Skeletal Framework for Large Language Model Operations (LLMOps) using existing CI/CD pipeline

Narayana Darapaneni Anwesh Reddy Sudha B G Karthick Kumarasamy

Nethaji BS Shweta Kulkarni Surendar Raj Premkumar R

January 2025

Contents

List of Figures

Abstract

Large language models (LLMs) have changed many industries, but their use in production comes with many challenges such as scaling up, being reliable, and following ethical rules. This paper shows a simple form for LLM Operations (LLMOps), which helps to manage key parts in the lifecycle of an LLM. The framework points out basic steps like implementing models into action, improving infrastructure, monitoring on performance, and always making things better while trying to save money and meet ethical needs. As a broad guide, we look at ways in the CI/CD pipeline to make it easier to keep the LLM model working well in production with focus on growth, potential effectiveness and proper AI conduct.

Chapter 1

Introduction

1.1 Background

1.1.1 LLMs-Large Language Models

Natural language processing originated from computational linguistics, starting in the 1950s. Early models had rule-based systems that tried to use strict grammar rules for language. But these did work well for complex real-world language. In the 1990s, statistical methods like Hidden Markov Models (HMMs) and n-grams came up using big data to identify language patterns with probabilities. This set up later probabilistic ways for NLP. In 2000s, machine learning techniques such as SVM (Support Vector Machines) and decision trees were used in NLP tasks too. But what really changed the game was deep learning, Deep learning which started gaining attraction in the 2010s with models like RNN (recurrent neaural networks) and LSTM (long-short-term memory) .They showed they could handle sequences of text better by analyzing large datasets effectively. In 2017, a key paper by Vaswani et al., called "Attention is All You Need"[1] brought forth the Transformer model. Unlike earlier models like RNNs or LSTMs, Transformers used self-attention allowing words to be processed at once instead of one after another leading to much quicker and larger scale improvements within NLP models overall that gave rise to Large Language Models also known as LLMs.

1.1.2 Evolution of MLOps

The growing need of machine learning models in real-world settings has resulted in new field called **MLOps** (Machine Learning operations)[2]. MLOps is the collection of methods, tools, and procedures aimed at deploying, evaluating, and monitoring whether ML models working well in actual applications[3]. MLOps are important for making sure these models can be expanded, combined with other systems, and applied safely and dependably.

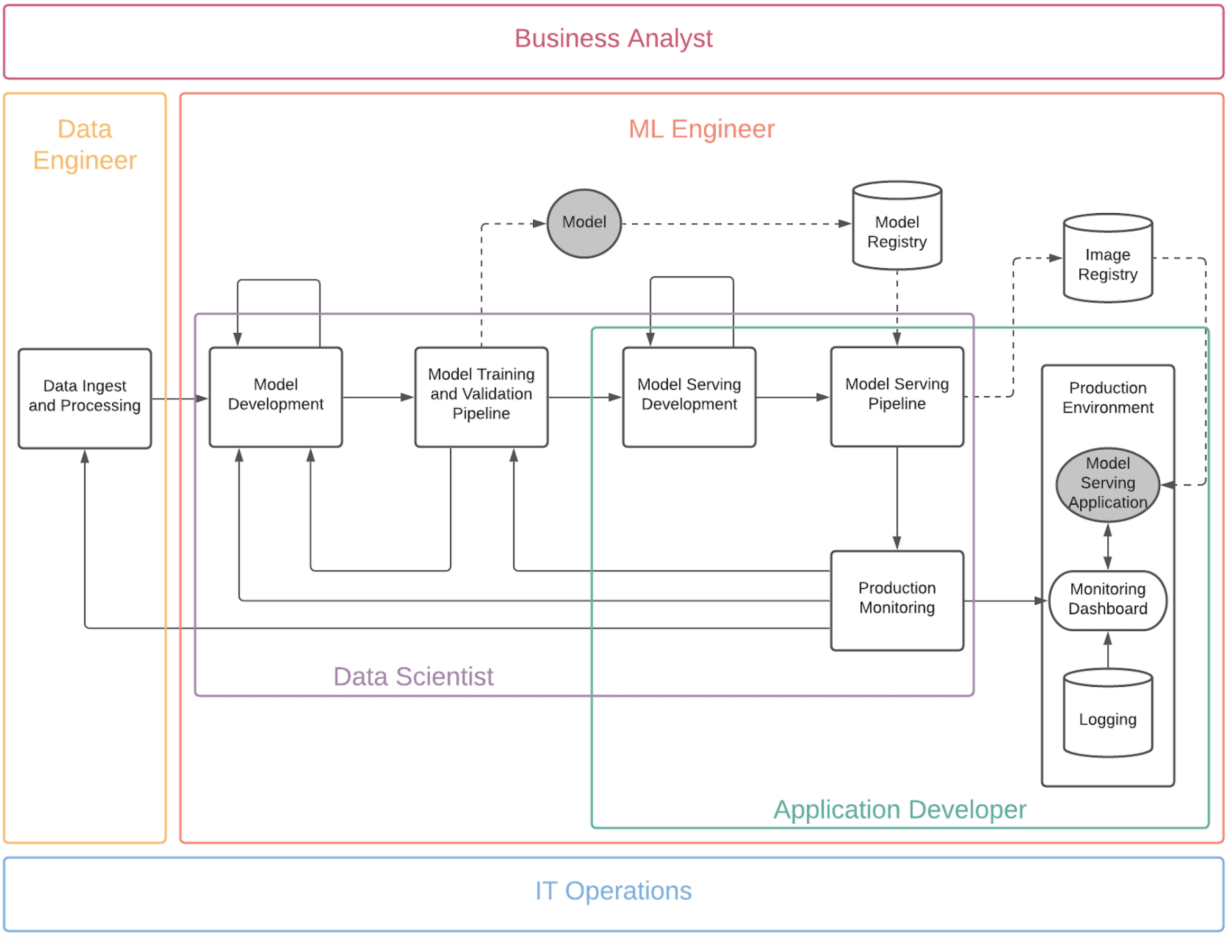

Figure 1.1: MLOps Architecture
[4]

1.1.3 Emergence of FMOps

MLOps is about managing how custom models lifecycle,it needs loads of work in gathering data, training the model, and getting it to production.But **FMOps** (foundation Model Operations) [5] deviates here—it uses pre-trained models to make things easier for development and makes the deployment of Generative AI faster.

Generative AI, known as **GenAI**, refers to artificial intelligence that can generate new content like words, pictures, tunes, clips, and others. These systems get trained on piles of information represented as dataset and learn patterns. After learning this way, they can produce fresh outputs based on what they've figured out before. This makes them useful for many different uses.

Foundation Models do not embed the machine learning model but connect to pre-trained model through an API call.Foundational Model address three main categories of generative AI Use cases.

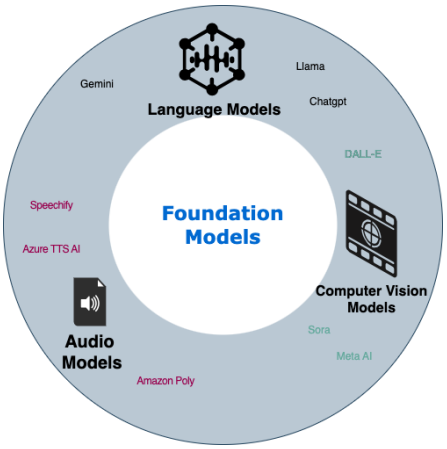

Figure 1.2: Foundation Models

1.1.4 Exaptation of LLMOps

Large Language Model Operations (LLMOps) involve the methods, plans, and tools for managing large language models when they are used in real-world production systems.[6] LLMOps are subsets of FMOps which in turn is a subset of larger MLOps.

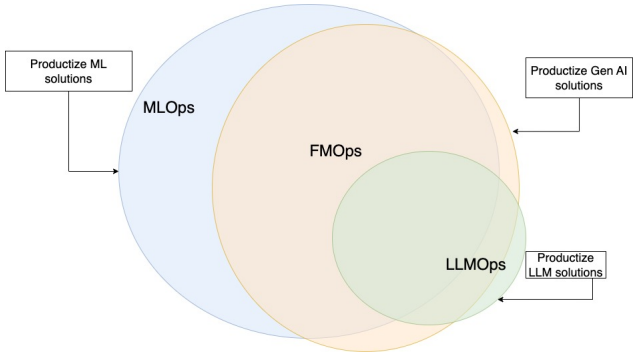

Figure 1.3: LLMOps

Training LLMs(Text-to-Text) like GPT, Claude and LLaMa can be an expensive affair. Many organizations lack the budget, advanced infrastructure, and machine learning expertise needed to productionize them.

Some Unique Aspects required for managing Large language Model(LLMs)[7] are listed below:

1. **Heavy Computational Resources:** Training and fine-tunning a large language model involves significant large calculations and these require special hardware like GPU's.
2. **Transfer Learning:**This nature of LLMs which commence with "Foundation model",and then specializes with fine-tunning,achieving high levels of performance would require special attention in managing the model in production.
3. **Human Feedback:**Human feedback is crucial for determining how well the model is working.The gathered feedback can be used for fine-tunning.This approach of incorporating real-world user feedback is not commonly seen in traditional MLOps.
4. **Performance Metrics:**Evaluation large language Models requires a different set of metric and scoring such as bilingual evaluation understudy (BLEU) and Recall-Oriented understudy for gisting evalua-

tion(ROUGE)[8].These metrics require extra consideration unlike traditional metrics like accuracy,AUC,F1 score.

5. **Prompt Engineering:**Prompt engineering is where we create and improve the inputs (prompts) given to a large language model (LLM) to control how it acts and produce the desired outputs.It is a critical skill for working with LLMs because these models respond to instructions in a highly contextual and refined manner.

As a result, instead of building a foundation model from scratch, many businesses turn to more cost-effective alternatives to integrate LLMs into their operations. However, these options still require a learning process and the right tools to ensure successful development, deployment, and ongoing maintenance.

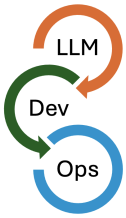

Figure 1.4: LLMops flow

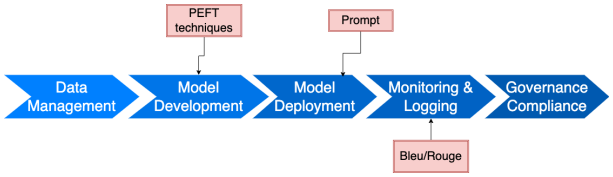

Figure 1.5: Specific stages for LLMOPS

Chapter 2

Literature Review

2.1 LLMOps: Definitions, Framework and Best Practices

2.1.1 Introduction

This paper[5] explains how using large language models is different from regular ML solutions. Experiences from the MLOps model show that more complexity arises when scaling. To address problems with LLMs being deployed, a subset of FMOps(Foundation model operations)was created. It helps businesses deploy, monitor, and update their LLMs easily.

