhead of token increment and the benefits of accuracy improvement based on the real-time query load and user demands.

Chapter 4

Prodigy: An Elastic and Robust Transformer Serving System via Token-Reduction Warm-up

As established in chapter 3, token reduction has become a critical technique for enabling elastic Transformer serving by dynamically determining the number of tokens during inference to match available hardware resources and dynamic request loads. However, accuracy rapidly degrades as the number of reduced tokens increases. This weak robustness necessitates accurately identifying redundant or useless tokens in existing works. In this chapter, a further step is taken to involve the fine-tuning process to extend the robustness of Transformer models for token reduction. We consider a novel serving scheme that prepares a set of fine-tuned models towards different reduction ratios to 'warm up' the serving system. Our concept is grounded in the understanding that fine-tuning with awareness of token reduction contributes significantly to enhancing inference accuracy. However, exhaustive fine-tuning for all ratios leads to prohibitive computational costs. As a solution, we propose PRODIGY, an elastic and robust Transformer serving system that only requires fine-tuning a lim-

ited set of models while scaling to arbitrary reduction ratios. Specifically, we adopt an optimization algorithm that strategically selects a subset of reduction ratios for fine-tuning, and a model ensemble to leverage different models for inference. The experimental results show that PRODIGY improves accuracy by 27% with limited fine-tuning burden.

4.1 Introduction

Transformer model empowered architectures have become a pillar of intelligent cloud services, offering unprecedented capabilities for creating intelligent applications and handling complex tasks, e.g., generative AI, drug discovery, etc [3]. Transformers are adept at capturing long-range dependencies in input data through the attention mechanism [77]. This computation mechanism also incurs a significant inference latency, which presents a hurdle for the extensive deployment of advanced Transformers.

Deploying such sizable models poses significant challenges due to constrained hardware resources and fluctuating query demands. The rapid expansion in Transformer model size exacerbates a substantial burden on the serving system. From the application's perspective, significant variations in request loads can strain the system's ability to meet users' stringent latency demands, leading to an unsatisfactory experience.

To address the above challenges, token reduction, which reduces the token number of Transformer, becomes an important method to expedite model inference by establishing an elastic serving system for Transformers [26, 9]. Generally, token reduction includes token pruning and token merging. Token pruning identifies and removes unnecessary tokens that contribute little to the final prediction, such as those representing the background, with negligible loss of accuracy [68, 50]. Token merging aims at combining similar tokens together with a lightweight matching strategy [2, 54]. To fully harness the potential of token reduction, the serving system needs to dynami-

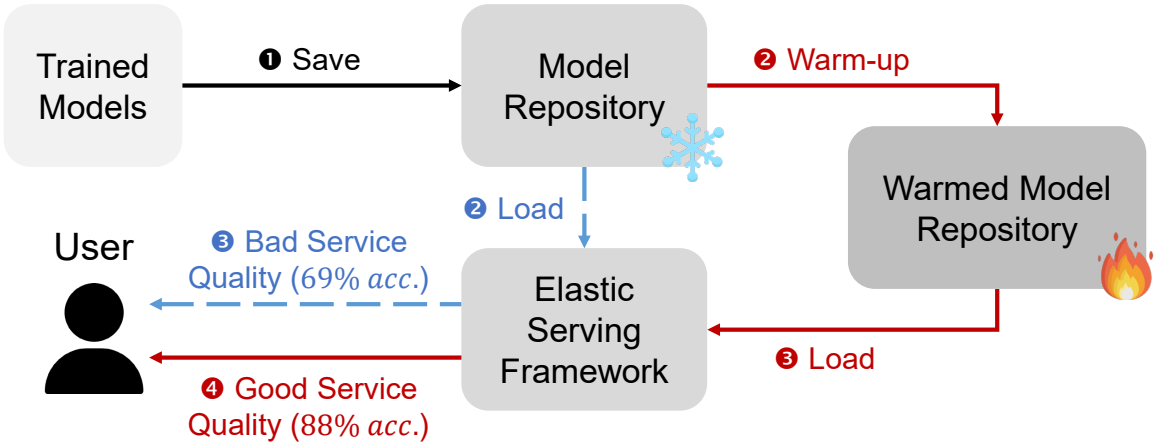


Figure 4.1: The workflow and service quality of the existing works without token-reduction warm-up and ours.

cally select a reduction ratio for a batch of queries according to available hardware resources, current query loads and the inference latency requirement, finding a trade-off between accuracy and efficiency [9].

However, a large token reduction ratio incurs a significant accuracy drop that diminishes the advantages of token reduction, thus negatively influencing the serving quality. While designing more advanced token reduction algorithms can enhance the performance of the serving system, the potential improvement may be limited due to the inevitable information loss resulting from large reduction ratios. In this chapter, we address this issue from the perspective of system design and propose a novel serving system, called PRODIGY, aiming at enhancing the efficacy and elasticity of Transformer deployment. As shown in Figure 4.1, our system is based on a new idea called *token-reduction warm-up*, which warms elastic model serving via fine-tuning and adapts to token reduction in advance. Specifically, we prepare multiple versions of serving models by fine-tuning models with different reduction ratios. This is motivated by the key insights that fine-tuning a model with token reduction before inference markedly improves the model performance during inference with token reduction. This is because the models have been adapted to the shifted feature space resulting from token reduction, which mitigates the loss of information and diminishes

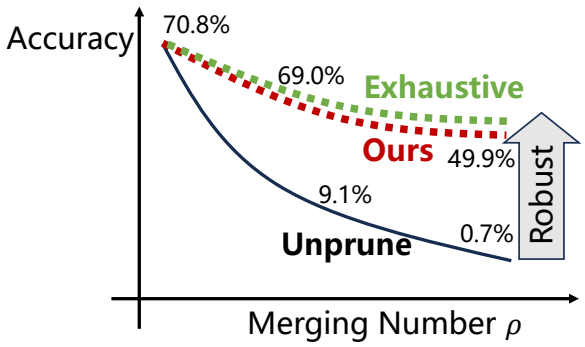


Figure 4.2: The robustness for token reduction. The robustness means that the accuracy remains stable with the increase in the merging number. The numbers in the figure correspond to Figure 4.3.

the potential for overfitting. These models preview the pattern of token reduction, thus improving the accuracy and successfully serving queries with different ratios.

Despite the promising potential, fine-tuning models across all ratios results in significant training expenses and large storage costs. To reduce the resource consumption of token-reduction warm-up, we delve into the relationship between model performance and the reduction ratio during the fine-tuning process. Our findings reveal that token reduction exhibits *downward robustness*, meaning that a model fine-tuned at a larger ratio can also accommodate lower reduction ratios. Based on this, we design an optimization algorithm that selects a limited subset of ratios for fine-tuning, aiming to maximize average accuracy within the bounds of restricted training expenses. Our method is straightforward and effective, ensuring good model performance as shown in Figure 4.2.

During the serving process, we choose an appropriate fine-tuned model for inference with a given testing ratio in offline profiling. Specifically, we evaluate the accuracy of all models offline and then select the one with the highest accuracy. Furthermore, we develop a model ensemble algorithm that tailors a composite model for a specific testing ratio, thus enabling the inference with arbitrary token reduction ratios.

The contributions of the chapter are summarized as follows.

- We develop a novel concept called *token-reduction warm-up* to investigate the benefits of token reduction aware fine-tuning for Transformer serving.

- We propose Prodigy, an elastic and robust Transformer serving system by fine-tuning Transformer models with a limited set of token reduction ratios. We develop an efficient fine-tuning planner and a model ensembler to maximize the average accuracy during serving with limited fine-tuning costs.

- We conduct comprehensive evaluations for Prodigy. The results show that Prodigy can improve the average accuracy by 27% with high fine-tuning efficiency, thus achieving satisfactory token reduction robustness.

This chapter is organized as follows. Sections 4.2 and 4.3 provide the background and motivation of this chapter. In section 4.4, we introduce the system design and our methodology, including Fine-tuning Planner, Model Ensembler and Reduction Scheduler. Sections 4.5 and 4.6 show the implementation and experimental results. We conclude this chapter with section 4.7.

4.2 Background

Elastic Serving. Nowadays, numerous widely-used applications are powered by the Transformer architecture, providing intelligent services for billions of users [21, 62]. To satisfy the increasing demands of users and provide high-quality experiences, offering precise and instantaneous services is crucial. Unfortunately, the Transformer is known for its substantial computational overheads, resulting in significant inference delays [75]. Therefore, building an elastic serving system for the Transformer is of great importance. An elastic serving system should adjust the inference stage dynamically based on hardware resources, system load, and user requirements.

As discussed in chapter 2 and chapter 3, there are three classes of work for building

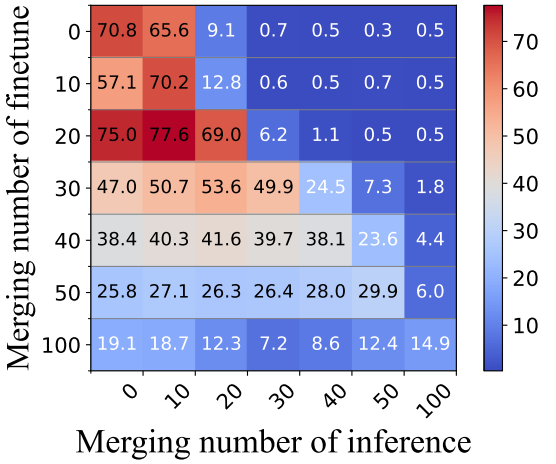


Figure 4.3: Accuracy comparison with various token merging numbers.

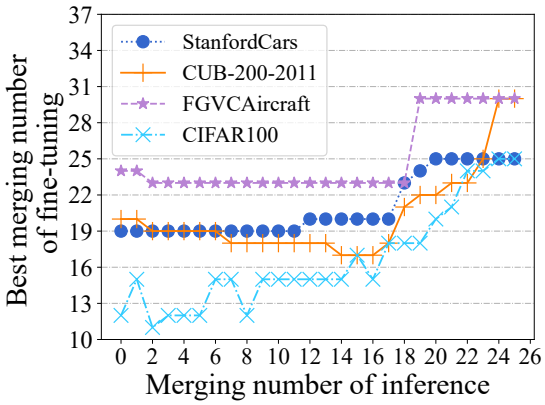


Figure 4.4: Best merging numbers for fine-tuning across different inference settings.

an elastic serving system. The first class of works, called model adaptation, pre-trains multiple versions of models with different sizes and adaptively selects a suitable model for service [70, 79, 23]. The second class of works, called resource scaling, dynamically manages the available hardware resources, such as virtual machines and containers, to accommodate the fluctuating resource demands [90, 48]. The third class of works, called token reduction, specifically designed for Transformer, adjusts the processing tokens during inference [9, 86]. Token reduction achieves remarkable results with low training and I/O costs. This chapter focuses on the model serving for Transformers based on token reduction.

Issues of existing efforts. To minimize the negative impact of token reduction on model accuracy, existing works design strategies to ignore less important tokens and merge similar tokens during the inference stage accurately. However, no matter the token reduction approach, increasing the reduction ratio inevitably leads to accuracy degradation. The inaccurate inference results significantly undermine the service quality of a serving system. To verify this phenomenon, we conduct experiments using a token merging algorithm (i.e., TOME [2]) on the CIFAR100 dataset. When the token merging number of each Transformer layer increases from 0 to 25, the

accuracy plummets from 91.7% to 19.3%. The experimental results indicate that the current model cannot deal with samples with high token reduction ratios, thus failing to leverage the benefits of token reduction and provide a good service experience.

4.3 Motivations

Inspired by a famous quote, "**Before anything else, preparation is the key to success.**", instead of following the existing works to design better strategies to ignore less important tokens during the inference stage, we propose a new concept called *token-reduction warm-up*. The concept is based on the intriguing observation that by integrating token reduction during the fine-tuning stage, the model can learn to cope with merged tokens, thereby enhancing the model's resilience to token reduction during the inference stage. However, we also note that the concept introduces a considerable computational and storage overhead when implemented in a naive manner. In the following, we investigate the idea in more detail and explore the potential for its application to a serving system.

4.3.1 Warm-up is All you Need

The initial step is to examine the impact of the fine-tuning stage with token reduction on the performance during the inference stage with token reduction. We use a ViT model pre-trained on the ImageNet 21K dataset as a backbone. We fine-tune the model on StanfordCars [41]. The token reduction is based on TOME [2] with a merging number from 0 to 100.

Observation 1. *Appropriate token reduction during the fine-tuning stage can enhance the model's accuracy during the inference stage. Conversely, inappropriate token reduction (e.g., an excessively high or low pruning ratio) during fine-tuning can result in a decline in the accuracy during inference.*

Figure 4.3 illustrates the accuracy achieved with varying token merging numbers during the fine-tuning and inference stages for the StanfordCars dataset. The results show that the fine-tuning stage with moderate token reduction improves accuracy during the inference stage with token reduction. For example, when the merging number is 20 during inference, the accuracy of a vanilla model drops from 70.8% to 9.1%. In contrast, the accuracy remains at 69.0% if the model is fine-tuned with a merging number of 20.

Fine-tuning with a high pruning ratio or a large merging number may decrease accuracy during inference. For example, in Figure 4.3, the optimal accuracy for merging number 10 is 77.6%. However, it drops to 18.7% for a model fine-tuned with a merging number of 100.

Observation 1 motivates the development of some token-reduction warm-up models within an elastic serving system with token reduction. These models are designed to be fine-tuned with token reduction before inference. However, an elastic serving system has many possible merging numbers during inference to meet users with diverse demands. One possible approach is to fine-tune models with all available merging numbers and then select the optimal fine-tuned model for a given merging number during inference. Nevertheless, it is both computationally and storage-intensive, resulting in a significant burden on resources. In order to alleviate the burden, in the following, we delve into two characteristics of token-reduction warm-up: downward compatibility for token-reduction warm-up (subsection 4.3.2) and token-reduction warmed model ensembling (subsection 4.3.3).

4.3.2 Downward Compatibility for Warm-up

To apply the idea of token-reduction warm-up at a low cost, it is necessary to identify a small set of merging numbers for selective fine-tuning. This subsection delves into the relationship between inference accuracy and models that have been fine-tuned

with different levels of token reduction.

Observation 2. *For a given merging number during inference, a model fine-tuned with a higher merging number performs better than one fine-tuned with a lower merging number.*

According to Figure 4.3, downward compatibility exists for token-reduction warm-up. In other words, the values in the lower triangle are more closely aligned with those along the diagonal than those in the higher triangle. As shown in Figure 4.3, for an inference stage with a merging number of 20, a model fine-tuned with a merging number of either 30 or 40 performs better than that fine-tuned with a merging number of either 0 or 10.

Observation 2 motivates us to warm up a model using a high merging number during fine-tuning to serve an inference stage with lower merging numbers. However, according to **Observation 1**, an excessively high merging number during fine-tuning may result in an inference accuracy decline. Therefore, as shown below, we investigate the selection of a small set of high merging numbers during the fine-tuning stage without a corresponding decrease in inference accuracy.

Observation 3. *For some merging numbers during the inference stage, there exists a similar merging number during the fine-tuning stage that performs optimally.*

As shown in Figure 4.4, for each merging number during inference, we record the corresponding optimal merging number during fine-tuning for four image classification datasets. The best fine-tuning merging number remains relatively stable at approximately 20 for all datasets initially, and the value increases to a new stable point when the inference stage's merging number exceeds 20. For example, for the FGVCAircraft dataset, the optimal merging number for fine-tuning is 23 when the inference stage's merging number is below 20. This number slightly increases to 30 as the inference stage's merging number rises.

Table 4.1: Accuracy for various warmed models supported by model ensemble, when the merging number is 15 during the inference stage. (x, y) in the first column means the combination weight of two models. F.T. indicates fine-tuning.

Merging num. during F.T.	CIFAR	CUB	FGVCA	SCars
10	91.41	78.56	47.45	61.45
15	91.37	80.11	48.98	70.13
20	91.08	79.88	54.15	77.19
10&20 (0.5,0.5)	92.39	80.07	53.84	74.4
10&20 (0.1,0.9)	91.58	80.45	54.54	77.3
10&20 (0.2,0.8)	91.79	80.98	54.43	77.21

Observation 3 motivates us that it is possible to select a small set of high merging numbers for the fine-tuning stage. In other words, it is possible to apply the idea of token-reduction warm-up into an elastic Transformer serving system at a low cost. Nevertheless, it remains a significant challenge to identify the optimal set, given the high cost of exhaustively examining the accuracy of every possible pair of fine-tuning and inference stages' merging numbers.

4.3.3 Token-Reduction Warmed Model Ensemble

Due to resource constraints, we can only have a small set of models fine-tuned with token reduction, i.e., token-reduction warmed models. In this subsection, we demonstrate how to extend the generalization of token-reduction warmed models via model ensemble, i.e., weighted averaging of the model parameters to generate a new model [84, 83, 34, 35, 11].

