Module 6
LLMOps

Learning Objectives

By the end of this module you will:

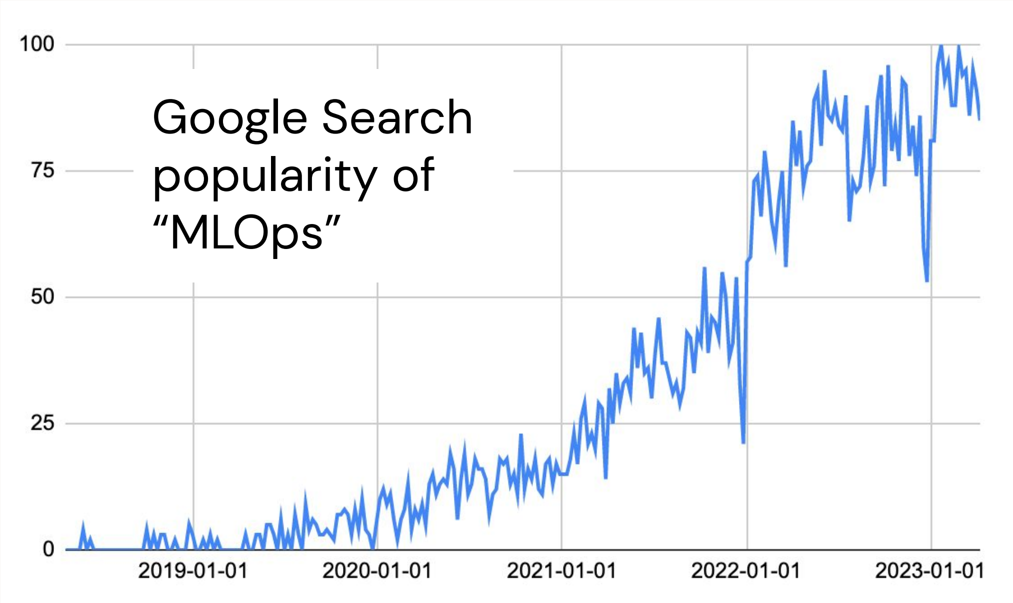
- Discuss how traditional MLOps can be adapted for LLMs.
- Review end-to-end workflows and architectures.
- Assess key concerns for LLMOps such as cost/performance tradeoffs, deployment options, monitoring and feedback.
- Walk through the development-to-production workflow for deploying a scalable LLM-powered data pipeline.

MLOps

ML and AI are becoming critical for businesses

Goals of MLOps

- Maintain stable performance
  - Meet KPIs
  - Update models and systems as needed
  - Reduce risk of system failures



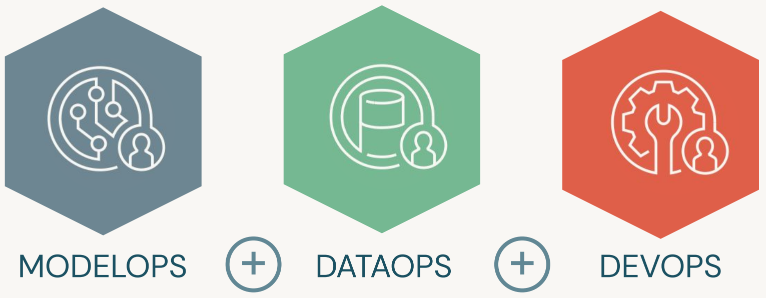
- Maintain long-term efficiency
  - Automate manual work as needed
  - Reduce iteration cycles dev→prod
  - Reduce risk of noncompliance with requirements and regulations

Source: google.com

Traditional MLOps:
“Code, data, models, action!”

