GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

LoRA and Efficient LLM Serving for Financial Expert Agents

Priyanshu Gupta

MatrNr: 19750534

Supervisor: Chirag Mandal

Georg-August-Universität Göttingen
Institute of Computer Science

September 30, 2025

Abstract

The rapid proliferation of Large Language Models (LLMs) has unlocked transformative opportunities in specialized domains such as finance, where expert-level agents are required for complex analytical tasks. However, full fine-tuning of massive models like GPT-3 175B is prohibitively expensive, while deploying numerous independently fine-tuned instances introduces severe system-level bottlenecks. Traditional serving architectures, designed for single, monolithic models, struggle with inefficient memory allocation, inference latency, and the "adapter proliferation" problem arising from parameter-efficient tuning techniques. This report investigates Low-Rank Adaptation (LoRA), a parameter-efficient fine-tuning (PEFT) strategy that reduces trainable parameters via rank decomposition matrices, in conjunction with advanced multi-tenant serving frameworks such as vLLM, Punica, S-LoRA, and dLoRA. These systems employ unified paging, custom CUDA kernels for heterogeneous batching, and workload-aware scheduling to enable scalable, high-throughput deployment of thousands of concurrent LoRA adapters with up to $4\times$ higher throughput than baseline systems. Through analysis and financial domain case studies such as FinGPT and FinLoRA, we demonstrate that the synergistic integration of LoRA with specialized serving architectures provides a cost-effective and scalable paradigm for deploying robust financial intelligence agents, ultimately democratizing access to domain-specific AI capabilities.

Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this work I have used ChatGPT or another AI as follows:

- ☐ Not at all
- ☑ During brainstorming
- ☐ When creating the outline
- ☑ To write individual passages, altogether to the extent of 30% of the entire text
- ☐ For the development of software source texts
- ☐ For optimizing or restructuring software source texts
- ☑ For proofreading or optimizing
- ☐ Further, namely: -

I hereby declare that I have stated all uses completely.
Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Tables

List of Figures

List of Listings

List of Abbreviations

AGMs Annual General Meetings

API Application Programming Interface

CoLA Corpus of Linguistic Acceptability

CUDA Compute Unified Device Architecture

DAPT Domain-Adaptive Pretraining

DCF Discounted Cashflow

JSON JavaScript Object Notation

GPT Generative Pre-trained Transformer

GPU Graphics Processing Unit

I/O Input/Output

LLM Large Language Model

LoRA Low-Rank Adaptation

dLoRA Dynamic Low-Rank Adaptation

HPC High Performance Computing

KV Key-Value

MSE Mean Squared Error

OHLCV Open, High, Low, Close, Volume (Financial Data)

PEFT Parameter-Efficient Fine-Tuning

PDF Portable Document Format

RAG Retrieval-Augmented Generation

RLHF Reinforcement Learning from Human Feedback

ROUGE Recall-Oriented Understudy for Gisting Evaluation

S-LoRA Switchable Low-Rank Adaptation

SLO Service Level Objective

SGMV Segmented Gather Matrix-Vector Multiplication

SQL Structured Query Language

SEC Securities and Exchange Commission

vLLM Very Large Language Model

XML Extensible Markup Language

XBRL eXtensible Business Reporting Language

1 Introduction

1.1 The Rise of Domain-Specific LLMs in Finance

A significant current area of non-trivial problem-solving involves adapting Large Language Models (LLMs) to specialized domains, such as finance, where acquiring the necessary specialized knowledge, covering niche terminologies and complex multi-step processes, is often time-intensive [Kal+24]. The dominant approach to achieving task-specific performance is the "pretrain-then-finetune" paradigm, where a general model is refined using domain-specific data [Hu+21; She+24]. Leveraging LLMs to emulate financial domain experts promises to significantly offload routine responsibilities from human experts, allowing them to focus on more strategic tasks [Kal+24].

1.2 Why Fine-Tuned LLMs Don't Scale

As LLMs continue to grow exponentially in scale, with parameter counts reaching 175 billion and beyond, the conventional fine-tuning approach faces a critical scalability challenge [Hu+21]. The major downside of full fine-tuning is that the newly trained model contains as many parameters as the original, making storing and deploying many independent instances of specialized models (like 175B Generative Pre-trained Transformer (GPT)-3 instances) a crucial deployment obstacle [Hu+21]. For LLM serving, the autoregressive generation process results in the KV cache consuming a large portion of Graphics Processing Unit (GPU) memory, and since its size dynamically grows and shrinks, inefficient contiguous memory management leads to severe memory fragmentation, limiting the maximum achievable batch size and overall throughput [Kwo+23]. Furthermore, when developing true expert agents, researchers must address inherent LLM limitations such as hallucination, lack of long-term memory, and inconsistency when following complex, multi-step instructions [Kal+24].