Observation 4. *The model ensemble allows for combining the strengths of models fine-tuned by different merging numbers to adapt to more merging numbers during inference.*

As shown in Table 4.1, the combination of two models, each fine-tuned with a different token merging number (10 and 20), has been demonstrated to enhance inference performance compared to a single model in some cases. For example, for CIFAR, if

we combine two warmed models with coefficients of 0.5 and 0.5, the accuracy can reach 92.39%, representing an accuracy increase of 1% compared to a single model. When fine-tuned with a lower merging number, the model can learn the original image features. When fine-tuned with a larger merging number, the model can learn how to process with token merging. Therefore, the model ensemble can combine these advantages and improve performance. Besides, the optimal combination coefficients vary for different datasets.

Observation 4 motivates the combination of warmed models and the careful design of the ensemble method, which can help to improve the overall generalization performance of a small set of warmed models.

4.3.4 Explainability for Token-Reduction Warm-up

This subsection analyzes the reasons for performance improvement caused by the idea of token reduction warm-up. To this end, we consider the following points. **1) Fine-tuning and inference alignment:** Token reduction transforms the features from fine-grained image features to coarse-grained local features or merged features. This requires the Transformer layers to adjust to new feature inputs and adapt to the coarse-grained features well. **2) Regularization:** Token reduction can be regarded as a data augmentation method, thereby reducing overfitting risk. **3) Reducing noise:** Token reduction can reduce the amount of information the model needs to process, thereby effectively reducing the noise in the input and ignoring unimportant or repetitive features. The model can focus on the features most important for inference.

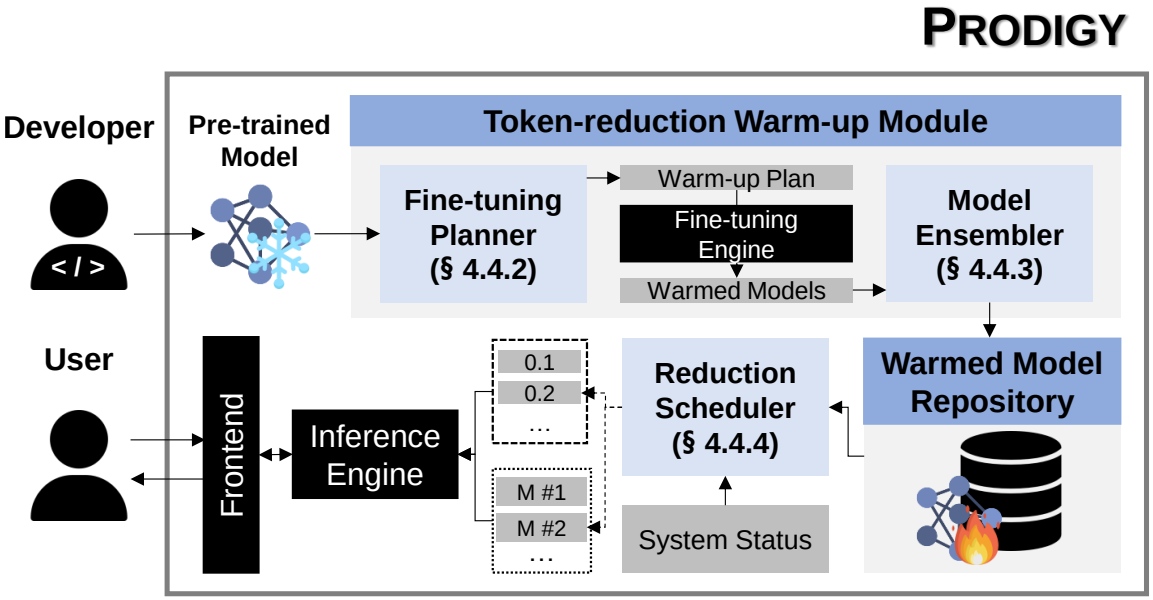


Figure 4.5: The system architecture of PRODIGY.

4.4 Prodigy Design

4.4.1 System Overview

According to section 4.3, to mitigate the performance degradation caused by token reduction during the inference stage and apply the idea of token-reduction warm-up at a low cost, we propose a new elastic serving system called PRODIGY. The system architecture is shown in Figure 4.5.

When a developer registers a new model, the system will pre-process the model through a *token-reduction warm-up module*. The newly introduced module conforms to the following design principles: 1) **Robustness**: PRODIGY should maintain high accuracy over different merging numbers during the inference stage; 2) **Computational efficiency**: The warm-up for each newly registered model should be completed as quickly as possible; 3) **Storage efficiency**: The additional storage space used by the new module should be limited.

For these objectives, the module is composed of two sub-modules as follows. First, a *fine-tuning planner* generates a plan for merging numbers during the fine-tuning stage. The plan is based on the characteristics of the newly registered model and the warm-up budget (e.g., computational costs and storage expenses) specified by the developer (refer to subsection 4.4.2). Given a fine-tuning plan, the fine-tuning engine fine-tunes the newly registered model with a set of merging numbers and gets a set of warmed models. Second, to improve the overall generalization performance of a small set of warmed models, a *model ensembler* will combine the warmed models further by selecting a combination coefficient (refer to subsection 4.4.3). All warmed models are stored on the warmed model repository.

Once a user has initiated a model serving request (or a batch) to the system, according to the warmed model repository and the system status (e.g., query load, available resources, and user demands), in PRODIGY, a *reduction scheduler* determines which merging number and which warmed model will be employed by the inference engine for elastic Transformer serving (refer to subsection 4.4.4).

4.4.2 Fine-tuning Planner

To search for a small set of merging numbers for token-reduction warm-up, it is necessary to consider a trade-off between two factors: (1) *robustness*: the average accuracy across all merging numbers during the inference; and (2) *system efficiency*: the computational and storage costs for the warmed models. We visualize the trade-off in Figure 4.6. A method named Unprune, which deploys a vanilla model without token reduction during the fine-tuning stage, maintains high system efficiency but results in low robustness. To achieve high robustness, a method named Exhaustive, which fine-tunes the model with all possible merging numbers, will be overwhelmed by significant computational and storage costs. The objective is to identify an optimal point on the trade-off curve that achieves high robustness and system efficiency.

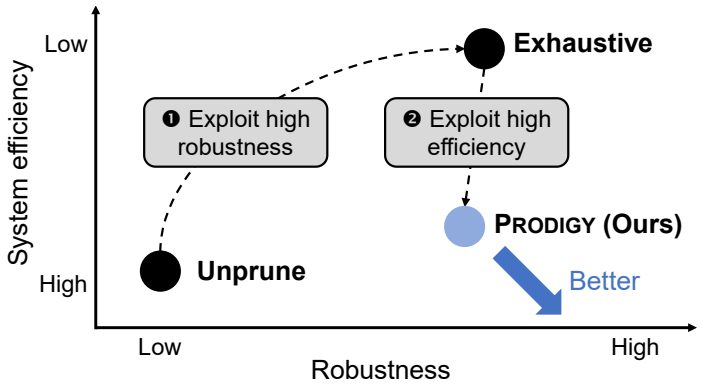


Figure 4.6: Comparison between the Unprune method, the Exhaustive method, and the fine-tuning planner in PRODIGY.

Problem formulation. An optimization problem is formulated to identify a set of merging numbers for fine-tuning that maximize robustness during inference while maintaining system efficiency. We denote the set of available token merging numbers as R, where each specific number in this set is denoted by $r \in R$. The notation $\gamma_r \in \{0, 1\}$ signifies whether the model is fine-tuned with the r-th merging number. The inference accuracy with r-th token reduction number in the fine-tuning stage is a_r. To maximize the average accuracy for all merging numbers by selecting suitable merging numbers for fine-tuning, we formulate the problem in Eq. (4.1).

$$\max_{\gamma_r} \quad \frac{1}{|R|} \sum_{r \in R} a_r \tag{4.1}$$

$$s.t. \quad \sum_{r} (\gamma_r \times c_r) < \alpha; \tag{4.1a}$$

$$\sum_{r} (\gamma_r \times o_r) < \beta. \tag{4.1b}$$

For the system efficiency, we denote the time and storage costs for fine-tuning the r-th number as c_r and o_r, respectively. We define two bounds α and β to restrict the warm-up consumption in Eq. (4.1a) and Eq. (4.1b).

Outline. Solving the problem is challenging because we do not have any information about a_r before fine-tuning the model with the r-th merging number, making apply-

ing traditional optimization algorithms hard. Thus, we design a heuristic algorithm motivated by the following observation about token-reduction warm-up. According to subsection 4.3.2, the warm-up effect can be divided into two stages: (1) Flat region: as the merging number increases, the accuracy remains stable, and a model with a larger merging number can generalize to these small merging numbers. For example, a model fine-tuned with a merging number of 20 can perform well when tested on cases with a merging number lower than 20. (2) Declining region: the accuracy significantly decreases as the merging number increases. Models should be specifically fine-tuned for these merging numbers to adapt to shifted features caused by token reduction and compensate for the loss of information. Therefore, we define a parameter r_{mid} to split ratios into two pairs and fine-tune models for numbers greater than r_{mid}.

Detailed design. In the search for optimal merging numbers, we select those that, when utilized for fine-tuning models, yield the greatest enhancement in accuracy. Specifically, we design a pair $[r_l, r_h]$ for the numbers (e.g., [20,30]), beginning at r_l and concluding at r_h. At each iteration, we select a pair with the largest accuracy difference and use the midpoint of this pair as the fine-tuning number. The pseudocode is shown in Algorithm 4. At first, we select the two smallest and largest ratios for fine-tuning (i.e., r_{mid} and r_{max}) and estimate their fine-tuning time costs, storage costs and accuracy on the profiling set. From line 9 $\sim$ 24, we find a new feasible merging number. We first sort the pairs according to their accuracy differences and proceed to iterate through each pair. We calculate a new number r_{new} in line 15 and profile the corresponding costs. We find a new merging number if the value meets the constraint. Otherwise, we gradually increase the merging number and check if it meets the requirements because increasing the merging number can reduce fine-tuning costs (Line 20 $\sim$ 24). Upon obtaining an appropriate number, we add this number to R^* and divide the chosen pair into two smaller pairs. Finally, we update the overall warm-up cost and check if it exceeds the threshold.

Algorithm 4: Find warm-up ratios.

Input: time constraint α; memory constraint β; middle ratio r_{mid};
Output: warm-up ratios R^*;

$R^* \leftarrow \{\}$, $I \leftarrow \{\}$; // Initialization
// Always include two end ratios
$c_{mid}, o_{mid} \leftarrow \text{ProfileCost}(r_{mid})$; // c: time cost
$c_{max}, o_{max} \leftarrow \text{ProfileCost}(r_{max})$; // o: memory cost
I.add($[r_{mid}, r_{max}]$); // Store pairs
Finetune two models with r_{mid} and r_{max};
R^*.add(r_{mid}); R^*.add(r_{max});
$C \leftarrow c_{mid} + c_{max}; O \leftarrow o_{mid} + o_{max}$; // All cost
while *True* **do** // Iteratively find a new ratio
    Profile the accuracy difference between each pair in I, based on which sort I in descending order;
    $find \leftarrow False$;
    for $[r_l, r_h] \in I$ **do** // Iteratively select a pair
        I.delete($[r_l, r_h]$);
        if $r_h - r_l == 1$ **then** // Cannot split
            continue;
        $r_{new} \leftarrow \frac{r_l + r_h}{2}$; // A new fine-tuning ratio
        $c_{new}, o_{new} \leftarrow \text{ProfileCost}(r_{new})$;
        if $C + c_{new} < \alpha$ *and* $O + o_{new} < \beta$ **then**
            $find \leftarrow True$; break; // Under bounds
        else if $O + o_{new} < \beta$ **then**
            // Increase r_{new} to reduce c_{new}
            while $C + c_{new} \geq \alpha$ *and* $r_{new} + 1 < r_h$ **do**
                $r_{new} \leftarrow r_{new} + 1$;
                $c_{new}, o_{new} \leftarrow \text{ProfileCost}(r_{new})$;
            if $r_{new} + 1 < r_h$ **then** // Find a r_{new}
                $find \leftarrow True$; break;
    if $find == False$ **then**
        break;
    I.add($[r_l, r_{new} - 1]$); I.add[r_{new}, r_h];
    Finetune a model with r_{new};
    R^*.add(r_{new}); $C \leftarrow C + c_{new}; O \leftarrow O + o_{new}$;
    if $C \geq \alpha$ *or* $O \geq \beta$ **then**
        break;
return R^*;

Algorithm 5: Model ensemble.

Input: profiling Set D_p; models M;
Output: weight W;

```
Initialize weight W ← {};
for r ∈ R do
    A = {};
    for m ∈ M do
        acc = ProfileAcc(M, r);
        A.add(acc);
    A ← Normalize(A);
    W[r] ← A;
return W;
```

Analysis. Our approach is based on a key principle that selects the midpoint of the pair exhibiting the greatest accuracy difference. From the analysis in section 4.3, it is observed that models with large merge numbers suffer more degradation of accuracy caused by token reduction, which means selecting a ratio from the steepest pair can lead to the greatest accuracy enhancement. We select the midpoint because it can benefit more ratios in the pair, thereby contributing to the improvement of the average accuracy. By increasing the value of α and β, our algorithm can generalize to exhaustive tuning.

4.4.3 Model Ensembler

As illustrated in subsection 4.3.3, ensembling models can significantly enhance the accuracy of token merging. Recent works also demonstrate that averaging weights of multiple fine-tuned models improves accuracy without introducing additional inference time [84, 83]. The reason is that the resulting model has the ability to acquire diverse knowledge from different models, thereby improving its ability to process both the raw image features and the merged features. However, designing an optimal model ensemble scheme for token reduction is a non-trivial task. First, determining the optimal combination weights is a major challenge, as it is crucial for enhancing

performance. Second, generating too many models will consume significant memory resources. To deal with these challenges, we propose an adaptive model ensemble scheme to determine the combination weights. Additionally, we design a resource-aware model ensemble method to optimize resource usage.

Adaptive model ensemble. We perform model ensemble on the parameter space to derive a new model for the unwarmed ratios. The primary task is to find optimal combination weights W for different warmed models. These optimal weights should be able to perceive the adaptability of different models to the target ratio. To achieve this, we design an adaptive model ensemble method based on an offline profiling approach to obtain the combination coefficients, as outlined in Algorithm 5. The core idea is to evaluate the model's compatibility for a ratio r by considering the profiling accuracy. In line $2 \sim 6$, we calculate the accuracy for testing ratio r using fine-tuned model m on the validation dataset. Then, we normalize the accuracy set and use it as the weight for the combination.

Resource-aware model ensemble. Generating an excessive number of models results in significant storage overhead. To optimize resource usage, we adjust the number of merged models to maximize average accuracy under resource constraints. This problem can be formulated as a variant of the multidimensional knapsack problem. In this context, each ratio can be viewed as an object in the knapsack problem, where the accuracy represents the value and storage cost represents the size of each object. The objective is to maximize the total value (accuracy) while ensuring that the overall size (cost) of the objects remains within the budget. Various solutions for the knapsack problem, such as greedy algorithms and dynamic programming, can be employed to address this issue. By utilizing this method, the system can adapt smoothly to different reduction numbers with minimal overhead.

4.4.4 Reduction Scheduler

Once the warmed models have been obtained, a significant challenge arises in selecting an appropriate model from the warmed model repository for each merging number during inference. This is a crucial step in enhancing the quality of service for a Transformer serving system. To solve the challenge, we design a warmed model selection scheme for the reduction scheduler based on offline profiling. Specifically, we first record the accuracy of all warmed models on the validation set with different merging numbers. For each merging number, the model achieving the highest accuracy for inference will be recorded. Therefore, during the inference phase, given a merging number for a batch of requests, the reduction scheduler can select the corresponding warmed model based on the offline profiling results.

Moreover, to maximize the potential of token reduction, the token reduction scheduler is responsible for dynamically adjusting the token number based on factors such as query load, available resources, and user demands. To optimize resource utilization and maximize user experience, this chapter adopts a throughput-based method to allocate the token number. First, we profile the throughput of the Transformer model with different token number settings on the profiling dataset. During online serving, the system estimates the query load (i.e., the number of queries per second) during the inference of the last batch and assigns a maximum token number for a new batch whose throughput is larger than the estimated query load. Finally, Prodigy can execute the inference for a batch of requests with a selected token number with the corresponding optimal warmed model.

4.5 Implementation

We implement a prototype for Prodigy. We use PyTorch to define the model that takes a batch of samples and a reduction number as input and outputs the prediction

result. PRODIGY can be supported by various prevalent inference engines, such as Triton [61], FastTransformers [60], etc. The server needs to export all fine-tuned models and build a model repository. The client-side sets up a connection with the inference server and sends an inference request to the inference server.

The token reduction method is inserted into the vision Transformer model, which is extended from current merging approaches. The implementation of token merging is based on ToME [2], where similar tokens are merged according to the similarity between features.

PRODIGY provides two interfaces. One interface named Find_Ratio accepts a dataset and a pre-trained Transformer model as inputs from developers and produces warmed models as outputs. It finds a merging number according to Algorithm 4 and invokes the fine-tuning and profiling interfaces of the model. The other interface, named Serving, can be invoked by users and processes a batch of samples along with a specified merging number to perform inference with token reduction.

