



MLOps Lifecycle Framework and LLMOps Integration – Repository Analysis

Repository Structure and Key Components

The repository **MA_Mlops-** is centered around a comprehensive research work titled "*Navigating MLOps: Insights into Maturity, Lifecycle, Tools, and Careers.*" It primarily consists of a unified **MLOps lifecycle framework** documentation, along with comparative analyses, roles and tools mapping, and cost considerations. The content is structured like an academic paper with sections for Introduction, a proposed MLOps lifecycle (with integrated LLMOps), a mapping of roles and tools across the lifecycle, and conclusions. Key contributions of this repository include: a **consolidated MLOps lifecycle framework** (synthesizing best practices from industry and academia), the **incorporation of LLMOps** into this lifecycle, a detailed catalog of **roles involved** in each phase, and an overview of **supporting tools (with associated costs)** aligned to the MLOps phases ¹. Figures and tables are used extensively to summarize insights – for example, **Figure 1** presents the proposed lifecycle diagram, and tables detail MLOps maturity levels, MLOps vs LLMOps comparisons, roles, and tools. The repository's structure can be viewed as containing:

- **MLOps Lifecycle Framework Documentation:** The core write-up describing each phase of the lifecycle and how they interconnect (with diagrams).
- **Comparative Analyses:** Tables comparing existing MLOps maturity models and contrasting MLOps vs LLMOps processes.
- **Roles and Tools Mapping:** A mapping of various team roles to lifecycle phases, and a list of tools/technologies for each phase (including whether they are open-source and their cost estimates).
- **Diagrams and Tables:** Supporting visuals, including a consolidated lifecycle flow (Figure 1) and a decision flowchart for LLMOps (Figure 2), as well as summary tables (for maturity levels, MLOps vs LLMOps differences, role mappings, tool costs, etc.).

This structured approach provides both **conceptual guidance** (phases, practices, and LLMOps integration) and **practical considerations** (what roles and tools are needed, and potential costs) in a single repository.

Unified MLOps Lifecycle Framework

A centerpiece of the repository is the **unified MLOps lifecycle framework** proposed by the authors. This framework consolidates and builds upon multiple earlier models to cover the end-to-end machine learning workflow from project inception to monitoring in production. In the Introduction, the authors note that existing MLOps lifecycle definitions vary widely; for instance, tech companies and researchers have defined different maturity levels and stages ² ³. *Table I* in the document compares these **MLOps maturity models** from various sources (Google, Microsoft, Amazon, IBM, and academic work), highlighting how they generally progress from manual processes to fully automated pipelines ² ³. This comparison sets the stage for the authors' unified lifecycle. They aim to integrate best practices into a single coherent framework ¹.

Framework Phases: The proposed lifecycle is illustrated in **Figure 1** and described in detail in the text. It defines a sequence of phases, each with specific objectives and outputs, creating a continuous loop for iterative improvement ⁴. The major phases in this framework include:

- **Business Needs (BN):** Establish the business objectives and success criteria for the ML project. This phase ensures that any ML effort is aligned with strategic challenges and key performance indicators. Business stakeholders (e.g. *Business Analysts*) set clear goals and **OKRs** so that model development targets real needs ⁵. This phase drives everything downstream, as it prioritizes what problems to solve and defines success in business terms.
- **Data Collection/Generation:** Once goals are set, data required to achieve them is collected or generated. Data can come from internal databases, web scraping, sensors, or crowd-sourced efforts. Ensuring sufficient quantity and quality is key; techniques like data augmentation or transfer learning may be used if data is limited ⁶ ⁷. Proper data management practices (versioning, lineage, quality control) are put in place at this stage.
- **Data Preparation:** The raw data is then cleaned and preprocessed to make it suitable for modeling. This involves handling missing or erroneous values, normalizing formats, and possibly feature engineering or data integration from multiple sources ⁸. The goal is to improve data quality and consistency, since robust input data is crucial for model performance.
- **Administrative & Setup (AS):** In parallel with data preparation, an administrative setup phase lays the groundwork for the project's infrastructure. This includes provisioning **project repositories** (code and data version control), creating **ML workspaces** for development (using platforms like MLflow, Kubeflow, JupyterHub, Databricks, etc.), preparing data labeling pipelines, setting up compute resources (CPU/GPU/TPU servers or cloud instances), defining user roles and access permissions, and establishing monitoring systems ⁹ ¹⁰. The administrative setup ensures the team has the needed tools and environments (dev, test, prod) to collaborate efficiently and scale as needed. Security and governance are also addressed here (e.g. using IAM for access control) ¹¹. This phase essentially creates the **infrastructure and devops foundation** for the MLOps pipeline.
- **Model Development (MD):** This phase covers the core modeling work. It is further divided into sub-stages in the authors' framework. The team conducts exploratory data analysis and feature engineering to transform prepared data into informative features ¹² ¹³. Based on the problem, appropriate algorithms or model architectures are selected (e.g. regression, decision trees, neural networks), and experiments are run to train models on the prepared data ¹⁴. Hyperparameter tuning and cross-validation are performed to optimize performance while avoiding overfitting ¹⁵. The framework also emphasizes **Responsible AI** in this phase – ensuring fairness, explainability, and ethical considerations are addressed during model training and evaluation ¹⁶. Once a model is trained, it is evaluated on validation data (using metrics like accuracy, F1, AUC, etc.) and checked against business objectives defined in the BN phase ¹⁷ ¹⁶. This MD phase is iterative – data scientists may go back to data prep or feature engineering multiple times ("Iterate") to improve results. Notably, **Figure 1** highlights that this Model Development phase includes a special sub-cycle (marked in forest-green) for LLMOps, which we will detail in the next section ⁴.
- **Version Control (VC):** The framework inserts a dedicated phase for version control of models and data. Here, all assets from the development phase are versioned: datasets, features, model artifacts, and configuration/hyperparameters are tracked using tools like Git and DVC (Data Version Control) ¹⁸. This ensures reproducibility and enables the team to rollback to earlier model versions if needed ¹⁹. Version control in MLOps extends beyond code to include data and models, providing an audit trail for experiments and facilitating collaborative development.

