Review

Systems Engineering of Large Language Models for Enterprise Applications

Shubham Singh *

Posted Date: 9 January 2025

doi: 10.20944/preprints202501.0715.v1

Keywords: large language models; fine tuning; system engineering

Review

Systems Engineering of Large Language Models for Enterprise Applications

Shubham Singh

New York University; shubham.singh@nyu.edu

Abstract: This report provides a comprehensive systems engineering analysis of Large Language Model (LLM) adoption in enterprise environments, emphasizing the need for a structured implementation approach to ensure success. It examines key aspects of LLM deployment, including customization techniques such as Parameter-Efficient Fine-Tuning (PEFT), Low-Rank Adaptation (LoRA), and Retrieval-Augmented Generation (RAG), while addressing challenges related to computational resource management, model safety, and ethical considerations. The study outlines a framework covering the entire LLM lifecycle, from initial needs assessment to deployment and maintenance, with a focus on data management strategies, infrastructure requirements, and integration methods. Additionally, it offers a comparative analysis of fine-tuning existing models versus developing in-house solutions, providing practical guidance for overcoming implementation challenges. The findings underscore the importance of balancing technical capabilities with operational constraints and ensuring robust security, compliance, and ethical standards.

Keywords: large language models; fine tuning; system engineering

1. Introduction

Large Language Models or LLMs have transformed artificial intelligence accelerating adoption of AI in enterprise applications giving businesses new ways to use natural language data to improve their operations and customer experiences. Advanced AI models like GPT-4 (Open AI, Achiam *et. al*, 2023), Claude (Anthropic, 2024), and open source versions of equally capable models like LLAMA 3 (Meta, Dubey *et. al*, 2024) are pushing the limits of understanding and generating natural language with exceptional accuracy, almost indifferentiable from human level in a lot of tasks such as generating code for programming applications (Kim *et. al*, 2023), or reasoning in recommender systems (Hou, Y. *et al*., 2024).

As businesses across various sectors continue adopting LLMs, they are changing how companies interact with data, customers, and internal processes as they move towards relying on organizations like OpenAI, Anthropic and Meta for the foundational LLMs on which their systems are built. Companies can automate complex tasks, extract valuable insights from unstructured data, and provide more intuitive user experiences without spending millions (Shan, 2024) on creating specialized models for each task like the norm before. The adaptability and learning abilities of LLMs make them especially well-suited to the rapidly changing and innovating nature of modern business environments, where flexibility and accuracy are essential with low costs.

As it becomes more and more important for enterprises to adopt AI in their everyday operations, This research report explores the application of systems engineering methods to enable better adoption of Large Language Models (LLMs) in enterprise settings. Aiming to provide a comprehensive framework that enables businesses to integrate LLMs into their operations more quickly and effectively, addressing both technical and organizational needs. Through a thorough analysis of technical and organizational considerations, we strive to equip enterprises with the knowledge and strategies necessary to successfully integrate LLMs into their operations while addressing the inherent challenges and ethical implications.

 doi:10.20944/preprints202501.0715.v1

2. Background

LLMs represent the latest advancement in natural language processing, built on the foundation of deep learning and transformer architectures (Vaswani et. al, 2017). These models that started with a simple idea of attention mechanism and Reinforcement learning with Human feedback have transformed into large (200 Billion parameters) models that are trained on large amounts of multimodal datasets allowing them to understand and generate text, image and audio across a wide range of topics and tasks. The development of LLMs has been marked by rapid progress over the past 3 years, with each new iteration bringing significant improvements in performance and capabilities.

LLMs have had a tremendous impact across various fields, including healthcare, education, law, finance, and scientific research (Kukreja et. al., 2024). In healthcare, LLMs have been used for tasks like medical diagnosis based on foundational models (Vorontsov, E., Bozkurt, A., Casson, A. et al., 2024) and extracting biological information, but researchers have also raised concerns about misinformation and patient privacy (Haltaufderheide, J., Ranisch, R., 2024). In education, LLMs have been explored for helping with standardized tests and writing, while facing challenges related to plagiarism, inaccuracy and an inherent bias (Wang *et. al*, 2024). In law, LLMs have been used to analyze legal documents and predict judgments (Lai *et. al.* 2023). In finance, LLMs have been used for analyzing financial sentiment, but with careful monitoring due to potential risks to financial markets. In scientific research, LLMs have been used to help with various stages of the research process, from literature reviews to data analysis, while continually improving their reliability (Kukreja et. al., 2024). These adoptions of LLMs in different industries have been reported to have provided a significant boost to the productivity and output at a pace never experienced before.

Key milestones in LLM Research:

2017: Attention is all you need (Vaswani et. al, 2017)

2018: Introduction of BERT (Bidirectional Encoder Representations from Transformers) by Google, revolutionizing natural language understanding (Delvin et. al, 2018).

2019: Release of GPT-2 (Radford et. al, 2019) by OpenAI, demonstrating impressive text generation capabilities.

2022: Introduction of ChatGPT, bringing conversational AI to the mainstream.

2023: Release of GPT-4 and LLAMA, pushing the boundaries of LLM performance and accessibility.

2024: LLAMA 3 (open source) surpasses GPT4, Claude 3 is launched

As LLMs have evolved from text only to multimodal, their potential applications have expanded dramatically, from enhancing customer support systems (Rome et. al, 2024) to generate complex research reports.

However, the adoption of LLMs in enterprise environments presents unique challenges. These include managing massive datasets for fine tuning the model specifically for organization's internal purposes, ensuring model safety, privacy and ethical behavior (Schillaci et. al, 2024). Given these challenges and opportunities, a systems engineering approach is crucial for successful LLM adoption in enterprise settings. This approach is needed to provide a structured approach for companies for addressing the challenges of LLM integration, ensuring that technical, organizational, and ethical considerations are addressed.

2.1. System Engineering for Developing AI Pipelines and Its Importance for LLM Adoption

Systems engineering is crucial to any project from designing automotive systems (D'Ambrosio, 2017) to setting up complex software projects, and as adoption of AI grows there's a need for a framework for designing these Large Language systems for enterprise applications.

The approach needs enterprises to view AI pipelines as interconnected systems rather than isolated components independent of companies software infrastructure. By applying systems engineering principles, organizations can more effectively define and manage requirements, design

 doi:10.20944/preprints202501.0715.v1

architectures, and implement comprehensive testing strategies. This methodology considers the entire lifecycle of AI systems, from conception through deployment and maintenance, while also facilitating risk management and performance optimization across the entire pipeline.

Systems engineering emphasizes thorough documentation and traceability, crucial for maintaining complex AI systems over time, and aligns well with the iterative nature of AI development. These systems engineering principles can help companies create efficient and manageable AI and LLM pipelines by effectively addressing challenges posed by tasks.

3. Concept Development Stage

This section lays out a methodology based on the system engineering principles to develop LLM pipelines in an organization. System engineering lifecycle can be seen in Figure 1.