4.6 Experiment

4.6.1 Experimental Setup

Workloads. We use the ViT-Base, ViT-Large and DeiT models [18] pre-trained on ImageNet 21K as the backbone. We evaluate the proposed system on four image classification datasets (including StanfordCars [41], CUB-200-2011 [78], FGVCAircraft [53] and CIFAR100 [42]). For CIFAR100, we randomly select 1/5 of the training data as the profiling set in Algorithm 4 and Algorithm 5. For the other datasets, we use the validation set defined in Coop [98] as the profiling set in our system. We use two query traces for evaluation: 1) The synthetic trace. We generate a query load according to the Poisson distribution. For each query, we randomly sample an

arriving time and assign an input image for it. The query rate fluctuates between 1 Req/s to 700 Req/s, and the duration is 10 minutes [70]. 2) The real-world trace. We adopt a production trace from Microsoft Azure Functions (MAF) [90, 48]. The query rate fluctuates from 100 Req/s to 300 Req/s.

Node setup. We evaluate PRODIGY on Desktop/Server platforms: one is equipped with an NVIDIA GeForce RTX 4080 (12th Gen Intel(R) Core(TM) i9-12900K CPU with 16 cores and 64GB of memory) machine to serve the ViT-Base and DeiT models, and the other is equipped with two NVIDIA Tesla V100 GPUs to serve the ViT-Large model (16-core Xeon Gold 5218R processor with 121.8 GB of memory). The V100 platform installs CUDA-11.8, and the RTX 4080 platform installs CUDA-12.0.

Hyper-parameters. PRODIGY fine-tunes models with 30 epochs. The available merging numbers are from 0 to 25, plus the additional set $\{30, 40, 50, 100\}$. We constrain Algorithm 4 to fine-tune 6 models. The value of r_{mid} is 20.

Baseline. We compare PRODIGY with four baselines.

- Clipper [13]: A vanilla serving system. This system groups queries into batches and makes an inference with a ViT model. However, it lacks the elasticity to adjust effectively with varying query loads.

- OTAS [9]: A serving system that dynamically adjusts the token number to improve elasticity. It optimally allocates a token number for a batch based on the query load and resource conditions. However, it directly deploys a fine-tuned model for serving, which lacks robustness for different reduction settings.

- Maxprune: A system with a model fine-tuned with the maximum pruning ratio or merging number.

- Exhaustive: A strawman scheme that fine-tunes models with all ratios and numbers. It prepares a large model zoo for serving queries.

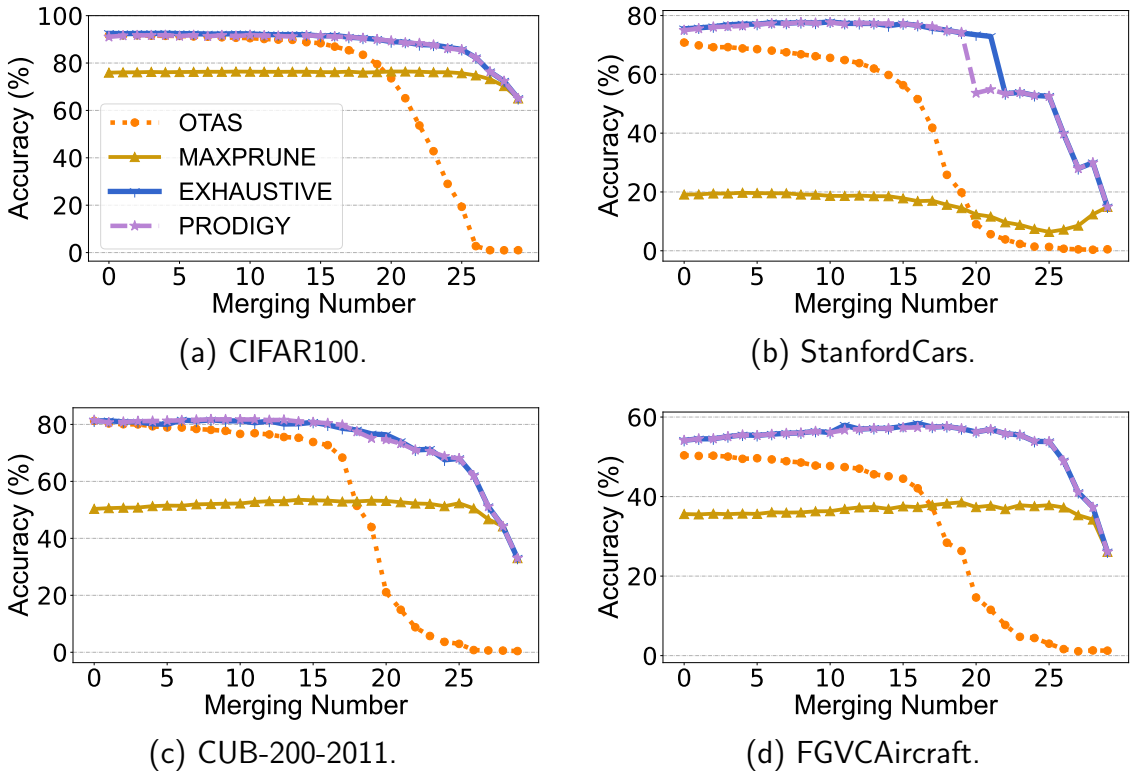
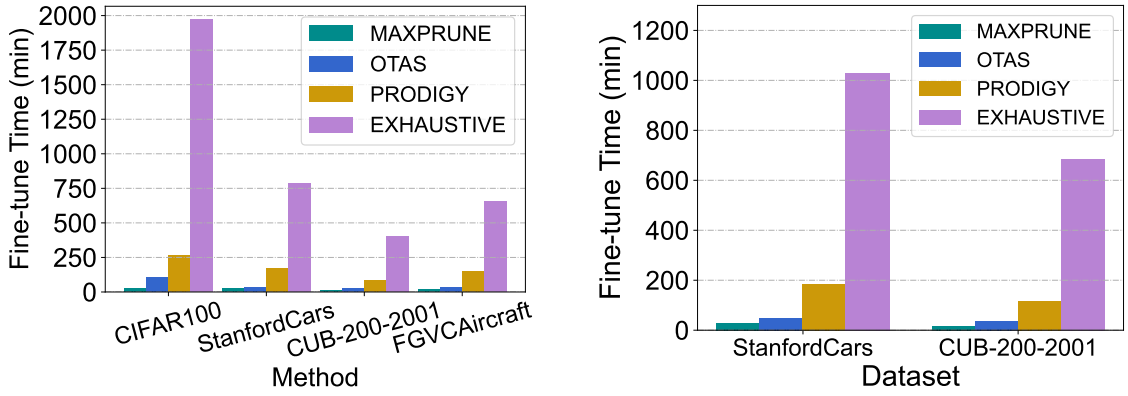


Figure 4.7: Accuracy values of a ViT-Base model during the inference stage with different token merging numbers.

Metrics. We validate the performance with the following metrics. 1) *Accuracy*, which reveals the generalization ability across various pruning ratios and merging numbers; 2) *warm-up cost*, which reveals the computational time costs and storage costs for the token-adaptive warm-up; 3) *throughput*, which is defined as successfully served query number per minute. The system successfully serves a query by returning an accurate result within the latency constraint; 4) *latency*, which includes queuing latency and inference latency, is an important matrix for evaluating the system's responsiveness.

4.6.2 Main Results

Accuracy during inference: Token-reduction warm-up increases the accuracy by up to 27%. The accuracy of token merging across different merging num-



(a) ViT-Base. (b) ViT-Large.

Figure 4.8: Fine-tuning costs of different token merging methods.

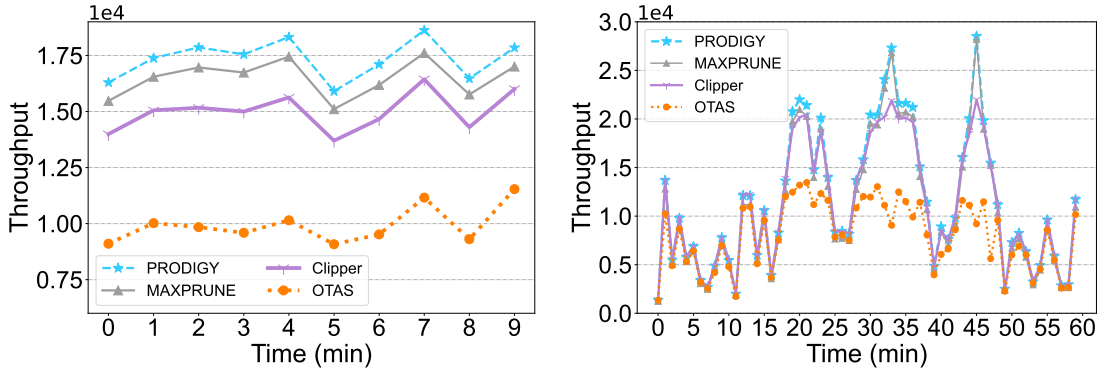
bers during inference is presented in Figure 4.7. Clearly, the token-reduction warm-up module significantly enhances the accuracy of token merging, ensuring the robustness of the serving system. For example, as the merging number increases, Exhaustive and PRODIGY retain high accuracy values across a wide range of merging numbers during inference. In Figure 4.7(a), the accuracy value remains near 90% when the merging number increases from 0 to 20 on the CIFAR-100 dataset, indicating that token reduction can accelerate inference with minimal accuracy loss. Their performance is significantly better than OTAS when evaluated with a large merging number, as they learn the token reduction pattern during the fine-tuning process beforehand. Token-reduction warm-up enables the serving system to provide accurate results for more requests and accelerate inference with diverse token merging numbers. On the contrary, OTAS and Maxprune cannot achieve a high level of robustness. The model of OTAS struggles to handle queries with higher token merging numbers, and its accuracy drops drastically when the merging number exceeds 15, thereby failing to deliver high-quality service to users. Compared to OTAS, PRODIGY can improve the average accuracy by 27% and generalize well to different token merging numbers. The model of Maxprune fails to achieve high accuracy for queries with small merging numbers due to excessive information loss during the fine-tuning stage, rendering it incapable

Dataset	Ours	Ours + Ensemble
CIFAR100	88.29%	88.68%
StanfordCars	65.53%	70.76%
CUB-200-2011	74.37%	74.07%
FGVCAircraft	53.66%	54.16%

Table 4.2: The accuracy of model ensemble.

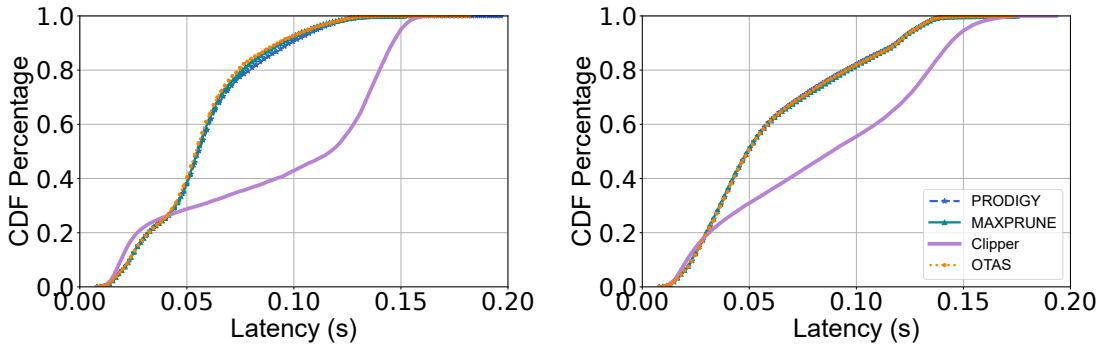
of processing the detailed information present in the original data. The above results demonstrate that our system can execute inference using a range of token merging numbers while still delivering precise outcomes for users, thereby fully leveraging the benefits of token merging. The average accuracy of the model ensemble for token merging is presented in Table 4.2. The accuracy can be further improved through the integration of diverse models, leveraging their collective strengths.

Warm-up cost: Prodigy reduces computational time costs and storage costs by 4×**.** Although Exhaustive achieves good performance with a strong model zoo, it has high computational costs and storage burdens to fine-tune and store models with all merging numbers. As depicted in Figure 4.8(a), the Exhaustive approach requires considerable computational time for fine-tuning. In contrast, PRODIGY cuts down the fine-tuning expenses by a minimum of 4×, maintaining a cost marginally above that of the two lightweight strategies. PRODIGY only selects a small set of merging numbers for fine-tuning, such as {20, 24, 30, 40, 50, 100}, and the fine-tuned models are representative and can serve requests with different merging numbers. The Exhaustive system prepares models for all merging numbers, consuming about 10GB of storage to store the parameters for each task. This method may result in significant I/O delays or cause memory overflows when deploying a serving system on a resource-constrained device. In contrast, PRODIGY only requires about 2GB of storage to store the model zoo. The evaluation results indicate that our system can deliver optimal service quality with minimal resource overheads, making it suitable for a variety of execution environments.



(a) Synthetic trace. (b) Azure trace.

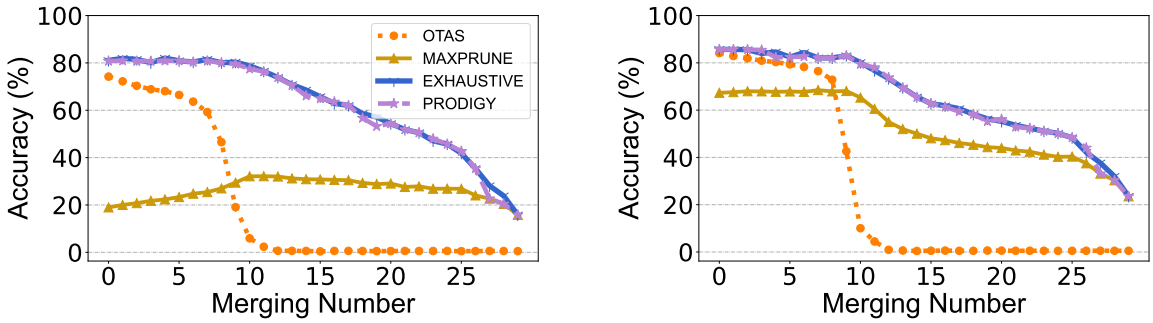
Figure 4.9: The throughput of the serving system.



(a) Synthetic trace. (b) Azure trace.

Figure 4.10: Request latency of the serving system.

Throughput and latency: Prodigy improves the serving throughput by up to 36% and reduces the latency by 62%. We deploy a ViT-Base model with token merging for serving queries sampled from the CIFAR100 dataset. The latency constraint is set as 0.1 seconds. The system successfully serves a query if it can return an accurate result within this latency constraint. The throughput for successfully served queries across different systems is presented in Figure 4.9(a) and Figure 4.9(b). Prodigy serves a higher number of requests under various query loads, improving throughput by up to 36% compared to OTAS. In the Azure trace, Prodigy improves the throughput from 4% to 23%. The Maxprune system also performs well because the query load is relatively flat in this setting, and an appropriate token merging



(a) StanfordCars. (b) CUB-200-2011.

Figure 4.11: Accuracy values on a ViT-Large model.

number is always selected for serving. The success of PRODIGY can be attributed to two key factors: (1) elasticity, allowing it to dynamically adjust the token reduction settings to accelerate inference, and (2) robustness, maintaining high accuracy even with high pruning ratios and merging numbers. The CDF plot of latency is presented in Figure 4.10. PRODIGY reduces the latency by 62% compared to Clipper. More than 80% requests can be processed within 0.1 seconds due to the significant acceleration in inference provided by token merging. In contrast, Clipper processes only about 40% and 60% of queries within the latency constraint in two traces, failing to meet the user's service latency objectives. These results illustrate the superiority of PRODIGY in enhancing serving quality.

4.6.3 Other Evaluation

Results on ViT-Large. We present accuracy values on the ViT-Large model for token merging in Figure 4.11. Exhaustive and PRODIGY still have the highest accuracy in this setting, providing a desirable serving experience for users. OTAS cannot generalize to large merging numbers, with its accuracy declining significantly once the merging number exceeds 10. Maxprune cannot ensure high accuracy when evaluating with a small merging number, as relying on a single model compromises robustness across different merging numbers. The merging numbers of PRODIGY during the fine-

tuning phase are $\{10, 14, 19, 24, 30, 100\}$. Figure 4.8(b) shows that the computational costs can be reduced by 5×.

Results on Token Pruning. We also extend our idea to token pruning with DynamicViT [68], and the available pruning ratios are set as 0 to 0.7. In this setting, the performance of PRODIGY is still comparable to Maxprune and maintains high accuracy across different pruning ratios. For example, the accuracy on the CIFAR100 dataset is larger than 80% when the system prunes up to 50% of the tokens. OTAS experiences a significant accuracy degradation as the pruning ratio rises, with the value dropping below 10% in some cases, thereby compromising the system's robustness. PRODIGY selects the pruning ratios of {0.2,0.4,0.7} for the fine-tuning stage. It achieves high computational efficiency during fine-tuning, making it a lightweight solution for the serving system.