1.3 Memory Fragmentation and the Cost of Efficient Fine-Tuning

Historically, parameter-efficient methods sought to mitigate the high cost of fine-tuning by adapting only a subset of parameters or learning external modules, such as adapter layers or prefix tuning [Hu+21]. However, these techniques often introduced significant inference latency by extending model depth or reduced the usable sequence length, posing a trade-off between efficiency and model quality, often failing to match the performance baseline of full fine-tuning [Hu+21]. Similarly, existing LLM serving systems, like FasterTransformer and Orca, struggle with inefficient KV cache management due to fragmentation (internal and external) because they require tensors to be stored in contiguous memory space [Kwo+23]. When serving multiple models fine-tuned from the same base, utilizing existing systems that treat each model independently leads to redundant weight copies, a high memory footprint, and subsequent low GPU utilization [Wu+].

1.4 Scalable Serving Architectures for LoRA-Enhanced LLMs

The core solution relies on LoRA, which cApplication Programming Interface (API)talizes on the hypothesis that weight changes during model adaptation reside on a low intrinsic dimension [Hu+21]. LoRA addresses the training cost by freezing the large pre-

trained weight matrix (W_0) and optimizing only small rank decomposition matrices ($A \in \mathbb{R}^{h\times r}, B \in \mathbb{R}^{r\times d}$) such that the update $\Delta W = BA$ is applied in parallel to W_0, where the rank r is significantly smaller than the dimensions h and d [Hu+21]. LoRA reduces trainable parameters by up to $10,000\times$ and GPU memory usage by $3\times$ for GPT-3 175B, while eliminating inference latency by merging ΔW with W_0 at deployment [Hu+21]. Building upon this, high-throughput serving architectures, such as vLLM (with PagedAttention), Punica, Switchable Low-Rank Adaptation (S-LoRA), and Dynamic Low-Rank Adaptation (dLoRA), have been developed. These systems enable scalable serving by efficiently managing memory via techniques like Unified Paging for concurrent KV cache and adapter weights, and employ advanced custom Compute Unified Device Architecture (CUDA) kernels for heterogeneous batching of requests destined for different adapters [She+24]. For specific financial expert agents, a comprehensive framework enhances the underlying LoRA-LLM with a layered architecture, incorporating data extraction, Python scripting for complex analysis, and a vector database memory layer for storing expert processes and keyword knowledge [Kal+24].

1.5 Empirical Validation of LoRA in Financial LLMs

The effectiveness of LoRA has been validated across various models (RoBERTa, DeBERTa, GPT-2, GPT-3), frequently matching or exceeding the model quality of full fine-tuning [Hu+21]. When applied to the financial domain (FinLoRA), parameter-efficient fine-tuning achieved substantial performance gains of 36% on average over base models when tackling complex eXtensible Business Reporting Language (XBRL) analysis tasks [Wan+25]. Specialized serving systems demonstrate superior efficiency: S-LoRA is capable of serving thousands of LoRA adapters concurrently, achieving throughput up to $4\times$ higher than naive vLLM implementations and $30\times$ higher than HuggingFace PEFT [She+24]. Furthermore, dLoRA demonstrates optimal performance under skewed workloads by dynamically switching between merged and unmerged inference, achieving up to $1.8\times$ lower average latency compared to S-LoRA. The iterative agent framework (Kaler et al., 2024) significantly improved accuracy on complex financial analysis questions, rising from $\sim 66.79\%$ (coding layer with retries) to $\sim 82.86\%$ (with the addition of the memory layer) in a zero-shot setting [Kal+24].

1.6 Contributions

The work described draws upon and synthesizes several key contributions from the literature:

- LoRA: Proposing and demonstrating the efficacy of freezing pre-trained weights and injecting low-rank decomposition matrices for parameter-efficient adaptation [Hu+21].
- PagedAttention and vLLM: Introducing block-based KV cache memory management, inspired by operating system paging, to drastically reduce memory fragmentation in LLM serving [Kwo+23].
- Punica's SGMV Kernel: Designing the SGMV CUDA kernel to enable efficient batching of requests destined for multiple, different LoRA models simultaneously on a shared GPU [Che+23].
- S-LoRA's Unified Paging: Developing a scalable serving system featuring a unified memory pool to jointly manage dynamic adapter weights and KV cache tensors in a paged fashion, enabling the concurrent serving of thousands of adapters [She+24].

• dLoRA's Dynamic Orchestration: Proposing a system that dynamically switches between merged and unmerged LoRA inference modes and uses adapter-request co-migration to efficiently handle diverse and skewed workloads across a cluster [Wu+].

• Financial Expert Agent Framework: Demonstrating an iterative, zero-shot framework that layers domain-specific capabilities (data extraction, Python scripting, and memory) onto LLMs to achieve high performance in complex financial analysis [Kal+24].

1.7 Report Outline

The remainder of this report details the foundational technology of LoRA, reviews the challenges and innovations in high-performance serving systems, and illustrates their critical application in building specialized financial domain expert agents. Section 2 provides background on the Transformer architecture, the KV cache, and the mechanism and benefits of LoRA. Section 3 analyzes the state-of-the-art serving systems (vLLM, Punica, S-LoRA, and dLoRA) focusing on their memory management and heterogeneous batching strategies. Section 4 explores specific applications in finance, including iterative agent architectures and key fine-tuning projects (FinLoRA, FinGPT). Section 5 summarizes relevant evaluation metrics. Finally, Section 6 provides a comprehensive conclusion.