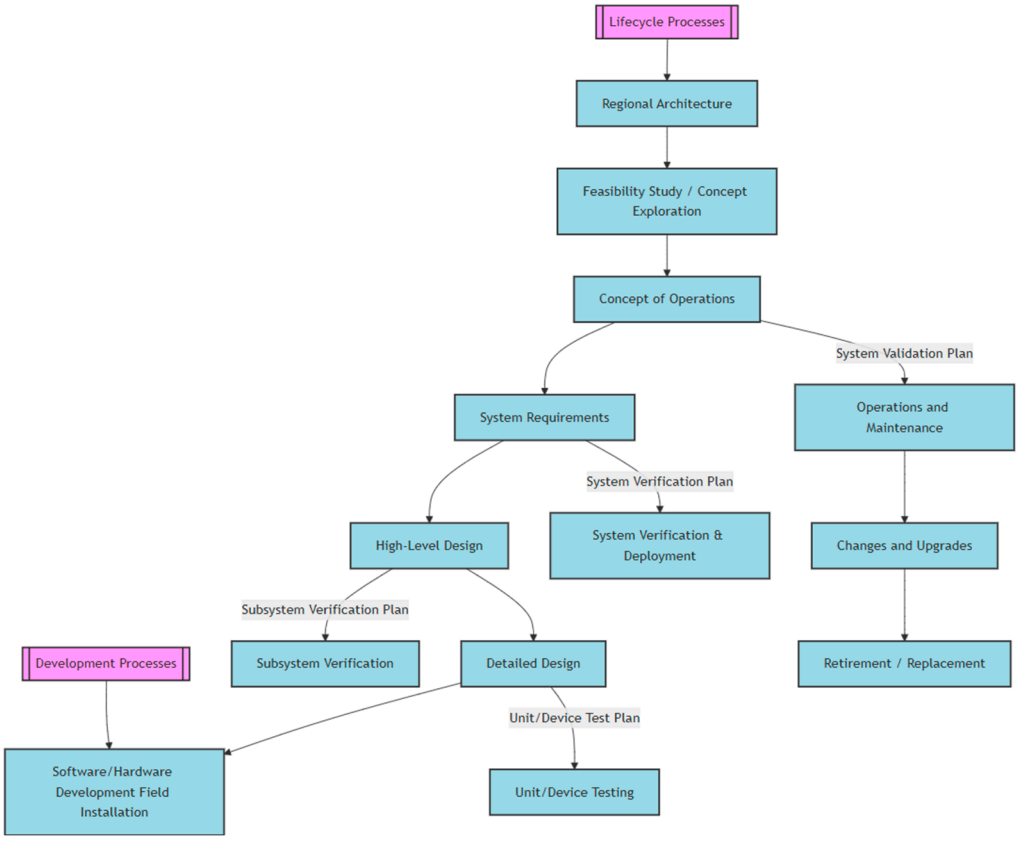

Figure 1. System Engineering lifecycle.

3.1. Needs Analysis

From a systems engineering perspective, the needs analysis requires an evaluation of the organization's requirements across departments and functions. This involves identifying specific use cases where LLMs can add value such as automating code generation, getting insights about a topic/ internal documentation or providing decision support in complex scenarios.

Systems engineers must work closely with stakeholders to define clear objectives, performance criterias, and constraints that will serve as a guideline for the LLM integration to the enterprise's functions as illustrated in Figure 2. This stage also involves assessing the organization's existing technological infrastructure and data resources to determine the feasibility and scope of implementation.

 doi:10.20944/preprints202501.0715.v1

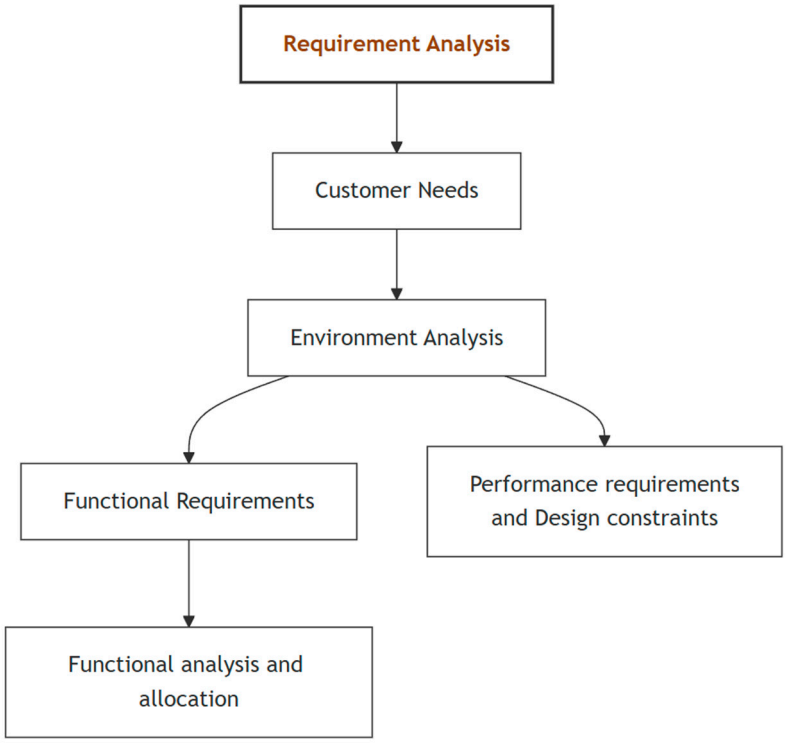

Figure 2. Needs/ Requirement analysis for system engineering.

3.2. Concept Exploration

System engineers should evaluate different architectures and fine tuning methodologies to determine the most suitable approach for the enterprise's needs. This involves analyzing various models including proprietary models such as ChatGPT and Claude as well as OpenSource license models like LLama 3, comparing their capabilities on specific tasks (depending on organization's requirements), and assessing their potential for customization with organization's tech stack as well as data.

3.3. System Definition

The concept definition stage involves clearly outlining the scope, scale, and capabilities of the chosen LLM within the context of our organization. KPIs such as accuracy, inference speed, cost and interpretability of the model outputs must be defined, by taking into account the specific requirements.

This stage also involves identifying and analyzing potential trade-offs between model performance, computational resources, and implementation costs. The organizations need to create a detailed specification that outlines the expected functionalities and scalability capabilities to make sure that the solutions align with our long-term goals and operational needs.

4. Development of Enterprise Systems

There are two primary approaches to developing AI for enterprise applications: customizing existing pre-trained models or developing in-house solutions from the ground up.

Customizing pre-trained models through fine-tuning offers a resource-efficient strategy, allowing businesses to rapidly leverage established technologies while tailoring them to specific requirements. Techniques such as LoRA, PEFT, and RAG enable efficient customization, delivering high performance without necessitating extensive retraining. These methods are particularly advantageous for organizations seeking prompt deployment with reduced computational costs. However, it is essential to recognize that not all models are compatible with every customization

 doi:10.20944/preprints202501.0715.v1

technique. For example, methods like PEFT and DPO require direct access to model parameters, which is not feasible with proprietary models like ChatGPT, Claude, or Gemini, as these platforms provide only API access.

In such cases, alternatives such as prompt engineering, external knowledge integration, or the use of open-source models like Llama, Qwen, or Mistral may be considered. Moreover, utilizing open-source models affords the additional benefit of enabling organizations to host and store model weights internally, thus enhancing model privacy and security.

Conversely, developing in-house models offers complete control over the AI system, allowing organizations to build highly specialized solutions tailored to their unique needs. While this approach is more resource-intensive, it provides significant advantages for entities with the computational resources, expertise, and need for maximum flexibility. In-house development is often preferred when proprietary or off-the-shelf models fail to meet specific industry demands or when total control over the architecture is required.

The decision between these two approaches is contingent upon factors such as available resources, project timelines, performance (accuracy and error metrics) and the degree of customization necessary. Customizing pre-trained models is typically more rapid and cost-effective, while in-house development is more suited to organizations with specialized requirements that necessitate full autonomy over the AI system.

This report also includes a collection of relevant resources, including code and reference papers, to assist users in applying these methods to their own LLMs as needed.

4.1. Enterprise Adoption Through Customization and Fine-Tuning

Enterprises looking to adopt Large Language Models (LLMs) can significantly benefit from customization and fine-tuning techniques that enable domain-specific adaptation with minimal computational resources.