Results on DeiT. We apply our idea to another model, DeiT, with token merging on the StanfordCars dataset. The average accuracy of PRODIGY and Exhaustive are 45.24% and 50.04% respectively, while the computational time of PRODIGY is approximately 5 times smaller than Exhaustive. The average accuracy of OTAS and Maxprune is less than 30%, which cannot provide high robustness.

4.7 Chapter Summary

In this chapter, we present PRODIGY, an innovative elastic serving system with token reduction based on token-reduction warm-up. PRODIGY allows Transformer-based application developers to fine-tune a limited set of models while achieving robust generalization ability for arbitrary token reduction ratios during inference. Experimental results show that PRODIGY improves the average accuracy by 27%. The reason for the performance improvement is that our method can proactively learn the patterns of token reduction and adjust to incomplete inputs.

Chapter 5

Fast Multimodal Edge Inference via Selective Feature Distillation

Inferring user status at the edge is essential for delivering personalized services, such as detecting emotional states. However, deploying large-scale models directly on user devices is impractical due to substantial computational overhead and the scarcity of labeled data. Conversely, uploading raw data to the cloud (as implemented in chapter 3 and chapter 4) for processing raises significant privacy concerns and incurs prohibitive communication costs. To address this challenge, we propose a privacy-preserving multimodal inference framework that leverages large-scale public data while safeguarding sensitive information and optimizing computational efficiency. Specifically, we first train a teacher model in the cloud using publicly available data. Through a feature distillation process, the knowledge from this teacher model is transferred to a lightweight encoder deployed at the user end. This transfer is tailored to the user's data, ensuring that only relevant knowledge is distilled. To accommodate varying communication constraints, we introduce a feature compression mechanism that significantly reduces communication overhead without compromising inference accuracy. Extensive experiments on emotion recognition tasks demonstrate that the proposed

framework effectively balances privacy preservation, resource efficiency, and inference accuracy, facilitating seamless collaboration between cloud and edge devices.

5.1 Introduction

With the emergence of a large number of edge devices and the development of artificial intelligence technologies in recent years [73, 59], providing intelligent services to users at the edge has become a critical need. These applications often require the use of deep learning models to process multimodal data generated by edge devices, such as images, text, and other types of data [5]. Although the richness of data greatly enhances the user experience, the increasing scale of deep learning models has made it increasingly difficult for edge devices to deploy full-scale large models, which severely hinders the development of edge AI applications [36].

A common approach to addressing this challenge involves deploying large models in the cloud and transmitting raw data generated by edge devices to the cloud for processing [49, 97]. While this method can achieve high inference accuracy, it introduces significant communication overhead in multimodal data scenarios, making it difficult to ensure real-time inference under limited bandwidth conditions. Furthermore, it poses substantial risks to user privacy. An alternative strategy is to deploy smaller models on edge devices to accommodate the resource constraints of user hardware. Although this method enables real-time inference for multimodal data at the edge, it often suffers from reduced accuracy. Some studies have proposed fine-tuning small models with data collected from edge devices [51]; however, the inherent knowledge limitations of these models continue to hinder improvements in their generalization capabilities. Moreover, fine-tuning small models on edge devices requires labeled data, which is typically unavailable, as users are generally unwilling to engage in manual data labeling.

Given the comprehensive insights provided by extensive public datasets, leveraging large models in the cloud, such as pre-trained models on large-scale datasets, holds significant promise. For resource-constrained devices, knowledge distillation [30] serves as an effective technique for transferring knowledge from a sophisticated teacher model to a smaller student model, thereby reducing training costs while maintaining satisfactory accuracy with a smaller dataset. Building on the advantages of such transfer learning, deploying lightweight models on resource-limited local devices becomes a practical approach to approximating the performance of larger, cloud-based models. However, a key challenge remains: lightweight models trained on public data often struggle to generalize effectively on edge devices due to the substantial distributional gap between public datasets and real-world edge data.

To address the aforementioned challenges, we propose a cloud-edge collaborative paradigm that leverages a selective distillation method. This approach enables efficient knowledge transfer from a large cloud-based model to a smaller model deployed on edge devices. At the same time, it ensures that the small model is adapted to the specific data characteristics of the edge environment. Specifically, features of the edge data are first extracted by inputting them into the small model, and these features are then transmitted from the edge device to the cloud. Subsequently, a subset of public data is selected for each specific device based on the similarity (e.g., feature distance) to the extracted edge features. This selected public data is then used to distill relevant knowledge from the cloud-based teacher model to the small student model, which is subsequently synchronized with the model on the local device. Since neither raw data nor labels are transmitted, the privacy of the user's multimodal data is preserved. To further mitigate privacy risks, we propose transmitting prototypes of the local data (computed as averages in the feature space) to construct the distillation dataset. Additionally, we introduce an adaptive feature compression method designed to accommodate dynamic communication channels, thereby optimizing the latency between the edge device and the cloud.

The contributions of the chapter are summarized as follows.

- We present a privacy-preserving inference framework designed to operate on lightweight edge devices, enabling the inference of user status from multimodal data. To achieve this, we introduce a *selective feature distillation* method that leverages features extracted from edge data, rather than raw data, to construct a compact distillation dataset from the larger pool of public data. This approach ensures that only relevant knowledge is transferred to a small model, which is subsequently synchronized with the edge device during the inference stage.

- To further enhance data privacy and reduce communication costs during feature synchronization, we design a prototype-based feature synchronization scheme that uploads only the clustering feature centroids to the server. Additionally, we propose a feature compression mechanism that further reduces the size of these feature prototypes, minimizing communication overhead while maintaining inference performance.

- Our established theories show that the generalization error of the trained model on the edge data is bounded.

- We conducted extensive experiments on a representative edge inference scenario, namely audio-visual emotion recognition, using two widely adopted datasets. The results demonstrate that our framework effectively delivers privacy-preserving, low-latency, and accurate edge intelligence applications.

The remainder of the chapter is organized as follows. Section 5.2 shows the system model for the proposed cloud-edge inference system. Section 5.3 presents the detailed design of the system. Section 5.5 provides experimental results to evaluate the performance of the proposed scheme. Finally, Section 5.6 concludes the chapter.

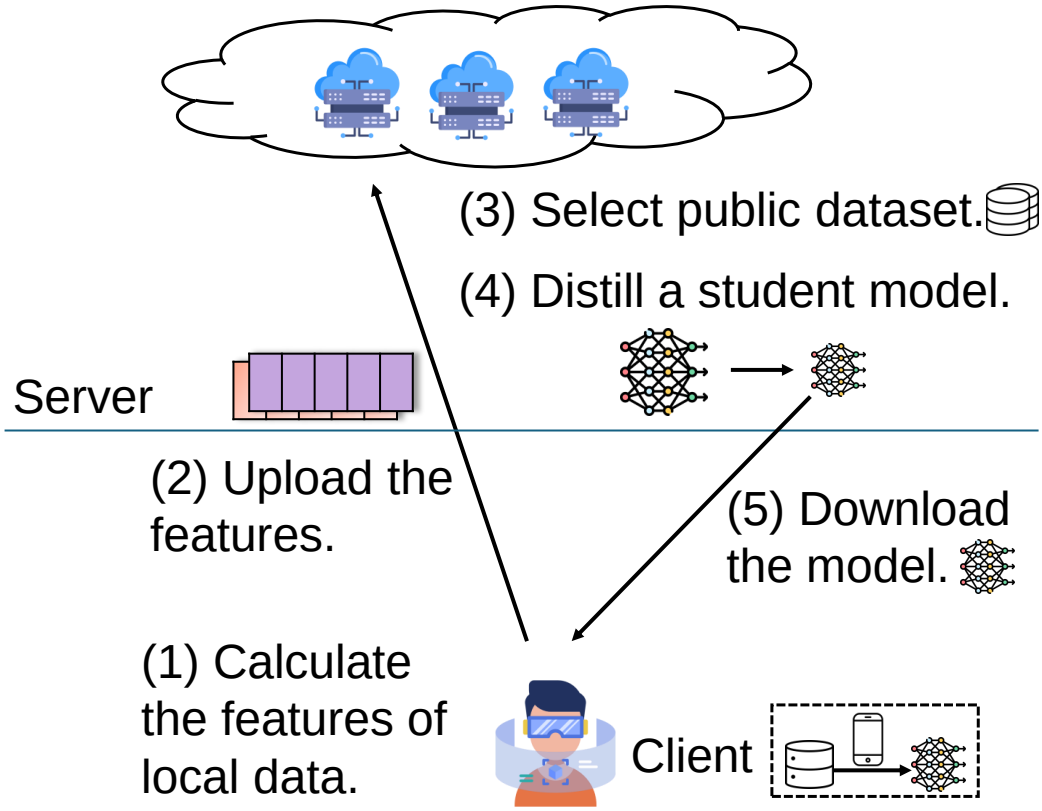


Figure 5.1: System model of knowledge distillation and local inference. We distill a meta-model to a student model with the selected dataset for personalization.

5.2 System Model

Modern edge environments increasingly require intelligent systems that can dynamically adapt to users' evolving contexts and behaviors. To address this need, our system focuses on inferring user status, which represents a user's dynamic contextual, emotional, and behavioral conditions based on multimodal data collected by edge devices. By accurately understanding user status, our approach enables more responsive and personalized services, enhancing the user experience across a wide range of applications. In particular, we emphasize emotional states as a primary example of user status. Emotional states encompass the affective conditions of a user, such as *happiness, sadness, anger, fear, surprise,* and *neutrality.* Recognizing these emotional states is crucial for applications like adaptive human-computer interaction,

where systems can tailor their responses based on the user's mood, such as adjusting the tone of virtual assistants or personalizing content delivery in educational tools. Emotional state detection is also vital for mental health monitoring, as continuous tracking of emotional fluctuations can help detect early signs of stress, anxiety, or depression, enabling timely interventions. Furthermore, emotional states influence user preferences, facilitating context-aware recommendations where systems suggest content (e.g., music, movies) that aligns with the user's current mood.

To effectively infer user status in edge environments, our system employs a collaborative cloud-edge paradigm optimized for efficient multimodal inference. As illustrated in Figure 5.1, the client device operates with limited resources, such as computing power and communication bandwidth, making it challenging to perform inference solely on the local device or entirely in the cloud. Local inference is computationally intensive, while transmitting raw data to the cloud incurs high bandwidth costs and poses privacy risks. To address these challenges, our system enables collaborative training and inference between the client and the server, leveraging private user data on the client side and public data in the cloud to enhance efficiency and accuracy.

Specifically, we employ lightweight encoders at the user side to extract features from each modality, which are optimized through a secure training interaction with the cloud. First, the client transmits only the extracted feature f_c computed by these encoders $\theta_c.enc$ from a local data sample x, where $f_c = g_{c.enc}(x_c; \theta_{c.enc})$. Since raw data is neither transmitted to the server nor shared with others, the risk of privacy leakage is significantly mitigated. Subsequently, the server computes the prediction probability by processing the received feature f_c through the prediction head of a large multimodal model (i.e., $\theta_p.head$) and aims to minimize the prediction error for

clients. The objective function is formulated as follows:

$$\min_{\theta_c} \frac{1}{|D_c|} \sum_{(x_c,y_c)\in D_c} \mathcal{L}_{CE}(g(\theta_p.head, f_c), y_c) \tag{5.1}$$
$$= \frac{1}{|D_c|} \sum_{(x_c,y_c)\in D_c} \mathcal{L}_{CE}(g(\theta_p.head, g_{c.enc}(x_c; \theta_{c.enc})), y_c),$$

where D_c represents the local dataset, and $g(\theta_p.head, f_c)$ denotes the predicted probability.

At the cloud side, the server has access to abundant public datasets from the cloud community, enabling the training of large-scale multimodal models with strong classification capabilities. These models can then be leveraged to optimize local encoders. For the selective distillation framework to function effectively, the public datasets should be closely aligned with the local task, such as emotional state detection, and should include multimodal data, such as audio-visual information, to ensure compatibility. Large-scale, diverse, and well-labeled datasets are crucial for generalization and for training robust teacher models that guide the distillation process. These datasets can be sourced from publicly available repositories or domain-specific annotations.

In this chapter, we propose leveraging these pretrained models to optimize the local encoders $\theta_c.enc$ through feature distillation, as formulated in Eq. (5.1). The distribution of local datasets across clients may differ from that of public datasets due to variations in personalized user status. To enhance the efficiency of distillation, we actively select subsets of public data that closely resemble the user's local data, enabling the selective transfer of knowledge from the larger encoder to the digital twin of the user's smaller local encoder. The key symbols used in this chapter are summarized in Table 5.1.

Table 5.1: A summary of main mathematical symbols

Symbol	Definition
θ_c	The parameters of the local encoder in client
θ_p	The model parameters in the server
(x, y)	The data sample and its corresponding label
β_p	The prediction probability for sample x_p
$g(\cdot)$	The function obtaining the prediction probability
$f(\cdot)$	The function outputting the intermediate feature
D_p	The public dataset in the server
D_s	The selected dataset from the public dataset
$\mathcal{L}_{CE}(\cdot)$	The cross-entropy (CE) loss function
$\mathcal{L}_{MSE}(\cdot)$	The mean of square error (MSE) function

5.3 Design for Fast Multimodal Inference

In this section, we introduce the design of the proposed inference framework, which incrementally addresses multimodal inference under resource-constrained conditions while ensuring the protection of raw data. Given the limited capacity of the local encoder, it is crucial to selectively extract relevant knowledge from the pre-trained model while preventing the transfer of irrelevant information. To achieve this, our method identifies a subset from the public dataset whose distribution closely aligns with that of the local dataset. Moreover, directly uploading data features poses a risk of privacy leakage. Therefore, we employ a feature clustering mechanism, utilizing prototypes of local datasets instead of raw data features for subset selection. Additionally, to account for the heterogeneity in users' bandwidth resources, we propose a dynamic compression scheme that adapts to varying communication conditions.

5.3.1 Overview

Figure 5.2 illustrates the inference framework designed for fast multimodal edge inference, as detailed in section 5.3. The process begins on the client side with *Feature Extraction*, where lightweight encoders are used to process multimodal data. To

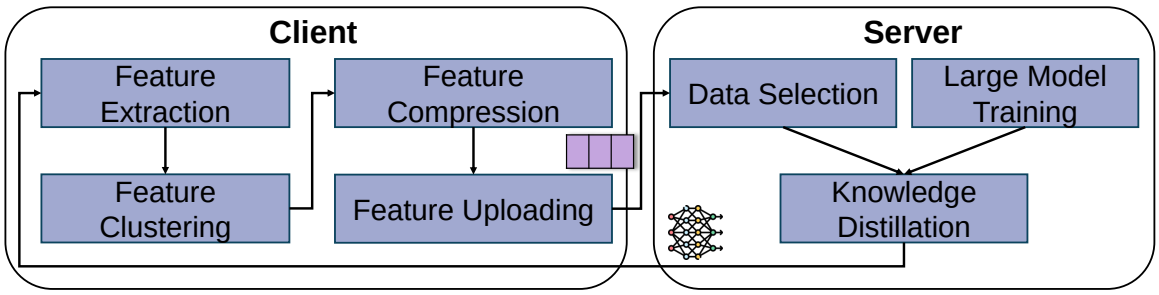


Figure 5.2: Inference framework: The client is responsible for extracting local features, while the server handles data selection and knowledge distillation.

enhance privacy, the extracted features undergo *Feature Clustering* (discussed in section 5.3.4), which groups features into prototypes, reducing the risk of privacy leakage. These clustered features are then compressed through a *Feature Compression* module (covered in section 5.3.5), aimed at minimizing communication overhead and adapting to varying bandwidth conditions before being uploaded to the server.

On the server side, the uploaded features are utilized in the *Data Selection* module (explained in section 5.3.3), where a subset of the public dataset is chosen based on similarity to the client's local data. This selection ensures that only relevant knowledge is transferred, optimizing the personalization process. Subsequently, *Knowledge Distillation* (also in section 5.3.3) is performed using the selected dataset, transferring knowledge from a large pre-trained model, which is trained as described in *Large Model Training* (section 5.3.2), to a smaller, efficient student model. This distilled model is then synchronized with the client device, enabling accurate, resource-efficient inference while preserving user privacy.

The framework thus highlights a coordinated approach to edge-cloud inference by integrating privacy-preserving feature extraction and compression on the edge device with powerful, large-scale model capabilities in the cloud. By minimizing data exchange and carefully selecting only relevant portions of a public dataset for knowledge distillation, this method effectively balances performance, resource usage, and confidentiality requirements.