- **Continuous Integration (CI):** (Sometimes considered part of the deployment pipeline in other frameworks, the authors call it out explicitly.) In this phase, automated tests are run whenever new code or models are committed. **CI pipelines** execute unit tests, integration tests, and evaluate model performance on a hold-out test set to ensure that any new model meets quality standards before release ²⁰. This helps catch issues early and maintains stability as the project evolves. (CI is tightly integrated with version control in practice – e.g., triggering tests from a GitHub Actions workflow on each commit.)
- **Model Deployment (D):** Once a model passes CI tests, it moves into deployment. Deployment is broken into a **Test environment** and then **Staging/Production** in the framework ²¹ ²². In the **Test phase (DT)**, the model is deployed in a production-like environment for further validation. Batch and real-time inference tests are conducted to check performance under realistic conditions (throughput, latency) ²³. **Quality assurance (QA)** checks (e.g. smoke tests, integration tests, and fairness/ethics checks) are applied to ensure reliability and compliance before fully releasing the model ²³. If the model meets all criteria, a “**Gated Approval**” process is used – meaning a responsible person or team must sign off before promotion to production ²⁴. This gate prevents un-vetted models from affecting end users. Finally, in **Staging/Production**, the model is deployed to serve real users (for example, as a batch job, a real-time API, or an embedded model in an application) ²². The framework notes that performance benchmarks like MLPerf can be used to evaluate the deployment’s efficiency ²⁵. At this stage, the model is delivering value in a live environment. Any supporting infrastructure (CI/CD pipelines, containerization, Kubernetes, etc.) is utilized to manage scaling and reliability for end-user access.
- **Monitoring (M):** After deployment, continuous monitoring is critical. The Monitoring phase tracks the model’s performance in production (and also collects feedback from the test/staging environments) ²⁶. It covers metrics like prediction accuracy, data drift, latency, and system health indicators. Automated alerts or triggers are set up – for instance, if model accuracy drops or data characteristics shift, the system can trigger notifications or even pipeline actions (like initiating a new data collection or retraining cycle) ²⁷ ²⁶. Monitoring ensures the model remains aligned with business needs over time and that any issues are promptly addressed. Insights from monitoring feed back into the **Business Needs** or **Model Development** phases, creating a closed-loop for continuous improvement ⁴. In essence, the lifecycle is iterative: outputs from later phases inform updates to earlier phases, enabling adaptive MLOps practices.

Novel Contributions: The proposed framework is described as “*novel*” in that it **combines best practices from both industry and academia into one lifecycle** ¹. Unlike some earlier models that might omit certain steps, this framework explicitly includes phases like **Administrative Setup**, **Version Control**, **CI**, and distinct Test vs Production deployment stages to cover the full spectrum of MLOps tasks. It also uniquely embeds **LLMOPs (Large Language Model Ops)** within the standard ML pipeline (more on this below). By aligning their lifecycle with business objectives at the start and using monitoring to close the loop, the authors stress that this framework helps ensure ML deployments deliver *tangible value* and can evolve with changing needs ²⁸ ²⁹. The inclusion of roles and cost considerations (addressed later) is also a distinguishing aspect aimed at practical adoption. In summary, this unified MLOps framework provides a **comprehensive, phase-by-phase blueprint** for organizations to implement MLOps, from project conception (business case) all the way to running models and maintaining them in production.

Integration of LLMOps within the MLOps Lifecycle

A significant addition in this repository is the integration of **LLMOps** (Large Language Model Operations) into the standard MLOps lifecycle. Recognizing the rise of large language models (LLMs) in industry, the authors extended the framework to address the unique challenges of developing and deploying LLMs alongside typical ML models ¹ ³⁰. In **Figure 1**, the portion of the pipeline specific to LLMOps is highlighted in forest-green, indicating an “*LLMOps development cycle*” that intersects the Model Development phase ⁴. The framework makes clear that LLMOps does *not* stand alone – it requires all the surrounding MLOps phases (data, deployment, monitoring, etc.) to be in place – but certain tasks within Model Development differ when an LLM is involved ⁴.