2 Background Theory

The modern foundation of Large Language Models rests on the Transformer architecture, a sequence-to-sequence model introduced by Vaswani et al. in 2017 that relies heavily on self-attention mechanisms [Hu+21]. The Transformer typically consists of an encoder and a decoder. The encoder processes input data (e.g., a query) into context-rich vectors, essentially deeply understanding the input. The decoder then uses this context along with the partially generated output to predict the next token, utilizing masked self-attention to prevent information leakage from future tokens.

2.1 The Critical Role of the KV Cache

A core component within the Transformer's self-attention module is the generation of query (W_q), key (W_k), and value (W_v) vectors through linear transformations applied to the input hidden states. During the generative inference process, LLMs exhibit an autoregressive pattern, generating one token at a time [Wu+; Kwo+23]. Since the generation of a new token requires attention to all preceding tokens in the sequence (both the prompt and previously generated output), the key and value vectors of these tokens are cached in GPU memory to avoid redundant re-computation, a mechanism known as the KV cache [Wu+].

The KV cache poses significant challenges for high-throughput serving. The memory consumption of the KV cache is proportional to the number of tokens and grows dynamically throughout the request lifetime. For instance, the KV cache for a single token in the OPT 13B model can require 800 KB of space, making KV cache management critical for determining the maximum achievable batch size. This dynamic nature, combined with the often unpredictable output lengths of user requests, is a primary cause of memory-bound performance and fragmentation in traditional LLM serving systems [Kwo+23].

2.2 Mechanism and Benefits of LoRA

LoRA emerged as a technique to mitigate the escalating cost of fine-tuning large pre-trained models. The key insight behind LoRA is the hypothesis that the learned over-parametrized models and the change in weights during adaptation reside on a low intrinsic dimension or rank [Hu+21].

2.2.1 LoRA Technical Implementation

For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA introduces an update ΔW constrained by a low-rank decomposition $\Delta W = BA$, where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, and $r \ll \min(d, k)$ [Hu+21; Wu+]. During the fine-tuning process:

1. The original weights W_0 are frozen and do not receive gradient updates [Hu+21; Wu+].

2. Only the rank decomposition matrices A and B contain trainable parameters [Hu+21].

3. The matrix A is typically initialized using a random Gaussian distribution, while B is initialized to zero, ensuring that $\Delta W = BA$ is zero at the beginning of training [Hu+21].

4. The final adapted weight matrix is $W = W_0 + BA$ [Hu+21].

LoRA is typically applied to the weight matrices within the Transformer's self-attention module, specifically the query (W_q), key (W_k), value (W_v), and output (W_o) projection matrices [Hu+21].

2.2.2 Performance and Efficiency Gains

The low-rank representation drastically reduces the number of trainable parameters. For GPT-3 175B, LoRA can reduce the number of trainable parameters by $10,000\times$ and cut the GPU memory requirement by $3\times$ during training. This reduction in trainable parameters also results in a significant speedup during training (e.g., a 25% speedup was observed on GPT-3 175B compared to full fine-tuning) [Hu+21].

Crucially, LoRA eliminates inference latency compared to a fully fine-tuned model. This is achieved by explicitly computing and storing the merged weight matrix $W = W_0 + BA$ when deploying the model for production. When switching between downstream tasks, the system can quickly recover W_0 by subtracting BA and adding a different $B'A'$ (a quick operation with very little memory overhead) [Hu+21].

Empirical studies confirm that LoRA matches or exceeds the performance quality of full fine-tuning across models like RoBERTa, DeBERTa, and GPT-3. For instance, when scaled up to GPT-3 175B, LoRA matched or exceeded the fine-tuning baseline on WikiStructured Query Language (SQL), MNLI-m, and SAMSum datasets. The observed effectiveness suggests that the update matrix (ΔW) during adaptation often possesses a very low "intrinsic rank" [Hu+21].

3 High-Throughput LoRA Serving Systems

The efficiency of LoRA solves the fine-tuning and storage problem, but the subsequent deployment of thousands of concurrent LoRA adapters requires specialized serving systems to handle multi-tenancy, dynamic batching, and memory constraints [Hu+21]. Several advanced systems leverage operating system principles and custom kernels to address these High Performance Computing (HPC) challenges.

3.1 vLLM: PagedAttention for Efficient Memory Management

The vLLM serving system was developed specifically to overcome the inherent inefficiency of KV cache management in traditional systems, which suffer from fragmentation and memory waste due to storing KV cache in contiguous memory. PagedAttention: vLLM introduces PagedAttention, an attention algorithm inspired by virtual memory and paging techniques in operating systems [Kwo+23].

• Non-Contiguous Memory: PagedAttention partitions the KV cache of each sequence into fixed-size KV blocks. Crucially, these blocks are not required to be stored in contiguous physical memory, similar to how virtual pages are mapped to physical memory frames [Kwo+23].

• Fragmentation Reduction: This design alleviates internal fragmentation by utilizing small blocks allocated on demand and eliminates external fragmentation as all blocks share the same fixed size. This efficient management reduces memory waste from 60-80% in existing systems to near-zero waste [Kwo+23].