Fine-tuning pre-trained models using methods like LoRA (Low-Rank Adaptation) and Parameter-Efficient Fine-Tuning (PEFT) allows organizations to adapt models to their specific requirements even with smaller datasets and constrained computing power. These methods maintain high model performance while minimizing resource consumption, making them ideal for enterprises with limited computational capacity. Additionally, optimization techniques such as Bayesian Optimization and Direct Preference Optimization (DPO) can further refine model parameters, aligning outputs with organizational preferences. However, it is important to reiterate that methods like PEFT and DPO require direct access to model parameters, which is not feasible with proprietary models, such as ChatGPT, Claude, or Gemini, as they only provide API access and do not allow modifications at the model parameter level. As discussed earlier, in such cases, alternatives such as prompt engineering, external knowledge integration, or utilizing open-source models like Llama, Qwen, or Mistral may provide viable solutions. Open-source models also offer the advantage of enabling organizations to host and store model weights internally, improving privacy and security.

In addition, advanced techniques like Reinforcement Learning and Retrieval Augmented Generation (RAG) contribute to the continuous, dynamic enhancement of model capabilities, allowing models to adapt to evolving requirements and domain-specific data.

 doi:10.20944/preprints202501.0715.v1

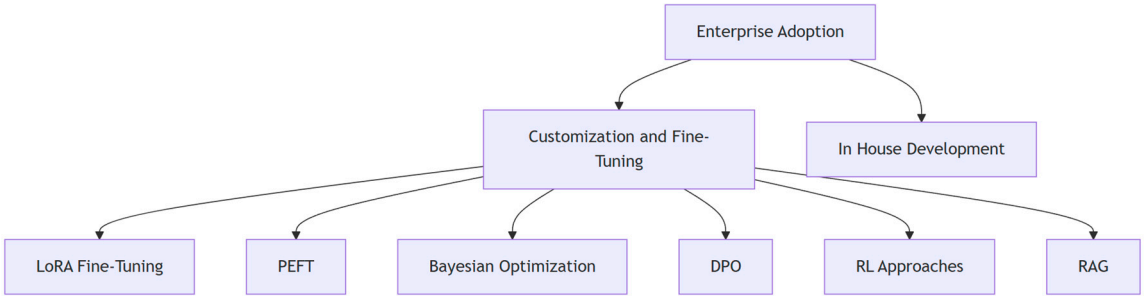

Figure 3. Enterprise adoption methods.

These customization techniques offer a flexible, cost-effective approach for enterprises to integrate powerful AI solutions into their systems, enabling scalable adaptation without the high costs associated with training models from scratch.

4.2. Parameter-Efficient Fine-Tuning (PEFT) Techniques

Parameter-Efficient Fine-Tuning (PEFT) encompasses a family of techniques designed to adapt large language models while minimizing computational overhead and memory usage. The method works by employing various approaches such as quantization, adapter layers, LoRA and prompt tuning to modify only a small subset of model parameters while keeping the majority frozen. For example, QLoRA (Quantized Low-Rank Adaptation) combines 4-bit quantization with low-rank matrix decomposition, reducing memory requirements by up to 75% compared to traditional fine-tuning methods.

PEFT techniques dramatically reduce memory usage and training time, making it possible to fine-tune large models on consumer hardware. The approach maintains model performance while requiring significantly fewer computational resources than full fine-tuning. The primary disadvantages include potential performance degradation on complex tasks due to reduced parameter count, the need for careful hyperparameter selection, and possible quantization artifacts affecting model outputs.

PEFT include several ways to drastically reduce model size and resource usage such as:

- Low-Rank Adaptation (LoRA): One of the most popular and efficient PEFT methods
- Adapter Layers: Introduces small trainable modules between frozen layers
- Prompt Tuning: Employs learnable soft prompts
- Prefix Tuning: Adds trainable prefix tokens

4.3. Fine-Tuning with LoRA (Low-Rank Adaptation)

Introduced by Microsoft Research, Low-Rank Adaptation (Hu et. al, 2021) is a highly efficient fine-tuning technique that enables the customization of large language models with significantly reduced computational resources. This approach operates by freezing the original model's weights and introducing trainable low-rank matrices, which adjust only a subset of the model's parameters, as opposed to the full parameter updates required in traditional fine-tuning. LoRA achieves this by decomposing weight matrices into smaller, lower-rank matrices, thereby reducing the number of trainable parameters from billions to millions. For instance, LoRA can reduce the 175 billion parameters of GPT-3 to approximately 17.5 million trainable parameters.

LoRA offers several key advantages, particularly in terms of memory and computational efficiency. By reducing the number of trainable parameters, it lowers memory requirements and training time, making it feasible to fine-tune large models on consumer-grade GPUs and shortening the training process from weeks to hours. Importantly, LoRA does not introduce additional inference

 doi:10.20944/preprints202501.0715.v1

latency, as the trainable matrices are merged with the frozen model weights during deployment. This ensures that the fine-tuned model retains the performance of the original model while offering the benefits of task specialization.

LoRA also allows organizations to maintain multiple specialized versions of their models with minimal storage overhead. This flexibility makes it easy to switch between different tasks or domains by simply swapping the LoRA weights, thereby enabling efficient task-specific adaptation without incurring significant storage costs. As a result, LoRA provides substantial cost savings while often achieving performance comparable to, or even exceeding, that of traditional full fine-tuning.

LoRA-fine-tuned models become highly specialized for specific tasks or domains, which may necessitate additional fine-tuning or readjustment when new requirements emerge. While LoRA preserves the original model's general knowledge, it may not capture the full range of adaptations that could be achieved through complete fine-tuning, potentially limiting its applicability in cases where broader model adjustments are required. Therefore, while LoRA is an efficient method for domain adaptation, its scope may be restricted in highly dynamic or rapidly evolving application environments.

To conclude this section, a recommendation of relevant papers and code implementations are provided:

Code Implementation of LoRA: https://github.com/microsoft/LoRA

Collection of Papers on LoRA:

- LoRA+: Efficient Low Rank Adaptation of Large Models (Hayou et. al., 2024)
- Sparse Low-rank Adaptation of Pre-trained Language Models (Ding et. al., 2023)
- QA-LoRA: Quantization-Aware Low-Rank Adaptation of Large Language Models (Xu et. al., 2023)
- LoRA-FA: Memory-efficient Low-rank Adaptation for Large Language Models Fine-tuning (Zhang et. al., 2023)

A brief reading of these research papers would help understand application of LoRA method in different applications and constraints.

4.4. Adapter Layers

Introduced in 2023, adapter layers for Parameter-Efficient Fine-Tuning present an approach to model adaptation by introducing small, trainable modules into pre-trained neural networks while keeping the original model parameters frozen. This method strategically places these modules between the frozen layers of the pre-trained model, typically after attention mechanisms and feed-forward networks. The adapter architecture generally consists of down-projection and up-projection layers, utilizing a bottleneck structure to minimize the number of trainable parameters. This design is essential for reducing the memory footprint while preserving the performance of the original model.

The implementation of adapter layers follows a structured pattern that typically includes a down-projection layer, an activation function (e.g., ReLU), an up-projection layer, and layer normalization. The down-projection layer reduces the input dimensionality, the activation function introduces non-linearity, the up-projection layer restores the original dimensionality, and layer normalization ensures stable training. This structure enables efficient parameter utilization while maintaining high model performance.(Hu et. al., 2023)

One of the significant advantages of adapter layers lies in their parameter efficiency. With only a small number of trainable parameters—often less than 1% of the original model—adapter layers drastically reduce the computational resources required for fine-tuning. This efficiency ensures that the model retains its high quality while significantly decreasing memory requirements. Additionally, adapter layers support the efficient storage and retrieval of multiple task-specific adaptations, making them ideal for multi-task learning, domain adaptation, and rapid prototyping scenarios.

 doi:10.20944/preprints202501.0715.v1

In terms of performance, adapter layers achieve results that are near comparable to full fine-tuning. The pre-trained model's knowledge is effectively preserved while achieving robust generalization across various tasks.