MLOps = DevOps + DataOps + ModelOps

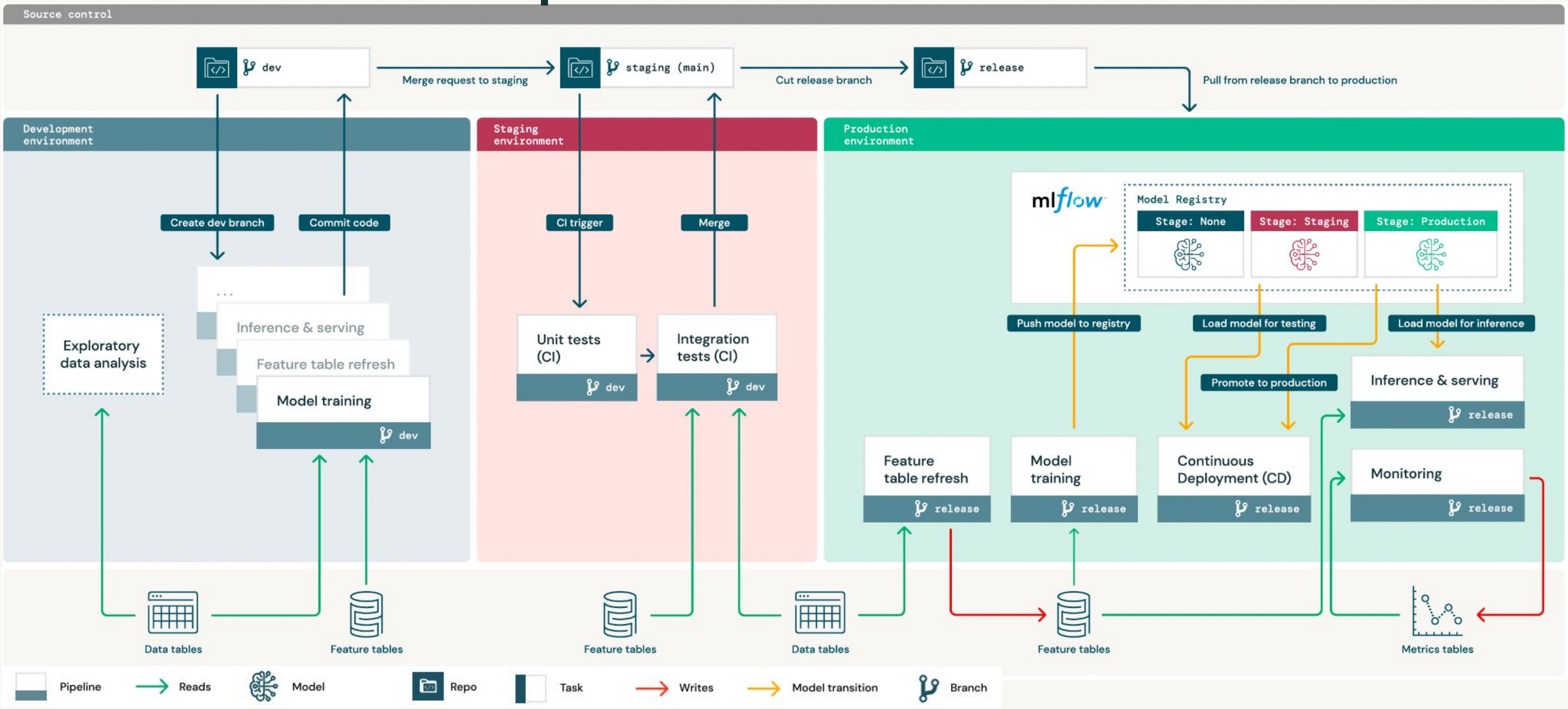
A set of processes *and automation*
for managing ML *code, data and models*
to improve perfor*mance and long-term efficiency*