This paper offers a clear definition of LLMOps and a solid framework, along with best practices gathered from our experience in various areas. Lastly, it aims to guide AI professionals who wish to implement their GenAI applications smoothly.

2.1.2 Architecture

The basic LLMOps framework that serves as a support for the enterprise-level GenAI solutions have been developed recently. The LLMOps framework has the following key steps:

1. **Creating a Project Charter** This step is about setting project aims, limits, and measures of success. It is crucial to include main stakeholders and experts to build a clear charter with specific goals and expectations.

2. **Choosing Generative AI Use Cases** In this phase, valuable GenAI use cases and complexity is evaluated. Evaluating use case feasibility with a qualification matrix that considers all significant points regarding risks and returns is done.

3. **Solution Design** Building a scalable and cost-efficient architecture is vital. This involves working with experts and ML engineers to construct strong architectures.

4. **Data Pipeline Design** This step includes figuring out data sources, how to integrate data, and how data will flow. This also aids in forming a solid data strategy and pinpointing data sources.

5. **Collecting Data** The necessary contextual data for training the LLMs must be available to minimize errors. Key actions include using automated scripts for data collection from company systems and databases to get contextualized and standardized data formats, as well as using data connectors that work well with different sources.

6. **Changing Data** This phase focuses on cleaning noisy text data and addressing any missing data. It also involves fixing inconsistencies and removing duplicates.

7. **Storing Data** Traditional database systems are not suitable or fast enough for retrieving unstructured data in the form of embeddings. Thus, using vector databases is advised for this type of data. There are many vector databases available today, both open-source and closed-source, which we will discuss later.

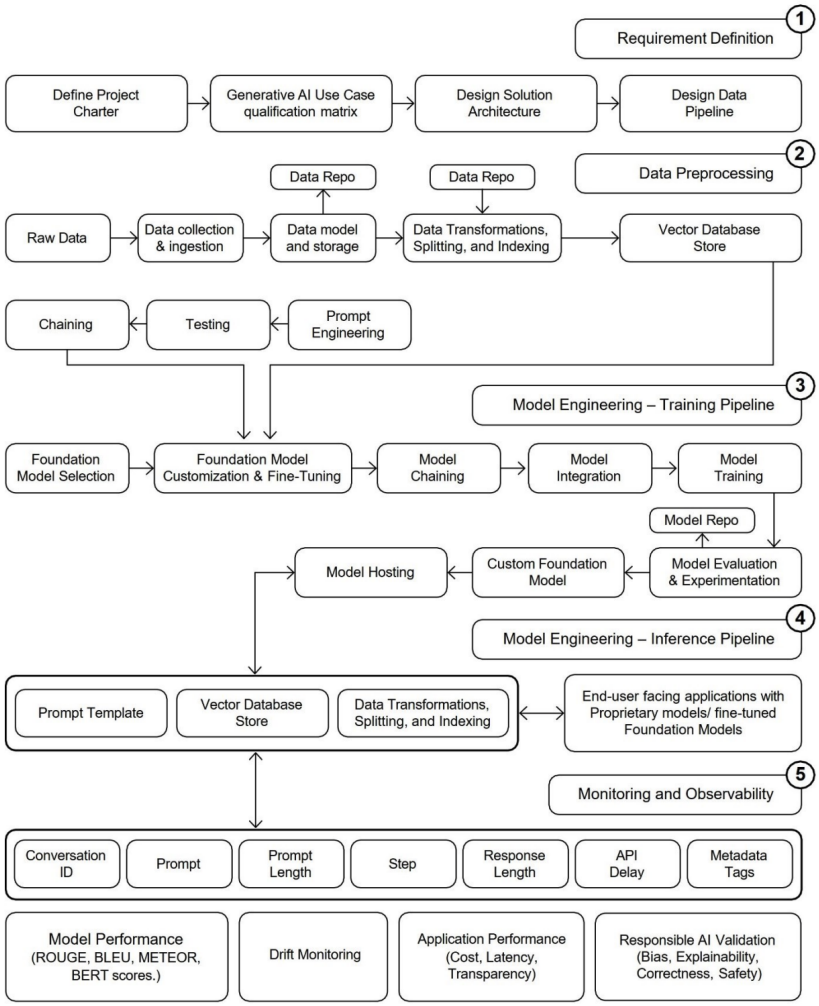

Figure 2.1: LLMOps Framework
[5]

8. **Handling Data** Besides standard AI data pipeline creation,GenAI incorporates techniques like prompt engineering, which must be thoughtfully designed to ensure that the LLM produces the intended output. Training data for an LLM needs to be tokenized and normalized. The next two steps are crucial for data preparation: Versioning Data keeps track of the development process. Encrypting Data and Access Controls protect data through encryption and enforce access restrictions, like role-based access, to guarantee secure data management.

9. **Training the Model** After preparing the data through the steps outlined, the LLM is considered ready for training. The data scientist must decide between prompt tuning, few-shot tuning, or fine-tuning the LLM to create a tailored model. It's important to fine-tune the model performance using well-known libraries and methods to improve the model's abilities in particular areas with business-specific contextual data. This requires a careful balance between performance and expenses to determine which tuning method fits the problem best.

10. **Evaluating the Model** When evaluating trained or fine-tuned Large Language Models (LLMs), set metrics are used on benchmark tasks to assess their performance. This includes checking their ability

to yield accurate, coherent, and contextual outputs. Additionally, metrics such as factual accuracy, robustness, and trustworthiness of the model outputs are also important to consider.

11. **Governing the Model** An LLM must be monitored for performance, making adjustments when necessary and decommissioning it when it's no longer valuable. This entails defining and identifying relevant metrics. Every run of the model in production must capture the prompt, response, latency, and metadata. It is also considered good to set thresholds and alerts to retrain the LLM when deviations occur.

12. **Model Inference** For model inference, it is important to choose a suitable deployment strategy that fits the budget, security needs, and infrastructure demands. Specific architectural patterns should be identified to implement the fine-tuned models in production. Additionally, matters like concurrent users, scalability, and performance must be addressed for using available LLM options.

13. **Model Serving Model** serving refers to the approach to make the LLM accessible to users. The choice of LLM serving hinges on the associated costs. It considers how to best utilize computing resources. While many cloud providers deliver these services, toolkits like Ray Serve and others offer features like custom response streaming and dynamic request batching for LLMs. We will cover tool selection further in the next section.

14. **Model Monitoring** The accuracy needed from LLM outputs changes with each use case, even within the same field; for instance, a clinician's treatment choice requires higher accuracy than sorting patient log records.

2.2 Large Language Model Operations (LLMOps): Definition, Challenges, and Lifecycle Management

2.2.1 Introduction

Designing and maintaining large language models (LLMs) like GPT-3 and BERT is really hard and needs a lot of resources. This involves huge amounts of data, strong computer power, and high costs. There are problems with keeping data good, reducing biases, and dealing with old knowledge. Also, fine-tuning can be inefficient because it uses too much memory.

Even with these issues, LLMs have useful functions; they can produce text that makes sense in context. To make using them easier, Large Language Model Operations (LLMOps) has come up to handle the lifecycle of LLMs. On the other hand, Foundation Model Operations (FMOps) focuses on using generative AI for different applications.

This paper [6] looks at LLMOps for model creators who train models and those who adjust them for special tasks. It explains the method used, points out major challenges, and describes how to put this into practice.

2.2.2 Architecture

The stages identified and classified in LLM lifecyle are:

Data Management – Gathers, processes, tags, and sorts data for good input. Involves storage options, version keeping, and making things repeatable.

1. **Model Selection** – Picks the best pre-made or custom model based on use case, dataset details, and limits in computing power.

2. **Adaptation** – Tweaks models through prompt tricks, fine-tuning stuff, or retrieval-boosted generation (RAG) for specific tasks in a field.

3. **Evaluation** – Checks how well the model works using old-school methods and LLM-specific measures like accuracy checkups, coherence tests, and relevance scoring. Automated tools speed up this checking process.

4. **Deployment** – Deals with CI/CD setups, growth potential issues, performance fixes-up of how things go fast-efficient techniques like reducing size.

5. **Monitoring** – Watches over models doing their job including hallucinations, timing responses back to users' questions to keep a steady operation going well.

6. **Data Privacy Security** – Applies ways to hide info (anonymization), converts it into codes (encryption), trains against attacks (adversarial training), plus conducts inspections to protect private info while sticking to rules.

7. **Ethics Fairness** – Tackles bias issues; aims to be fair by involving diverse participants in the responsible work of AI that is rightly developed.

2.3 Enterprise LLMOps: Advancing Large Language Models Operations Practice

2.3.1 Introduction

This paper [7] looks at how LLMs work in companies and gives a structure to help businesses use these models well. The aim is to support smart, fast, and careful use of LLMs.Problems with Using LLMs in Companies as emumerated below:

1. **Technical Issues** Making LLMs suitable for business while fitting them into what's already there. Handling big data while keeping it safe and private (like following GDPR rules). Keeping accuracy and dependability so as not to get wrong or biased results.

2. **Ethical and Legal Matters** LLMs might bring along biases from data they were trained on, which can be a problem ethically in delicate areas. The "black box" part of LLMs makes them hard to explain, which is bad in fields like finance and healthcare. Keeping up with changing AI laws across different places.

3. **Money and Resource Challenges** Big upfront cost for tech, data, and skilled workers. Smartly using computing power and people resources for setup and upkeep. Regular updates are needed to keep things running well.

2.3.2 Architecture

The 4D Steps of LLMOps, The LLM steps have four main parts: **Discover, Distill, Deploy, and Deliver (4D)**, making it easier to operatinalize on Large Language Models (LLMs).

1. **Discover** – Looks for the need for an LLM, checks what it can do, and sets goals. Important tasks are getting data ready, checking base models, and creating prompts.

2. **Distill** – Works on processing data and training the model. This stage includes fine-tuning the model, making it run better (like quantization), and being ready to grow.

3. **Deploy**– Links the trained model to IT systems while managing time of use, containers, API linking up with cloud/on-site putting in action.

4. **Deliver**– Keeps check on how well the model works long-term through watching over its performance, keeping safe standards met with rules while improving based on real-life results.

Problems in LLMOps

1. Scalability: Solved with cloud methods plus expandable system designs made easy.

2. New Model Changes Keeping Track: Handled using continuous integration/deployment pathways along with strong version control measures.

3. Fairness Issues: Keeping fairness clear is done via rule-making groups focusing safety placements with ongoing reviews.

This plan helps businesses get most out of their LLMs while sorting out issues linked to performance faults as well as security matters dealing staying fair through AI uses.