Adapter layers also offer significant resource optimization, substantially reducing GPU memory consumption and enabling training on consumer-grade hardware. The reduced memory demands and shorter training times make adapter layers an attractive alternative to full fine-tuning, especially in resource-constrained environments.

The practical applications of adapter layers are diverse and include domain adaptation for specialized tasks, multi-task learning, and scenarios requiring rapid experimentation. By providing a cost-effective means of customizing pre-trained models, adapter layers have become an essential tool for organizations looking to adapt large-scale models to specific needs with minimal computational overhead.

Code Implementation of Adapter tuning:

https://github.com/AGI-Edgerunners/LLM-Adapters

Collection of Papers on Adapter tuning:

- X-PEFT: eXtremely Parameter-Efficient Fine-Tuning for Extreme Multi-Profile Scenarios (Kwak & Kim, 2024)
- Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey (Han et. al., 2024)
- Compacter: Efficient Low-Rank Hypercomplex Adapter Layers (Mahabadi et. al., 2021)

A brief reading of these research papers would help how Adapters work, and how Adapter tuning can help in the fine tuning of LLMs for efficient usage of resources.

4.5. Prompt Tuning

Prompt tuning is an important technique within the Parameter-Efficient Fine-Tuning (PEFT) basket for fine tuning NLP models, focusing on the modification of input representations rather than altering the parameters of a pre-trained model. This approach enables the adaptation of large language models (LLMs) to task-specific requirements while minimizing computational overhead. Prompt tuning is generally classified into two primary categories: hard prompt tuning and soft prompt tuning. Both methods aim to guide the model's behavior by shaping input data in ways that align with the objectives of a given task. By avoiding changes to the model's parameters, prompt tuning presents a resource-efficient means of customizing pre-trained models.

Hard prompt tuning involves the manual construction of textual instructions that are appended to the input data in order to provide task-specific guidance. For instance, a hard prompt for a classification task might consist of instructions such as "Classify this sentence as positive or negative." These prompts are explicitly designed by human experts and directly influence the model's response based on clear, interpretable instructions. The straightforward nature and interpretability of hard prompts make them useful for the rapid adaptation of models to new tasks. However, their effectiveness is contingent upon the expertise of the designer, and the process of crafting precise prompts can prove challenging, particularly for more complex tasks. Furthermore, hard prompts may not fully exploit the latent knowledge within the model, potentially restricting their generalization capabilities. Although Hard prompts are technically PeFT, they are not considered PeFT (Xing et. al, 2024)

In contrast, soft prompt tuning replaces manually crafted textual instructions with learnable continuous vectors that are optimized during training (Lester et. al, 2021). These vectors are embedded directly into the model's input layer, facilitating a more automated approach to task-specific adaptation. Unlike hard prompts, soft prompts capture complex patterns and relationships within the model's knowledge space through optimization. Soft prompts demand significantly fewer computational resources compared to traditional fine-tuning, as only a limited set of parameters are

 doi:10.20944/preprints202501.0715.v1

trained. This characteristic makes them particularly well-suited for scenarios with constrained computational resources or where frequent adaptations are necessary. Additionally, soft prompts have demonstrated strong performance across a variety of tasks by effectively leveraging the model's inherent understanding of language and context.

Both hard and soft forms of prompt tuning are widely applied in various natural language processing tasks, including sentiment analysis, summarization, translation, and question answering. Hard prompts are typically employed in situations where quick deployment or interpretability is prioritized, whereas soft prompts are preferred for tasks requiring greater adaptability and efficiency.

Prompt tuning provides several notable advantages, such as computational efficiency, adaptability, and ease of implementation. While hard prompts are relatively straightforward to design and interpret, they may require considerable effort for more complex tasks. Soft prompts are highly scalable and demonstrate robust performance across diverse tasks, although they may lack the direct interpretability associated with hard prompts.

Code Implementation of Prompt tuning:

https://github.com/google-research/prompt-tuning_

Collection of Papers on Prompt tuning:

- The Power of Scale for Parameter-Efficient Prompt Tuning (Lester et. al, 2021)
- Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm (Reynolds & McDonell, 2021)

4.6. Bayesian Optimization

Bayesian optimization (BO) is a probabilistic model-based approach designed to efficiently optimize complex, expensive-to-evaluate functions. In machine learning, particularly with Large Language Models (LLMs), BO is employed to fine-tune hyperparameters, select optimal prompts, and enhance model performance.

LLMs possess numerous hyperparameters, such as learning rates, batch sizes, and architectural configurations. Manually tuning these parameters is resource-intensive. BO automates this process by constructing a probabilistic model of the objective function and iteratively selecting hyperparameters that are likely to improve performance. (Liu et. al, 2024) This method balances exploration of new hyperparameter regions with exploitation of known promising areas, leading to more efficient optimization.

The effectiveness of LLMs often depends on the prompts provided. Selecting optimal prompts can be formulated as a combinatorial optimization problem. Applying BO in a continuous embedding of the combinatorial space allows for efficient search of effective prompts without requiring access to the LLM's internal parameters. This approach has been demonstrated to enhance performance across various tasks (Sabbatella et. al, 2023).

In applications where user preferences guide model outputs, BO can be utilized to actively elicit preferences through natural language queries (Austin et. al, 2023). By integrating LLMs with BO, systems can generate queries that effectively reduce uncertainty over item utilities, leading to improved recommendations. This method has shown significant improvements in mean average precision after multiple turns of cold-start natural language preference elicitation dialogue.

To incorporate Bayesian optimization into machine learning workflows, one should define the objective function clearly, select a surrogate model to approximate the objective function, choose an acquisition function (e.g., Expected Improvement, Upper Confidence Bound) that guides the search for optimal configurations by balancing exploration and exploitation, and engage in iterative optimization by using the acquisition function to select new configurations, evaluate them, and update the surrogate model with the new data. This process is repeated until a stopping criterion is met, such as a predefined number of iterations or satisfactory performance.

Bayesian Optimization (BO) offers several advantages when applied to Large Language Models (LLMs). It enhances efficiency by reducing the number of evaluations needed to identify optimal hyperparameters, which is particularly valuable given the computational expense of training LLMs.

 doi:10.20944/preprints202501.0715.v1

Integrating LLMs into BO further accelerates the optimization process by leveraging their pre-trained knowledge and contextual understanding.

However, there are limitations to consider. While BO reduces evaluation costs, updating surrogate models becomes computationally expensive in high-dimensional spaces typical of LLMs. Integrating LLMs into BO frameworks adds further resource demands. Additionally, BO may converge to local optima if the surrogate model or acquisition function is poorly designed. Zero-shot warm starting with LLMs can mitigate this issue but does not eliminate it entirely. Implementing Bayesian Optimization for LLMs requires expertise in probabilistic modeling and optimization algorithms. Incorporating LLMs adds additional complexity due to their large-scale architecture and resource requirements. The success of BO depends on accurate evaluations of hyperparameters. For LLM-enhanced BO, poor-quality data or noisy observations can degrade performance.