5.3.2 Accurate Large Model over Public Data

Training a Large Model. First, the cloud server trains a large model using the abundant public dataset, which is assumed to achieve superior performance on the specified multimodal classification task, as requested by the user, while also exhibiting strong generalization capabilities across other tasks. We use the p and c to denote the server and client side respectively, for model parameters and datasets. Let D_p represent the public dataset, where each sample x_p consists of two modalities, such as an audio modality $x_{p,a}$ and a video (image) modality $x_{p,v}$. The parameters of the large pre-trained model in the cloud are denoted as θ_p, which include two encoders $\theta_p.enc$ and a fusing module for classification, $\theta_p.head$. Consequently, the prediction probability for a sample x_p is given by:

$$\beta_p = g(x_p, \theta_p). \tag{5.2}$$

Given the label y_p of the sample, the loss function is computed using the cross-entropy function, denoted as $\mathcal{L}_{CE}(\beta_p, y_p)$. Thus, the objective function is formulated as:

$$\arg\min_{\theta_p} \frac{1}{|D_p|} \sum_{(x_p, y_p) \in D_p} \mathcal{L}_{CE}(\beta_p, y_p). \tag{5.3}$$

Distilling a Small Model Using the Entire Public Dataset. In edge computing, users frequently rely on lightweight end devices for tasks such as emotion recognition. However, these devices possess limited computational resources, rendering them unsuitable for the direct deployment of large multimodal models. Deploying such models on end devices introduces several challenges, including high latency, excessive computational overhead, increased energy consumption, and elevated device temperatures. These issues are further exacerbated by the growing demand for edge applications. To mitigate these challenges, knowledge distillation techniques are employed. In this approach, a large teacher model is first trained, and its knowledge is

subsequently transferred to a smaller student model. This model efficiently analyzes user data while adhering to resource constraints on edge devices.

The primary computational burden in existing neural networks typically stems from the encoder. To address this challenge, we distill a compact student encoder from a strong encoder using a public dataset. This lightweight student model is specifically designed to extract features from client-specific data. The knowledge distillation process is conducted independently for each modality. Let the parameters of a student's model for a given client be denoted as θ_c, consisting of a small encoder $\theta_{c.enc}$ and a classifier head $\theta_{c.head}$. For an input x_p from the public dataset D_p, the feature representation generated by the student model is expressed as: $f_c = g_{c.enc}(x_p; \theta_{c.enc})$. Similarly, the corresponding feature representation derived from the teacher model is given by: $f_p = g_{p.enc}(x_p; \theta_{p.enc})$. The distillation objective, formulated in Eq. (5.4), aims to minimize the mean squared error (MSE) loss, thereby optimizing the student model parameters.

$$\arg\min_{\theta_c} \frac{1}{|D_p|} \sum_{(x_p, y_p) \in D_p} \mathcal{L}_{MSE}(f_p, f_c). \tag{5.4}$$

The training pipeline is outlined in Algorithm 6. At this stage, a high-performance model is deployed on the server, while a student model with two encoders has been distilled using the entire public dataset. Each encoder is specifically designed to extract features for a distinct modality.

5.3.3 Personalized model via Selective Distillation

Although a large volume of data is stored on the server, local clients typically possess datasets with diverse and heterogeneous distributions. As a result, a small model distilled solely on the public dataset may exhibit suboptimal performance across different clients due to domain drift. To mitigate this issue, we propose the development of personalized models through selective distillation. Specifically, our approach aims

Algorithm 6: Distillation on public data.

Input: Public dataset D_p;
Output: Teacher model θ_p, θ_c;
1 **for** *each epoch* **do**
2 Sample a batch B_p from public dataset D_p;
3 Training the meta-model according to the Eq.(5.3);
4 Distill a student model according to the Eq.(5.4);
5 **return** θ_p, θ_c;

to identify and select a representative subset of public data that closely aligns with the client's data distribution. This enables the distillation of a compact model with enhanced accuracy on the client's dataset.

The primary challenges in this process are twofold: (1) establishing an effective metric to quantify the similarity between local and public samples, and (2) preserving user privacy while leveraging the distinctive characteristics of the data. To address these challenges, we compute feature representations on local clients and upload these representations instead of raw data to the server, thereby ensuring data privacy. Subsequently, sample selection is performed based on the similarity of prediction probabilities between local and public samples, enabling the construction of personalized and privacy-preserving distilled models.

The pipeline is illustrated in Figure 5.3, and the corresponding pseudocode is provided in Algorithm 7. In each outer epoch, a subset of public data is selected for distillation. The distillation process itself consists of multiple inner epochs. In Line 2, for each outer epoch, we first compute the features on the client side, denoted as $f_c = g_c(x_c; \theta_c)$. These features from both modalities are then uploaded to the server for further analysis (Line 3). Once on the server, the features are processed by the fusion model to generate the prediction probabilities, represented as θ_c (Line 4). Additionally, the prediction probabilities of the samples from the public dataset, denoted as θ_p, are computed using the teacher model, which has superior capability in producing accurate results (Line 5).

After obtaining the prediction probabilities for both the local and public datasets, we identify and select the most similar public data samples for model distillation. The detailed pseudocode for this selection process is outlined in Lines 14 $\sim$ 21 of Algorithm 7. The fundamental objective of the data selection algorithm is to find public dataset samples that closely resemble the client samples and belong to the same class. The similarity between samples is quantified using the cosine similarity between the prediction probabilities θ_c and θ_p, as defined by:

$$\text{sim} = \cos(\theta_c, \theta_p). \tag{5.5}$$

For each sample in the client dataset, we select the most similar, previously unchosen samples from the public dataset (Lines 16 $\sim$ 20).

Compared to feature-based matching, utilizing prediction probabilities offers several distinct advantages. Prediction probabilities directly capture the likelihood of class membership, effectively identifying samples belonging to the same class and mitigating the effects of non-IID data across clients. This approach ensures that the distilled model is better aligned with each client's data distribution. In contrast, feature-based matching is more susceptible to noise and variability, often resulting in less accurate matches. Moreover, feature-based methods tend to be computationally intensive, thereby reducing the overall efficiency of the distillation process.

After sampling a representative dataset, we can execute the knowledge distillation on the server. We train the student model according to the objective of Eq.(5.4), and the model is initialized with the previous model that is distilled on the whole public dataset. For each distillation epoch, we first sample a batch from the selected dataset, and perform gradient descent for the objective of Eq.(5.4). By executing these steps for a few times, we can distill a personalized small model that can have a good performance for local data (Lines 7 $\sim$ 10).

Minimizing Prediction Error through Privacy-Preserving Selective Distillation: In our

proposed framework, we aim to reconcile the trade-off between minimizing prediction errors and preserving client data privacy through an innovative approach to selective feature distillation. In the following, we will analyze the impact of selective distillation on minimizing prediction errors across client data.

The objective of the system is to minimize the prediction error for each client, as shown in Eq.(5.1). The cross-entropy loss can be expressed as:

$$\min_{\theta_c} \frac{1}{|D_c|} \sum_{x_c, y_c \in D_c} y_c^T \log g\left(\theta_{p\text{·}head}, f_c\left(x_c, \theta_c\right)\right). \tag{5.6}$$

However, since labels are not available on clients, we leverage feature knowledge distillation to optimize θc as shown in Eq.(5.4). Ideally, θ_c would be obtained by distilling knowledge from D_c, but this is not feasible due to privacy concerns. Instead, we use a subset of public data (p') for distillation. The parameters of the client encoder, obtained from Eq.(5.4), can be denoted as:

$$\theta_c^{p'} = arg\min_{\theta_c} \mathop{\mathbb{E}}_{x_{p'} \sim \mathrm{P}_{p'}(\mathrm{X})} \left[f_p\left(x_{p'}\right) - f_c\left(x_{p'}, \theta_c\right)\right]^2, \tag{5.7}$$

where $\mathrm{P}_{p'}(\mathrm{X})$ represents the distribution of the subset selected from the public dataset.

This approach introduces a gap in the prediction loss on client data, defined as:

$$\begin{aligned} G = & \mathop{\mathbb{E}}_{x_c, y_c \sim \mathrm{P}_c(\mathrm{X},\mathrm{Y})} y_c^T \log g\left(\theta_{p\text{·}head}, f_c\left(x_c, \theta_c\right)\right) \\ & - y_c^T \log g\left(\theta_{p\text{·}head}, f_c\left(x_c, \theta_c^{p'}\right)\right) \\ = & \mathop{\mathbb{E}}_{x_c, y_c \sim \mathrm{P}_c(\mathrm{X},\mathrm{Y})} y_c^T \log \frac{g\left(\theta_{p\text{·}head}, f_c\left(x_c, \theta_c\right)\right)}{g\left(\theta_{p\text{·}head}, f_c\left(x_c, \theta_c^{p'}\right)\right)}, \end{aligned} \tag{5.8}$$

where $\mathrm{P}_c(\mathrm{X}, \mathrm{Y})$ denotes the data and label distribution of the client. This gap primarily arises from the divergence in output probabilities caused by the distributional differences between $\mathrm{P}_c(\mathrm{X})$ and $\mathrm{P}_{p'}(\mathrm{X})$. Consequently, selecting public samples with distributions more similar to the local client data can mitigate this gap. This insight

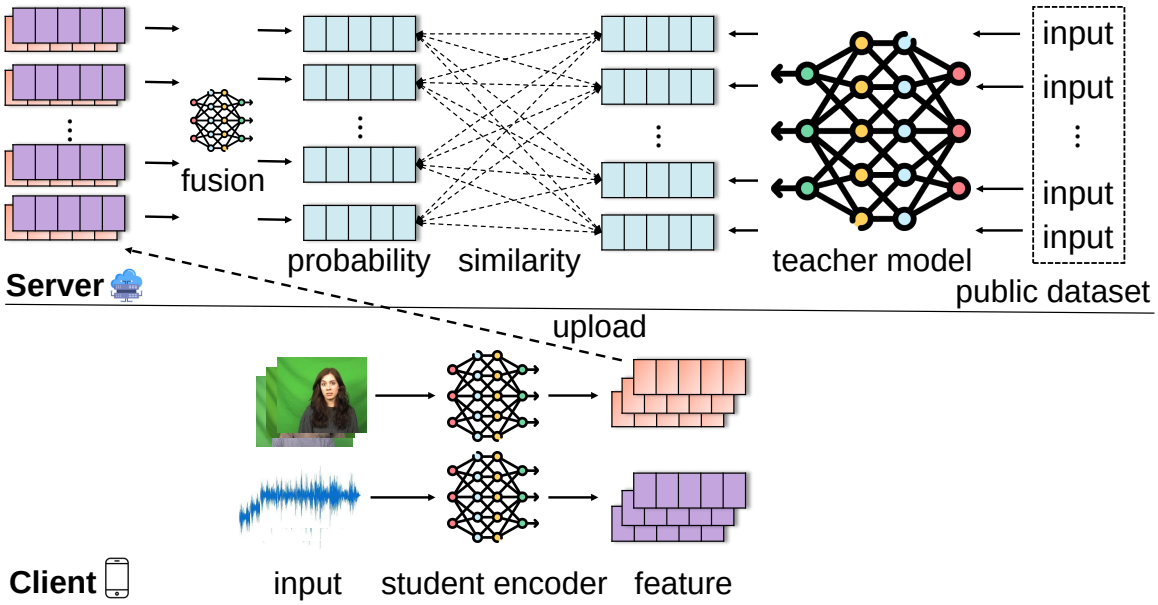


Figure 5.3: Data Selection Process. The local features are uploaded to the server and the public dataset is selected according to the probability similarity.

also validates our strategy of using prediction probabilities for sample selection.

5.3.4 Privacy Preservation via Output Clustering

Although the uploaded features can effectively select the corresponding samples, there remains a significant risk of exposing the user's raw sample data through potential reconstruction from the features. To address this privacy concern, we enhance the framework by incorporating privacy-preserving measures. Specifically, we cluster the output probabilities and upload only the aggregated group characteristics, thereby reducing the risk of data leakage.

On the client side, the local device extracts features f_c and computes the prediction probability β_c. To enhance privacy, we employ the k-means algorithm to cluster the predicted probabilities into prototypes. The k-means algorithm initially selects k centroids, calculates the distances between samples and centroids, and assigns pseudolabels to the samples accordingly. It then iteratively updates the centroids and

Algorithm 7: Distillation on selected data.

Input: Public dataset D_p; client dataset D_c; student model θ_c; teacher model θ_p;
Output: Student model for a client θ_c;

for *each outer epoch* **do**
  Get client feature $F_c = g_c(x_c; \theta_c), \forall x_c \in D_c$;
  Upload client features to the cloud;
  Calculate the probability of client's data $\beta_c = g_{p.head}(f_c, \theta_{p.head}), \forall f_c \in F_c$;
  Calculate the probability of public data $\beta_p = g_p(x_p, \theta), \forall x_p \in D_p$;
  $D_s = \text{SelectData}(\beta_c, \beta_p, D_p)$;
  for *each inner epoch* **do**
    Sample a batch B_s from selected dataset D_s;
    Training the student model according to Eq.(5.4);
  Download the student model θ_c;
return θ_c;
Function `SelectData`(β_c, β_p, D_p)**:**
  Compute similarity matrix $S_{c,p}$ between β_c and β_p;
  Initialize selected dataset $D_s = \{\}$;
  for *each row* S_c *in* $S_{c,p}$ **do**
    $D_n \leftarrow$ Select n samples from D_p (excluding those in D_s) with the highest values in S_c;
    $D_s \leftarrow D_s \cup D_n$;
  return D_s;

reassigns the samples until convergence is achieved. Subsequently, the prototypes are uploaded to the server, where samples from the public dataset are selected based on probability similarity. Following established methodologies, we then perform knowledge distillation to train a student model.

5.3.5 Adaptive Feature Compress for Dynamic Channel

Although we distill a lightweight model for client-side inference to reduce computational overhead, the extracted features must still be uploaded to the server for further analysis and storage. This process incurs significant communication costs and can lead to high latency, particularly under constrained bandwidth conditions. To mitigate these challenges, we propose a feature compression module that adaptively

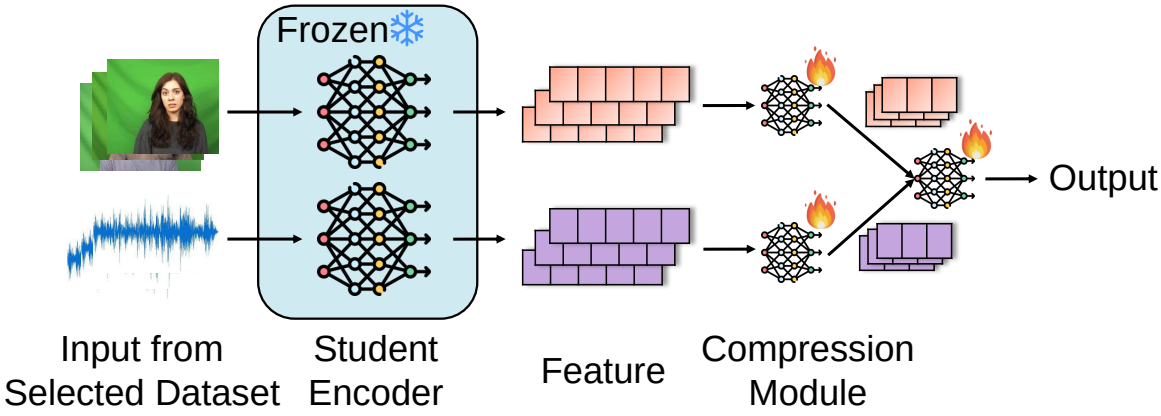


Figure 5.4: Feature compression. We design some modules to compress the features to reduce the communication burden.

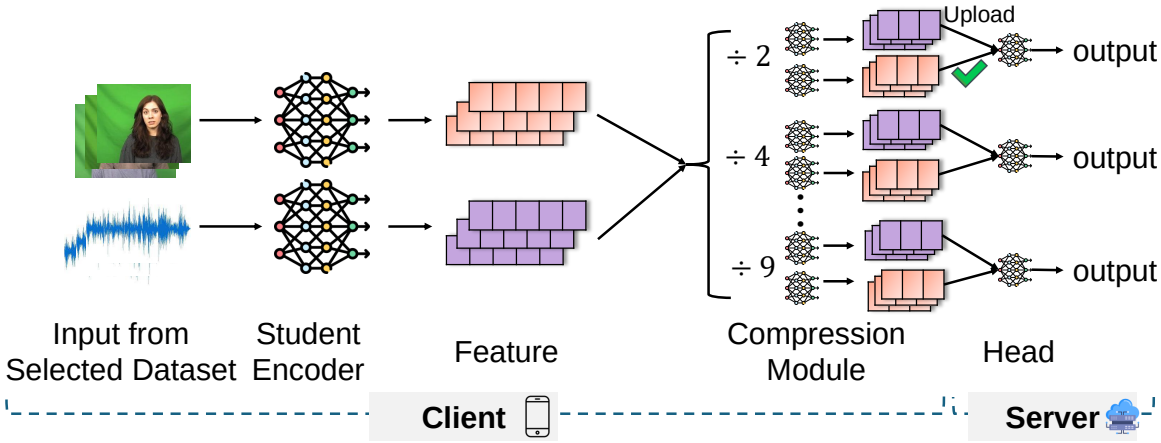


Figure 5.5: Inference with different compression modules.

compresses features based on network conditions. For instance, when the original feature dimension is 512, we can reduce the transmitted data by compressing the feature dimension to 256. We denote the compression ratio as γ and design a dedicated feature compression module, $\theta_{c,\gamma}$, along with a corresponding classification head, $\theta_{c.head,\gamma}$, to facilitate efficient transmission and classification.