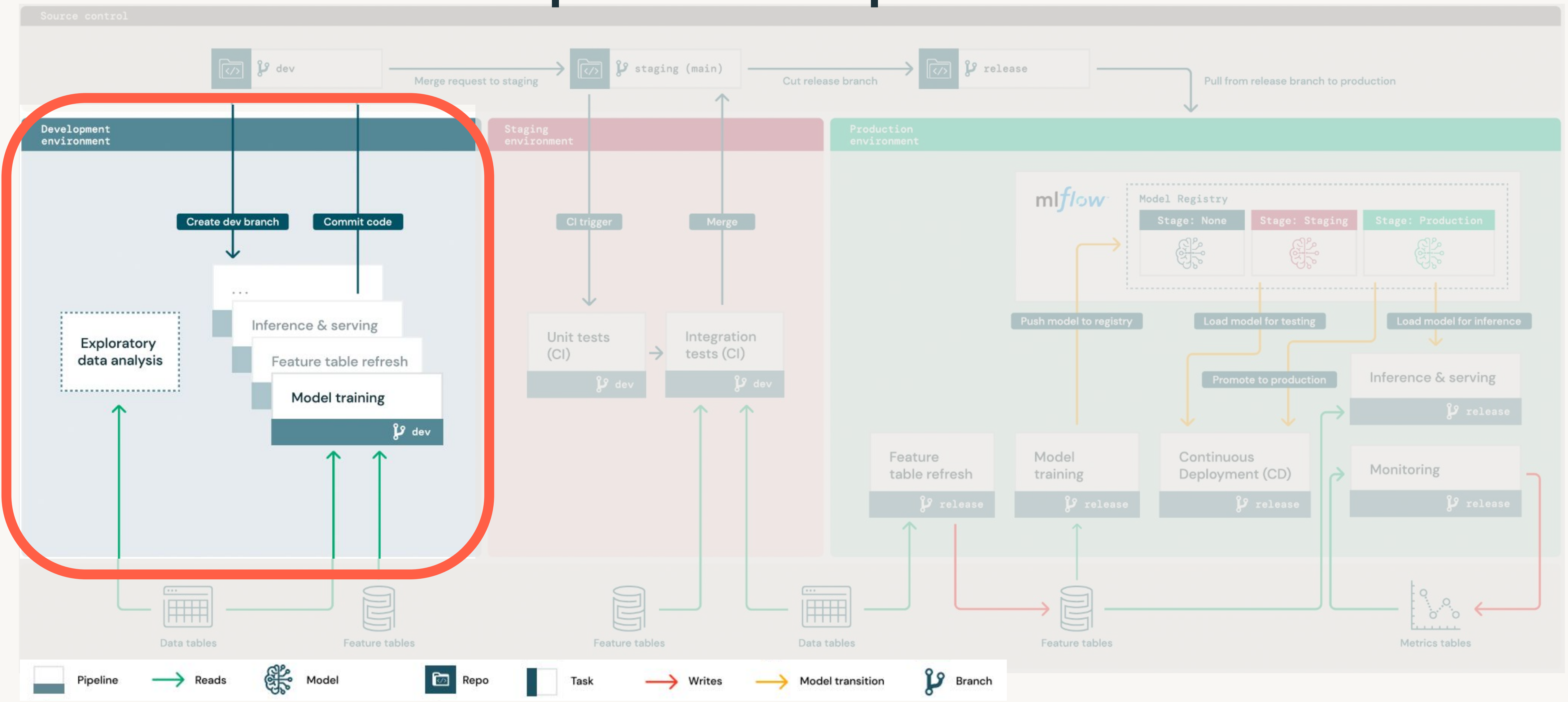
- Dev-staging-prod workflow
- Testing and monitoring
- CI/CD
- Model Registry
- Feature Store
- Automated model retraining
- Scoring pipelines and serving APIs
- …