Code Implementation of Bayesian Optimization:

https://github.com/tennisonliu/LLAMBO _

Collection of Papers on Bayesian Optimization:

- Large Language Models to Enhance Bayesian Optimization (Liu et. al, 2024)
- Bayesian Optimization with LLM-Based Acquisition Functions for Natural Language Preference Elicitation (Austin et. al, 2024)

4.7. Direct Preference Optimization (DPO)

Direct Preference Optimization (DPO) is an emerging technique designed to align large language models (LLMs) with human preferences using a simplified and efficient training approach. Unlike traditional reinforcement learning-based methods, DPO adjusts model parameters directly through a binary cross-entropy objective, eliminating the need for complex reward modeling. This simplifies the alignment process, transforming it into a binary classification task where the model learns to distinguish between preferred and non-preferred responses. The method operates by utilizing paired examples of preferred and non-preferred responses, with the key steps involving a binary classification loss function. Preferred responses are assigned higher probabilities, while non-preferred ones are suppressed. The loss function is reparameterized to influence the model's policy, enabling alignment with human preferences without requiring an explicit reward signal. Through iterative training, the model continuously adjusts, increasing the likelihood of generating desired outputs while reducing undesired ones (Rafailov, 2024).

One of the main advantages of DPO is its simplicity and efficiency. By bypassing the computational complexity of reward modeling and reinforcement learning, it significantly reduces the training overhead. DPO can be implemented using standard classification techniques, making it more accessible and faster to train than traditional methods. Additionally, minimal hyperparameter tuning is required, further streamlining the process. DPO has demonstrated strong performance across a variety of tasks, such as sentiment modulation, summarization, and dialogue generation. In many cases, it performs comparably to or even surpasses more complex approaches, such as reinforcement learning with human feedback (RLHF). Its straightforward implementation also makes it easier to adopt in real-world applications.

However, DPO does have some limitations. The effectiveness of the method is highly dependent on the quality of the initial supervised fine-tuning. If the model is poorly fine-tuned, it may struggle to align effectively with human preferences. Moreover, DPO tends to prioritize avoiding undesirable outputs over improving the quality of preferred responses, which can limit its ability to generate highly optimized outputs in certain scenarios. Additionally, when preferred and non-preferred responses are very similar, DPO may struggle to differentiate between them, reducing its performance in more nuanced or subtle alignment tasks.

Code Implementation of DPO:

https://github.com/phymhan/llm-dpo

Collection of Papers on DPO:

 doi:10.20944/preprints202501.0715.v1

- Direct Preference Optimization: Your Language Model is Secretly a Reward Model (Rafailov, 2024)

4.8. Reinforcement Learning Approaches

Reinforcement Learning (RL) is a key approach for fine-tuning large language models (LLMs) (Ouyang et. al, 2022), encompassing both traditional RL techniques and Reinforcement Learning from Human Feedback (RLHF). These methods enable models to align with desired behaviors and human preferences by optimizing their outputs through reward-based systems. Traditional RL methods focus on optimizing an agent's policy through interactions with an environment.

Notable approaches include Q-Learning, which is an off-policy method using Q-tables to map state-action pairs to rewards, learning the optimal policy through iterative updates, but struggles with large state-action spaces. SARSA, an on-policy method, updates values based on the actual actions taken, offering a more conservative approach (Gholamian & Huh, 2024).

Monte Carlo methods learn from complete episodes of experience without requiring full knowledge of the environment, making them useful for episodic tasks but computationally expensive. Policy gradient methods directly optimize the policy by computing gradients of expected rewards, particularly suited for continuous action spaces, while temporal difference learning combines Monte Carlo and dynamic programming techniques, enabling learning from incomplete episodes and updating estimates iteratively.

Reinforcement Learning from Human Feedback (RLHF) is a specialized form of RL designed to align LLMs with human preferences by incorporating human feedback into the training process (Kirk et. al, 2023). RLHF involves three main stages: Supervised Fine-Tuning (SFT), where a pre-trained model is fine-tuned using labeled data to establish a baseline for further optimization; Reward Model Training, where a reward model is trained using human feedback, such as ranking responses or providing binary feedback to guide the model's understanding of preferences; and Policy Optimization, where the model is fine-tuned using Proximal Policy Optimization (PPO), which balances exploration and exploitation while optimizing the reward model's outputs. KL divergence penalties are often used to prevent the model from deviating too far from its baseline behavior.

RL and RLHF offer several key advantages. RL enables continuous learning without complete retraining, allowing models to adapt to changing requirements and user needs. It also enhances alignment with specific goals, such as safety, helpfulness, and reduced bias, and improves response quality through iterative optimization based on direct feedback. Furthermore, RLHF improves instruction-following capabilities, making models more reliable in real-world applications. However, both RL and RLHF come with limitations. Designing effective reward functions is challenging, as they must balance competing objectives and avoid unintended behaviors like reward hacking. RLHF also requires substantial human involvement for feedback collection and evaluation, making it resource-intensive and costly to implement at scale. Moreover, RL methods can be prone to training instabilities, particularly when reward models are inaccurate, or optimization leads to undesirable outputs. Both traditional RL and RLHF involve significant computational overhead, especially when scaling to large models like GPT-4 or LLaMA.

In comparison, traditional RL and RLHF differ in their focus and application. Traditional RL optimizes behavior in simulated environments and uses environment-based rewards, employing algorithms like Q-Learning, SARSA, and policy gradients. In contrast, RLHF focuses on aligning LLMs with human preferences, using human-provided feedback and algorithms such as PPO and KL divergence. While traditional RL has broad applicability across domains, RLHF offers stronger alignment with human values but is constrained by challenges such as high costs and reliance on human input. In conclusion, both traditional RL and RLHF are crucial for fine-tuning LLMs, but each method has its strengths and limitations. Traditional RL excels in general-purpose optimization tasks, while RLHF offers a more targeted approach for aligning AI systems with complex human values and preferences. Both methods face significant challenges in terms of computational cost, reward design, and stability.

 doi:10.20944/preprints202501.0715.v1

Code Implementation of RL for NLP tasks:
https://github.com/CarperAI/trlx
Collection of Papers on RL for LLMs:

- Fine-Tuning Language Models with Advantage-Induced Policy Alignment (Zhu et al., 2023)
- Reinforcement Learning for Generative AI: A Survey (Cao et al., 2023)
- Secrets of RLHF in Large Language Models Part I: PPO (Zheng et. al, 2023)
- Pairwise Proximal Policy Optimization: Harnessing Relative Feedback for LLM Alignment (Wu et al., 2023)
- DPO Meets PPO: Reinforced Token Optimization for RLHF" (Zhong et al., 2024)

4.9. Retrieval Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is a framework that enhances Large Language Models (LLMs) by integrating information retrieval with natural language generation. This approach enables LLMs to access and utilize external knowledge sources, thereby improving the accuracy, relevance, and contextual understanding of their outputs. (Gao et. al, 2023) RAG operates through a structured process that combines retrieval and generation.

First, external data, such as documents or databases, is converted into vector embeddings—numerical representations of text—and stored in a vector database. This step facilitates efficient retrieval of relevant information. When a user submits a query, the system retrieves pertinent documents or data from the vector database using techniques like semantic search or similarity matching. (Wu et. al, 2024) The retrieved documents may then be scored and ranked by relevance using a reranker, ensuring the most pertinent information is selected. Finally, the retrieved information is combined with the original query and passed to the LLM, which generates a response that integrates both its pre-trained knowledge and the retrieved data. This process enables RAG systems to provide up-to-date, domain-specific, and contextually accurate responses without the need for retraining the LLM itself.

One of the key advantages of RAG is enhanced accuracy. By incorporating external knowledge, RAG reduces hallucinations (fabricated responses) and improves factual correctness. It ensures that outputs are aligned with current, domain-specific information, making it particularly useful for dynamic fields such as customer support or specialized industries. In addition, RAG offers practical benefits such as real-time updates, as it can access live data sources, ensuring responses remain relevant without retraining the model. Its flexibility also allows integration with any LLM and adaptation to various domains or tasks by connecting to different knowledge bases. RAG further excels in handling queries outside the LLM's training data by retrieving relevant external information, thus improving out-of-distribution performance.