• Memory Sharing: PagedAttention enables memory sharing at the block granularity, which is vital for complex decoding algorithms. For instance, in parallel sampling (generating multiple outputs from one prompt), the KV cache of the shared prompt is mapped to the same physical blocks. vLLM uses a copy-on-write mechanism, similar to OS virtual memory, at the block granularity when a shared block needs modification, minimizing unnecessary memory duplication. This sharing achieves significant memory savings, such as \$37.6% - 55.2%\$ in beam search scenarios [Kwo+23].

3.2 Punica: Multi-Tenant Batching with SGMV

Punica is designed explicitly as a multi-tenant serving framework for LoRA models that share a pre-trained backbone model. Its architecture focuses on maximizing GPU efficiency by consolidating multiple LoRA serving workloads onto a small number of GPUs [Che+23].

Core Strategy and SGMV: Unlike the conventional LoRA deployment approach of merging weights (which is necessary for zero latency in single-adapter inference), Punica operates on unmerged weights to enable multi-tenant batching. The computation is separated into the batched base model computation ($\mathbf{x}W$) and the batched LoRA addition ($\mathbf{x}AB$), which is calculated on-the-fly [Che+23].

• Segmented Gather Matrix-Vector Multiplication (SGMV): Punica's key novelty is the SGMV CUDA kernel. SGMV allows batching GPU operations for multiple, different LoRA models concurrently. It parallelizes feature-weight multiplication for different re-

quests and groups requests corresponding to the same LoRA model to enhance operational intensity and utilize GPU Tensor Cores [Che+23].

• Deployment Efficiency: By keeping the base model computation separate and batching the LoRA additive computations, Punica allows a GPU to hold only a single copy of the underlying pre-trained model while serving many different LoRA models. This consolidation significantly improves memory efficiency and allows for a fast cold-start since only the smaller A and B matrices need to be loaded for a new LoRA model [Che+23].

3.3 S-LoRA: Scalable Serving with Unified Paging

S-LoRA is a system specifically designed for the scalable serving of thousands of concurrent LoRA adapters on a single machine or across GPUs [She+24]. Unified Paging: To manage the two major sources of dynamic memory usage—the dynamically sized KV cache tensors and the dynamically loaded adapter weights of various ranks—S-LoRA introduces Unified Paging [She+24].

• Unified Paging uses a unified memory pool to jointly manage both KV cache blocks and adapter weights. Both are stored in this pool in a paged manner: a KV cache tensor consumes S pages (where S is sequence length), and a LoRA weight tensor of rank R consumes R pages. This unified, paged memory structure significantly reduces memory fragmentation caused by dynamic loading/offloading of different-sized adapters and varying KV cache sizes [She+24].

• S-LoRA stores all adapters in main memory (CPU RAM) and only fetches the necessary adapters into the GPU memory for the currently running batch. This approach allows the number of served adapters to be constrained only by the available main memory size, enabling scaling to thousands of adapters [She+24].

• S-LoRA proactively addresses Input/Output (I/O) latency through dynamic prediction and prefetching of adapters needed for the next batch while the current batch is running, overlapping I/O with computation [She+24].

Custom Kernels and Parallelism: S-LoRA employs custom CUDA kernels—such as Multi-size Batched Gather Matrix-Matrix Multiplication (MBGMM) for the prefill stage and MBGMV for the decode stage—that operate directly on the non-contiguous, paged memory structure. Additionally, S-LoRA introduces a novel tensor parallelism strategy that aligns the LoRA computation partitioning with the Megatron-LM strategy used for the base model, minimizing communication costs by scheduling communications on small intermediate tensors [She+24].

3.4 dLoRA: Dynamic Orchestration and Load Balancing

dLoRA (Dynamically Orchestrating Requests and Adapters for LoRA LLM Serving) builds on techniques like PagedAttention and fine-grained scheduling but focuses on dynamically adapting to changing workload patterns and improving efficiency across clustered replicas [Wu+].

Dynamic Cross-Adapter Batching: dLoRA addresses the inefficiency arising from skewed request types within a single replica by proposing dynamic cross-adapter batching. This technique dynamically switches between two inference modes at runtime:

1. Merged Inference: Where the adapter weights are fused into the base model (standard, low-latency approach, similar to single-adapter LoRA deployment). This is benefi-

cial when requests are highly skewed toward a single adapter type, minimizing computational overhead [Wu+].

2. Unmerged Inference: Where the computation of the base model ($\mathbf{x}W$) and the adapter ($\mathbf{x}AB$) are separated, allowing batching across different adapter types. This is crucial when request types are diverse, maximizing GPU utilization. The system uses an adaptive threshold tuning algorithm, guided by the workload ratio, and a credit-based mechanism to prevent starvation of low-frequency adapter types [Wu+].

Adapter-Request Co-migration: To manage load imbalance across multiple replicas (caused by variable input/output lengths and bursty request patterns), dLoRA employs an adapter-request co-migration technique. This mechanism reactively migrates both LoRA adapters and the corresponding requests (including their intermediate KV cache states) from overloaded replicas to underloaded ones. This problem is modeled and solved using an Integer Linear Programming (ILP) formulation to determine the optimal placement plan that minimizes overall running time across the cluster. By leveraging selective migration and constraint relaxation, dLoRA efficiently solves the optimal migration plan within milliseconds, effectively balancing load and improving stability [Wu+].