LLM-specific Tasks: During the **Model Development (MD)** phase, the authors introduce a sub-phase labelled **MDL** (Model Development for LLMs). This sub-phase comprises tasks that are specific to working with large language models ³¹ ³²:

- *Data Privacy Governance*: Ensuring that the training data for LLMs does not violate privacy regulations (like GDPR/CCPA) or contain sensitive information, given LLM training often involves massive, diverse datasets ³³.
- *Vector Embedding*: Preparing and storing high-dimensional data (text, tokens) as numeric vectors, which is important for LLM workflows (e.g., embedding text for similarity search or as model input) ³³. This is particularly relevant for LLM-powered applications and is highlighted as beneficial for LLMOps ³⁴.
- *Existing LLM Selection*: Deciding whether to use a pre-trained large model versus training one from scratch. The framework suggests identifying suitable existing models (e.g., Meta’s **LLaMA**, EleutherAI’s **GPT-J**, Google’s **Flan-T5**, or image models like **Stable Diffusion**) that could be fine-tuned to the task ³⁵. Leveraging a proven model can greatly accelerate deployment and reduce the need for huge proprietary datasets and compute resources, if it meets the application’s needs ³¹.
- *Fine-Tuning & In-Context Learning*: If using an existing LLM, techniques such as fine-tuning (updating the model’s weights on domain-specific data) or in-context learning (providing carefully engineered prompts or examples to guide the model) are employed ³⁶. **Prompt engineering** becomes crucial here: crafting prompts or few-shot examples that coax the LLM to produce the desired output without needing full re-training ³⁷ ³⁸. The document notes examples of prompt engineering methods (zero-shot, few-shot, chain-of-thought, etc.) and emphasizes it as a key skill in LLMOps ³⁹ ³⁷. Fine-tuning, when done, is focused on the model’s later layers to specialize it efficiently, often improving accuracy and cost-efficiency compared to training from scratch ⁴⁰.
- *Evaluate & Refine with RLHF*: LLMs often require human-in-the-loop evaluation. The framework mentions **RLHF (Reinforcement Learning with Human Feedback)** as a technique for refining LLM outputs by having humans rate model responses and feeding that back into training ⁴¹. They highlight that LLMOps typically uses specialized metrics like BLEU, ROUGE (common for language tasks), and may involve human evaluators to judge model quality in addition to automated metrics ⁴² ⁴³. Ensuring fairness, accountability, and transparency (sometimes abbreviated FATE) in model outputs is also stressed, as LLMs can inadvertently produce biased or undesired content ⁴⁴.

These LLMOps tasks (marked in Figure 1 with dotted green boxes) complement the standard model development activities (data exploration, feature engineering, algorithm selection, etc., which the authors label as **MDT**, the traditional MD sub-section ⁴⁵). By including MDL, the framework explicitly covers what additional steps teams must consider when dealing with LLMs versus regular ML models.

LLM Decision Flow: The repository also provides guidance on **whether to train a custom LLM or use an existing one**. **Figure 2** is a decision flowchart that asks questions about data availability, computational resources, expertise, and maintenance requirements to help teams decide between building a new large model or leveraging a pre-trained LLM ³¹ ⁴⁶. The figure (and accompanying explanation) suggests that **using an existing LLM is generally preferable** unless the team truly needs a very specialized model and has “*ample high-quality data, significant computational resources, specialized ML expertise, and can continuously maintain the model*” ³¹. If those stringent conditions are not met, fine-tuning or prompt-engineering an existing large model is the recommended path for most cases, due to the saved time and innovation leverage ⁴⁷ ⁴⁸. The flowchart starts from business needs and asks, for example, if a “*specialized model*” is required, if *high-quality proprietary data* is available, if *high compute resources* are accessible, and if *experts and maintenance plans* are in place ⁴⁹. Only if most answers are “Yes” would it lead toward training a custom LLM; otherwise it points to using an existing model ⁴⁶ ⁵⁰. This kind of decision framework is a novel inclusion, recognizing the trade-offs in LLMOps.

MLOps vs LLMOps Comparison: To underscore how LLMOps extends traditional MLOps, the authors included *Table II* which directly **compares MLOps and LLMOps** along several aspects ⁵¹ ⁵². A brief summary of these comparisons:

- **Computational Demands:** Standard MLOps can often rely on CPUs or modest GPUs for model training, whereas LLMOps “**relies heavily on high-performance GPUs or TPUs**” due to the scale of models and data involved ⁵³. Training or fine-tuning large language models is extremely compute-intensive.
- **Cost Structure:** In typical MLOps, costs are driven by data collection, experimentation, and iterative development. In LLMOps, a huge portion of cost comes from **GPU-based computation and possibly licensing fees** for proprietary models ⁵⁴. Running large models in production (inference) also incurs significant ongoing compute cost.
- **Data Management:** Traditional ML projects require extensive preprocessing of structured datasets. LLMOps, by contrast, involves curating massive text corpora and may rely on techniques like tokenization and prompt engineering even with smaller data samples ⁵⁵. The emphasis is on quality and representativeness of data (to avoid biased or inappropriate outputs) rather than sheer volume alone.
- **Experimentation & Tuning:** Regular MLOps focuses on hyperparameter tuning and experimentation to optimize accuracy. LLMOps still experiments, but largely around **fine-tuning foundation models and refining prompts**, balancing model performance with computational efficiency and cost constraints ⁵⁶. The scope and techniques of experimentation differ (e.g., adjusting prompt formats or few-shot examples in LLMOps vs. tuning many algorithm parameters in MLOps).
- **Transfer Learning:** In conventional MLOps, teams sometimes train models from scratch for each task (minimal transfer learning). In LLMOps, **transfer learning is fundamental** – large models are pretrained on vast data and then *fine-tuned* for specific tasks rather than training from scratch ⁵⁷.
- **Model Evaluation & Human Feedback:** MLOps model evaluation typically uses standard metrics (accuracy, precision, F1, etc.) and usually does not require human intervention beyond perhaps error analysis. LLMOps relies on **specialized metrics** for language quality (e.g., BLEU, ROUGE scores for text generation) and often incorporates **human feedback loops** (via RLHF or manual review) to assess output quality like correctness, coherence, or absence of “hallucinations” ⁵⁸ ⁵².
- **Prompt Engineering:** For most traditional ML (structured data tasks), the concept of prompt engineering doesn’t apply. But in LLMOps, prompt engineering is “**crucial for optimizing**

performance, reducing hallucination, and ensuring reliable responses” ⁵⁹. Crafting the right prompts is a new discipline that has no analogue in classical MLOps.

- **Deployment Complexity & Latency:** Deploying a typical ML model might be straightforward on standard infrastructure with only moderate latency concerns. Deploying LLMs, however, often “**requires dedicated GPU infrastructure, complex pipelines (e.g., using frameworks like LangChain or LlamaIndex for chaining LLM operations), and faces significant latency challenges**” when serving real-time due to model size ⁶⁰ ⁶¹. Ensuring responsiveness of LLMs can be difficult, sometimes needing techniques like model distillation or approximation.
- **Governance & Monitoring:** MLOps governance mostly involves standard model monitoring and ethical practices, whereas LLMOps has heightened risks such as model hallucinations, data leakage, or misuse of generated content. This necessitates “**rigorous, advanced governance frameworks and continuous model monitoring**” beyond what simpler models might require ⁶². Organizations must implement policies to filter outputs, audit model behavior, and guard against misuse when deploying large language models.

Overall, by embedding LLMOps steps into the lifecycle and explicitly comparing them, the repository treats LLMOps as an evolution of MLOps. It shows that while the **surrounding pipeline remains similar**, the **development phase for LLMs** demands additional expertise, tools, and considerations. This integration ensures that teams adopting the framework can extend their MLOps practices to modern AI use-cases involving LLMs, rather than treating LLMOps as an entirely separate silo. The approach is forward-looking, given the increasing popularity of LLM-based applications, and fills a gap by marrying LLMOps with classical MLOps in one lifecycle model ³⁰.

Roles Mapped to MLOps Phases

Another valuable feature of this repository is the clear mapping of **team roles** to the MLOps lifecycle phases. The authors enumerate the various roles involved in a successful MLOps implementation, from the planning stages through to deployment and monitoring. In the paper, these roles are listed in *Table III*, which maps each role to the phases where it is most active and even aligns them with U.S. Department of Labor (DOL) job codes and average salaries (to aid in resource planning) ⁶³ ⁶⁴. By identifying the human resources needed, the framework helps organizations understand what skill sets and team members are required at each stage of the MLOps process.

Key Roles and Responsibilities: The roles span both **business-oriented** and **technical** positions, reflecting the interdisciplinary nature of MLOps. Below is a summary of the key roles and the lifecycle phases they impact (phase abbreviations in parentheses correspond to Figure 1, e.g. BN = Business Needs, AS = Administrative Setup, MD = Model Development, VC = Version Control, D = Deployment, M = Monitoring):

- **Business Analyst (BN, M):** Focuses on the **Business Needs** phase and later monitoring outcomes. Business Analysts ensure the project’s objectives align with strategic goals and define KPIs/OKRs ⁵. They translate business requirements into ML project plans and, post-deployment, verify that the model’s performance meets business metrics. (*DOL title: “Business Operations Specialists”*) ⁶³.
- **Data Architect (BN, MD, M):** Active in early phases to design the **data strategy**. They decide how data will be collected, stored, and accessed. During model development, Data Architects help ensure the data pipeline supports the model’s needs, and they monitor data quality and integrity in production. (*DOL: “Database Architects”*) ⁶⁵.