RAG also has certain limitations. The introduction of additional components, such as the retriever, reranker, and vector database, increases system complexity and maintenance requirements. Furthermore, retrieval processes can add computational overhead, potentially causing latency issues during real-time applications. The quality of the generated responses is also heavily dependent on the relevance and accuracy of the retrieved documents. Poorly curated or incomplete knowledge bases can result in suboptimal outputs. Lastly, scaling RAG systems to handle diverse tasks or large datasets may require significant resources for data ingestion, indexing, and retrieval. (Zhao et. al, 2024)

Code Implementation of RAG:
https://github.com/langchain-ai/rag-from-scratch
Collection of Papers on RAG:
awesome-generative-ai-guide/research_updates/rag_research_table.md at main

4.10. Inhouse AI Models

Developing in-house Large Language Models (LLMs) requires a structured systems engineering approach, spanning multiple stages from concept development to deployment and maintenance.

 doi:10.20944/preprints202501.0715.v1

This ensures the creation of highly specialized models that align with organizational goals while addressing the technical and operational challenges that arise during development.

1. Concept Development

The first step involves identifying the specific needs and use cases of the enterprise. A comprehensive needs analysis is performed to evaluate potential applications, define key performance metrics, and establish success criteria for the LLM. This phase helps clarify the business objectives and sets the foundation for subsequent stages of development.

2. Architecture Planning

Once the goals are established, the architecture planning phase begins. This involves evaluating various model architectures, such as transformer-based models, and determining the optimal scale and capabilities for the LLM. The choice of architecture significantly impacts the model's performance, scalability, and training requirements. Additionally, training methodologies must be defined to ensure the model can be effectively trained on the available computational infrastructure.

3. Technical Requirements

In this phase, the computational infrastructure and data management strategies are addressed. High-performance computing clusters, distributed training systems, and scalable storage solutions are essential to handle the massive datasets required for LLM training. A robust network infrastructure is also necessary to manage data movement efficiently during training and inference. Domain-specific data is crucial for model specialization, requiring comprehensive data cleaning, preprocessing pipelines, and quality assurance protocols to ensure the integrity and relevance of the data.

4. Development Process

The development process is divided into several key stages:

- **Model Training**: This involves initial pre-training using large-scale datasets, followed by domain adaptation and specialization. Hyperparameter optimization is performed to fine-tune the model for maximum performance. Throughout this phase, performance validation ensures that the model meets the defined success criteria.
- **Safety and Robustness**: Addressing ethical considerations is a priority in this phase. Bias mitigation techniques and safety protocols are implemented to ensure the model's outputs align with ethical standards and avoid harmful or biased results. Compliance with relevant standards ensures that the model adheres to legal and regulatory requirements.

5. Deployment Considerations

Deployment requires careful planning around scalability, performance, and cost optimization. A scaling strategy is defined to manage resource allocation efficiently, while latency management ensures real-time or near-real-time performance. The deployment phase also requires continuous monitoring of the model's performance, with strategies in place to detect emergent behaviors that could impact operational reliability.

6. Maintenance Protocol

Post-deployment, the model must be regularly updated to incorporate new data and maintain performance. A maintenance protocol is implemented, which includes performance monitoring, periodic retraining, and the detection of any issues related to emergent behavior. Regular updates ensure the model remains effective as requirements and data evolve over time.

7. Engineering Challenges

In-house LLM development is fraught with engineering challenges, particularly related to resource management and model optimization:

- **Resource Management**: The development of LLMs requires significant computational resources, including handling massive datasets and managing the computational demands of

 doi:10.20944/preprints202501.0715.v1

training. Balancing the size of the model with its performance is a critical challenge, as larger models often require exponentially more resources.

- **Model Optimization**: Ensuring that the model generalizes well across different tasks while remaining specialized in its domain is a key challenge. It is important to maintain robustness throughout the development process and manage the interdependencies between various model components to prevent training instability.

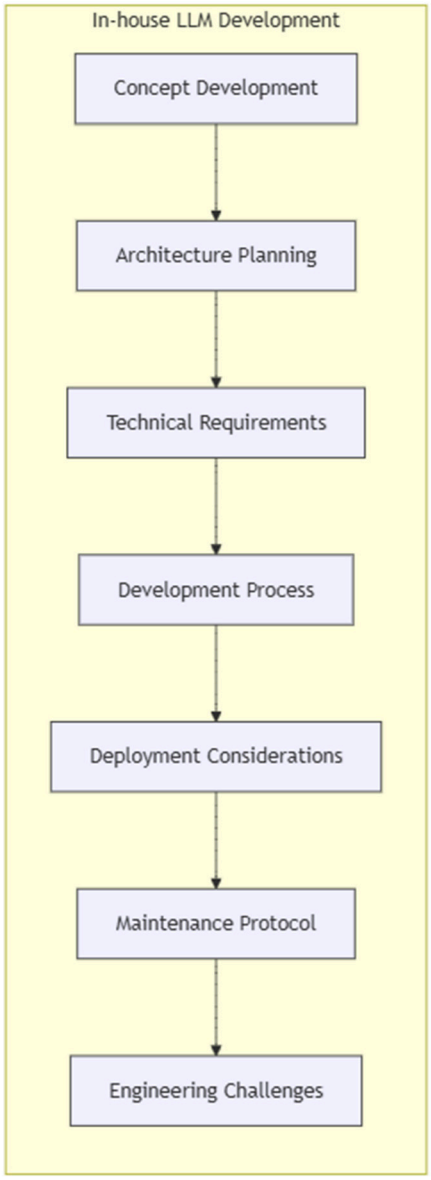

Figure 4. In-House AI development process.

The success of in-house LLM development hinges on the integration of the right infrastructure, expert resources, and a systematic approach that addresses technical, ethical, and operational challenges at each stage of the process.

5. Post-Adoption Stage

After deploying Large Language Models (LLMs) in enterprise applications, it is crucial to implement robust monitoring, maintenance, and security protocols to ensure optimal performance, adaptability, and protection against potential threats.

 doi:10.20944/preprints202501.0715.v1

Deployment and Scaling Strategies

Efficient deployment and scaling of LLMs across enterprise applications require careful consideration of latency, cost, and performance optimization. Implementing strategies such as load balancing, model quantization, and distributed computing can enhance scalability and responsiveness. Additionally, optimizing resource allocation and monitoring system performance are essential to maintain cost-effectiveness and meet user expectations.

Ongoing Model Updates and Maintenance

Establishing a framework for continuous updates and maintenance ensures that LLMs evolve with new data and emerging use cases. Regular retraining with updated datasets, performance benchmarking, and version control are vital practices. Automated testing pipelines and performance monitoring facilitate the detection of issues and the implementation of necessary improvements. This proactive approach helps maintain model accuracy and relevance over time.

Monitoring and Addressing Emergent Behaviors

Implementing robust monitoring systems is essential to detect and address emergent behaviors or anomalies in LLMs. Real-time monitoring tools can track metrics such as latency, token usage, and output quality. Behavioral analysis frameworks, like the Fiddler LLM Enrichments Framework, augment inputs and outputs with quality scores and calculate context-dependent accuracy metrics to identify unexpected model behaviors. Additionally, emergent behavior detection frameworks, such as EmergentEval, apply regression analysis to identify anomalous patterns and monitor for discontinuous performance jumps. These measures ensure adherence to safety, ethics, and compliance standards throughout the model's lifecycle.

Safety and Security Measures

Ensuring the security and safety of LLMs involves implementing comprehensive measures to protect against vulnerabilities and malicious exploitation. Key areas include:

Data Security: Protecting the data used for training and inference is crucial. This involves filtering out sensitive information and biased content from the data used to train LLMs, ensuring the model doesn't learn from the wrong sources.