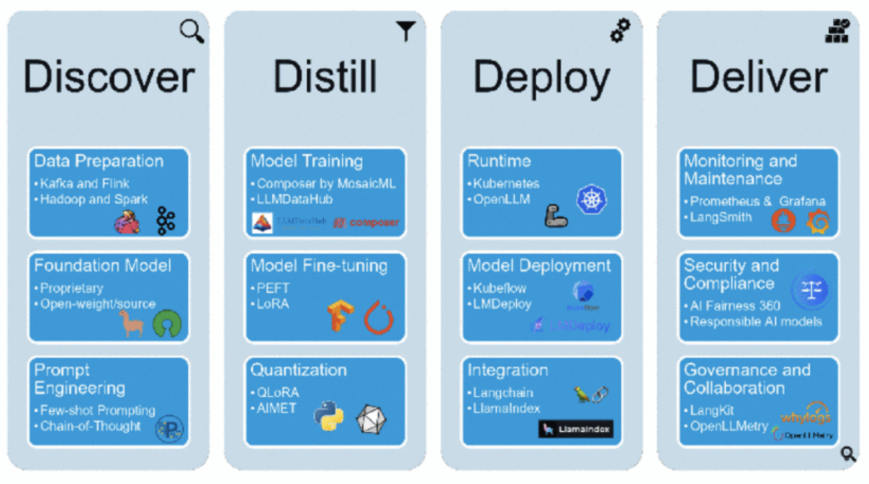

Figure 2.2: Four Ds Model
[7]

The effective use of LLMOps in companies depends on several tech tools and open-source software that help with making, deploying, and monitoring on models.

1. Data Tools
   Apache Kafka - Handles live data streams.
   Apache Flink - Works for stream and batch stuff.
   Hadoop Spark - Process big chunks of data to train and tweak LLMs.

2. Models
   Proprietary models: OpenAI (GPT), Google (Gemini), Anthropic (Claude).
   Open-source models: Eleuther AI, xAI Grok-1 for managing LLMOps separately.

3. Prompt Tricks
   Few-shot prompting - Gives examples to help guide what LLM says.
   Chain of Thought reasoning - Improves logical thinking when solving problems.

4. Training Models
   Composer (MosaicML) – Makes large training easier.
   LLMDataHub – Place for chatbot training info.

5. Fine-Tuning Models
   PEFT – Changes models with less computing power.

LoRA – Saves costs by changing fewer parameters during fine-tuning.
QLoRA – Uses smaller weights to make fine-tuning faster.

6. Quantization
   AIMET – Makes LLM work better through quantizing and compressing it.

7. Running It
   Docker Kubernetes – Helps deploy LLMs in a scalable way using containers.
   OpenLLM – Open-source way to manage how LLM works across settings.

8. Deploying Models
   Kubeflow - Automates machine learning tasks within Kubernetes setups.
   LMDeploy – Focuses on compressing and serving LLMs well.

9. Chaining
   LangChain - Gives structure for building NLP apps using LLMs.
   LlamaIndex - Merges structured/unstructured info into applications powered by LLMs easily.

10. Keeping Track
    Prometheus Grafana– Collect and show performance details from the model.
    TruLens, Evidently, Fiddle, Whylabs — Spot changes in model or data quality over time.
    LangSmith– Helps find bugs, test things out, keep track of apps using LLM technology.

11. Ethical AI Stuff
    AI Fairness 360 (IBM) — Spots bias put into AI models so they can be fixed up right.
    AI Risk Management Framework— Offers rules to follow for compliance needs.

12. Governance Openness
    LangKit—Keeps tabs on LM performance with text metrics.
    OpenLLMetry—Boosts visibility into LLMS through extra cloud features.

2.4 Challenges and Opportunities in integrating LLM into CI/CD pipeline

2.4.1 Introduction

This paper [9] discusses challenges on how to manage LLMs, CI/CD Pipes and lifecycle handling.

2.4.2 Architecture

The main objective of this paper is to find out whether an LLM model can be fit into the CI/CD pipeline, identify challenges and assert whether continuous monitoring of the model is possible and beneficial. The literature survey reviews LLMs ,underlying transformer architecture and usage of LLMs in different domains. It details how to Operationalize and Manage LLMs.The Key LLM Operations components are:

- DataOps – Collects, preprocess, and manage training data while ensuring quality, security, and privacy.
- ModelOps – Develops, tests, and deploys models while maintaining scalability and robustness.

- Prompt Engineering – Designs effective prompts to optimize LLM responses.
- Human Feedback Loop – Incorporates user feedback to improve model alignment, fairness, and safety.
- Responsible AI – Ensures accountability, transparency, and ethical AI deployment.

The Main focus is around benefits and challenges with integrating LLMs with CI/CD.
Benefits:

1. Faster Deployment – Accelerates time-to-market while improving customer satisfaction.
2. Higher Quality Reliability – Reduces errors and ensures consistent performance.
3. Efficient Management – Simplifies complex workflows, improving scalability and flexibility

Challenges:

1. Customizing CI/CD Pipelines – Adapting pipelines to LLM-specific language complexity and ethical considerations.
2. Developing Reliable Testing Methods – Implementing unit, integration, and user acceptance testing for LLMs.
3. Continuous Monitoring Updating – Ensuring models remain fair, relevant, and aligned with real-world data.

Along with Step or stages as mentioned in previous papers.This paper also discusses the measurement of results. LLM outperformed baseline models in ROUGE scores, producing more relevant and accurate summaries. Achieved comparable or better results than state-of-the-art models like PEGASUS, BART, and ProphetNet. Web service performance (latency, throughput, availability) meet CI/CD quality standards.

2.5 Architecting MLOps in the Cloud: From Theory to Practice

2.5.1 Introduction

This paper [10] discusses the success rate of deploying ML models in 2020's which was around 22 percent.It identifies the gaps when deploying ML model on cloud platform and maintaining the same.Cloud platforms now serve as the backbone of many IT setups. Big cloud vendors such as aws,GCP and azure provide many tools along with many open source libraries for deployment and maintaining of ML models in cloud infrastructure.For businesses, the real puzzle is not just choosing and combining the right bits to suit their needs.This paper serves a tutorial introducing the basic steps for deploying and maintaining the modelin cloud infrastructure.

2.5.2 Summary

Key insights from this paper:

- Public cloud vendors dish out bundles of tools ML adventures. We have a look at different architectures followed by each vendor and consider key concepts.
- Selecting a particular cloud provider and its set of tools for MLOps can be a real headache. careful evaluation should be done if it satisifies all the requirements for deployment and maintenance of ML model in question.
- In the end, getting a hold on what you need, core blueprints, and the design and techical options in MLOps really boosts the success of ML models.

2.6 MLOps: Five Steps to Guide its Effective Implementation

2.6.1 Introduction

DevOps practices have become a cornerstone in software engineering and practice.Like DevOps, MLOps does not have one clear meaning.It is known as a mixture of machine learning with DevOps practices while others call it a smooth plan for handling the ML cycle .MLOps process kicks off with experiments that grab data, pull insights, and set up models. Then training, testing, and CI/CD jump into action before operations take over to push out to production work and monitor the model. The paper mentions [11] five key moves that help both greenhorns and veteran data engineers who sometimes find MLOps , a baffling maze. Its take comes from real hands-on experimentation mingled with lots of reading on machine learning, DevOps, and MLOps.

2.6.2 Architecture

The five crucial Steps are :

1. CHANGE THE WAY YOU THINK

   Rethink how you work. Teams working on MLOps must ditch devops routines they're molded into. Everyone—from top management to the folks down on the shop floor—has to get ready for new procedure changes. This is a tricky cultural shift which would effect the complete way as how traditional software systems were deployed and maintained.

2. PREPARING TO BUILD THE MODEL

   Prep comes first before any training starts up. Think of your machine learning model as more than a clever algorithm—it's a wild mixture of training data, performance checks, hyperparameters, and test sets. Every ingredient might toss a challenge: maybe a quirky algorithm choice, hidden data tangles.Look over your data carefully and match its quirks to the heart of your project. And picking the right tools while keeping version control honest helps keep your model with ease.

3. CHOOSING THE BEST TOOLS

   Nailing the right toolset is a must for crafting a machine learning model. Data is like a wild bush that never stops growing, so steer clear of manual routines.It is a long-term upkeep, the ease of spinning up CI/CD , the josh of containerization, or rock-solid version tracking are parts of ML ops journey.

4. SETTING UP AUTOMATED WORKFLOWS

   Push your model into action with a splash of automation. Build a pipeline that runs every part on its own without fuss. Implement continuous integration and continuous delivery/deployment to test every step, making sure nothing slips past.

5. KEEPING A CLOSE EYE ON PERFORMANCE

   Even after launch, your model and its data are in constant motion. What if new input data drifts away from the training tune? The performance might stumble, and the model could become yesterday's news. Setting up a few touchstones and keeping a casual eye on data shifts lets you catch little hiccups before they grow into real headaches.

2.7 ROUGE: A package for automatic evaluation of summaries

2.7.1 Introduction

[12]In a system that generates text summaries,where we need to rate each summary based on following parameters

- Coherence (Does it make sense?)
- Conciseness (Is it short but informative?)
- Grammar (Is it grammatically correct?)
- Readability (Is it easy to understand?)
- Content (Does it cover the important points from the original text?)

It would be too large of a task for a human to perform.Thus researchers designed a system of comparing the machine-generated summary with a human-written summary using mathematical methods.

One of the first approaches was:

- **Cosine Similarity** – Compares how similar two summaries are based on their words.
- **N-gram Overlap (Unigram/Bigram)** – Checks how many individual words or word pairs appear in both summaries.
- **Longest Common Subsequence (LCS)** – Finds the longest sequence of words that appear in both summaries in the same order.

These methods helped, but researchers weren't sure how well they matched human judgment.this lead to proposing another metric called ROUGE.

2.7.2 Architecture

Types of ROUGE Scores:

ROUGE-N (n-gram overlap)
Measures how many individual words (ROUGE-1) or word sequences (ROUGE-2, ROUGE-3, etc.) match between the generated and reference summaries.

$$ROUGE\text{-}N = \frac{\sum_{\mathbf{gram}_n \in \mathbf{ReferenceSummaries}} \mathbf{Count_{match}}(\mathbf{gram}_n)}{\sum_{\mathbf{gram}_n \in \mathbf{ReferenceSummaries}} \mathbf{Count}(\mathbf{gram}_n)}$$

n is the n-gram size (e.g., 1 for ROUGE-1, 2 for ROUGE-2).

$$\mathbf{Count_{match}}(\mathbf{gram}_n)$$

is the number of n-grams that appear in both the candidate summary and the reference summaries.

$$\mathbf{Count}(\mathbf{gram}_n)$$

is the total number of n-grams in the reference summaries.