- **Data Engineer (BN, MD, M):** Responsible for **data ingestion and preparation**. In the BN phase, Data Engineers set up data collection pipelines. In **Model Development**, they implement data transformations, feature engineering, and ensure datasets are available for training. They also handle data versioning (VC phase) and troubleshoot data issues in Monitoring. (*DOL: grouped under "Data Scientists" occupation*) ⁶⁵.
- **Data Scientist (BN, MD, M):** Focuses on the **core modeling** in MD. Data Scientists perform EDA, select algorithms, train models, and tune hyperparameters. They work closely with Data Engineers to get the right data and with ML Engineers for deployment. They also help interpret monitoring results (M) to decide if model retraining is needed. (*DOL: "Data Scientists"*) ⁶⁵.
- **ML Engineer (MD, D, M):** Often overlapping with Data Scientist in skills, the ML Engineer leans more into the **engineering side of model development and deployment**. They take models (sometimes from Data Scientists) and optimize them for production, handling issues like scaling, containerization, and CI/CD. In deployment (D), they integrate models into applications or pipelines. In Monitoring (M), ML Engineers set up logging, manage model version updates, and ensure infrastructure supports the model's runtime needs. (*DOL: "Computer Occupations, All Other" – reflecting a mix of skills*) ⁶⁶.
- **Prompt Engineer (MDL, D, M):** A newer role emerging with LLMOps. The Prompt Engineer specializes in crafting and refining prompts for LLMs (during the **LLM-specific development** sub-phase MDL) to achieve desired outputs ³⁹. They also help design how the LLM is integrated into products at deployment (e.g., constructing prompt templates, managing prompt-response flows), and evaluate outputs during monitoring (checking for issues like hallucinations or user feedback). (*Not a standard DOL category; mapped to a general software occupation code*) ⁶⁶.
- **System Administrator (AS, VC, D, M):** Sets up and maintains the **infrastructure**. In the Administrative Setup phase (AS), System Admins provision servers, networking, and cloud resources. They manage user accounts and permissions. During Version Control (VC) and Deployment (D), they might maintain CI/CD servers, artifact repositories, and environment configurations. In Monitoring (M), they ensure system-level metrics (CPU/GPU utilization, memory, etc.) are collected and that the system stays secure and up-to-date. (*DOL: "Network and Computer Systems Administrators"*) ⁶⁷.
- **Network Administrator (AS, VC, D, M):** Similar to System Administrator, with emphasis on **network architecture** – setting up VPNs, firewall rules, data transfer pipelines, and ensuring connectivity between components (e.g., between data storage and training servers, or between model API and users). Active in infrastructure setup and maintenance phases. (*DOL codes same as SysAdmin*) ⁶⁷.
- **DevOps Engineer (AS, MD, VC, D, M):** Bridges software development and operations in the ML context. DevOps Engineers design automated build/test/deploy pipelines (CI/CD), manage containerization and orchestration (Docker, Kubernetes), and integrate model code with infrastructure. They work throughout the pipeline: during development (MD) they set up continuous integration, in deployment (D) they handle releases and scaling, and in monitoring (M) they implement logging/alerting for the ML system. In this framework, **MLOps Engineer** is listed similarly; many organizations may use the terms interchangeably or consider MLOps Engineer a specialization of DevOps for ML. (*Both mapped to DOL's "Software Developers" category*) ⁶⁸.
- **Software Engineer (AS, MD, VC, D, M):** Traditional software developers also play a role – building the **application logic** that uses the model. They contribute in Model Development by helping wrap models into applications or APIs, in Deployment by integrating model endpoints into production services, and in Monitoring by maintaining the overall software system in which the model operates. They ensure that the **surrounding application** (e.g., a web service or user interface) can consume the model's predictions correctly ⁶⁹ ⁷⁰. (*DOL: "Software Developers"*).
- **QA Engineer (DT, VC):** Quality Assurance engineers design and run tests to validate the ML system. In the **Deployment-Test (DT)** phase, they perform testing of the model in a staging environment –

this includes not only standard software tests (unit/integration tests using frameworks like JUnit or PyTest) but also validation of model accuracy on test data and checks for regression compared to prior models ⁶⁹. They ensure the model's outputs meet specifications and that new deployments haven't broken any functionality. QA may also be involved in version control by verifying that all changes are properly tracked and reproduced for testing. (DOL: "Software Quality Assurance Analysts") ⁷¹.

- **Security and Compliance Officer (BN, AS, MD, VC, D, M):** Oversees **governance, ethical, and security aspects** across the entire lifecycle. In Business Needs (BN), they might ensure the project plans align with regulatory requirements (for instance, if working with sensitive data). During Administrative Setup (AS), they enforce security best practices (access control, data encryption). In Model Development (MD), they verify that data privacy is maintained (no leaking of personal data into models) and that the models themselves are secure (robust against adversarial attacks). During Deployment (D), they assess the security of the deployment environment and that compliance standards (like GDPR, HIPAA if applicable) are met. In Monitoring (M), they track any security incidents or compliance violations and ensure audits are conducted. Essentially, this role is about **risk management** in ML Ops – making sure the entire pipeline and product adhere to laws and policies, and protecting against breaches or misuse. (DOL: "Information Security Analysts") ⁷².

Table III in the document provides further detail by aligning each of these roles with official labor categories and even listing **monthly wage estimates** for each (based on 2023 data) ⁶³ ⁷³. This is quite practical – it allows organizations to anticipate the **human resource costs** of MLOps initiatives. For example, roles like Data Scientist or Software Engineer had estimated monthly wages around \\$9k–\\$11k, whereas a Business Analyst was around \\$7.4k ⁶³. By including these, the authors make the point that adopting MLOps is not just a technical endeavor but also a planning exercise to staff the right roles and budget for them.