Model Security: Safeguarding the integrity and confidentiality of the models themselves is essential. Implementing firewalls, intrusion detection systems, and secure network protocols are key measures to prevent unauthorized access and cyber attacks on the infrastructure supporting LLMs.

Infrastructure Security: Protecting the physical and virtual environments that host and run LLMs is vital. This includes securing APIs, ensuring secure communication channels to prevent unauthorized access, and implementing role-based access control.

Ethical Considerations: Addressing ethical concerns, such as bias mitigation and compliance with regulations like HIPAA and GDPR, is crucial to ensure responsible AI usage.

Implementing these measures requires a blend of traditional cybersecurity techniques and protective measures specific to LLMs. Regular security audits, penetration testing, and adherence to frameworks like the NIST Cybersecurity Framework are essential to maintain a robust security posture.

By integrating comprehensive monitoring, maintenance, and security protocols, organizations can ensure that their LLMs operate effectively, adapt to evolving requirements, and remain protected against emerging threats.

6. Systems Engineering Challenges

6.1. Data and Computational Resource Management

 doi:10.20944/preprints202501.0715.v1

One of the primary challenges in implementing Large Language Models (LLMs) in enterprises is the need for massive datasets and computational resources. For example, One of the smallest variations of large language models, the Llama 2 7-billion parameters model uses 20 Gigabytes of RAM. Larger ones require having good in-house infrastructures with 40 Gigabytes of RAM, and Enterprise workstation CPU like Intel Xeon, and dedicated Graphics Card like Nvidia Blackwell.

Engineering teams must design scalable data pipelines to handle large volumes of data and its retrieval by designing scalable systems for data ingestion, preprocessing, and storage, as well as developing computational infrastructure that supports LLM fine-tune level training and inference. We must also keep in mind to manage computational workloads, implement data lifecycle management strategies, and optimize energy consumption to balance performance with environmental considerations.

6.2. Model Safety, Robustness, and Ethical Behavior

Ensuring the safety, privacy and ethical behavior of LLMs is another critical challenge. We must design comprehensive testing frameworks that evaluate model responses across a wide range of test scenarios, edge cases and potentially harmful inputs.

Tackling the problem requires implementing content filtering mechanisms, developing ethical guidelines for model outputs, and creating feedback loops for continuous model improvement. It is crucial to design guardrails to manage situations where the outputs may be problematic in the form of hallucinations in Large Language models. The system must remain reliable and trustworthy especially organizations where it matters a lot.

6.3. Performance Optimization

Managing the complex interdependencies of different components and subcomponents of the systems is often directly proportional to a performance of the model. Data quality for fine tuning tasks, and efficient integration and creation of software pipeline to ensure good real-world performance is a significant challenge. For that we must develop optimization strategies that balance these factors to achieve optimal performance for specific enterprise use cases .

This involves designing adaptive training pipelines that efficiently scale with growing customers and data, while also maintaining or improving performance metric or cutting down costs. Implementation of monitoring tools to continuously evaluate the impact of changing parameters such as number of users and how it affects performance, can help make data-driven decisions to optimize these elements .

Trying techniques to make models less computationally heavy is also one way to do this, by using techniques like Quantization of models and Sharding of databases, one can make the performance much better and less reliant on need for computational resources.

6.4. Bias Mitigation and Fairness

Ensuring bias mitigation and fairness in enterprise LLM applications is a critical systems engineering challenge. Developing evaluation frameworks that assess model outputs across diverse demographic groups and use cases is essential. Additionally, implementing ongoing monitoring and adjustment mechanisms, including feedback loops and audits, helps detect and correct biases that may emerge during operational use to maintain fairness throughout the system's life cycle and beyond.

6.5. Balancing Generalization and Specialization

Another complex challenge is balancing an LLM's ability to generalize across multiple domains while specializing for particular enterprise needs. Engineers must design flexible model architectures and training strategies that enable efficient domain adaptation without compromising the model's broader knowledge base. This involves implementing modular approaches that allow a base model

 doi:10.20944/preprints202501.0715.v1

to be augmented with domain-specific components and developing dynamic fine-tuning pipelines that can quickly adapt the model to new tasks or domains. Evaluation frameworks are needed to assess both the model's general capabilities and its performance on specialized tasks.

6.6. Integration with Existing IT Infrastructure

Integrating LLMs with existing IT infrastructure presents unique challenges. Systems engineers must develop strategies for seamless integration with current enterprise systems, ensuring compatibility with existing data processing workflows and addressing security and compliance concerns. This involves implementing robust data privacy measures and ensuring adherence to data protection regulations such as GDPR. Engineers must also ensure that LLMs are compatible with the organization's broader IT architecture, while maintaining the integrity and efficiency of legacy systems.

6.7. Cost Optimization and Financial Planning

Cost optimization in LLM implementation focuses on financial planning and resource allocation. Enterprises face substantial upfront costs, and systems engineers must conduct thorough cost-benefit analyses to justify these investments. Predicting and measuring return on investment (ROI) over time is essential, considering both tangible and intangible benefits. Financial resources must be strategically allocated across different phases of LLM deployment, from initial setup to ongoing maintenance. Scaling LLM capabilities in a financially sustainable manner and negotiating vendor contracts effectively are also key components of cost optimization.

6.8. Ongoing Maintenance and Adaptation

Ensuring the long-term effectiveness of LLMs requires continuous maintenance and adaptation. Systems engineers must develop frameworks for ongoing model refinement and create efficient mechanisms for rolling out updates with minimal operational disruption. Performance monitoring systems should be in place to track LLM performance over time and identify areas for improvement. Resource-efficient fine-tuning methods are necessary to adapt models to new tasks or domains without requiring full retraining. Finally, systems engineers must plan for the eventual obsolescence or major overhaul of LLM systems as technology advances.

By addressing these challenges, organizations can maximize the potential of LLM technologies while ensuring sustainable, efficient, and ethical deployment in enterprise settings.

7. Conclusions

The expected outcomes of this study offer a holistic framework for enterprises adopting LLM technologies, ensuring a structured and efficient implementation process. The first outcome focuses on system design frameworks and best practices, providing detailed architectural guidelines, scalable infrastructure patterns, and strategies for seamless integration with existing systems. Additionally, it emphasizes maintaining high-quality documentation and quality assurance protocols to ensure reliability. The second outcome delves into customization and deployment strategies, comparing techniques such as PEFT, LoRA, and adapters, while outlining implementation methodologies for various enterprise use cases. Guidelines for resource optimization, performance monitoring, and scalability ensure organizations can meet growing demands without compromising efficiency.

The third outcome addresses key implementation challenges and provides mitigation guidelines for data privacy, security, ethical considerations, and cost management. It emphasizes a risk-aware approach, ensuring that enterprises comply with regulatory standards and sustain long-term viability through robust maintenance procedures. Lastly, the fourth outcome introduces an implementation toolset, comprising technical specifications for infrastructure setup, performance monitoring tools, quality assurance frameworks, and security audit checklists to uphold compliance. Together, these outcomes form a comprehensive roadmap that enables enterprises to effectively