4 Applications in the Financial Domain

The deployment of specialized Large Language Models (LLMs) in high-stakes environments, such as the financial sector, requires models that possess domain-specific knowledge, precision, and the capability for complex, multi-step reasoning. Leveraging LLMs to emulate financial domain experts can significantly offload routine responsibilities from human experts, freeing them to focus on more strategic tasks [Kal+24]. The efficiency of LoRA combined with high-performance serving systems makes the mass deployment of these specialized agents feasible.

Financial LLMs generally focus on three major application areas:

1. Fundamental Analysis: This involves analyzing a company's financial health and valuation, typically by reviewing official documents like SEC filings (10-K, 10-Q, 8-K).

2. Technical Analysis: Focuses on predicting future price movements based on historical price data and volume, utilizing technical indicators and quantitative strategies.

3. Qualitative Analysis: Pertains to interpreting unstructured text data, such as news articles, market events, Annual General Meetings (Annual General Meetings (AGMs)), conference call transcripts, and earning reports, to assess market sentiment or risk factors.

Achieving expert-level performance in these areas requires highly customized LLMs developed using various techniques and trained on specialized data sources.

4.1 Architecture Overview for Financial Domain Expert Agents

A Retrieval-Augmented Generation (RAG) pipeline alone is insufficient for constructing financial domain expert agents, as such systems must support multi-source data access, analytical reasoning, and domain-specific terminology grounding [Kal+24]. A modular, layered architecture provides a more robust foundation.

(1) **Data Extraction Layer:** The language model generates structured queries (e.g., SQL or API calls) to retrieve heterogeneous financial data, including structured statements, time-series OHLCV and technical indicators, as well as unstructured sources such as earnings call transcripts, AGMs, news, and research reports. A tool executes these

queries and returns the results to the model. However, this layer alone exhibits high failure rates on complex queries and may introduce compliance risks when relying on remote execution.

(2) **Scripting Layer:** To support analytical computation beyond declarative querying, a scripting interface (e.g., Python execution) enables statistical modeling, indicator computation, and visualization. The model decomposes tasks into sequenced data extraction and scripting steps. Error feedback from failed executions enables self-correction, while sandboxed execution ensures security.

(3) **Memory Layer:** This enhances an agent by providing domain expertise, enabling it to handle complex, multi-step processes more accurately. Unlike relying solely on zero-shot reasoning or a few static examples, it allows the agent to store and recall relevant knowledge, interpret specialized terminology, and learn from past interactions. This structured memory improves the agent's ability to answer detailed questions with precision and reduces errors. In testing, incorporating the Memory Layer increased accuracy on complex tasks, such as financial analysis processes, from approximately 67% to 83%, demonstrating its effectiveness in enhancing domain-specific reasoning [Kal+24].

Together, these layers form a unified framework that integrates heterogeneous data ingestion, analytical computation, and long-term process memory, enabling scalable financial LLM systems that operate beyond conventional RAG-based designs.

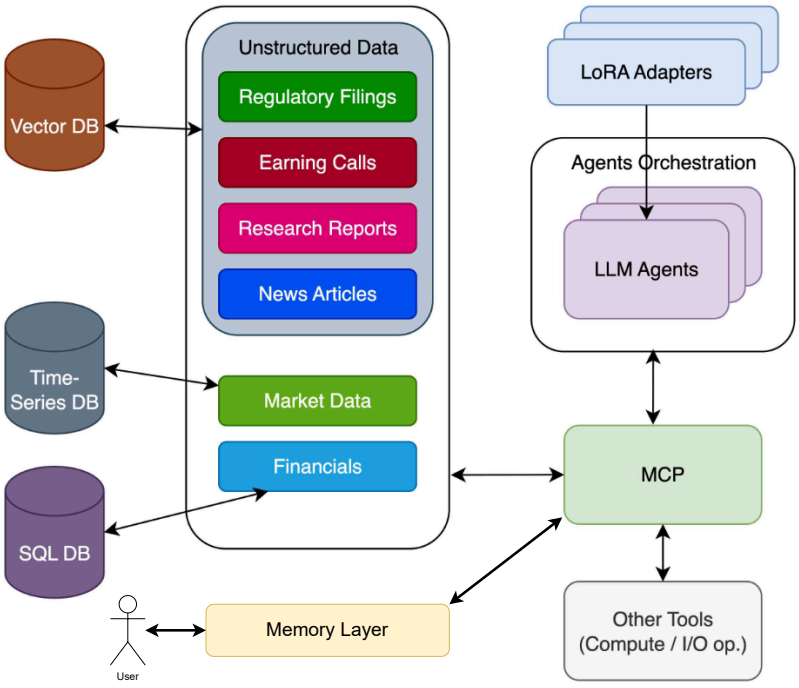


Figure 1: Financial LLM Architecture Overview

4.2 Different Techniques for Making Finance Specific LLMs

Building a financial domain expert agent can be approached through a spectrum of methods, ranging from basic prompting to highly resource-intensive pretraining. These tech-

niques often prioritize different goals, such as maximizing accuracy, minimizing cost, or enhancing domain internalization.