See "The Big Book of MLOps" for an overview

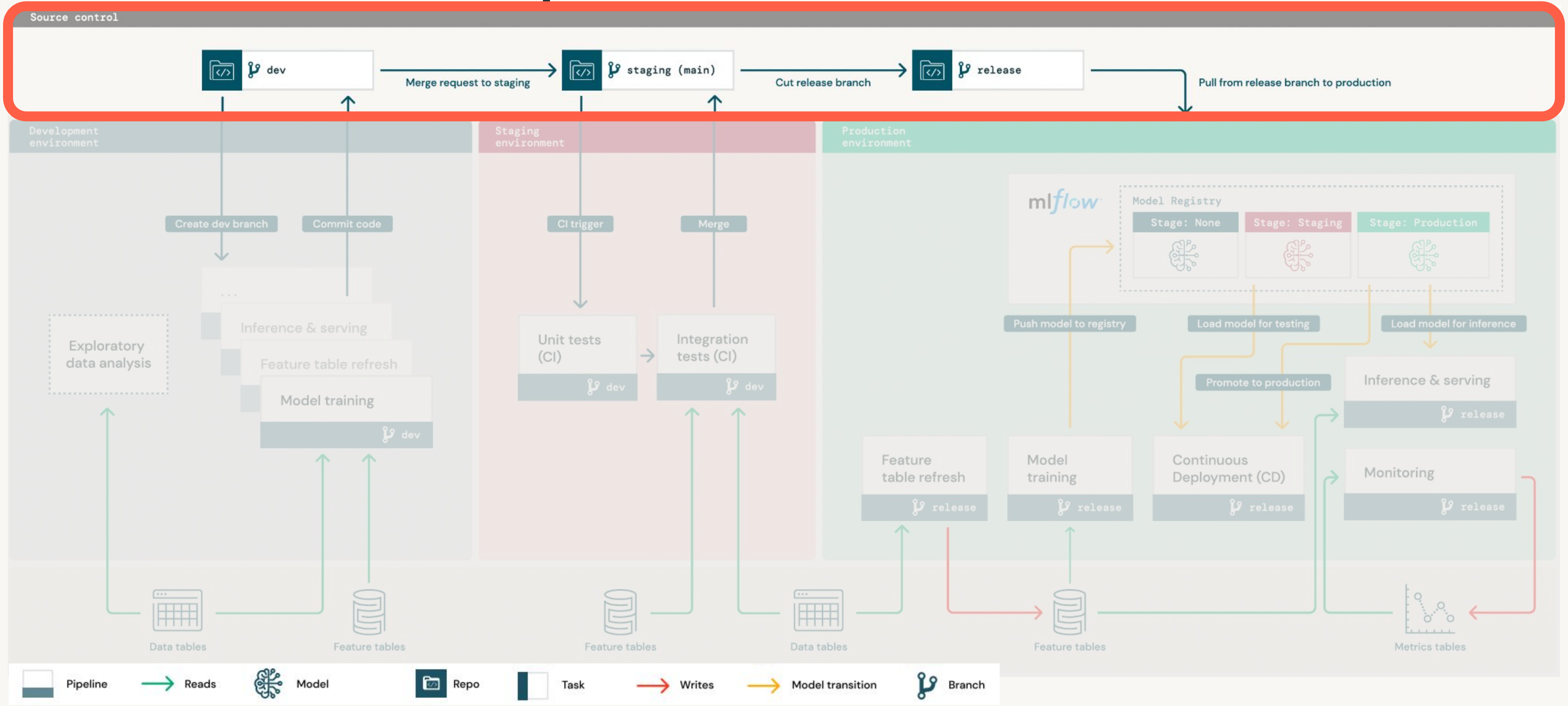
Traditional MLOps architecture



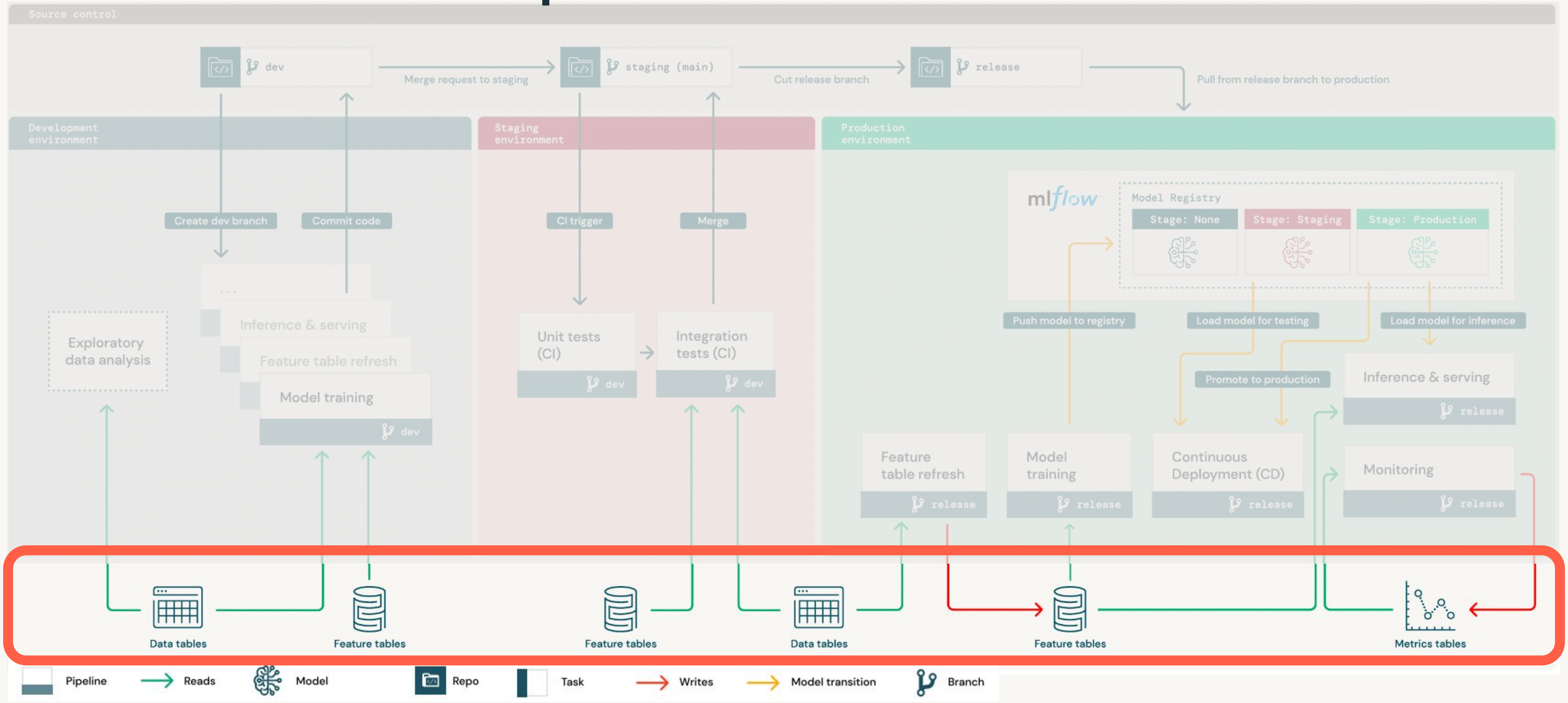
Traditional MLOps: Development environment