 doi:10.20944/preprints202501.0715.v1

adopt, customize, and scale LLM technologies while mitigating risks and ensuring operational sustainability.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S. and Avila, R., 2023. Gpt-4 technical report. arXiv preprint arXiv:2303.08774.
2. https://www.anthropic.com/news/claude-3-family
3. Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A. and Goyal, A., 2024. The llama 3 herd of models. arXiv preprint arXiv:2407.21783.
4. Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2024. Language models can solve computer tasks. In Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS '23). Curran Associates Inc., Red Hook, NY, USA, Article 1723, 39648–39677.
5. Hou, Y. *et al.* (2024). Large Language Models are Zero-Shot Rankers for Recommender Systems. In: Goharian, N., *et al.* Advances in Information Retrieval. ECIR 2024. Lecture Notes in Computer Science, vol 14609. Springer, Cham. https://doi.org/10.1007/978-3-031-56060-6_24
6. R. Shan and T. Shan, "Enterprise LLMOps: Advancing Large Language Models Operations Practice," 2024 IEEE Cloud Summit, Washington, DC, USA, 2024, pp. 143-148, doi: 10.1109/Cloud-Summit61220.2024.00030.
7. Vaswani, A., 2017. Attention is all you need. Advances in Neural Information Processing Systems.
8. Sanjay Kukreja, Tarun Kumar, Amit Purohit, Abhijit Dasgupta, and Debashis Guha. 2024. A Literature Survey on Open Source Large Language Models. In Proceedings of the 2024 7th International Conference on Computers in Management and Business (ICCMB '24). Association for Computing Machinery, New York, NY, USA, 133–143. https://doi.org/10.1145/3647782.3647803
9. Vorontsov, E., Bozkurt, A., Casson, A. et al. A foundation model for clinical-grade computational pathology and rare cancers detection. Nat Med (2024). https://doi.org/10.1038/s41591-024-03141-0
10. Haltaufderheide, J., Ranisch, R. The ethics of ChatGPT in medicine and healthcare: a systematic review on Large Language Models (LLMs). npj Digit. Med. 7, 183 (2024). https://doi.org/10.1038/s41746-024-01157-x
11. Wang, S., Xu, T., Li, H., Zhang, C., Liang, J., Tang, J., Yu, P.S. and Wen, Q., 2024. Large language models for education: A survey and outlook. arXiv preprint arXiv:2403.18105.
12. Devlin, Jacob, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." North American Chapter of the Association for Computational Linguistics (2019).
13. Radford, Alec, Jeff Wu, Rewon Child, David Luan, Dario Amodei and Ilya Sutskever. "Language Models are Unsupervised Multitask Learners." (2019).
14. Scott Rome, Tianwen Chen, Raphael Tang, Luwei Zhou, and Ferhan Ture. 2024. "Ask Me Anything": How Comcast Uses LLMs to Assist Agents in Real Time. In Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24). Association for Computing Machinery, New York, NY, USA, 2827–2831. https://doi.org/10.1145/3626772.3661345
15. Schillaci, Z., 2024. LLM Adoption Trends and Associated Risks. In Large Language Models in Cybersecurity: Threats, Exposure and Mitigation (pp. 121-128). Cham: Springer Nature Switzerland.
16. J. D'Ambrosio and G. Soremekun, "Systems engineering challenges and MBSE opportunities for automotive system design," 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, Canada, 2017, pp. 2075-2080, doi: 10.1109/SMC.2017.8122925.

a. _____Lora_____

17. Hu, Edward J., 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).
18. Hayou, Soufiane, Nikhil Ghosh, and Bin Yu. "Lora+: Efficient low rank adaptation of large models." *arXiv preprint arXiv:2402.12354* (2024).
19. Ding, Ning, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. "Sparse low-rank adaptation of pre-trained language models." *arXiv preprint arXiv:2311.11696* (2023).

20. Xu, Yuhui, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. "Qa-lora: Quantization-aware low-rank adaptation of large language models." *arXiv preprint arXiv:2309.14717* (2023).
21. Zhang, Longteng, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. "Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning." *arXiv preprint arXiv:2308.03303* (2023).
22. Hu, Zhiqiang, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. "Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models." arXiv preprint arXiv:2304.01933 (2023).
23. Kwak, Namju, and Taesup Kim. "X-PEFT: eXtremely Parameter-Efficient Fine-Tuning for Extreme Multi-Profile Scenarios." arXiv preprint arXiv:2401.16137 (2024).
24. Han, Zeyu, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. "Parameter-efficient fine-tuning for large models: A comprehensive survey." arXiv preprint arXiv:2403.14608 (2024).
25. Karimi Mahabadi, Rabeeh, James Henderson, and Sebastian Ruder. "Compacter: Efficient low-rank hypercomplex adapter layers." Advances in Neural Information Processing Systems 34 (2021): 1022-1035.
26. Jialu Xing, Jianping Liu, Jian Wang, Lulu Sun, Xi Chen, Xunxun Gu, Yingfei Wang, A survey of efficient fine-tuning methods for Vision-Language Models — Prompt and Adapter, Computers & Graphics, Volume 119,2024,103885,ISSN 0097-8493, https://doi.org/10.1016/j.cag.2024.01.012.
27. Lester, Brian, Rami Al-Rfou, and Noah Constant. "The power of scale for parameter-efficient prompt tuning." arXiv preprint arXiv:2104.08691 (2021).
28. Reynolds, Laria, and Kyle McDonell. "Prompt programming for large language models: Beyond the few-shot paradigm." In Extended abstracts of the 2021 CHI conference on human factors in computing systems, pp. 1-7. 2021.
29. Liu, Tennison, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. "Large language models to enhance bayesian optimization." arXiv preprint arXiv:2402.03921 (2024).
30. Sabbatella, Antonio, Andrea Ponti, Antonio Candelieri, Ilaria Giordani, and Francesco Archetti. "A Bayesian approach for prompt optimization in pre-trained language models." arXiv preprint arXiv:2312.00471 (2023).
31. David Austin, Anton Korikov, Armin Toroghi, and Scott Sanner. 2024. Bayesian Optimization with LLM-Based Acquisition Functions for Natural Language Preference Elicitation. In Proceedings of the 18th ACM Conference on Recommender Systems (RecSys '24). Association for Computing Machinery, New York, NY, USA, 74–83. https://doi.org/10.1145/3640457.3688142
32. Rafailov, Rafael, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. "Direct preference optimization: Your language model is secretly a reward model." *Advances in Neural Information Processing Systems* 36 (2024).
33. Gholamian, Sina, and Domingo Huh. "Reinforcement Learning Problem Solving with Large Language Models." *arXiv preprint arXiv:2404.18638* (2024).
34. Zhu, Banghua, Hiteshi Sharma, Felipe Vieira Frujeri, Shi Dong, Chenguang Zhu, Michael I. Jordan, and Jiantao Jiao. "Fine-tuning language models with advantage-induced policy alignment." *arXiv preprint arXiv:2306.02231* (2023).
35. Cao, Yuanjiang, Quan Z. Sheng, Julian McAuley, and Lina Yao. "Reinforcement learning for generative ai: A survey." *arXiv preprint arXiv:2308.14328* (2023).
36. Zheng, Rui, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu et al. "Secrets of rlhf in large language models part i: Ppo." *arXiv preprint arXiv:2307.04964* (2023).
37. Wu, Tianhao, Banghua Zhu, Ruoyu Zhang, Zhaojin Wen, Kannan Ramchandran, and Jiantao Jiao. "Pairwise proximal policy optimization: Harnessing relative feedback for llm alignment." *arXiv preprint arXiv:2310.00212* (2023).
38. Zhong, Han, Guhao Feng, Wei Xiong, Xinle Cheng, Li Zhao, Di He, Jiang Bian, and Liwei Wang. "Dpo meets ppo: Reinforced token optimization for rlhf." *arXiv preprint arXiv:2404.18922* (2024).
39. Gao, Yunfan, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. "Retrieval-augmented generation for large language models: A survey." *arXiv preprint arXiv:2312.10997* (2023).

 doi:10.20944/preprints202501.0715.v1

40. Wu, Shangyu, Ying Xiong, Yufei Cui, Haolun Wu, Can Chen, Ye Yuan, Lianming Huang et al. "Retrieval-augmented generation for natural language processing: A survey." arXiv preprint arXiv:2407.13193 (2024).
41. Zhao, Penghao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. "Retrieval-augmented generation for ai-generated content: A survey." *arXiv preprint arXiv:2402.19473* (2024).