Multiple Reference ROUGE-N - When multiple reference summaries are available, ROUGE-N multi is calculated by:

Computing ROUGE-N scores for the candidate summary against each reference summary. Selecting the maximum ROUGE-N score among all pairwise scores.

$$ROUGE\text{-}N_{\mathbf{multi}} = \arg\max_i ROUGE\text{-}N(r_i, s)$$

ROUGE-L (Longest Common Subsequence)

ROUGE-L measures the quality of a summary by checking the Longest Common Subsequence (LCS) between a candidate summary and a reference summary. Instead of just looking at n-gram matches (like ROUGE-N), ROUGE-L considers word order and sentence structure, making it more flexible.

$$ROUGE\text{-}L = \frac{LCS(X,Y)}{\mathbf{length}(Y)}$$

$$P_{LCS} = \frac{LCS(X,Y)}{\mathbf{length}(X)}$$

$$F_{LCS} = \frac{(1+\beta^2) \cdot R_{LCS} \cdot P_{LCS}}{R_{LCS} + \beta^2 \cdot P_{LCS}}$$

Where:

- - X = Candidate summary (machine-generated)
- - Y = Reference summary (human-written)
- - $LCS(X,Y)$ = Length of the Longest Common Subsequence between X and Y
- - length(Y) = Total number of words in the reference summary
- - P_{LCS} = Precision based on LCS
- - R_{LCS} = Recall based on LCS
- - F_{LCS} = F1-score for ROUGE-L
- - β is typically set to 1, making recall and precision equally important.

ROUGE-L is useful for capturing sentence structure rather than just word overlap. It works better than ROUGE-N when word order matters. Higher ROUGE-L means the generated summary follows the reference summary more closely in terms of sentence structure.

2.8 BLEU: a Method for Automatic Evaluation of Machine Translation

2.8.1 Introduction

BLEU (Bilingual Evaluation Understudy) [13] is a widely used automatic evaluation metric for machine translation. It is used to measure the quality of a machine translated text by comparing with one or more human written reference translations.

2.8.2 Architecture

BLEU evaluates translations based on:

N-gram Precision → How many n-grams in the machine-generated translation match those in the reference translations.

Brevity Penalty → Penalizes overly short translations that may score high on precision but lack completeness.

The BLEU score is computed as:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

where:

- BP is the brevity penalty.
- p_n is the precision for n-grams.
- w_n is the weight for each n-gram level (usually equal for 1-gram, 2-gram, 3-gram, and 4-gram).
- N is the maximum n-gram length (typically 4).

N-gram Precision Calculation

The n-gram precision is defined as:

$$p_n = \frac{\sum_{\text{all n-grams}} \text{Count}_{\text{match}}(n\text{-gram})}{\sum_{\text{all n-grams}} \text{Count}_{\text{total}}(n\text{-gram})}$$

where:

- $\text{Count}_{\text{match}}(n\text{-gram})$ is the number of n-grams in the candidate translation that also appear in the reference translations.
- $\text{Count}_{\text{total}}(n\text{-gram})$ is the total number of n-grams in the candidate translation.

Brevity Penalty (BP)

$$BP = \begin{cases} 1, & \text{if } c > r \\ e^{(1-\frac{r}{c})}, & \text{if } c \leq r \end{cases}$$

where:

- c is the length of the candidate translation.
- r is the length of the closest reference translation.

2.9 LLM Fine Tuning with PEFT Techniques

2.9.1 Summary

In this Article [14],the author introduces PEFT the acronym for Parameter Efficient Fine Tuning.In a machine learning model, we adjust these coefficients to minimize errors and make accurate predictions. In the case of LLMs, which have billions of parameters, changing all of them during training can be computationally expensive and memory-intensive.Fine-tuning comes to play here. It grabs a model that already knows a bit about language and nudges it to nail a specific task. Sometimes, you go with a task called PEFT—selectively urging only a handful of paramters while leaving the rest to their own .

Two techniques,Low-Rank Adoption (LoRA) and Quantization + LoRA (QLoRA), are used for this purpose.

LoRA (Low-Rank Adaptation): LoRA capitalizes on the fact that not all parameters in a model contribute equally. Instead of fine-tuning the entire weight matrix, it factorizes it into two smaller matrices, allowing only a subset of parameters to be adjusted. The rank factor (R) controls the number of parameters involved in fine-tuning—choosing a lower R reduces computational overhead while maintaining performance efficiency.

Quantization: Quantization compresses model parameters by converting high-precision floating-point values into lower-bit representations, such as 4-bit integers. While this process introduces some loss of precision, it significantly reduces memory usage and speeds up computation. During operations, these quantized values are dequantized to minimize errors and maintain accuracy.

For practical implementation of PEFT (Parameter Efficient Fine-Tuning), Below are the key steps involved in fine-tuning using PEFT:

- Data Preparation: Organize your dataset according to your task requirements. Clearly define inputs and expected outputs, particularly when working with models like Falcon 7B.
- Library Setup: Install essential libraries such as HuggingFace Transformers, Datasets, BitsandBytes, and WandB for tracking training progress.
- Model Selection: Choose the LLM you want to fine-tune, such as Falcon 7B.
- PEFT Configuration: Set up LoRA parameters, including the choice of layers and the rank factor (R), which determines the subset of weights to be adjusted.
- Quantization: Select an appropriate level of quantization to optimize memory usage while maintaining acceptable accuracy.
- Training Arguments: Define key hyperparameters, including batch size, optimizer, learning rate scheduler, and checkpointing strategy.
- Fine-Tuning: Use HuggingFace's Trainer with PEFT configurations to fine-tune the model, while tracking training progress with WandB.
- Validation: Monitor both training loss and validation loss to prevent overfitting and assess model performance.
- Checkpointing: Save checkpoints periodically, enabling you to resume training from a specific stage if needed.

2.10 A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications

2.10.1 Introduction

Prompt engineering [15] is a powerful technique designed to optimize the performance of pre-trained large language models (LLMs) and vision-language models (VLMs). Rather than modifying a model's internal parameters, it focuses on crafting task-specific prompts that direct model outputs efficiently.

This method has revolutionized AI adaptability, enabling models to perform a wide range of tasks without the need for extensive retraining or fine-tuning. The real strength of prompt engineering lies in its ability to shape model behavior through carefully crafted instructions, representing a shift from traditional machine learning approaches.

Advantages of Prompt Engineering

Boosts Model Flexibility: Enables LLMs and VLMs to handle specialized tasks without additional training.

Expands AI Capabilities: Enhances performance in language generation, question answering, coding,

reasoning, and multimodal tasks.
Addresses Research Gaps: Provides task-specific optimizations through systematically designed prompts.

2.10.2 Architecture

Modern breakthroughs fuel a burst of prompt ideas, ranging from basic to advanced techniques, including:

New Task Adaptation

- Zero-Shot Prompting allows LLMs to complete tasks without prior training, relying only on well-structured prompts.
- Few-Shot Prompting enhances model understanding by providing a few input-output examples, improving performance on complex tasks but requiring additional tokens.

Reasoning and Logic Enhancement

- Chain-of-Thought (CoT) Prompting structures LLM reasoning in step-by-step sequences, improving problem-solving in math and commonsense tasks.
- Auto-CoT automates the generation of reasoning chains, boosting efficiency and reducing manual effort.
- Self-Consistency refines CoT responses by sampling multiple reasoning paths, significantly improving accuracy.
- Logical CoT (LogiCoT) integrates symbolic logic principles, reducing errors and hallucinations.
- Tree-of-Thoughts (ToT) Graph-of-Thoughts (GoT) introduce hierarchical and network-based reasoning, allowing LLMs to explore multiple reasoning paths dynamically.

Reducing Hallucination and Improving Accuracy

- Retrieval-Augmented Generation (RAG) enhances factual accuracy by incorporating external knowledge retrieval.
- ReAct Prompting combines reasoning and actions to improve decision-making, especially in fact-checking tasks.
- Chain-of-Verification (CoVe) Chain-of-Note (CoN) ensure response reliability by encouraging models to verify their outputs systematically.

Code Generation and Computational Optimization

- Scratchpad Prompting enables multi-step reasoning in programming tasks, improving code correctness.
- Program-of-Thoughts (PoT) Chain-of-Code (CoC) refine LLMs' ability to reason with numeric and symbolic expressions, leveraging executable code.
- Optimization by Prompting (OPRO) turns LLMs into iterative optimizers, improving solutions for mathematical and NLP tasks, outperforming human-designed prompts by up to 50

Metacognition and User Interaction

- Rephrase and Respond (RaR) Prompting helps LLMs clarify ambiguous questions for improved comprehension and response accuracy.
- Step-Back Prompting enhances higher-order reasoning by prompting LLMs to abstract key concepts before solving problems.

Chapter 3

Foundations

3.1 Rationale and Techniques in LLM

Key to LLMs performing well is the encoder-decoder setup, a model that has helped in areas like changing languages, shortening texts, and talking AI. The Encoder-Decoder design was brought out to deal with Seq2Seq issues, showing big progress in managing data sequences.

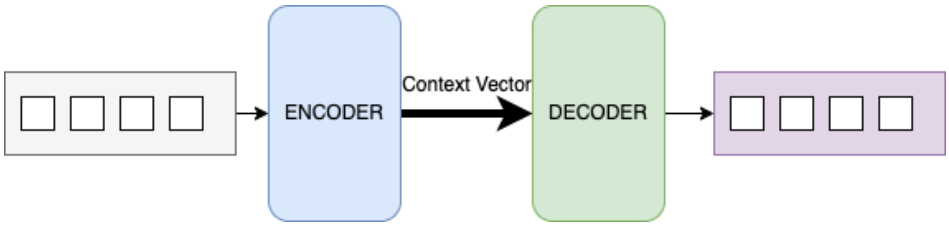

Figure 3.1: Seq2Seq2 Model

3.1.1 RNN

To process sequences like time series and language construct, Reccurant Neural Networks are designed. Unlike feedforward neural network, RNNs have a special design where neuron connections loop back on themselves to preserve previously fed data. This cyclical connection enables RNNs to process and make predictions based on temporal patterns in sequential data. Significant Seq2Seq Model Architecture:

Architecture and working of an RNN is explained below.

Functional details of Recurrent Neural Network

An RNN executes sequences one step at a time. To preserve previous step information, it maintains a hidden state. The key components of an RNN unit are:

- **Input (x_t):** The sequence's input data at each time step t.
- **Hidden State (h_t):** At each time stage t, a hidden state is maintained which holds information of x_t and $h_t - 1$. This is carried on from one stage to the next stage, thereby retaining memory of past data feeds.
- **Output (y_t):** At each time stage t, based on the current h_t, the output is calculated. This output may be used for prediction or provided onto the network's subsequent tier.