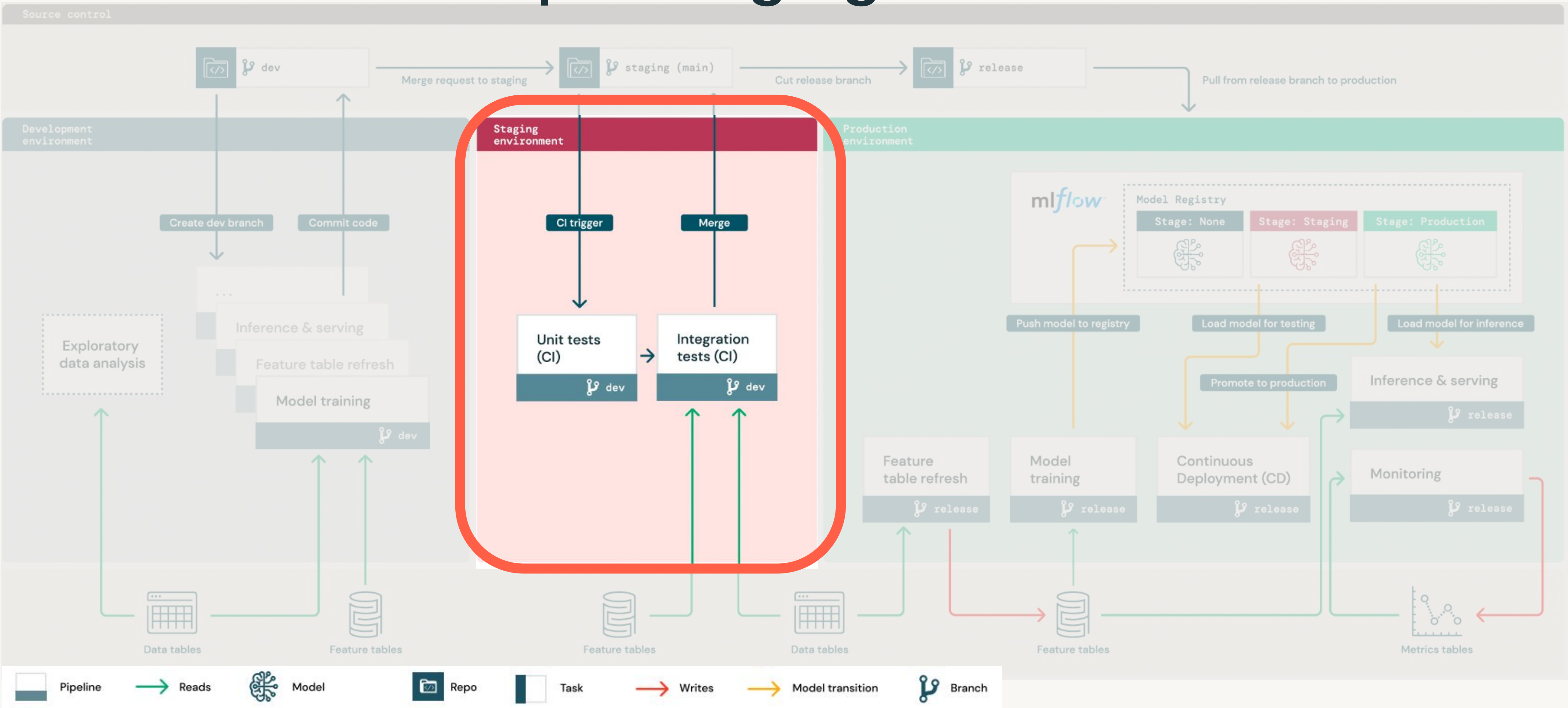
Traditional MLOps: Source control



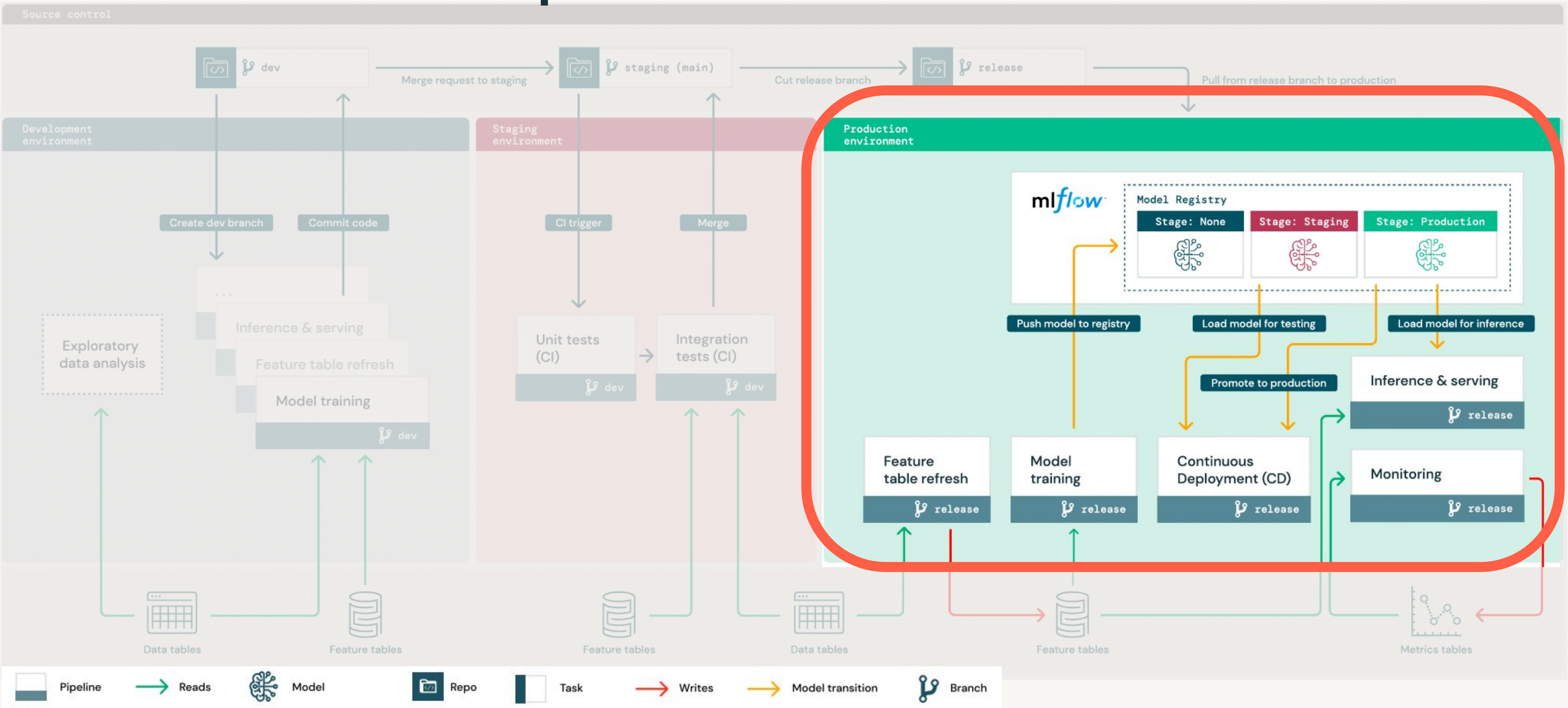