Table 1: Techniques for Making Finance Specific LLMs

Goal	**Best Method**
Q&A over financial documents	RAG (Retrieval-Augmented Generation)
Internalize financial knowledge style	LoRA or Instruction Tuning
Lightweight deployment	Prompt Engineering + LoRA
Deep adaptation to domain	Domain-Adaptive Pretraining (DAPT) + Finetuning
Report automation or analysis agent	Tool-Augmented LLM Agent

4.2.1 Prompt Engineering and System Prompts

This method relies on carefully crafted instructions to guide the LLM's behavior, making it act like a financial analyst. This includes role-playing prompts (e.g., "You are a CFA-certified analyst...") and providing few-shot examples to demonstrate desired outputs. It may also incorporate tool or function-calling instructions (e.g., "Run a Discounted Cashflow (DCF) with the following inputs..."). A major drawback is that complex tasks can lead to high token usage.

4.2.2 Retrieval-Augmented Generation (RAG)

RAG leverages a vector database containing finance-specific documents (e.g., 10-Ks, earnings calls, research reports). The LLM retrieves relevant information from this external knowledge base to formulate its response. This technique requires no explicit model fine-tuning and offers flexibility and interpretability. However, initial setup costs can be high, and it is less suited for real-time high-frequency data compared to other methods.

4.2.3 Tool-Augmented Agents (AutoGPT-style)

This approach combines the reasoning capabilities of an LLM with external tools. Tools typically include a Python or spreadsheet engine for complex financial calculations, a Web search API for real-time news, or PDF parsers for extracting data from reports. Agents like FinGPT agents are examples of this, capable of sophisticated actions such as "scrape earnings release, analyze it, generate report".

4.2.4 PEFT and LoRA

LoRA is specifically employed when the developer seeks to make the LLM internalize financial tone, structure, or language. It involves injecting trainable adapters into key model layers and training them on small, domain-specific datasets (e.g., financial Q&A or analyst reports). LoRA is recommended for internalizing financial knowledge style and for lightweight deployment when combined with prompt engineering.

4.2.5 Instruction Tuning

This technique involves fine-tuning the model with financial tasks phrased as explicit instructions (e.g., "Explain the difference between EBITDA and net income."). It often uses FLAN-style instruction tuning formats and is frequently combined with LoRA, serving to enhance the model's understanding of financial reasoning tasks.

4.2.6 DAPT

DAPT represents the most expensive but often the most effective method for creating a customized base model. It uses unsupervised learning on a vast financial corpus (news, filings, Reddit, blogs, research reports) to help the model adapt the style, jargon, and semantics specific to finance, often resulting in "FinanceBERT" or "FinGPT"-style foundation models. DAPT is the best method for deep adaptation to a domain.

4.3 Data Sources for Finance LLMs

Training and adapting LLMs for finance requires access to extensive, high-quality, and specialized datasets. The necessary data sources span diverse formats, from structured market indicators to complex legal text:

Table 2: Representative Data Sources and Formats for Finance LLMs

Source Type	**Examples**	**Data Formats**
Regulatory Filings	10-K, 10-Q, 8-K, S-1	XBRL, Portable Document Format (PDF)
Earnings Calls	Transcripts, Analyst Q&A	PDF
Research Reports	Equity Research, DCF Models, Market Outlooks	PDF, DOCX
News Articles	Reuters, Yahoo Finance, Bloomberg	Raw Text, JSON
Market Data	Technical Indicators, Open, High, Low, Close, Volume (Financial Data) (OHLCV) Prices	JavaScript Object Notation (JSON)

5 Evaluation Metrics

Effective evaluation of high-performance serving systems and specialized LLMs requires a comprehensive set of metrics covering quality, speed, and resource efficiency

5.1 LoRA-Specific Financial Models

5.1.1 FinLoRA

FinLoRA is an exemplary project demonstrating how PEFT can create specialized financial LLMs in an affordable and scalable manner.

Motivation and Goals

The project was developed in response to resource-intensive efforts like the BloombergGPT model, which highlighted the potential of specialized FinLLMs but required one million GPU hours, estimated to cost around $3 million in 2023. The core goal of FinLoRA is to democratize financial intelligence by reducing the computational cost of creating specialized models to less than $100.

Methodology

FinLoRA focuses on Llama 3.1 models and utilizes the LoRA fine-tuning method. The crucial domain area targeted is the eXtensible Business Reporting Language (XBRL), the global standard for digital business reporting, which is inherently complex due to its Extensible Markup Language (XML)-based structure [Wan+25].

1. **Datasets:** Using four novel XBRL analysis datasets derived from 150 SEC filings.
2. **LoRA Variants:** Employing multiple LoRA variants, including LoRA, QLoRA, DoRA, and rService Level Objective (SLO)RA. These variants were configured in various ways, such as 8-bit rank 8 and 4-bit rank 4 settings.
3. **Efficiency:** By using LoRA, the project reduced the number of trainable parameters to as little as $0.01%$ of the full model's parameter count.

Applications and Outcomes

FinLoRA targeted two key XBRL applications:

• **Financial Reporting:** Assisting small and medium-sized businesses in generating compliant financial reports in the XBRL format.