As illustrated in Figure 5.4, we train the compression modules on the selected dataset while keeping the pre-trained student encoders frozen. A dedicated compression module and classification head are introduced for each modality. Given a selected data sample x_s, the corresponding feature representation is extracted as $f_s = g_c(x_s; \theta_c)$.

Algorithm 8: Adaptive feature compression training.

Input: Public dataset D_p; client dataset D_c; student model θ_c; teacher model θ_p;
Output: Teacher model θ_p;

```
Freeze the encoder parameter θ_c of student model;
D_s = SELECTDATA(β_c, β_p, D_p);
for each epoch do
    Calculate the feature of the selected dataset with the student model
      F_s = g_c(x_s; θ_c), ∀x_s ∈ D_s;
    Calculate the compression feature F_{s,γ} = g_c(f_s; θ_{c,γ});
    Upload the compression feature to the cloud;
    Calculate the probability with the head β_{s,γ} = g_c(f_{s,γ}; θ_{c.head,γ});
    Train the compression module and head with the classification loss;
```

The extracted feature is then compressed according to a predefined compression ratio γ, yielding a lower-dimensional representation $f_{s,\gamma} = g_c(f_s; \theta_{c,\gamma})$. This compressed feature is subsequently processed by the classification head to obtain the prediction probability, expressed as $\beta_{s,\gamma} = g_c(f_{s,\gamma}; \theta_{c.head,\gamma})$. The parameters of the compression module and classification head are optimized using the classification loss function:

$$\frac{1}{|D_s|} \sum_{(x_s, y_s) \in D_s} \mathcal{L}_{CE}(\beta_{s,\gamma}, y_p), \tag{5.9}$$

where $\mathcal{L}_{CE}$ represents the cross-entropy loss. The detailed training procedure is outlined in Algorithm 8.

As illustrated in Figure 5.5, the inference process begins with feature extraction using the student encoder. Based on the prevailing network conditions, an appropriate compression module is selected to generate compressed features at a designated compression ratio. The compressed features are then uploaded to the server, where the final prediction is obtained through the classification head.

5.4 Theoretical Analysis

Considering that the neural network can be approximated by the linear model when its width is infinite, this chapter mainly analyzes the distillation performance of the proposed method in terms of the linear model. In fact, the argument has also been discussed by [37] in detail. We tend to first establish the theory for the generalization performance over the full public dataset and then analyze the performance of the selective distillation which is based on selected data samples. At last, we theoretically show that the selected distillation outperforms the full distillation in terms of generation performance over the client data.

Formally, we establish the theory for the error bound of distillation on model $g \in \mathcal{H}$ which is defined by $g(x) = \mathbb{1}\{\theta^T x \geq 0\}$ with parameter θ on any data sample $x \in \mathbb{R}^d$, where $\mathcal{H}$ is the hypothesis space. We denote $\mathcal{D}_p, \mathcal{D}_c, \mathcal{D}_s$ as the corresponding ground-truth distribution of the public, client, and selected datasets D_p, D_c, D_s. Besides, for the simplicity of presentation, we simplify the $g_c(x;\theta_c)$ as g_c and $g_p(x;\theta_p)$ as g_p. Our theories utilize the following reverse cdf function:

$$p(w) = P_{x\sim\mathcal{D}_p}\Big[\cos^{-1}\big(\frac{|\theta_p^T x|}{\|\theta_p\| \cdot \|x\|}\big) \geq w\Big], \tag{5.10}$$

for any $w \in [0, \pi/2]$. We use $d_{\mathcal{H}\Delta\mathcal{H}}$ to measure the domain discrepancy between two distributions, which is commonly used in the field of transfer learning [20]. Besides, we use $\mathcal{L}_D(g)$ to represent the loss of model g on the data distribution D. Our theories are built on the following existing theorems.

Proposition 1 (Theorem 1, Ben et al. [1]). Considering the distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, for every $g \in \mathcal{H}$ and any $\sigma \in (0, 1)$, with probability at least $1 - \sigma$ (over the choice of the samples), there exists

$$\mathcal{L}_{\mathcal{D}_1}(h) \leq \mathcal{L}_{\mathcal{D}_2}(h) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_1, \mathcal{D}_2) + \lambda, \tag{5.11}$$

where $\lambda = \mathcal{L}_{\mathcal{D}_1}(g_*) + \mathcal{L}_{\mathcal{D}_2}(g_*)$ with $g_* = \arg\min_{g\in\mathcal{H}} \mathcal{L}_{\mathcal{D}_1}(g) + \mathcal{L}_{\mathcal{D}_2}(g)$.

Proposition 2 (Theorem 3, Mary et al. [64]). For any training set $D = \{x_i \in \mathbb{R}^d | i = 1, \ldots, n\}$ sampled from the distribution $\mathcal{D}$, if g_t is the teacher model and g_c is a linear model learned by distilling from g_t, the following statements hold:

$$\mathbb{E}_{D\sim\mathcal{D}}[R(g_c)] = 0 \tag{5.12}$$

when $n > d$, and

$$\mathbb{E}_{D\sim\mathcal{D}}[R(g_c)] \leq p(\beta) + p(\pi/2 - \beta)^n \tag{5.13}$$

when $n < d$ and $\beta \in [0, \pi/2]$. $R(g) = P_{x\sim\mathcal{D}}[g_c(x) \neq g_t(x)]$ represents the probability of the student model g_c predicting differently from the teacher model g_t.

Now, we have the following theory for the generalization bound of the g_c over the distribution $\mathcal{D}_c$ of client data.

Theorem 2. *By distilling from the teacher model g_p on the public dataset $D_p \in \mathbb{R}^{d\times n}$, the generalization error of student model g_c on the client data distribution $\mathcal{D}_c$ is bounded:*

$$\mathcal{L}_{\mathcal{D}_c}(g_c) \leq C + \mathcal{L}_{\mathcal{D}_p}(g_c) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_p, \mathcal{D}_c) + \lambda_p \tag{5.14}$$

when $n \geq d$, and

$$\begin{aligned}\mathcal{L}_{\mathcal{D}_c}(g_c) \leq\ & C + \mathcal{L}_{\mathcal{D}_p}(g_c) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_p, \mathcal{D}_c) \\ & + \lambda_p + p(\beta) + p(\pi/2 - \beta)^n\end{aligned} \tag{5.15}$$

when $n < d$, where $\beta \in [0, \pi/2]$, $C > 0$ denotes the loss of teacher model g_p over the public dataset $\mathcal{D}_p$, and $\lambda_p = \mathcal{L}_{\mathcal{D}_c}(g_*) + \mathcal{L}_{\mathcal{D}_p}(g_*)$ with $g_* = arg\ min_{g\in\mathcal{H}} \mathcal{L}_{\mathcal{D}_c}(g) + \mathcal{L}_{\mathcal{D}_p}(g)$.

Proof. Considering the teacher model has been pre-trained and fixed in our method,

we consider that the loss of teacher model g_p over the public dataset $\mathcal{D}_p$ as a constant C. Then, based on Proposition 2, the generalization error of the student model g_c over the public dataset is bound by

$$\mathcal{L}_{\mathcal{D}_p}(g_c) \leq C \tag{5.16}$$

when $n > d$ and

$$\mathcal{L}_{\mathcal{D}_p}(g_c) \leq C + p(\beta) + p(\pi/2 - \beta)^n \tag{5.17}$$

when $n < d$. Besides, based on Proposition 1, we establish the error bound of the model g_c learned from $\mathcal{D}_p$ over the client data distribution $\mathcal{D}_c$:

$$\mathcal{L}_{\mathcal{D}_c}(g_c) \leq \mathcal{L}_{\mathcal{D}_p}(g_c) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_p, \mathcal{D}_c) + \lambda_p, \tag{5.18}$$

where λ_p is defined in Theorem 1. Considering (5.16) and (5.17) with (5.18) together yields the theorem. □

Remark 1. *Theorem 1 indicates that the generalization error is reduced when increasing the size of the public dataset. But, the error will not always be reduced with the size of public data due to the limited capability of the student model. Specifically, when the number of samples grows large enough, the domain discrepancy $d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_p, \mathcal{D}_c)$ between the distillation dataset and the client dataset becomes dominant.*

Based on the above analysis, we can conclude that reducing the discrepancy is key when the number of samples is large enough but the capability of the student model is limited. To show this, we further derive the generalization error of the student model g_c distilled from the selected dataset $\mathcal{D}_s$ over the client data distribution $\mathcal{D}_c$. Similar to Theorem 2, we derive the following corollary for the selected distillation.

Corollary 1. *By distilling from the teacher model g_p on the selected dataset $D_s \in \mathbb{R}^{d \times n_s}$ with n_s samples, the generalization error of student model g_c on the client data*

distribution $\mathcal{D}_c$ is bounded:

$$\mathcal{L}_{\mathcal{D}_c}(g_c) \leq C' + \mathcal{L}_{\mathcal{D}_s}(g_c) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_s, \mathcal{D}_c) + \lambda_s, \tag{5.19}$$

when $n_s \geq d$, and

$$\begin{aligned} \mathcal{L}_{\mathcal{D}_c}(g_c) \leq C' + \mathcal{L}_{\mathcal{D}_s}(g_c) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_s, \mathcal{D}_c) \\ + \lambda_s + p(\beta) + p(\pi/2 - \beta)^{n_s} \end{aligned} \tag{5.20}$$

when $n_s < d$, where $\beta \in [0, \pi/2]$, $C' > 0$ denotes the loss of teacher model g_p over the selected public dataset $\mathcal{D}_s$, and $\lambda_s = \mathcal{L}_{\mathcal{D}_c}(g_*) + \mathcal{L}_{\mathcal{D}_s}(g_*)$ with $g_* = arg\ min_{g \in \mathcal{H}} \mathcal{L}_{\mathcal{D}_c}(g) + \mathcal{L}_{\mathcal{D}_s}(g)$.

Remark 2. *Assuming the teacher model is well-trained (i.e., $C \approx C'$), the key difference between Corollary 1 and Theorem 2 lies in the domain discrepancy, $d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_s, \mathcal{D}c)$ vs. $d\mathcal{H}\Delta\mathcal{H}(\mathcal{D}_p, \mathcal{D}_c)$. Obviously, the selected dataset $\mathcal{D}_s$ based on the local information of the client is closer to $\mathcal{D}_c$) than $\mathcal{D}_p$, which shows that the error bound of the model learned from the selected dataset is generally smaller than learning from the full dataset. One concern is that the number of selected samples is not sufficient, i.e., $n_s \leq d$, while the total samples $n > d$. In this case, the extra term $p(\beta) + p(\pi/2 - \beta)_s^n$ may increase the error of the model, which performs worse than the model distilled from the full public dataset. Therefore, we should select data as close to the client data distribution as possible, but not select all the public data samples.*

5.5 Performance Evaluation

5.5.1 Experimental Setting

We use the CREMA-D dataset [6] and AVE dataset [76] for evaluation. The CREMA-D dataset is used for emotion analysis and contains visual and audio modalities. There

are six categories, including *happy, sad, anger, fear* and *neural*. The dataset is divided into 6698 samples as the public dataset and 774 as the private dataset. We allocate the private dataset to two clients, where client 1 has samples with classes 0-2 and client 2 has samples with classes 3-5. AVE dataset is an audio-visual dataset for video classification. There are 3741 samples for the public dataset and 402 samples for the private dataset, which are generated according to [76]. The dataset contains 28 classes and all the clips are collected from YouTube.

In the experiment, we use the ResNet101-based network [28] as a teacher model and the ResNet18-based network [28] as a student model. We transform the audio sample to a spectrogram with the dimension of 257×299 for CREMA-D and 237×1004 for AVE. Ten frames are extracted from the input video and we randomly sample 2 frames among them. We use an SGD optimizer with a momentum of 0.9 and a weight decay of 1e-4. The learning rate is set as 1e-3 and becomes 1e-4 in the 70 epochs [63]. We select the data every 10 epochs and train the model for 150 epochs. We group 3 samples as a cluster. We conduct the experiments on one NVIDIA GeForce RTX 3090 GPU.

We consider two kinds of scenarios: Latency Insensitive and Latency Sensitive. In the *Latency Insensitive* scenario, users are not sensitive to inference latency. Therefore, the workflow involves online selection of public datasets and distillation of a smaller model based on the test data from the local device. The distilled model is subsequently used for inference. In the *Latency Sensitive* scenario, users require minimal response time during inference. To achieve this, we split the local dataset into two parts: one part is used for data selection and model distillation, while the other part is reserved for testing. The workflow consists of offline data selection and model distillation, followed by online inference using the pre-distilled smaller model to ensure low latency.

We compare the performance of the following methods: (1) *AllDistill*: The student model is distilled on the whole public dataset and directly applied on the local dataset. (2) *AllDistill+random*: The student model is initialized with the *AllDistill* model.

Method	Latency-Insensitive		Latency-Sensitive	
	Client 1	Client 2	Client 1	Client 2
AllDistill	66.31%	55.59%	69.84%	57.30%
AllDistill+random	63.13%	59.13%	68.25%	62.70%
AllDistill+select	68.70%	61.31%	71.96%	63.78%
AllDistill+select+cluster	67.90%	61.04%	72.49%	61.62%

Table 5.2: Accuracy comparison on CREMA-D.

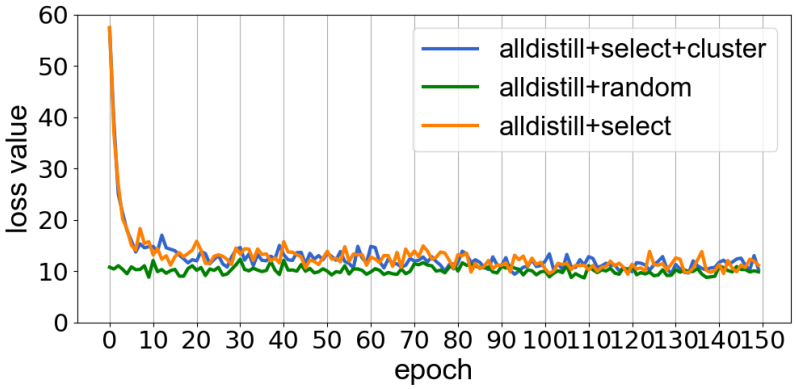


Figure 5.6: Loss change per epoch of client 1 in the latency-insensitive scenario on the CREMA-D dataset.

Then, we randomly select the public dataset and train the model again. (3) *AllDistill+select*: With the *AllDistill* for initialization, we train a student model on the select dataset illustrated in subsection 5.3.3. (4) *AllDistill+select+cluster*: Different from (3), we upload the prototype to the server for data selection.

5.5.2 Experimental Results

We first show the results on the CREMA-D dataset.

In this chapter, we propose utilizing probability prototypes for each class to guide the sample selection process on the server side. To validate the effectiveness of our method, we compare it with two alternative sample selection strategies: random selection and selection based on the full probability distribution of samples in each local

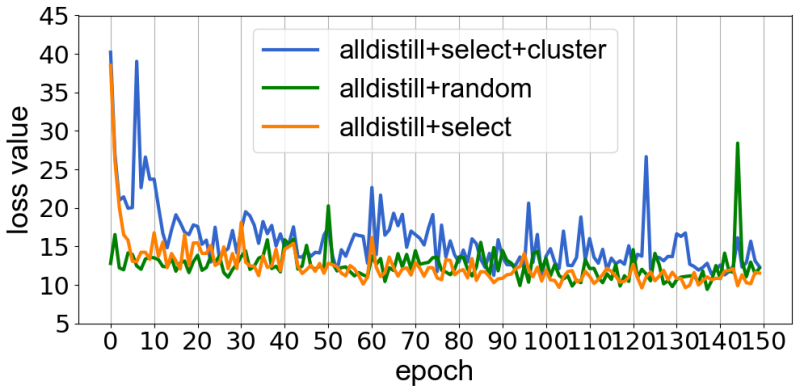


Figure 5.7: Loss change per epoch of client 2 in the latency-insensitive scenario on the CREMA-D dataset.