Traditional MLOps: Data



Traditional MLOps: Staging environment



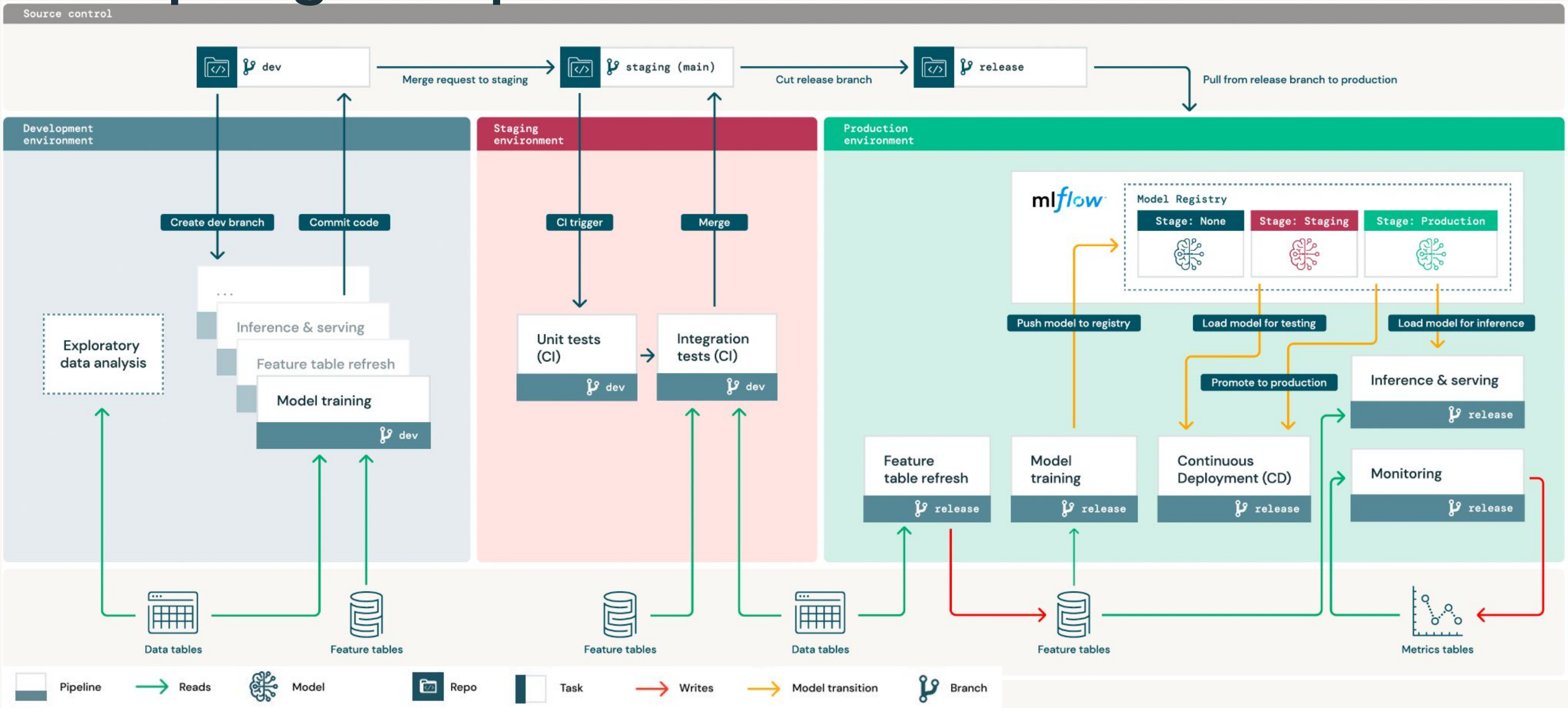