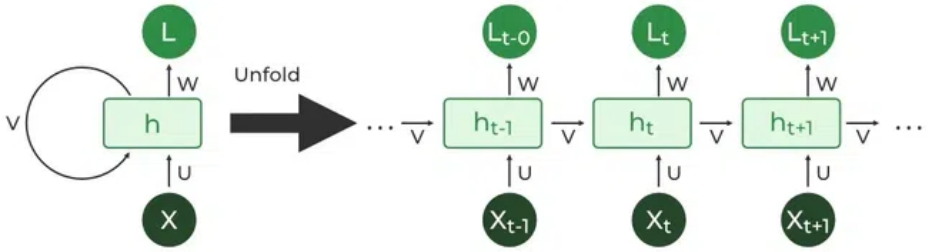

Figure 3.2: Working of RNN
[16]

RNN Mathematical Equations:

The RNN refreshes its hidden status at all times step t and determines the output based on following mathematical equations:

A. Hidden State Calculation: The hidden state h_t is found by using the earlier hidden state $h_t - 1$ along with input x_t at this moment: $h_t = f(W_t * h_(t-1) + W_x * x_t + b_h)$

This equation shows how the hidden state from before also the current input impact the current hidden state. The weight matrices W_h and W_x are adjusted during training.

B. Output Calculation: The output y_t is calculated on the current h_t. The output equation is:
$y_t = g(W_y * h_t + b_y)$

The output can be used in many ways, based on the task. For instance, in predicting sequences, the output at each step helps to guess the next value in the sequence.

Key Components of RNN:

A. Hidden State: The main memory in RNN is h_t , the hidden state. Every time step, it gets updated by looking at new input and previous state. This helps RNNs to keep information for a long time, which makes them good for working with sequences.

B. Weights: Weight Matrices: RNNs use several weight matrices:

- W_h for connections between hidden states
- W_x for connections between input and hidden state
- W_y for connections between hidden and output state.

The weight matrices are learned through backpropagation and gradient descent. **Bias Terms:** Bias terms b_h and b_y are used to offset the computations and provide flexibility in the model.

C. Activation Function: RNNs commonly use nonlinear activation functions such as **tanh** or **ReLU** in the h_t layer. The tanh is usually used because it outputs values between -1 and 1, which helps prevent large updates during training. However, **ReLU** can also be used for certain tasks.

D. Output Layer: The output layer uses the hidden state from each time step and calculates the final output with an activation function like **softmax** for classification or **linear** for regression.

Backpropagation Through Time (BPTT)

A main difference RNNs have from normal feedforward neural networks is that they can handle sequences and keep memory. When training RNNs, we apply a type of backpropagation known as **backpropagation through time** (BPTT).

How BPTT Works Forward Pass: During the forward pass, where each time step's hidden states along with outputs are calculated after its input sequence is sent through the RNN. **Loss Calculation:** AThe model calculates the loss determined by the output as well as the actual target values after processing the complete sequence. Any appropriate loss function, (which includes the weights along with biases. The gradients propagate backward via the sequence's **mean squared error** or **regression** or **cross-entropy** for classification), is available to compute this loss. **Backward Pass**: In the backward pass, the gradients of the loss are calculated with concerning the weights along with biases. The gradients propagate backward via the sequence's time steps. Thus, the model has the capability of updating the weights based on how much each time step contributed to the final loss. **Weight Updates**: The weight matrices W_h, W_x and W_y are updated using gradient descent or any other optimization algorithm (e.g., Adam, RMSprop).

Applications of RNN

RNNs are a good fit for tasks involving sequential data, including:

- **Natural Language Processing (NLP):** Language modeling, text synthesis, machine translation, along with sentiment analysis, among other tasks.
- **Speech Recognition:** Converting spoken language into text.
- **Time Series Prediction:** Forecasting stock prices, weather conditions, or demand in supply chain management.
- **Video Analysis:** Analyzing video sequences for action recognition or event detection.

3.1.2 LSTM

LSTM stands for **Long Short-Term Memory**,it is a type of RNN, or recurrent neural network. It handles common issues with RNNs, like vanishing and exploding gradients during training. LSTM networks learn dependencies over long periods and handle sequences well, It can handle time-series forecasting, speech understanding, and natural language work (NLP) very efficiently.

Details of LSTM architecture :

1. Basic Structure of LSTM:

There are four main components in LSTM unit, that can selectively remember or forget information over long sequences. The core components are:

Cell State (Ct):

1. Memory of LSTM network
2. Carries information throughout the sequence processing
3. Information's can be retained or forgot at each step by the gates

Hidden State (ht):

1. The output of the LSTM unit for a particular time step

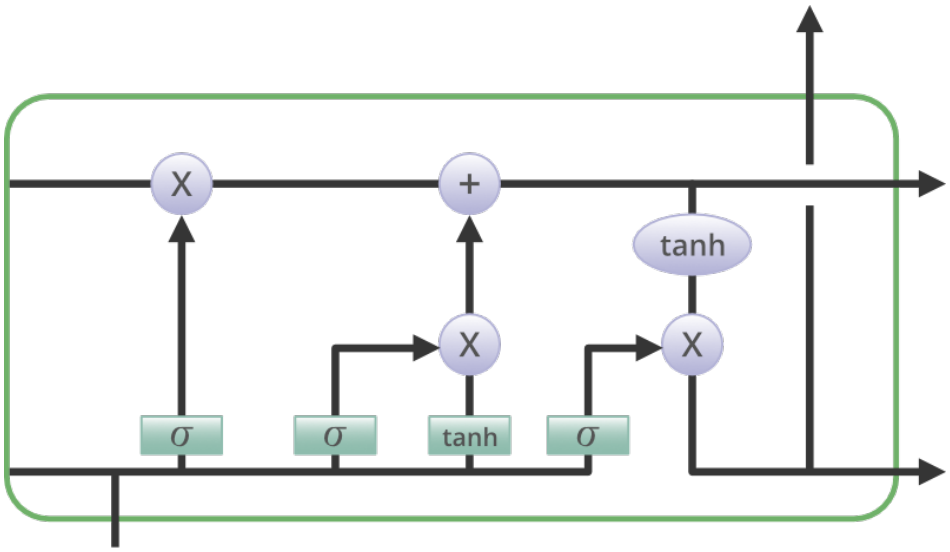

Figure 3.3: LSTM Architecture
[17]

2. It is passed on to the next layer as an output for the network

Gates: Flow of information is regulated by below 3 gates in LSTM

- **Forget Gate (ft)**
- **Input Gate (it)**
- **Output Gate (ot)**

These gates in LSTM act as a neural network. Those gates determine which information should be passed or modified

2. Key Components of LSTM:

Following are key gates and their duties in the LSTM structure:

A. Forget Gate (ft):

- The forget gate determines extent of previous cell state $C_t - 1$ taken to the next time step.
- The forget gate uses a **sigmoid function** for activation, where values range from 0 to 1. Here, 0 indicates "totally forget," while 1 refers to "fully keep." The calculation for the forget gate is: The forget gate is calculated as:

$$f_t = \sigma(W_f * [h_t - 1, x_t] + b_f)$$

B. Input Gate (it) and Candidate cell state(ĉt):

- The input gate looks at new information from present x_t and figures out how much to add to the cell state.
- The **candidate cell state(ĉt)** shows possible new information that might be added to the cell state.
- Both input gate and candidate cell state are found using a **tanh** activation type and a **sigmoid** activation type, in following ways:

$$\mathrm{i}_t = \sigma(W_i * [h_t - 1, x_t] + b_i)$$
$$\mathrm{C}_t = tanh(W_c * [h_t - 1, x_t] + b_c)$$

C. Cell State Update:

- The cell state gets updated by mixing the old cell state (governed by the forget gate) and fresh information (managed by input gate).
  The update rule for the cell state is:

$$\mathrm{C}_t = f_t * C_t - 1 + i_t * C_t$$

D. Output Gate (ot):

- The output gate makes a decision on which section of cell state C_t will be shown as hidden state h_t..
- The output gate is also calculated using a **sigmoid** activation function: $o_t = \sigma(W_o * [h_t - 1, x_t] + b_o)$

  The final hidden state h_t is calculated by multiplying the output gate's activation with the cell state passed through a **tanh** activation function:

$$h_t = o_t \cdot \tanh(c_t)$$

5. Benefits of LSTM:

- **Manages Long-Time Connections:** LSTMs are good at figuring out long-time connections in data, which normal RNNs have trouble with because of fading gradients.
- **Choice Memory:** LSTMs can choose to forget or keep information through their gate system.
- **Consistent Learning:** The gates help stop problems like fading and growing gradients, making it simpler to train LSTMs on long chains.

6. Applications of LSTM:

LSTM models are used a lot in many tasks that involve sequences like:

- **Time-Series Prediction:** Guessing future numbers from old data (for example, stock values, weather predictions).
- **NLP Tasks:** Jobs such as making language models, creating text, figuring out feelings in writing, and changing one language to another. Speech Recognition: Changing spoken words into written form.
- **Video Analysis:** Looking at time-based sequences in video pictures for jobs like recognizing actions.

3.1.3 Transformer

The **Transformer** uses an innovative method called **self-attention** This is different from old models like RNNs or LSTMs that relied on recurring structures. The Transformer model was introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017 that gave rise to the base of Transformer model which has become the state-of-the-art models in NLP, including **BERT**, **GPT**, **T5**, and many others.

Key Concepts of the Transformer Architecture:

The Transformer's central idea is the **self-attention methodology**, that lets model pay attention to various input sequence parts at same time, unlike RNNs and LSTMs that process sequences step-by-step.

The two main parts of Transformer architecture are:

1. **Encoder** (used for understanding the input sequence)
2. **Decoder** (used for generating the output sequence)

In some models (like BERT or GPT), only the encoder or decoder is used, but the full Transformer typically includes both.

1. Transformer Model Overview:

Below is a high-level breakdown of both:

- **Encoder:**
  - The encoder handles the input data and generates format that is usable by the decoder
  - There are several identical layers in each Encoder. (Generally 6 as per paper)
- **Decoder:**
  - The decoder produces the output data from the encoder's interpretation.
  - There are several identical layers in each Decoder too. (Generally 6 as per paper)

2. Transformer Encoder Layer:

There are two parts of each Enoder Layer:

- **Multi-Head Self-Attention Mechanism**
- **Feed-Forward Neural Network**

A. Multi-Head Self-Attention: It is a mechanism that enables every word (or token) in the fed line pay attention to each and every word when coding the line. In simpler terms, each token understands connections with itself and every other token in the line.

Scaled Dot-Product Attention:

Using this the contextual attention is computed. For each token in the sequence, the main idea is to take three components:

- **Query (Q)**
- **Key (K)**
- **Value (V)**

2. Multi-Head Attention: The Transformer uses **multi-head attention** instead of just computing one group of attention scores. This means the Machine Learning model can learn several different attention distributions, referred to as "**heads**," making it to understand various associations found in the input text.