Overall, mapping roles to phases clarifies *who* should be doing *what* at each stage of the pipeline. For instance, it becomes clear that you need business-facing roles early on, data-centric roles through development, engineering roles for deployment, and oversight roles (DevOps/MLOps, Security) consistently throughout. This mapping helps identify skill gaps – e.g., if a team lacks a specialist in one of these areas, that phase could be a bottleneck. The repository explicitly states that "*highlighting the necessary roles and skill requirements allows teams to identify gaps and plan for skill development and labor costs.*" ⁷⁴ ¹. By planning roles per phase, organizations can better ensure they have the right expertise at each step of the MLOps lifecycle.

Tools and Cost Considerations Across MLOps Phases

In addition to roles, the repository outlines a wide array of **tools and technologies** that support each phase of the MLOps lifecycle. This is presented in *Table IV*, which lists tools, their typical cost (per month), whether they are open-source, and checkmarks for which phases they apply to ⁷⁵ ⁷⁶. The inclusion of cost information (with assumptions like 10 users for pricing, and noting tools with consumption-based pricing as "CB") is particularly useful for **resource planning** ⁷⁷. By mapping tools to phases, the authors give readers a practical toolkit for implementing the framework, and a sense of budgeting for software/cloud services.

Phase-Wise Tool Examples: The tool list is extensive, covering everything from data management to model deployment and monitoring. Some notable examples for each phase include:

- *Business Needs / Planning:* Tools for project management and tracking business requirements, such as **Wekan** or **Taiga** (both project boards for agile planning). These help translate high-level business objectives into technical tasks ⁷⁸ ⁷⁹. Also listed are collaboration platforms like **Azure DevOps**, which can manage backlogs and track work items aligned with ML projects ⁸⁰. These planning tools often have free tiers or modest costs (Taiga, for instance, around \\$70/month for certain hosted plans) ⁸¹.
- *Administrative Setup:* Infrastructure-as-code and automation tools are key here. For example, **Terraform**, **Ansible**, **Bicep**, and **AWS CloudFormation (ARM)** are mentioned for provisioning cloud resources and configuring environments ⁸². These are usually open-source or free to use (cost is mainly the time/effort to write scripts). Containerization and environment management tools like **Docker** (implied via container registries) and environment managers like **Conda** could also be relevant, though specific mention in the table focuses on provisioning and config. Setting up data labeling and dataset management might involve tools like **Labelbox** or **SageMaker Ground Truth** (the text references these in Administrative Setup phase ⁸³), but in the tools table, we see **AWS Lake Formation** (for data cataloging) and **AWS SageMaker** itself as a platform that spans many phases (from data prep to deployment) ⁸⁴ ⁸⁵. Many of the cloud-based tools are marked "CB" (consumption-based pricing) – meaning their cost will scale with usage (AWS services, etc.).
- *Data/Model Development:* A plethora of **ML libraries and frameworks** are listed. Core libraries like **PyTorch**, **TensorFlow**, and **Pandas** are naturally included (these are open-source and cost \\$0 themselves) ⁸⁶. For experiment tracking and model management, **MLflow** is a key tool (open-source, used for logging experiments, storing models, etc.) and it appears in the table covering multiple phases (MD, VC, CI, D, M) ⁸⁷. **Databricks** (which offers a managed environment for data engineering and ML) is marked as consumption-based and spans from data engineering (AS) to model development (MD) ⁸⁸. **Azure ML Studio** (Microsoft's cloud ML platform) is another example, listed as \\$0 (likely for basic tier) and applicable to many phases from data prep to deployment ⁸⁸. For **LLMops-specific** development, tools like **LangChain**, **LlamaIndex**, and **Hugging Face Hub** are noted. LangChain and LlamaIndex (both open-source, \\$0) are libraries to build applications around LLMs (for chaining prompts, retrieval augmentation, etc.) ⁸⁹ ⁹⁰. Hugging Face is an open platform that provides model repositories and APIs for many pre-trained models (also \\$0 for basic use) ⁸⁷ – very relevant to LLMops phases (MDL, etc.). The presence of these indicates that the framework's toolset explicitly covers new **LLM tooling** in addition to traditional ML tools.
- *Version Control & CI/CD:* Standard software tools are included here: **Git** (and services like GitLab, Bitbucket) for code and model version control ⁹¹, and **DVC (Data Version Control)** for tracking data and model file versions alongside code ⁹¹. These are typically open-source (Git itself, DVC) or have free tiers (e.g., Bitbucket offers free repos up to a limit). For Continuous Integration, tools like **Jenkins** (open-source CI server) are listed ⁹². Jenkins is free, requiring only the server to run on, and is applicable to automating testing (CI phase) and even deployment steps. The table also lists **Azure DevOps** (which includes CI/CD pipelines) and **GitLab Enterprise** (which has CI runners) – these have subscription costs (Azure DevOps around \\$30/month for certain users, GitLab Enterprise around \\$290/month for 10 users as per table) ⁸⁷ ⁹³. Including these underscores that CI/CD pipeline setup is an integral part of MLOps toolchain.
- *Deployment & Model Serving:* Several tools address model serving and deployment. **Kubeflow** stands out – an open-source platform for orchestrating ML workflows on Kubernetes, which covers pipeline automation as well as model serving in production; it's shown as \\$0 and applicable to almost all