• **Financial Statement Analysis:** Facilitating the extraction of data from XBRL reports and enabling insightful analysis.

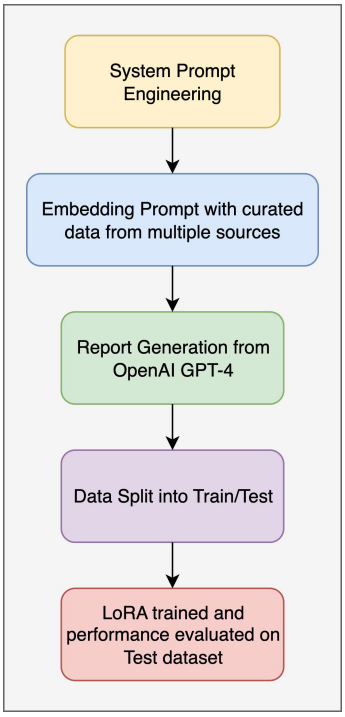
The fine-tuned adapters achieved substantial performance gains of 36% on average over the base models. This performance, achieved through lightweight, low-cost LoRA tuning, validated the parameter-efficient approach as a viable alternative to monolithic, expensive foundation models.

5.1.2 FinGPT

FinGPT is an open-source financial LLM project that operates as a Tool-Augmented Agent, integrating LLM capabilities with external data sources and complex techniques [YLW23].

Key Technology: A distinguishing feature of FinGPT is its incorporation of Reinforcement Learning from Human Feedback (RLHF), a technology noted as being absent from the initial BloombergGPT model. RLHF is crucial because it allows the LLM model to learn individual user preferences—such as their risk-aversion level, investing habits, or personalized robo-advisor needs—which is essential for advanced personalization.

Architecture and Workflow: FinGPT's operation relies heavily on external tools to acquire and structure data before feeding it to the core LLM for analysis.
Report Analysis Module: Data is collected from Securities and Exchange Commission (SEC) filings (10-K, 10-Q, 8-K via sec-API.io), market fundamentals and financial statements (via yfinance and FinancialModelingPrep), analyst recommendations, price targets, historical prices, and earnings call transcripts. Relevant sections are extracted using Retrieval-Augmented Generation (Retrieval-Augmented Generation (RAG)) pipelines (e.g., LangChain) and supplied to an LLM (e.g., GPT-4-turbo) via structured system prompts.
Forecasting Module: News (via Finnhub) and historical financial metrics (yfinance, Finnhub) are aggregated and converted into structured prompts. An LLM generates forward-looking assessments, which are labeled with observed future price movements (e.g., next-week return). The resulting dataset is used to fine-tune forecasting models using LoRA applied to architectures such as ChatGLM2-6B or LLaMA-2-7B.



Module Overview

5.2 Language Model Performance Metrics

These metrics assess the accuracy and quality of the specialized LLM's output for specific financial tasks:

- Accuracy/F1-Score: Standard classification metrics [Hu+24].
- Binary Accuracy (bin_acc): Used in forecasting to evaluate the accuracy of up/down/neutral trend direction predictions.
- Mean Squared Error (MSE): Measures the squared difference between predicted and ground-truth numerical prediction margins, crucial for quantitative analysis.
- Recall-Oriented Understudy for Gisting Evaluation (ROUGE) Scores (ROUGE-L, ROUGE-N): Measures the quality of machine-generated text (e.g., summarization or report generation) by comparing overlap of n-grams or longest common subsequences with human-written references [WM25; YLW23].
- Correlation Metrics: Pearson correlation (for tasks like STS-B) or Matthew's correlation (for Corpus of Linguistic Acceptability (CoLA)).

5.3 Serving System Metrics

These metrics quantify the efficiency and performance of the multi-tenant serving systems under dynamic load:

- Throughput (Tokens/sec, Requests/sec): Measures the rate at which the system processes output tokens or full requests, representing overall capacity [Zho+24].
- Latency: Critical metrics include Average Request Latency, First Token Latency (important for user experience), P90 Latency, and component-specific latencies (Decode, Prefill, Kernel Latency) [Zho+24].
- Resource Efficiency: Includes GPU Utilization (Streaming Multiprocessor/SM Utilization), KV Cache Memory Usage, and Memory Fragmentation levels.

• User-Centric Metrics: SLO Attainment (the percentage of requests meeting a Service Level Objective, typically a latency target) and User Satisfaction Score (a more fine-grained analysis of latency relative to the SLO).

• Scalability Overhead: Includes the measurable cost of operations introduced by advanced serving features, such as the Switching Overhead (for dLoRA's mode switching) and I/O Computation Overhead (for adapter swapping/loading).

5.4 LoRA-Specific Metrics

Evaluation often involves assessing the unique characteristics of LoRA deployments:

• Rank Deficiency: Investigating the "intrinsic rank" of the update matrix ΔW to determine the minimal sufficient rank (r) for optimal performance.

• Weight Selection Logic: Determining which specific Transformer layer weight matrices (W_q, W_k, W_v, W_o) should receive LoRA adapters for maximal downstream performance given a limited parameter budget. Empirically, adapting both W_q and W_v often yields the best performance.