The steps are:

- For each head, using different learned weight matrices, compute multiple sets of Q, K, and V.
- To each head, apply scaled dot-product attention.
- Add up the outcome of all heads. Then pass this through a linear layer.

B. Position-Wise Feed-Forward Networks: The attention mechanism is followed by a fully connected feedforward neural network (FFN) that applies the output to each point independently and identically. It has two layers,with the center being ReLU Activation.

Computation of FFN:
$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$

C. Residual Connection & Layer Normalization: In the encoder layer, there is a **residual connection** followed by **layer normalization** in each of the subcomponents (self-attention and feed-forward networks). This aids in training deeper networks by preventing vanishing/exploding gradients.

The encoder layer is thus structured as:

1. Input →**Multi-Head Self-Attention** →Add & Normalize
2. Output →**Feed-Forward Network** →Add & Normalize

3. Transformer Decoder Layer:

Apart from ensuring that it handles the decoder layer and the encoder layer's output are comparable.

The following elements can be found in this:

- **Masked Multi-Head Self-Attention:** By masking certain positions, it prevents focusing on the sequence's subsequent locations.
- **Multi-Head Attention (Encoder-Decoder Attention):** Using this mechanism the encoder's output becomes decoder's primary focus.
- **Feed-Forward Network (FFN):** Same applied to the decoder's hidden states, just like in the encoder layer.

The decoder layer structure is:

1. Input →**Masked Multi-Head Self-Attention** →Add & Normalize
2. Output→**Multi-Head Encoder-Decoder Attention** →Add & Normalize
3. Output →**Feed-Forward Network** →Add & Normalize

4. Positional Encoding:

The Transformer model uses supplementary positional encodings to the input embeddings to determine a method to include where each token is positioned, unlike RNNs and LSTMs which work through the input sequence step-by-step. This is accomplished by supplementing the input embeddings with positional encodings.

The input embeddings, which provide details regarding the absolute and relative positions of words in the phrase, are supplemented with these vectors. The sine as well as cosine functions of various frequencies have been used to make positional encodings in the original paper:
$PE(pos, 2i) = sin(pos/(10000^(2i/d_model)))$ $PE(pos, 2i + 1) = cos(pos/(10000^(2i/d_model)))$

5. Final Linear Layer and Softmax:

Then fed into the decoder layer. And the outcome is passed to a final **linear layer** along then to a **softmax** activation function to generate odds for each token across the language.

Advantages of Transformer Architecture:

1. **Parallelization:** Unlike RNNs or LSTMs that look at data one after another, Transformers can see through all parts of the sequence simultaneously. This makes them a lot quicker and better when training.

2. **Long-Range Dependencies:** Long term dependencies is more efficiently captured due to the self-attention mechanism than RNNs, which struggle with long-term memory due to vanishing gradients.

3. **Scalability:** The Transformer model scales well to large datasets and serves several purposes, such as language comprehension, text production, and machine translation.

3.2 Operationalizing and Managing LLMs

LLMs are susceptible to errors and biases, which can undermine their quality, reliability, and ethical integrity. These challenges raise significant concerns regarding the social impact of their deployment. Additionally, LLMs are dynamic and continuously evolving, necessitating ongoing monitoring and updating to incorporate the latest data and feedback. To mitigate these risks and ensure the effective operation of LLMs, it is crucial to adopt best practices and tools for their operationalization and management throughout their lifecycle. This involves implementing the principles of MLOps, a discipline designed to bridge the gap between machine learning (ML) development and devops.

3.2.1 Integrating LLMs into Pipelines/Chains

What are pipelines ?

A pipeline is an automated workflow that orchestrates the different stages of the machine learning lifecycle, i.e, from data preparation to model deployment and monitoring. Pipelines ensure that these stages are executed in a consistent, repeatable, and scalable manner, enabling teams to efficiently develop, test, and deploy machine learning models

Pipelines are also known as chains as some software frameworks use this term and both are used interchangeably.

Any LLM model to be deployed can be split into four components namely:[18]

- **Data pipeline**
- **Feature pipeline**
- **Training pipeline**
- **Inference pipeline**

As part of this study, we will cover each pipeline in a separate section.

3.2.2 Data Pipeline

The phrase often abbreviated as **GIGO** (garbage in,garbage out) succinctly captures this fundamental principle denoting the importance of quality of input.This principle has even greater significance,as the entire functionality and reliability of ML system depends on the data they are fed.

Important stages involved in Data pipeline are illustrated:[19]

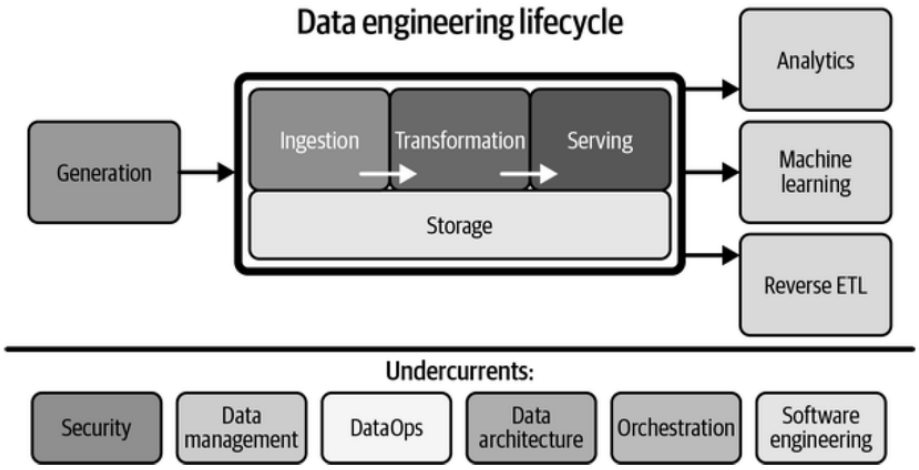

Figure 3.4: Stages in Data engineering
[19]

Data Generation This phase is about getting data in smart ways. Data can be many types like a store system, IoT gadget, flat files (like CSV or XML), RSS feeds, web services and more options too.

Data Engineers (DEs) should know how to work with these sources and talk to the stakeholders who handle them.

Data Ingestion Once the data is collected, it enters the ingestion phase and is transported to storage systems. This segment illuminates the architecture choices that govern this process.We differentiate between batch and real-tie ingestion,dissecting their applicability based on data velocity and volume.

Data Transformation At this phase, we change data into a clear format. On a simpler level, data often gets changed to other types or made uniform. Following that, there can be changes to the layout of information, making data consistent or putting it together in groups. The hardest (and most important) changes happen when we use business logic on the raw information. This is also known as **'data modeling'**.

Data Storage Storage is relevant everywhere in the lifecycle and can occur in various types at each stage. Selecting storage options is crucial and involves several compromises. Broadly, storage might take place on-site (on-prem) or online (cloud), which has seen wide acceptance lately for valid reasons.

Data Serving The final phase of the lifecycle is when you most engage with other parties involved. This section is where stakeholders get to enjoy the advantages.This data is served to a Machine learning model for prediction or decision making.

3.2.3 Feature Pipeline

A feature is a numeric representation of raw data. Many ways exist to change raw data into numeric, so features can appear many formats. Features need come from type of data that is available.

The feature pipeline's role is to take data points from integrated data serving component such as a warehouse,process them and load them into the feature store.

Custom properties of feature pipeline are:

- It processes different types of data.
- It contains processing steps for fine-tunning and RAG: cleaning, chunking and embedding.
- It creates a snapshot of data,after cleaning and after embedding
- It creates a logical feature.

RAG Feature Pipeline

Retrieval-augmented generation (RAG) is fundamental in most generative AI applications. RAG's core responsibility is to inject custom data into the LLM to achieve a given action (e.g., summarize, reformulate, and extract the injected data). We often want to use the LLM on data it wasn't trained on (e.g., private or new data). As fine-tuning an LLM is a highly costly operation, RAG is a compelling strategy that bypasses the need for constant fine-tuning to access that new data.

Retrieval-Augmented Generation (RAG) is a type of model designed to improve text generation jobs. It does this by using information that is found from external knowledge sources, for example , in question answers, summary making, and conversations.

A RAG system is composed of three main modules independent of each other:

- **Ingestion pipeline:** A batch or streaming pipeline used to populate the vector DB.
- **Retrieval pipeline:** A module that queries the vector DB and retrieves relevant entries to the user's input.
- **Generation pipeline:** The layer that uses the retrieved data to augment the prompt and an LLM to generate answers.

More on RAG would be covered as a subsequent Section.Following diagram illustrates generalized feature pipeline

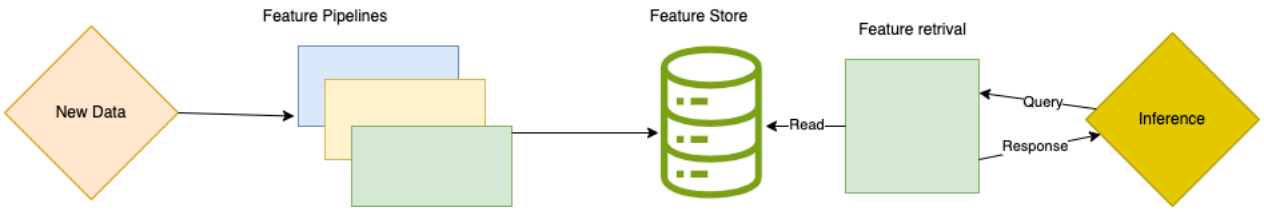

Figure 3.5: Feature Pipeline

3.2.4 Training Pipeline

The training pipeline takes input features and labels from the stored data and produces a model or multiple models. The model which is selected and stored is called model registry. This is similar to feature stores, but here, the focus shifts to the model . So, the model registry will keep track of, manage versions of, and share these models with another part known as the inference pipeline.

Many latest model registries have a metadata storage that lets you detail key parts of the model training. The key elements include features, labels, and the versions used in training the model.

Training stage also includes fine-tunning, a process that allows the model to perform better on particular tasks, like sentiment analysis, question answering, or domain-specific text generation, by adjusting its weights based on task-specific data.

PEFT stands for **Parameter-Efficient Fine-Tuning**, which are methods for optimizing huge, previously trained models (like huge LLMs) in a parameter-efficient way. The goal of PEFT is to reduce the computational cost and memory overhead associated with fine-tuning massive models, while still achieving good performance on specific downstream tasks.More on PEFT will be discussed in subsequent section

3.2.5 Inference Pipeline

The inference pipeline gets features and labels from feature store and also has the trained model from model registry. With these two, it can make predictions in batch mode or real-time mode easily.