phases (since Kubeflow has components for experiment tracking, serving, etc.)⁹⁴ ⁹⁵. For serving models (especially LLMs or other ML models at scale), **TensorFlow Serving** and **OpenLLM** are mentioned (both open-source)⁹⁶. **Seldon Core** is another open-source framework listed, used to deploy machine learning models on Kubernetes with advanced routing, which fits into deployment and monitoring phases⁹⁷. Container registries like **Azure Container Registry** are included (low cost, e.g., \\$5/month) since deploying models often involves containerizing them⁹⁸. The table also references **Composer by MosaicML** and **H2O.ai** (both are platforms to train models efficiently or provide AutoML; MosaicML's Composer is open-source for model training acceleration, H2O.ai offers an enterprise ML platform – marked CB)⁹⁵. These might be used in model development or deployment pipelines for optimizing models.

- **Monitoring & Observability:** Tools for logging, monitoring, and alerting are crucial in the M phase. The stack often includes **Prometheus** (metrics collection), **ELK Stack** – Elasticsearch, Logstash, Kibana (logging and analytics), and **Grafana** (dashboarding)⁹⁹. Indeed, the table lists the ELK stack and Grafana as \\$0 (open-source) for self-hosted versions⁹⁹. **Grafana Cloud** (hosted Grafana service) is listed with a small cost (e.g., \\$19/month)⁹⁹. **WhyLabs** and **Langsmith** are examples of newer ML-specific monitoring solutions: WhyLabs (with a cost around \\$250 for usage tier) provides AI observability to detect data/model drifts, and Langsmith (a tool from LangChain for tracing and debugging LLM applications, listed at \\$390 likely for a team plan) is specifically geared towards LLMOps monitoring⁹⁹. The inclusion of **OpenLLMetrics** (mentioned alongside ELK in the table) indicates tools emerging for telemetry of LLMs. Additionally, **MLFlow** appears again here – its model tracking and serving components aid in monitoring model versions and performance over time⁸⁷.

It's worth noting the repository not only enumerates these tools but also indicates which phase each is relevant to. Many tools span multiple phases (for example, MLFlow in development, version control, and deployment; or cloud platforms like SageMaker covering data prep through deployment). The table also flags whether tools are open-source (most are, which can help keep costs down). For proprietary or cloud services, providing the monthly cost (for a team of ~10) gives a ballpark for budgeting.

Cost Estimation and Resource Planning: One of the repository's goals is to aid in **cost estimation** for MLOps projects. By mapping both roles (with salary estimates) and tools (with service costs) to each phase, one can derive an estimate of the operational expense of implementing MLOps at a given maturity level. The authors explicitly state that "*mapping roles and tools to MLOps phases aids in cost estimation and resource allocation.*"¹⁰⁰. For instance, if an organization is at an early maturity level (largely manual processes), they might not yet invest in costly tools or many specialized roles; as they aim for higher maturity (automation, CI/CD, etc.), they can refer to the framework to see which new roles to hire (and their salary range) and which tools or platforms they may need to budget for. The provided wage and tool pricing data can thus be used to estimate monthly or annual costs of an MLOps initiative at scale. This kind of **financial planning insight** is a novel addition not commonly found in technical MLOps guides. It grounds the framework in real-world implementation concerns (headcount and software costs).

Additionally, cost-awareness shows up in the LLMOps guidance as well – for example, the decision to use an existing LLM versus train a new one is heavily influenced by resource availability and cost (since training a large model from scratch could be prohibitively expensive)³¹. The fine-tuning approach described is partly to improve **cost efficiency** by only training a portion of the model rather than an entire large model⁴⁰. They also note that optimizing prompts can reduce the need for costly retraining runs¹⁰¹ ¹⁰². Thus, cost considerations are woven throughout the lifecycle: from planning (aligning with business ROI), to choosing tools (open-source vs paid), to deciding ML vs LLM strategies.

In summary, the repository's tables of tools and roles serve as a **practical checklist** for each phase of MLOps – one can ask “Do we have the right person for this step? Do we have or can we afford the right tool to do it efficiently?” By answering those, an organization can map out the investments needed to reach a certain MLOps maturity. The comprehensive nature of these tables (covering everything from business planning software to model monitoring solutions) underscores the interdisciplinary effort required for effective MLOps.

Summary of Maturity Levels and Visual Aids

To tie everything together, the repository provides summary visuals and comparisons that help readers quickly grasp the state of MLOps practices:

- **MLOps Maturity Levels:** Table I (from earlier) encapsulates various maturity models. While each source uses different terminology and number of levels, the general progression is consistent: from no MLOps or manual processes (Level 0) up to fully automated continuous training and deployment (higher levels) ^{103 104}. The authors emphasize that higher maturity correlates with higher automation and better alignment between models and business outcomes ^{105 106}. Understanding these levels can help an organization self-assess where they stand (e.g., using ad-hoc scripts vs. having a CI/CD for ML) and what the next level of improvement entails. The framework they propose can be seen as a roadmap to achieving the higher maturity levels noted by Google, Amazon, etc.
- **Lifecycle Diagram (Figure 1):** The Machine Learning Lifecycle Development diagram (Figure 1) provides a **holistic view** of all phases and their interactions in a single schematic ⁴. It shows the flow from **Business Needs** at top-left, through **Administrative Setup**, into **Model Development** (with the LLMOps sub-cycle highlighted), then down to **Version Control**, **Model Deployment** (split into Test and Production stages), and finally **Monitoring**, which feeds back up to the start ⁴. Each phase block in the diagram is annotated with key activities or tools (for example, under Model Deployment it lists steps like QA tests, batch vs online inference, gated approval, etc., and under Monitoring it shows triggers for retraining) ^{107 108}. This figure is essentially a blueprint one could put on a wall to guide an MLOps implementation. The inclusion of the LLMOps cycle (in green) within Model Development is visually significant – it conveys that while developing large language models has some unique loops, it remains part of the broader ML lifecycle (needing version control, needing deployment, etc., just like any model). The figure also annotates phase abbreviations (BN, AS, MD, MDL, MDT, VC, CI, D, M), which tie back to the roles and tools tables (Table III and IV use those abbreviations) ^{109 77}. This consistency helps cross-reference where each role/tool is used.
- **LLM Decision Diagram (Figure 2):** As discussed, Figure 2 visualizes the decision logic for “*Train Custom LLM vs Use Existing LLM*.” It’s essentially a flowchart starting with “*Start*” (the project idea) and “*Business Needs / Specialized Model?*” then branching through questions about data, resources, expertise, maintenance, etc., leading to outcomes *Use Existing LLM* or *Train Custom LLM* ^{46 50}. This diagram is a handy summary for any team contemplating an LLM project – it distills a complex decision into a manageable set of considerations. By following it, teams can justify their approach (e.g., “**We chose to fine-tune GPT-3 instead of building our own because we answered NO to having proprietary data and unlimited compute**”). It also implicitly encourages teams to think about maintenance (“Regular Maintenance” and “Experts Available” are part of the criteria), which is sometimes overlooked when initiating ambitious AI projects.

- **Comparison Table (MLOps vs LLMOps):** Table II, as summarized above, gives a side-by-side comparison that is essentially a **cheat sheet** for differences one must plan for if dealing with LLMs. For instance, a project that was straightforward in MLOps might become significantly more expensive and complex in deployment if it shifts to LLMOps. This table helps set those expectations clearly (e.g., expecting higher latency, need for human evals, etc.) ⁵¹ ⁵⁹.
- **Roles and Tools Tables:** Tables III and IV act as summary references as well. Table III (roles) not only lists the roles but also shows their **DOL codes and wages**, which could be used in a management presentation to argue for certain hires (by pointing to standard occupational data) ⁶³ ⁷³. Table IV (tools) is practically a condensed **MLOps technology landscape**. It could guide tool selection by phase – for example, if a team is looking to improve their CI/CD, the table reminds them of options like Jenkins or GitLab CI; if they need a feature store or experiment tracker, it lists tools like Feast or MLflow (Feast is mentioned in lines around the table content as well, likely under feature store in VC phase) ⁷⁶ ⁹¹. The cost column in Table IV can also serve as a quick budgeting guide, as discussed.

In conclusion, the **MA_Mlops-** repository provides a **thorough analysis and framework for MLOps** that spans conceptual models to practical checklists. It introduces a unified lifecycle that can demystify MLOps for organizations, incorporates cutting-edge considerations like LLMOps, and doesn't shy away from the "people and cost" side of adopting these practices. By organizing the information into clear sections with visuals and tables, it allows different stakeholders to find relevant insights – engineers can look at the tools for each phase, project managers can look at roles and costs, and leadership can appreciate the maturity model and how the lifecycle ties ML efforts to business value. All these elements make it a valuable guide for navigating the complex landscape of operationalizing machine learning in the real world. The work serves as a reference blueprint, and as the authors suggest, it could be extended with case studies or security analyses in the future ¹¹⁰ ¹¹¹, but as is, it already maps the **who, what, and how** of MLOps in a comprehensive manner.

Sources:

- Stone *et al.*, “*Navigating MLOps: Insights into Maturity, Lifecycle, Tools, and Careers*,” arXiv preprint 2503.15577 (2025) ¹ ⁴ ⁶².
- MA_Mlops- Repository content (tables and figures) ² ⁶³ ⁷⁶ ¹⁰⁰.

https://www.researchgate.net/publication/390038755_Navigating_MLOps_Insights_into_Maturity_Lifecycle_Tools_and_Careers/fulltext/67dcd9c0e62c604a0df7b7a2/Navigating-MLOps-Insights-into-Maturity-Lifecycle-Tools-and-Careers.pdf

²⁷ (PDF) *Navigating MLOps: Insights into Maturity, Lifecycle, Tools, and Careers*
https://www.researchgate.net/publication/390038755_Navigating_MLOps_Insights_into_Maturity_Lifecycle_Tools_and_Careers

¹⁰⁵ ¹⁰⁶ *Navigating MLOps: Insights into Maturity, Lifecycle, Tools, and Careers*
<https://arxiv.org/html/2503.15577v1>