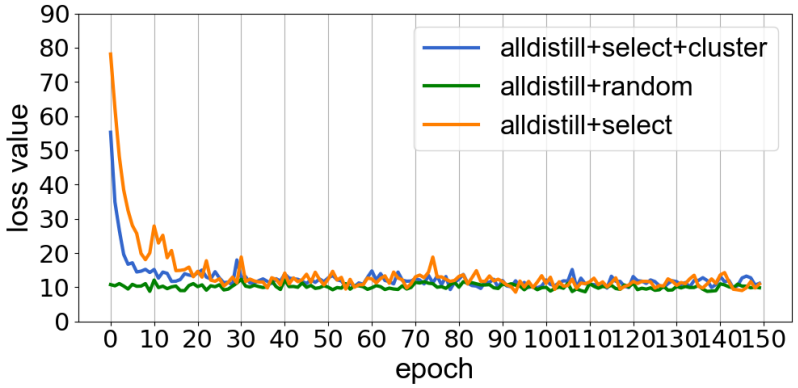


Figure 5.8: Loss change per epoch of client 1 in the latency-sensitive scenario on the CREMA-D dataset.

client (denoted as "AllDistill+select"). The results on the CREMA-D dataset are presented in Tables 5.2. Client 1 and Client 2 contain data from the first three and last three classes, respectively. The "AllDistill" strategy refers to performing distillation using the entire dataset on the server for one epoch. As observed, our method significantly outperforms random sample selection on both clients, regardless of whether the scenario is latency-insensitive or latency-sensitive. Compared to "AllDistill+select" in the latency-insensitive scenario, our method exhibits slightly lower performance (by less than one percentage point) but substantially reduces communication costs. In the latency-sensitive scenario, our method performs relatively worse on Client 2 while achieving a notable improvement on Client 1. This suggests that under certain data

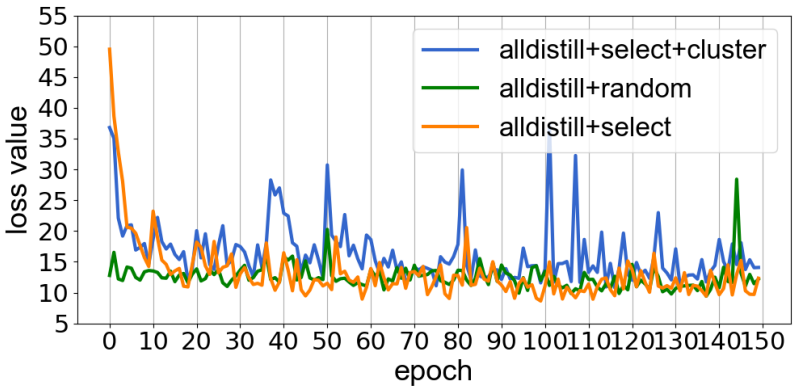
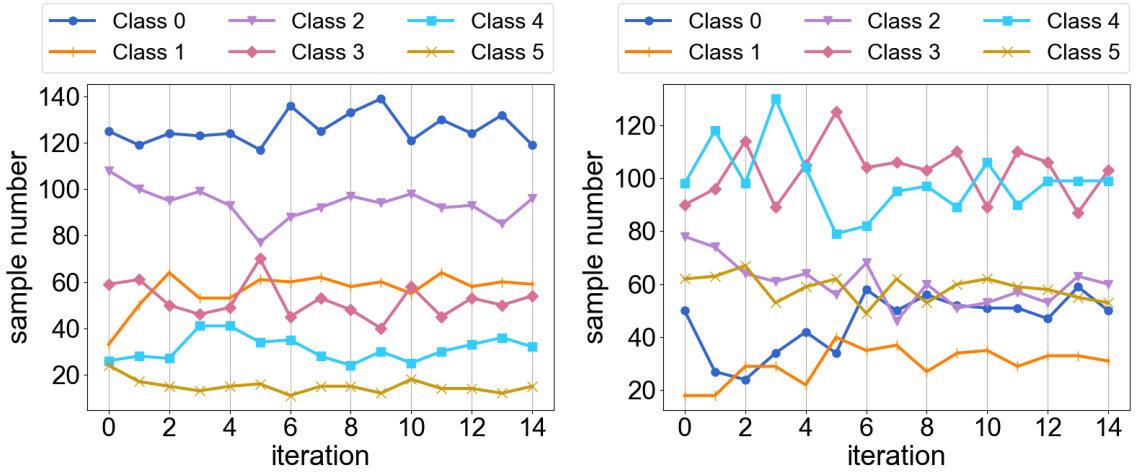


Figure 5.9: Loss change per epoch of client 2 in the latency-sensitive scenario on the CREMA-D dataset.

distributions, probability prototypes effectively capture the semantic information of each class, leading to sample selections that enhance the model's generalization ability. Overall, our method achieves comparable performance to sample selection based on the full probability distribution of local samples while offering improved efficiency.

In addition to the accuracy comparison, we also analyze the loss trends for the three sample selection strategies on the CREMA-D dataset, as illustrated in Figure 5.6∼Figure 5.9. It can be observed that the two sample selection methods based on local information exhibit higher losses during the initial epochs, followed by a sharp decline as training progresses until stabilization. In contrast, the loss for the random selection method remains relatively low throughout the entire training process. This occurs because randomly selected samples maintain a consistent distribution over time, whereas sample selection based on local data information allows models to gradually focus on samples more representative of the local data distribution, thereby enhancing their suitability for local tasks. Consistent with the final accuracy results, the loss for selection based on probability prototypes is slightly higher than that of selection based on all local probabilities.

Sample selection for each class. In the edge scenario, each client contains data from disjoint classes, necessitating the selection of appropriate samples from the server

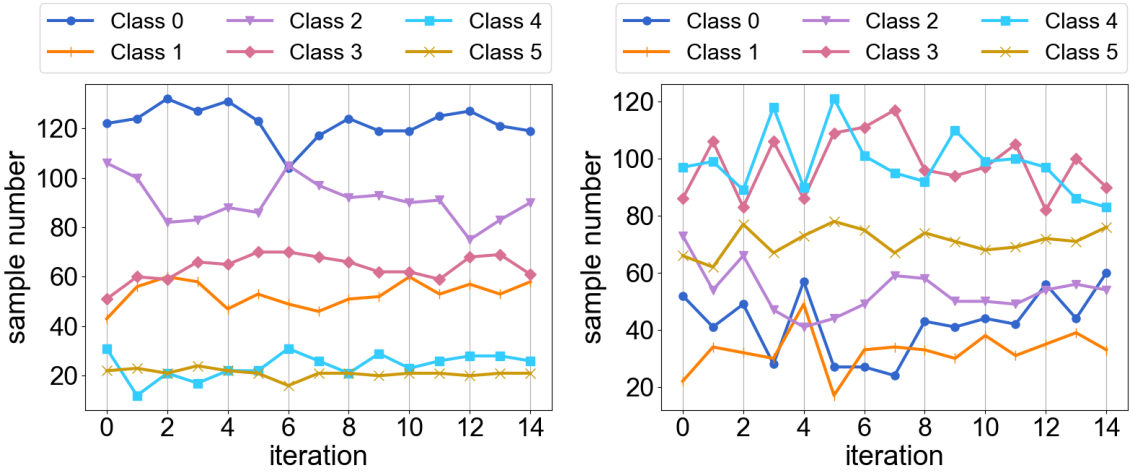


(a) Client 1. (b) Client 2.

Figure 5.10: Selected class number of two clients in the latency-insensitive scenario on the CREMA-D dataset.

dataset to facilitate a more effective local distillation process. The key intuition behind our approach is to prioritize selecting samples that belong to the same classes as those present in the local client data. To illustrate this, we analyze the number of selected samples for each class during training, with the results presented in Figure 5.10 and Figure 5.11. For Client 1, which contains data from classes 0∼2, the selected samples predominantly belong to these same classes. Although some samples from classes 3∼5 are also selected, they constitute only a small proportion. A similar trend is observed for Client 2, further validating the effectiveness of our method in aligning the sample selection process with the local data distribution. However, in some cases, missing classes may contain a comparable or even greater number of samples than the existing classes (e.g., class 1 and class 3 have similar sample sizes in Client 1), particularly in latency-sensitive scenarios. This indicates that while our method effectively selects relevant samples, it may also include some similar samples from missing classes. Optimizing this aspect remains an important direction for future work.

Analysis of compression ratio. In order to further reduce the communication



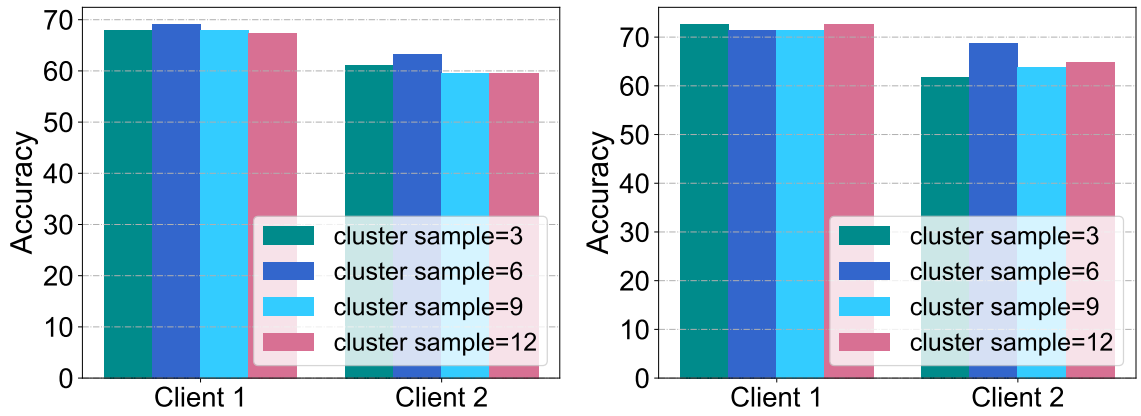
(a) Client 1. (b) Client 2.

Figure 5.11: Selected class number of two clients in the latency-sensitive scenario on the CREMA-D dataset.

Compress ratio	CREMA-D		AVE	
	Client 1	Client 2	Client 1	Client 2
1	72.49%	61.62%	60%	59.34%
1.5	69.31%	62.16%	53.91%	65.93%
2	68.78%	63.24%	58.26%	53.85%
3	71.96%	60%	56.52%	61.54%

Table 5.3: Accuracy comparison for different compression ratios under latency-sensitive scenarios.

cost during inference, we propose a feature compression module to reduce the feature dimensions. We analyze the performance of our method with different compression ratios γ. The results are shown in Table 5.3. As the compression rate increases, although the performance drops a little, it does not continue to decline significantly due to the increase in the compression rate (69.31% with $\gamma = 1.5$ and 71.96% with $\gamma = 9$ on client 1). The accuracy of Client 2 slightly improves, possibly due to the additional scaling network enhancing the model's expressive capability. It shows that our compression method is effective, and the original high-dimensional feature has too much redundant information, which is prominently eliminated by compression. The



(a) Latency-insensitive scenario. (b) Latency-sensitive scenario.

Figure 5.12: Accuracy comparison of different cluster samples.

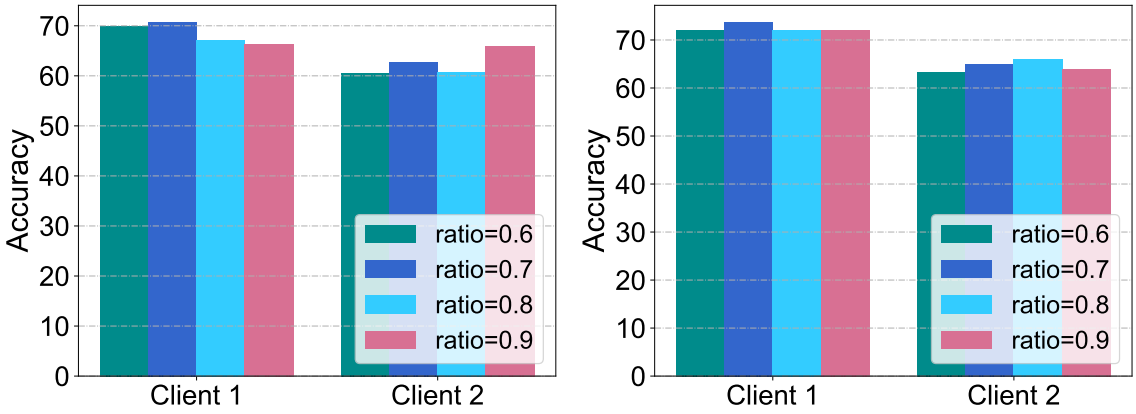
high compression rate does not significantly reduce performance but also brought a great reduction in communication cost.

The influence of the clustering sample. We investigate the influence of the number of samples in each clustering. As shown in Figure 5.12, we vary the sample number from 3 to 12. The clustering process has little influence on the performance. It shows that the student model can learn a good representation for local samples and calculate a representative prototype. In some cases, the algorithm with more clustering samples behaves better because it can avoid the negative influence of some outlier samples.

The influence of the uploaded sample number. In the subsection 5.3.3, we upload all the client samples for data selection. In Figure 5.13, we compare the accuracy when we randomly choose a small set of samples to upload. The accuracy value fluctuates between 60% and 75%. The data quality may increase when we reduce the sample number, leading to the improvement of the accuracy.

We further extend the experiments on the AVE dataset.

Comparison of different sample selection strategies for the AVE dataset. As shown in Table 5.4, our methods achieve the best performance with 61.88% and



(a) Latency-insensitive scenario. (b) Latency-sensitive scenario.

Figure 5.13: Accuracy comparison of different number of uploaded samples.

Method	Latency-Insensitive		Latency-Sensitive	
	Client 1	Client 2	Client 1	Client 2
AllDistill	56.05%	55.87%	54.78%	56.04%
AllDistill+random	60.99%	61.45%	58.26%	60.44%
AllDistill+select	61.88%	62.01%	60.87%	58.24%
AllDistill+select+cluster	61.88%	63.69%	62.61%	64.84%

Table 5.4: Accuracy comparison on the AVE dataset.

63.69% accuracy. In client 1, the *AllDsitill+random* method is worse than the *AllDistill* method because it converges to a small dataset and results in poor generalization. With the clustering methods, the accuracy can be improved by 1.68% accuracy, and the communication size can be also reduced. For a latency-sensitive scenario, we fine-tune the training epochs and get the results in Table 5.4. For client 1, our method have a better performance than random selection. For client 2, because there are few data samples at the client, the data selection algorithm cannot perform well.

Figure 5.14 and Figure 5.15 show the loss changes for different methods. Because there are few private samples in the AVE dataset, the loss curve of our method has a bigger oscillation. The *alldistill+random* method selects data that has a similar distribution with the whole dataset, so the loss value is nearly unchanged.

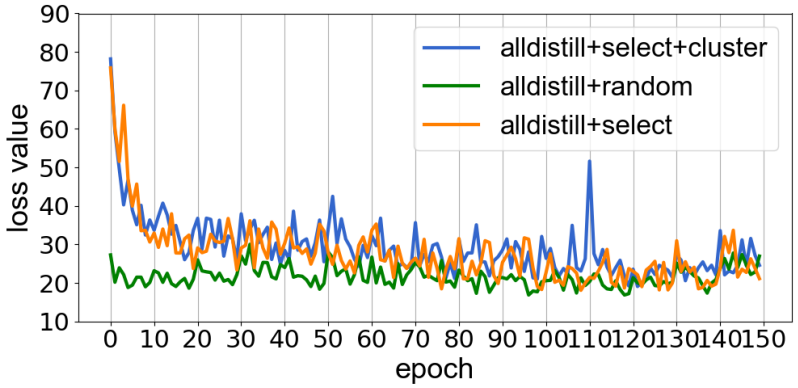


Figure 5.14: Loss change per epoch of client 1 in the latency-insensitive scenario for the AVE dataset.

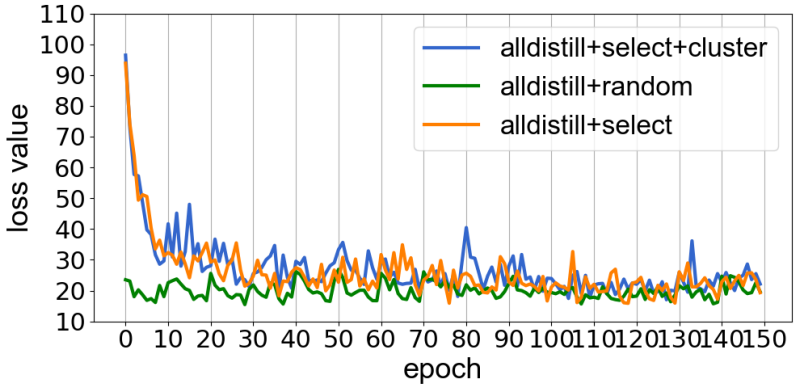


Figure 5.15: Loss change per epoch of client 2 in the latency-insensitive scenario for the AVE dataset.

Analysis of compression ratio for AVE dataset. We investigate the effect of various compression ratios on the AVE dataset. We train the modules on the whole public dataset. As shown in Table. 5.3, the accuracy of client 1 varies from 53.91% to 60%. Different compression ratios of client 2 have similar performance.

5.6 Chapter Summary

In this chapter, we propose a novel cloud-edge collaborative inference framework that enables efficient multimodal model training and inference for users. Our approach

leverages a large, globally trained model in the cloud and selectively transfers its knowledge to individual users based on a public data subset that closely aligns with their data features. To preserve user privacy, we utilize local data prototypes instead of per-sample features when constructing the distillation dataset. Furthermore, to facilitate real-time human inference under varying bandwidth conditions, we design a feature compression method that significantly reduces the communication overhead for feature transmission between the cloud server and edge devices. Extensive experiments demonstrate that the proposed framework enables privacy-preserving multimodal inference for edge users in resource-constrained environments, offering valuable insights for future intelligent edge applications.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis explores three efficient ways to enhance AI model serving, focusing on elasticity, robustness, and privacy-preserving optimization. First, it introduces an elastic Transformer serving system that flexibly adjusts the token numbers, thereby improving throughput under fluctuating query loads. By dynamically adding prompting tokens to boost accuracy and removing redundant tokens to accelerate inference, this mechanism efficiently balances performance and resource usage without requiring multiple pre-trained model variants. Second, the thesis proposes a robust Transformer serving approach, which mitigates unpredictable accuracy degradation caused by aggressive token reduction. Through a token-reduction warm-up strategy and a selective model ensemble method, it achieves consistently high performance across diverse reduction settings while also reducing fine-tuning overhead. Third, this work advances privacy-preserving optimization for AI model serving in edge environments through selective feature distillation. By uploading only feature representations and employing prototype-based clustering, the proposed framework significantly minimizes data leakage risk. An adaptive feature compression scheme further reduces communication

costs, ensuring secure and efficient inference on resource-constrained devices. Chapter 4 enhances the robustness of the methods presented in Chapter 3, while Chapter 5 extends the scope from cloud inference to edge-cloud collaborative inference. Together, these three innovations address urgent issues in large-scale AI deployment. They provide more scalable, reliable, and secure solutions, contributing to the broader goal of delivering high-performance yet cost-effective AI services.