Inference pipeline start with init phase. Here, trained model is obtained from model registry, libraries and dependencies are loaded up, and connection to outside resources like database or feature store are set up. In next phase, pipeline gets input data—often a prompt or question—does any pre-processing if needed, then makes predictions using LLM. After that in post-process steps for model's output maybe formatting it or filtering results through ranking them. If feature store is in use, pipe may pull relevant features before processing more changes and sending data to model for prediction making. This setup keeps things efficient and scalable when working with big models.

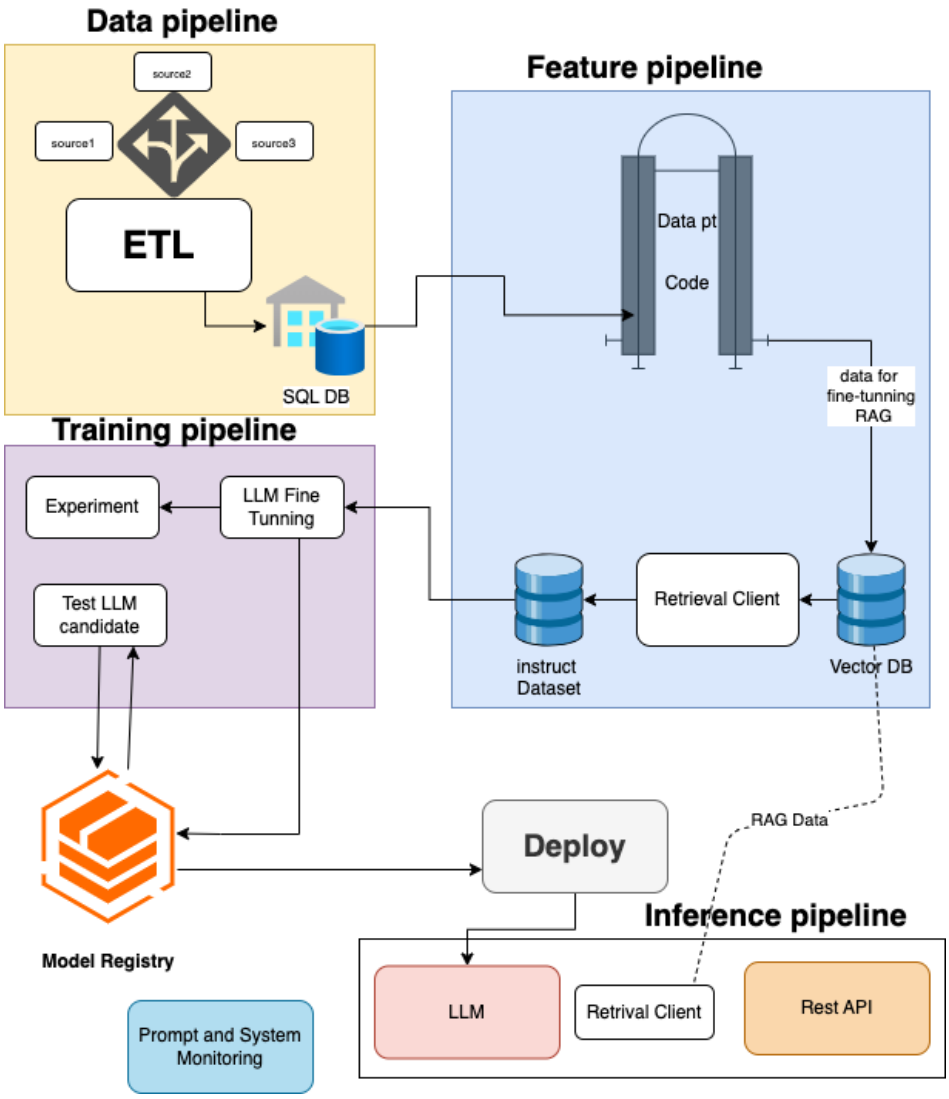

Figure 3.6: LLM Architecture

3.3 RAG

RAG enhances the accuracy and reliability of generative AI models with information fetched from external sources. It is a technique complementary to the internal knowledge of the LLMs. Before going into the details, let's understand what RAG stands for:

- **Retrieval:** Search for relevant data
- **Augmented:** Add the data as context to the prompt
- **Generation:** Use the augmented prompt with an LLM for generation

Any LLM is bound to understand the data it was trained on, sometimes called parameterized knowledge. Thus, even if the LLM can perfectly answer what happened in the past, it won't have access to the newest data or any other external sources on which it wasn't trained.

3.3.1 Problems RAG solves

The two fundamental problems that RAG solves:

- Hallucinations
- Old or private information

Hallucinations

A hallucination refers to when an AI system, such as a language model, generates information that is incorrect, misleading, or entirely fabricated. This can occur even though the AI presents the information confidently and in a manner that may seem authoritative or convincing.[20]

By introducing RAG, we enforce the LLM to always answer solely based on the introduced context. The LLM will act as the reasoning engine, while the additional information added through RAG will act as the single source of truth for the generated answer. By doing so, we can quickly evaluate if the LLM's answer is based on the external data or not.

Old information

Any LLM is trained or fine-tuned on a subset of the total world knowledge dataset. This is due to three main issues:

- **Private data:** You cannot train your model on data you don't own or have the right to use.
- **New data:** New data is generated every second. Thus, you would have to constantly train your LLM to keep up.
- **Costs:** Training or fine-tuning an LLM is an extremely costly operation. Hence, it is not feasible to do it on an hourly or daily basis.

RAG solves these issues, as you no longer have to constantly fine-tune your LLM on new data (or even private data). Directly injecting the necessary data to respond to user questions into the prompts that are fed to the LLM is enough to generate correct and valuable answers.

Embeddings and Vector database

The retrieval component of RAG systems typically relies on embeddings to represent text (queries, documents, etc.) in a high-dimensional space. These embeddings capture semantic meaning, enabling the system to retrieve relevant information even if the exact words in the query don't match those in the database.

Query Embeddings: When a user submits a query, the system first converts it into an embedding using a model like **BERT, RoBERTa, SBERT, or GPT-3**. This embedding is a numerical representation of the query's meaning.

Document Embeddings: Similarly, documents (or other knowledge sources) are converted into embeddings. These embeddings serve as the "indexed" content in the retrieval phase.

Vector Databases
Following the conversion of the query into an embedding, the data is stored and indexed in a vector database. embeddings of the documents (or knowledge base). When a query embedding is generated, the vector database searches for the nearest neighbors in the embedding space, retrieving the most semantically relevant documents based on the distance between their embeddings.

Vector databases like **Pinecone, FAISS, Milvus, and Weaviate** enable efficient similarity search by using algorithms such as **Approximate Nearest Neighbor (ANN)** search, which allows for quick retrieval even with millions or billions of vectors.

Cosine Similarity or **Euclidean Distance** is typically used as a measure of distance to determine the extent of similarity or proximity between the emdeddings in the vector space.Cosine similarity is a way to

measure how two non-zero vectors are in an inner product space. This method is often used in areas like information retrieval,text mining, and machine learning to show how close two documents or text pieces are when looking at their vector forms (embeddings).

The core idea behind cosine similarity is that it measures indicating the degree evaluating the cosine of two vectors' angles to determine how they are. their direction, rather than magnitude. A smaller angle The core idea behind cosine similarity is that it measures indicating the degree evaluating the cosine of two similar between the vectors indicates greater similarity.

$$\textbf{cosine\_similarity}(\text{A, B}) = \frac{(A \cdot B)}{\|A\|\|B\|}$$

3.4 Prompt and Prompting Techniques

A **prompt** is a statement or question given to the model to elicit a response. It defines the task, guides the model on what kind of answer to give, and may specify any particular format or constraints on the output.

3.4.1 Prompting Techniques

- **Direct prompts (Zero-shot)** This method uses a clear instruction or question without extra context. An example can be asking for new ideas or explaining a concept.. An example of this is idea generation,or explain a concept.
- **One-, few- and multi-shot prompts** This technique gives the model some examples of input-output pairs before the main prompt. This can help the model grasp the task better and produce more precise answers.
- **Chain of Thought Prompts** CoT prompting gets the model to split complicated reasoning into smaller steps, resulting in a more detailed and organized final response.

3.5 Fine-tuning

In ML, Retraining a trained model on a smaller, more focused dataset is known as fine-tuning. This method uses the knowledge the model learned before and adjusts it for new tasks without needing a lot of computing power.

Supervised Fine-Tuning (SFT) [21] is a technique that involves retraining a previously trained model using a labeled dataset relevant to the job at hand. The objective of supervised learning is to adjust the parameters of the model so that it can better predict outputs (like labels or continuous values) from given input data, by aiming to reduce a loss function.

3.5.1 Parameter-Efficient Fine-Tuning (PEFT)

PEFT (Parameter-Efficient Fine-Tuning) [22] a technique that concentrates on changing a small number of a model's parameters. The computational while memory expenses frequently associated with conventional fine-tuning techniques have been lowered by this technology. PEFT's primary concept is to alter a previously trained model for a particular job without the need to adjust all of its parameters.

Key PEFT Techniques:

- **Adapters** are small modules added to the model which has already been trained. Here, only adapter parameters are trained, while the model itself is kept unchanged

- **LoRA**, or Low-Rank Adaptation, adds low-rank matrices in some parts of the model, which cuts down the number of parameters that must be updated.
- **QLoRA**, or Quantized Low-Rank Adaptation, is a technique for adjusting big pre-trained models. This technique helps improve the process's effectiveness in terms of memory along with processing power. It mixes low-rank adaptation (LoRA) with quantization, enabling large models like GPT-3 and T5 to be fine-tuned for specific tasks using lesser resources.

3.6 CI/CD

3.6.1 Continuous Integration

CI is a software development technique in which the team members frequently store their code revisions in a shared location, typically at least once every day, according to Martin Fowler [23]. An automated build procedure is utilizing to verify each integration so includes testing, allowing teams to quickly identify and address integration issues. This approach helps minimize the risk of delivery delays, reduces the effort required for integration, and promotes a healthy codebase that supports the rapid addition of new features.

Main practices for Continuous Integration are:

- Store all in a version-controlled main line
- Automate the building process
- Ensure the build self-tests
- Everyone commits to the mainline daily
- Every update to the mainline should start a build
- Resolve broken builds
- A quick build process
- Hide work-in-progress
- Test in a replicated production environment
- Make everything visible to all
- Automate the deployment process

3.6.2 Continuous deployment

Continuous deployment (CD) is a method in software engineering where software features are released often and through automated processes.

Continuous deployment is not the same as continuous delivery, which is known as CD. In continuous delivery, software features are released frequently and may be ready for deployment, but they do not have to be deployed every time.Therefore, continuous deployment is seen as a more complete type of automation compared to continuous delivery.

Without continuous integration (CI), evaluation, and monitoring, LLM-based AI systems are at risk of performance degradation, which can lead to undesirable behaviors or even complete failure. Such issues can manifest as follows.