• Performance vs. Efficiency Tradeoff: Analyzing how changes in trainable parameters (or rank r) affect task performance, noting that performance may not always benefit monotonically from having more trainable parameters, especially for methods like prefix tuning.

6 Conclusion

The demand for specialized LLMs, particularly in high-stakes domains like finance, necessitates a transition from resource-intensive full fine-tuning to highly scalable, parameter-efficient solutions. LoRA serves as the fundamental technical enabler, dramatically reducing training costs, storage requirements (by $10,000\times$), and eliminating inference latency by merging adapter weights [Hu+21]. This technological foundation has spurred the development of specialized serving systems that overcome HPC challenges inherent in scaling LLM deployment:

• vLLM and its PagedAttention operator revolutionized KV cache management by applying OS paging concepts, enabling near-zero memory fragmentation and increasing batch capacity [Kwo+23].

• Punica enabled the concurrent batching of requests to multiple different LoRA adapters on a single shared GPU using the novel SGMV kernel, maximizing shared backbone efficiency.

• S-LoRA achieved massive scalability by introducing Unified Paging to manage dynamic KV cache and adapter weights jointly, allowing systems to serve thousands of distinct LoRA adapters constrained only by CPU memory [She+24].

• dLoRA addressed operational volatility across clusters by implementing dynamic batching (switching between merged and unmerged inference) and optimal adapter-request co-migration, yielding superior performance under skewed and variable workloads [Wu+].

In the financial domain, these innovations are already proving transformative. Projects like FinLoRA demonstrate that affordable, low-cost LoRA fine-tuning can achieve high performance on complex tasks like XBRL analysis (a \$36%\$ gain over base models), effectively lowering the barrier to entry for developing FinLLMs [Wan+25]. Furthermore,

the architectural framework for Financial Domain Expert Agents demonstrates that iteratively adding capabilities—including Python scripting for complex numerical analysis and a memory layer for storing expert processes, significantly enhances performance and reduces ambiguity in zero-shot execution [Kal+24].

Collectively, the integration of LoRA with high-performance serving architectures provides the required computational foundation to deploy specialized, high-accuracy, and scalable LLM agents, signaling a pivotal shift toward the practical democratization of domain-specific artificial intelligence.

References

[Che+23] Lequn Chen et al. *Punica: Multi-Tenant LoRA Serving*. Oct. 28, 2023. DOI: 10.48550/arXiv.2310.18547. arXiv: 2310.18547 [cs]. URL: http://arxiv.org/abs/2310.18547 (visited on 06/02/2025). Pre-published.

[Hu+21] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. Oct. 16, 2021. DOI: 10.48550/arXiv.2106.09685. arXiv: 2106.09685 [cs]. URL: http://arxiv.org/abs/2106.09685 (visited on 06/02/2025). Pre-published.

[Hu+24] Jiacheng Hu et al. "Optimizing Large Language Models with an Enhanced LoRA Fine-Tuning Algorithm for Efficiency and Robustness in NLP Tasks". In: *2024 4th International Conference on Communication Technology and Information Technology (ICCTIT)*. 2024, pp. 526–530. DOI: 10.1109/ICCTIT64404.2024.10928552.

[Kal+24] Gagandeep Singh Kaler et al. "A Natural Way of Building Financial Domain Expert Agents". In: (2024).

[Kwo+23] Woosuk Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. Sept. 12, 2023. DOI: 10.48550/arXiv.2309.06180. arXiv: 2309.06180 [cs]. URL: http://arxiv.org/abs/2309.06180 (visited on 06/02/2025). Pre-published.

[She+24] Ying Sheng et al. *S-LoRA: Serving Thousands of Concurrent LoRA Adapters*. June 5, 2024. DOI: 10.48550/arXiv.2311.03285. arXiv: 2311.03285 [cs]. URL: http://arxiv.org/abs/2311.03285 (visited on 06/02/2025). Pre-published.

[Wan+25] Dannong Wang et al. *FinLoRA: Benchmarking LoRA Methods for Fine-Tuning LLMs on Financial Datasets*. May 26, 2025. DOI: 10.48550/arXiv.2505.19819. arXiv: 2505.19819 [cs]. URL: http://arxiv.org/abs/2505.19819 (visited on 07/23/2025). Pre-published.

[WM25] Qiang Wang and Ning Ma. "A Comparative Analysis of Large Model Role-Dialogues Based on LoRA Fine-Tuning has been Conducted". In: *2025 8th International Conference on Advanced Algorithms and Control Engineering (ICAACE)*. 2025, pp. 1498–1502. DOI: 10.1109/ICAACE65325.2025.11019905.

[Wu+] Bingyang Wu et al. "dLoRA: Dynamically Orchestrating Requests and Adapters for LoRA LLM Serving". In: ().

[YLW23] Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. "FinGPT: Open-Source Financial Large Language Models". In: *FinLLM Symposium at IJCAI 2023* (2023).

[Zho+24] Yinmin Zhong et al. *DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving*. June 6, 2024. DOI: 10.48550/arXiv.2401.09670. arXiv: 2401.09670 [cs]. URL: http://arxiv.org/abs/2401.09670 (visited on 09/29/2025). Pre-published.