6.2 Future Work

Our work on serving systems is far from solving all problems. In the future, we mainly have three research directions.

(1) Efficiently deploying foundation models on heterogeneous devices. Current research primarily focuses on optimizing serving systems within homogeneous environments. However, with the increasing diversity of hardware, including computing devices from different manufacturers and various architectural versions, efficiently deploying large foundation models on heterogeneous infrastructures remains a significant challenge. Leveraging diverse hardware accelerators, such as NVIDIA GPUs, Huawei Ascend chips, and other domain-specific processors, is essential for optimizing performance across varied computing infrastructures. Furthermore, the efficient deployment of foundation models in heterogeneous edge-cloud environments requires a strategic integration of edge devices and cloud resources. To address these challenges, it is necessary to explore advanced model compression techniques, resource scaling strategies, and the design of parallel computing techniques.

(2) Efficiently deploying multi-modal serving systems. Current research primarily focuses on optimizing serving systems for single-modality models, such as vision Transformers and large language models. However, with advancements in AI research, sophisticated multi-modal models are now capable of seamlessly perceiving, processing,

and generating diverse data types, significantly enhancing the potential of foundation models across various applications. Despite these advancements, deploying large-scale multi-modal models remains a considerable challenge. First, different modalities often have varying model sizes and resource requirements, making it difficult to allocate computational resources efficiently. Second, the inconsistency in input formats across modalities can impact batch processing efficiency, leading to suboptimal throughput. Third, the access frequency of different modalities varies depending on the application scenario, complicating scheduling and resource allocation strategies. Lastly, the large number of parameters in multi-modal models poses challenges for efficient deployment on resource-constrained hardware. To overcome these challenges, it is essential to develop innovative strategies for optimizing multi-modal serving systems.

(3) Efficiently deploying foundation models based AI agents. Current research primarily focuses on optimizing serving systems for simple tasks, such as chatbots and search systems. However, as foundation models become more powerful, they can be developed into a wide range of agents with diverse workflows. These agents can access external knowledge bases, invoke external tools, and interact with their environment to accomplish complex tasks. Therefore, optimizing the efficiency of agent-based systems is also crucial. Despite these advancements, deploying serving systems for AI agents presents significant challenges. First, AI agents often involve complex workflows that require coordination among multiple models, necessitating workflow-aware resource scheduling and parallelization strategies to ensure efficient execution. Second, agents with strong chain-of-thought reasoning capabilities typically handle long input and output sequences, demanding highly efficient processing strategies to maintain responsiveness and minimize latency. Third, multi-agent frameworks introduce additional complexity due to the need for interactions among multiple agents, making deployment in resource-constrained environments particularly challenging. Addressing these issues requires novel system designs and optimization techniques to enhance the scalability and efficiency of agent-based serving architectures.

References

[1] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

[2] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token Merging: Your ViT but Faster. In *International Conference on Learning Representations*, 2023.

[3] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[5] Qixuan Cai, Xiulong Liu, Kaixuan Zhang, Xin Xie, Xinyu Tong, and Keqiu Li. ACF: an adaptive compression framework for multimodal network in embedded devices. *IEEE Trans. Mob. Comput.*, 23(5):5195–5211, 2024.

[6] Houwei Cao, David G. Cooper, Michael K. Keutmann, Ruben C. Gur, Ani Nenkova, and Ragini Verma. CREMA-D: Crowd-Sourced Emotional Multimodal Actors Dataset. *IEEE Transactions on Affective Computing*, 5(4):377–390, 2014.

[7] David F. Carr. Chatgpt's growth begins to flatten, up 12.6% from march to april. `https://www.similarweb.com/blog/insights/ai-news/chatgpt-growth-flattens/`, 2023. Accessed: 2025-02-17.

[8] Junyi Chai, Hao Zeng, Anming Li, and Eric WT Ngai. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, 6:100134, 2021.

[9] Jinyu Chen, Wenchao Xu, Zicong Hong, Song Guo, Haozhao Wang, Jie Zhang, and Deze Zeng. OTAS: An Elastic Transformer Serving System via Token Adaptation. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2024.

[10] Pranav Singh Chib and Pravendra Singh. Recent advancements in end-to-end autonomous driving using deep learning: A survey. *IEEE Transactions on Intelligent Vehicles*, 2023.

[11] Leshem Choshen, Elad Venezian, Noam Slonim, and Yoav Katz. Fusing finetuned models for better pretraining. *arXiv preprint arXiv:2204.03044*, 2022.

[12] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. InferLine: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 477–491, 2020.

[13] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A Low-Latency Online Prediction Serving System. In *NSDI*, volume 17, pages 613–627, 2017.

[14] Weihao Cui, Han Zhao, Quan Chen, Hao Wei, Zirui Li, Deze Zeng, Chao Li, and Minyi Guo. DVABatch: Diversity-aware Multi-Entry Multi-Exit Batching for Efficient Processing of DNN Services on GPUs. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 183–198, 2022.

[15] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. *arXiv preprint arXiv:2302.05442*, 2023.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[17] Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Hai-Tao Zheng, and Maosong Sun. Openprompt: An open-source framework for prompt-learning. *arXiv preprint arXiv:2111.01998*, 2021.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[19] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. Serverlessllm: Locality-enhanced serverless inference for large language models. *arXiv preprint arXiv:2401.14351*, 2024.

[20] Yaroslav Ganin and Victor S. Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32nd International Conference on Ma-*

chine Learning, ICML 2015, Lille, France, 6-11 July 2015, pages 1180–1189, 2015.

[21] GitHub. GitHub Copilot: Your AI Pair Programmer. `https://github.com/features/copilot`, 2023. Accessed: 2025-02-17.

[22] Gunasekaran, Jashwant Raj and Mishra, Cyan Subhra and Thinakaran, Prashanth and Sharma, Bikash and Kandemir, Mahmut Taylan and Das, Chita R. Cocktail: A multidimensional optimization for model serving in cloud. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1041–1057, 2022.

[23] Liwei Guo, Wonkyo Choe, and Felix Xiaozhu Lin. STI: Turbocharge NLP Inference at the Edge via Elastic Pipelining. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 791–803, New York, NY, USA, 2023. Association for Computing Machinery.

[24] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 982–995. IEEE, 2020.

[25] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021.

[26] Joakim Bruslund Haurum, Sergio Escalera, Graham W Taylor, and Thomas B Moeslund. Which tokens to use? investigating token reduction in vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 773–783, 2023.

[27] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[29] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

[30] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[31] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[32] HPC-AI Tech. Surpassing nvidia fastertransformer's inference performance by 50%, open source project powers into the future of large models industrialization. https://www.hpc-ai.tech/blog/surpassing-nvidia-fastertransformers-inference-performance-by-50-open-source-project-powers-into, 2022. Accessed: 2025-02-17.

[33] 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.

[34] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

[35] Gabriel Ilharco, Mitchell Wortsman, Samir Yitzhak Gadre, Shuran Song, Hannaneh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. Patching open-vocabulary models by interpolating weights. *Advances in Neural Information Processing Systems*, 35:29262–29277, 2022.

[36] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M. Hadi Amini. A survey on federated learning for resource-constrained iot devices. *IEEE Internet Things J.*, 9(1):1–24, 2022.

[37] Guangda Ji and Zhanxing Zhu. Knowledge distillation in wide neural networks: Risk bound, data efficiency and imperfect teacher. *Advances in Neural Information Processing Systems*, 33:20823–20833, 2020.

[38] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIII*, pages 709–727. Springer, 2022.

[39] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[40] Antoon WJ Kolen, Jan Karel Lenstra, Christos H Papadimitriou, and Frits CR Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

[41] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561, 2013.

[42] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, Toronto, Canada, 2009.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[44] Ivano Lauriola, Alberto Lavelli, and Fabio Aiolli. An introduction to deep learning in natural language processing: Models, techniques, and tools. *Neurocomputing*, 470:443–456, 2022.

[45] Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. Llm inference serving: Survey of recent advances and opportunities. *arXiv preprint arXiv:2407.12391*, 2024.

[46] Jing Li, Weifa Liang, Yuchen Li, Zichuan Xu, Xiaohua Jia, and Song Guo. Throughput Maximization of Delay-Aware DNN Inference in Edge Computing by Exploring DNN Model Partitioning and Inference Parallelism. *IEEE Transactions on Mobile Computing*, 22(5):3017–3030, 2023.

[47] Junyan Li, Li Lyna Zhang, Jiahang Xu, Yujing Wang, Shaoguang Yan, Yunqing Xia, Yuqing Yang, Ting Cao, Hao Sun, Weiwei Deng, et al. Constraint-aware and ranking-distilled token pruning for efficient transformer inference. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1280–1290, 2023.

[48] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, et al. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. *arXiv preprint arXiv:2302.11665*, 2023.

[49] Peng Liang, Yu Tang, Xiaoda Zhang, Youhui Bai, Teng Su, Zhiquan Lai, Linbo Qiao, and Dongsheng Li. A survey on auto-parallelism of large-scale deep learning training. *IEEE Trans. Parallel Distributed Syst.*, 34(8):2377–2390, 2023.

[50] Youwei Liang, GE Chongjian, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. EViT: Expediting Vision Transformers via Token Reorganizations. In *International Conference on Learning Representations*, 2021.

[51] Bingyan Liu, Yifeng Cai, Hongzhe Bi, Ziqi Zhang, Ding Li, Yao Guo, and Xiangqun Chen. Beyond fine-tuning: Efficient and effective fed-tuning for mobile/web users. In Ying Ding, Jie Tang, Juan F. Sequeda, Lora Aroyo, Carlos Castillo, and Geert-Jan Houben, editors, *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 2863–2873. ACM, 2023.

[52] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[53] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[54] Dmitrii Marin, Jen-Hao Rick Chang, Anurag Ranjan, Anish Prabhu, Mohammad Rastegari, and Oncel Tuzel. Token pooling in vision transformers for image classification. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 12–21, 2023.

[55] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. Towards efficient generative large language model serving: A survey from algorithms to systems. *arXiv preprint arXiv:2312.15234*, 2023.

[56] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 932–949, New York, NY, USA, 2024. Association for Computing Machinery.

[57] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. Spotserve: Serving generative large language models on preemptible instances. *arXiv preprint arXiv:2311.15566*, 2023.

[58] Microsoft. What are Tokens? `https://learn.microsoft.com/en-us/semantic-kernel/prompt-engineering/tokens`, 2023. Accessed: 2025-02-17.

[59] Phil K. Mu, Jinkai Zheng, Tom H. Luan, Lina Zhu, Zhou Su, and Mianxiong Dong. AMIS-MU: edge computing based adaptive video streaming for multiple mobile users. *IEEE Trans. Mob. Comput.*, 23(1):117–134, 2024.

[60] NVIDIA. NVIDIA FasterTransformer. `https://github.com/NVIDIA/FasterTransformer`, 2019. Accessed: 2025-02-03.

[61] NVIDIA. NVIDIA Triton Inference Server. https://docs.nvidia.com/deeplearning/triton-inference-server/, 2024. Accessed: 2025-03-20.

[62] OpenAI. Introducing ChatGPT. `https://openai.com/blog/chatgpt`, 2022. Accessed: 2025-02-17.

[63] Xiaokang Peng, Yake Wei, Andong Deng, Dong Wang, and Di Hu. Balanced Multimodal Learning via On-the-fly Gradient Modulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[64] Mary Phuong and Christoph Lampert. Towards understanding knowledge distillation. In *Proceedings of the 36th International Conference on Machine Learning, ICML, 9-15 June 2019, Long Beach, California, USA*, pages 5142–5151, 2019.

[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] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. *OpenAI*, 2018. Technical Report.

[67] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *OpenAI*, 2019. Technical report.

[68] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[69] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020.

[70] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. INFaaS: Automated Model-less Inference Serving. In *USENIX Annual Technical Conference*, pages 397–411, 2021.

[71] 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 *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 322–337, 2019.

[72] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E Gonzalez, et al. High-throughput generative inference of large language models with a single gpu. *arXiv preprint arXiv:2303.06865*, 2023.

[73] Tuo Shi, Zhipeng Cai, Jianzhong Li, Hong Gao, Tie Qiu, and Wenyu Qu. An efficient processing scheme for concurrent applications in the iot edge. *IEEE Trans. Mob. Comput.*, 23(1):135–149, 2024.

[74] Karen Simonyan. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[75] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient Transformers: A Survey. *ACM Comput. Surv.*, 55(6), dec 2022.

[76] Yapeng Tian, Jing Shi, Bochen Li, Zhiyao Duan, and Chenliang Xu. Audio-Visual Event Localization in Unconstrained Videos. In *ECCV*, 2018.

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[78] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[79] Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. Tabi: An Efficient Multi-Level Inference System for Large Language Models. In *Proceedings of*

the Eighteenth European Conference on Computer Systems, EuroSys '23, page 233–248, New York, NY, USA, 2023. Association for Computing Machinery.

[80] Siyuan Wei, Tianzhu Ye, Shen Zhang, Yao Tang, and Jiajun Liang. Joint Token Pruning and Squeezing Towards More Aggressive Compression of Vision Transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2092–2101, 2023.

[81] Ross Wightman. PyTorch Image Models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[82] Wikipedia contributors. Cumulative Distribution Function. `https://en.wikipedia.org/wiki/Cumulative_distribution_function`, 2023. Accessed: 2025-02-17.

[83] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR, 2022.

[84] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7959–7971, 2022.

[85] Yongji Wu, Matthew Lentz, Danyang Zhuo, and Yao Lu. Serving and Optimizing Machine Learning Workflows on Heterogeneous Infrastructures. *Proceedings of the VLDB Endowment*, 16(3):406–419, 2022.

[86] Fuzhao Xue, Valerii Likhosherstov, Anurag Arnab, Neil Houlsby, Mostafa Dehghani, and Yang You. Adaptive Computation with Elastic Input Sequence. In *International Conference on Machine Learning*, 2023.

[87] Shengyuan Ye, Jiangsu Du, Liekang Zeng, Wenzhong Ou, Xiaowen Chu, Yutong Lu, and Xu Chen. Galaxy: A Resource-Efficient Collaborative Edge AI System for In-situ Transformer Inference. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 1001–1010, 2024.

[88] Fuxun Yu, Di Wang, Longfei Shangguan, Minjia Zhang, Xulong Tang, Chenchen Liu, and Xiang Chen. A survey of large-scale deep learning serving system optimization: Challenges and opportunities. *arXiv preprint arXiv:2111.14247*, 2021.

[89] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1049–1062, Renton, WA, July 2019. USENIX Association.

[90] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. SHEPHERD: Serving DNNs in the wild. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 787–808, Boston, MA, April 2023. USENIX Association.

[91] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv.*, 52(1), February 2019.

[92] Yanqi Zhang, Íñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. Faster and Cheaper Serverless Computing on Harvested Resources. In *Proceedings of the ACM SIGOPS*

28th Symposium on Operating Systems Principles, SOSP '21, page 724–739, New York, NY, USA, 2021. Association for Computing Machinery.

[93] Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. CaM: Cache Merging for Memory-efficient LLMs Inference. In *Forty-first International Conference on Machine Learning*, 2024.

[94] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[95] Kongyange Zhao, Zhi Zhou, Xu Chen, Ruiting Zhou, Xiaoxi Zhang, Shuai Yu, and Di Wu. Edgeadaptor: Online configuration adaption, model selection and resource provisioning for edge dnn inference serving at scale. *IEEE Transactions on Mobile Computing*, 2022.

[96] Yiwu Zhong, Zhuoming Liu, Yin Li, and Liwei Wang. AIM: Adaptive Inference of Multi-Modal LLMs via Token Merging and Pruning, 2024.

[97] Yuchen Zhong, Guangming Sheng, Juncheng Liu, Jinhui Yuan, and Chuan Wu. Swift: Expedited failure recovery for large-scale DNN training. *IEEE Trans. Parallel Distributed Syst.*, 35(9):1644–1656, 2024.

[98] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Learning to prompt for vision-language models. *International Journal of Computer Vision*, 130(9):2337–2348, 2022.

[99] Zhe Zhou, Xuechao Wei, Jiejing Zhang, and Guangyu Sun. PetS: A unified framework for Parameter-Efficient transformers serving. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 489–504, Carlsbad, CA, July 2022. USENIX Association.