- Task failures

- Frustrating customer experiences
- Financial impacts from mishandling sensitive data.

To mitigate these risks and ensure AI safety, we propose integrating continuous evaluation of LLMs within Continuous Integration and Continuous Deployment (CI/CD) pipelines. CI/CD pipelines are essential for automating the processes of integrating, testing, and deploying code changes, enabling frequent updates while maintaining system stability. By incorporating continuous evaluation into these pipelines, we ensure that the LLMs are regularly tested and monitored for optimal performance, thereby preventing system degradation and minimizing potential risks.

Integrating LLMs into CI/CD pipelines offers several key benefits[9], including:

1. **Improved Quality and Consistency:** Continuous integration helps ensure that LLM-based products or services maintain high quality, reducing errors, bugs, and inconsistencies. This, in turn, improves the reliability of the system and promotes trust with users.
2. **Efficient Management:** CI/CD pipelines streamline the management of LLMs by simplifying the complexity inherent in these models. This integration provides scalability and flexibility, making it easier to handle updates, modifications, and the deployment of new versions of LLMs.

3.6.3 Advantages of LLMOps with CI/CD

Some advantages of using LLMOps with CI/CD

- **Version Control**: ML models, datasets, and configurations are tracked and versioned, similar to code, enabling reproducibility and traceability in the CI/CD pipeline.
- **Continuous Training**: CI pipelines are used to automatically retrain models using the latest data and continuously integrate new model versions.
- **Streamlined Testing and Debugging**: Unlike traditional software testing, LLMops allows us to integrate automated tests for ML models, including validation of model performance, robustness, and fairness.
- **Cost and Resource Management**: Through proper allocation of computational resources on a cloud infrastructure, companies can benefit by isolating the required resources for each stage rather than allocating the same computational heavy resource for all the stages. We don't need a heavy processing GPU based system during the data preparation stage.

3.6.4 Challenges

Despite these benefits, integrating LLMs into CI/CD pipelines presents certain challenges:

1. **Effective testing and evaluation:** Developing and implementing robust testing and evaluation methods for LLMs is critical. This consists of unit tests, integration tests, system tests, and user acceptance tests. It also involves setting and evaluating suitable performance measures and quality standards.Appropriate performance metric and criteria for quality should also be clearly specified.
2. **Continuous Monitoring and Updating:** LLMs require ongoing monitoring and refinement to ensure they remain aligned, fair, and safe. This includes collecting and analyzing new data, retraining the models, and deploying updated versions, while ensuring that ethical considerations are consistently addressed.

In this paper, we aim to demonstrate the application of LLMOps in a case study where an LLM is integrated into a CI/CD pipeline for a text summarization task. By showcasing this process, we highlight the importance of continuous evaluation in maintaining the performance, safety, and reliability of LLM-based systems within a CI/CD framework.

3.7 Methodology

The input data for our project will consist of news articles from reputable sources such as The Times of India and Sky News. We will begin by configuring the Continuous Integration (CI) pipeline, where initial tests will be conducted. The pipeline configuration will follow the specifications outlined in the YAML file located within the .github/workflows directory. Our approach involves implementing a text summarization model using fundamental Natural Language Processing (NLP) techniques and Reinforcement learning as part of the Human feedback loop mechanism. Specifically, we will employ the T5 transformer model and OpenAI's GPT-3.5 Turbo model for summarization tasks. We will also employ another model using NLP techniques(TF-IDF embedding) and based on sentence scoring using policy gradient methods we would summarize the text. We will write the necessary code for these models, ensuring that unit tests are defined using the pytest framework.

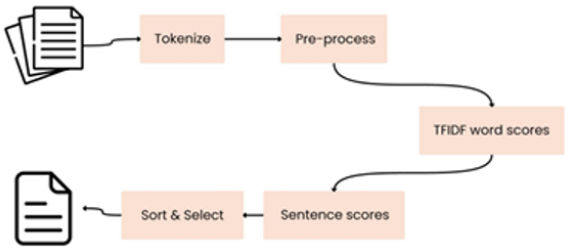

Fig. 3. Illustration of Sentence Scoring

Figure 3.7: Sentence scoring

To facilitate the development process, a separate branch will be created within GitHub Actions, and a push event will trigger the pipeline. If the results are satisfactory—i.e., no errors are encountered—the code will be merged into the main branch. This process will be repeated for each iteration of our text summarization models. To establish continuous evaluation for Large Language Models (LLM) in our CI/CD pipeline, we will leverage GitHub Actions for continuous integration and OpenAI's Python package for the evaluation process. The evaluation will be coded based on the metrics defined earlier in the methodology. Upon triggering the CI pipeline, the evaluation script will run automatically, recording the results. If the generated scores exceed a predefined minimum threshold, an error will be raised. This ensures that any undesirable changes are identified before they can propagate into the Continuous Deployment (CD) pipeline and affect production. In doing so, we will have successfully integrated continuous LLM performance evaluation into our CI/CD workflows.

Future Proposal: Out of Scope for This Experiment One potential avenue for future exploration is the inclusion of a Retrieval-Augmented Generation (RAG) system. A RAG system enhances a Large Language Model's (LLM) output by incorporating external or custom information, beyond the model's pre-trained knowledge base. While this approach offers promising benefits, it falls outside the scope of the current project and is therefore not included in our experiment.

Security Challenges While encryption, access controls, and enhanced monitoring are addressing some security challenges associated with integrating Large Language Models (LLMs) into CI/CD pipelines, there remains significant room for improvement. As LLMs continue to evolve and become more deeply embedded

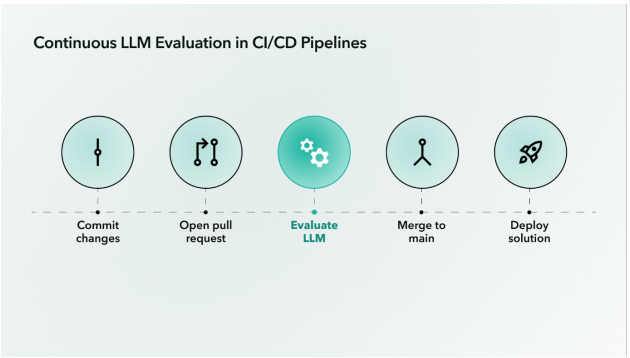

Figure 3.8: LLM ops CI/CD Pipeline

Bibliography

[1] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[2] Sridhar Alla and Suman Kalyan Adari. "Beginning MLOps with MLFlow". In: *Apress: New York, NY, USA* (2021).

[3] Beatriz M. A. Matsui and Denise H. Goya. "MLOps: Five Steps to Guide its Effective Implementation". In: *2022 IEEE/ACM 1st International Conference on AI Engineering – Software Engineering for AI (CAIN)*. 2022, pp. 33–34. DOI: 10.1145/3522664.3528611.

[4] *Enterprise MLOps Reference Design*. https://www.redhat.com/en/blog/enterprise-mlops-reference-design. Accessed: 2025-01-20. Dec. 2021.

[5] Megha Sinha, Sreekanth Menon, and Ram Sagar. "LLMOps: Definitions, Framework and Best Practices". In: *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET*. 2024, pp. 1–6. DOI: 10.1109/ICECET61485.2024.10698359.

[6] Josu Diaz-De-Arcaya et al. "Large Language Model Operations (LLMOps): Definition, Challenges, and Lifecycle Management". In: *2024 9th International Conference on Smart and Sustainable Technologies (SpliTech)*. 2024, pp. 1–4. DOI: 10.23919/SpliTech61897.2024.10612341.

[7] Richard Shan and Tony Shan. "Enterprise LLMOps: Advancing Large Language Models Operations Practice". In: *2024 IEEE Cloud Summit*. 2024, pp. 143–148. DOI: 10.1109/Cloud-Summit61220.2024.00030.

[8] Archanaa. N et al. "Comparative Analysis of News Articles Summarization using LLMs". In: *2024 Asia Pacific Conference on Innovation in Technology (APCIT)*. 2024, pp. 1–6. DOI: 10.1109/APCIT62007.2024.10673458.

[9] Tianyi Chen. "Challenges and Opportunities in Integrating LLMs into Continuous Integration/Continuous Deployment (CI/CD) Pipelines". In: *2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*. 2024, pp. 364–367. DOI: 10.1109/AINIT61980.2024.10581784.

[10] Indika Kumara et al. "Architecting MLOps in the Cloud: From Theory to Practice". In: *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. 2023, pp. 333–335. DOI: 10.1109/ICSA-C57050.2023.00076.

[11] Beatriz M. A. Matsui and Denise H. Goya. "MLOps: Five Steps to Guide its Effective Implementation". In: *2022 IEEE/ACM 1st International Conference on AI Engineering – Software Engineering for AI (CAIN)*. 2022, pp. 33–34. DOI: 10.1145/3522664.3528611.

[12] Chin-Yew Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*. 2004, pp. 74–81.

[13] Kishore Papineni et al. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[14] Awadhesh Srivastava. *LLM Fine Tuning with PEFT Techniques*. https://www.analyticsvidhya.com/blog/2023/10/llm-fine-tuning-with-peft-techniques/. Accessed: 2025-02-21. Dec. 2023.

[15] Pranab Sahoo et al. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*. 2024. arXiv: 2402.07927 [cs.AI]. URL: https://arxiv.org/abs/2402.07927.

[16] GeeksforGeeks. *Introduction to Recurrent Neural Networks*. https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/. Accessed: 2025-01-21. Nov. 2024.

[17] GeeksforGeeks. *what is LSTM*. https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/?ref=lbp. Accessed: 2025-01-21. Nov. 2024.

[18] *LLM Engineer's Handbook*. https://learning.oreilly.com/library/view/llm-engineers-handbook/9781836200079/. Accessed: 2025-01-20. Oct. 2024.

[19] Petra Heck. "What About the Data? A Mapping Study on Data Engineering for AI Systems". In: *2024 IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI (CAIN)*. 2024, pp. 43–52.

[20] G. Pradeep Reddy, Y. V. Pavan Kumar, and K. Purna Prakash. "Hallucinations in Large Language Models (LLMs)". In: *2024 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*. 2024, pp. 1–6. DOI: 10.1109/eStream61684.2024.10542617.

[21] Juan Martinez. *Supervised Fine-tuning: customizing LLMs*. https://medium.com/mantisnlp/supervised-fine-tuning-customizing-llms-a2c1edbf22c3. Accessed: 2025-01-20. Aug. 2023.

[22] Junyi Lu et al. "LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning". In: *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. 2023, pp. 647–658. DOI: 10.1109/ISSRE59848.2023.00026.

[23] Martin Fowler. *Continuous Integration*. https://martinfowler.com/articles/continuousIntegration.html. Accessed: 2025-01-20. Jan. 2024.