Traditional MLOps: Production environment



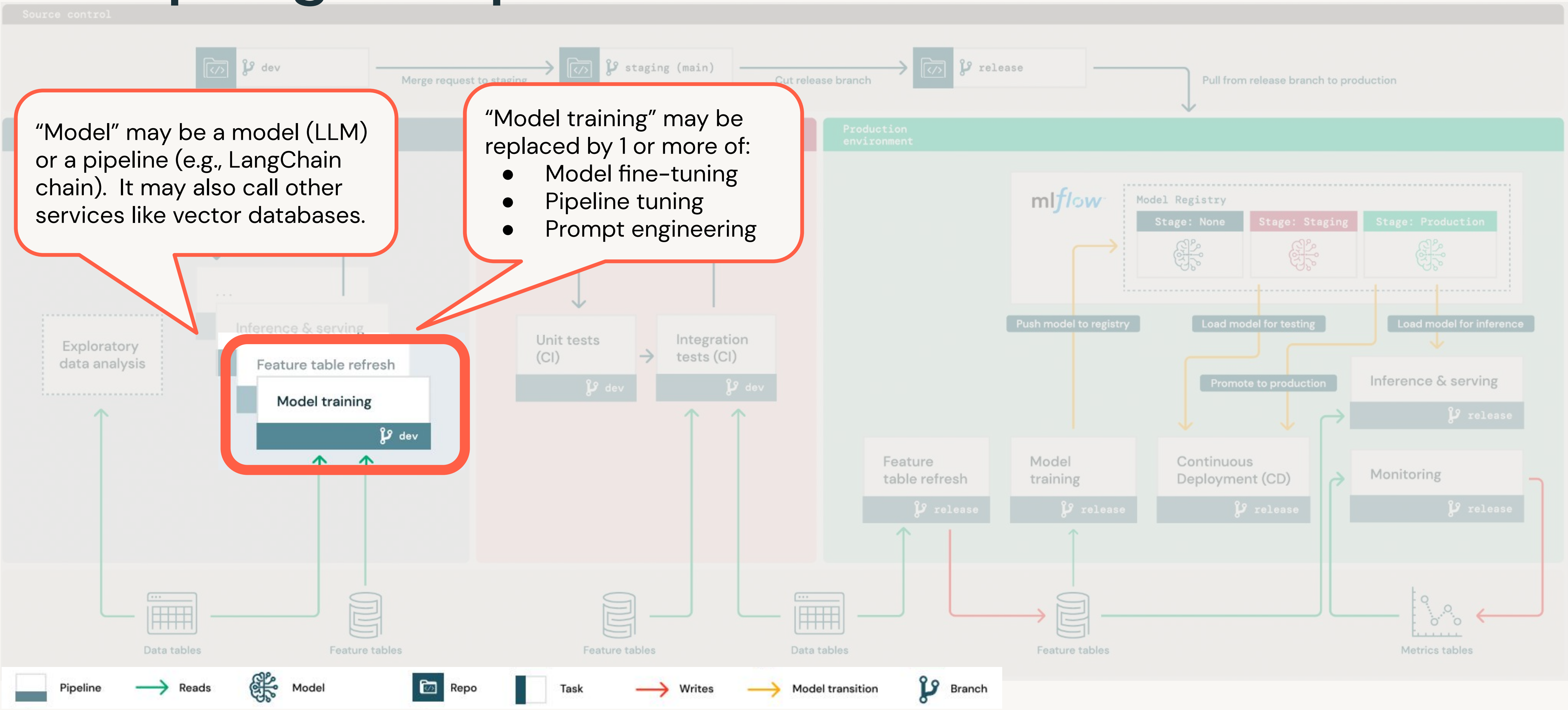
LLMOps:

“How will LLMs change MLOps?”

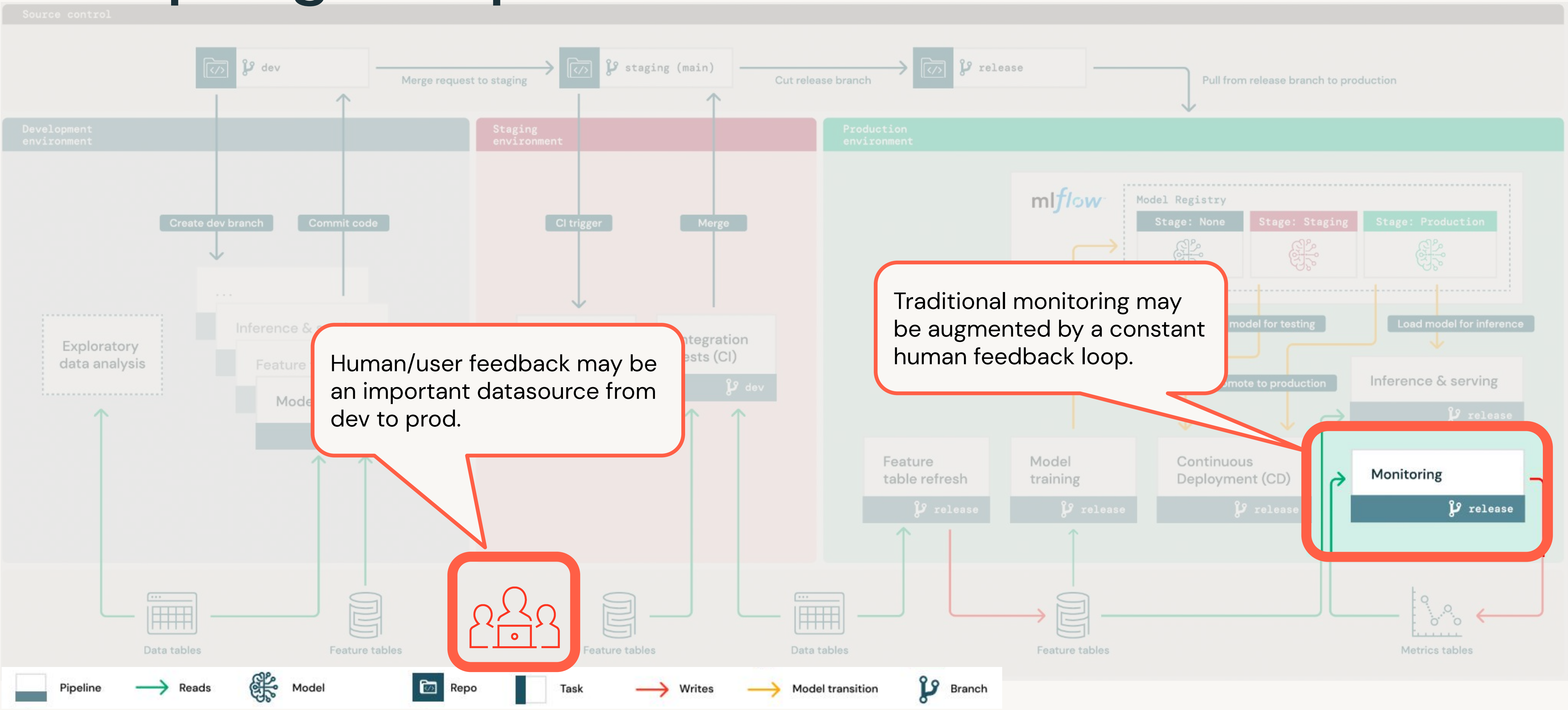
Adapting MLOps for LLMs



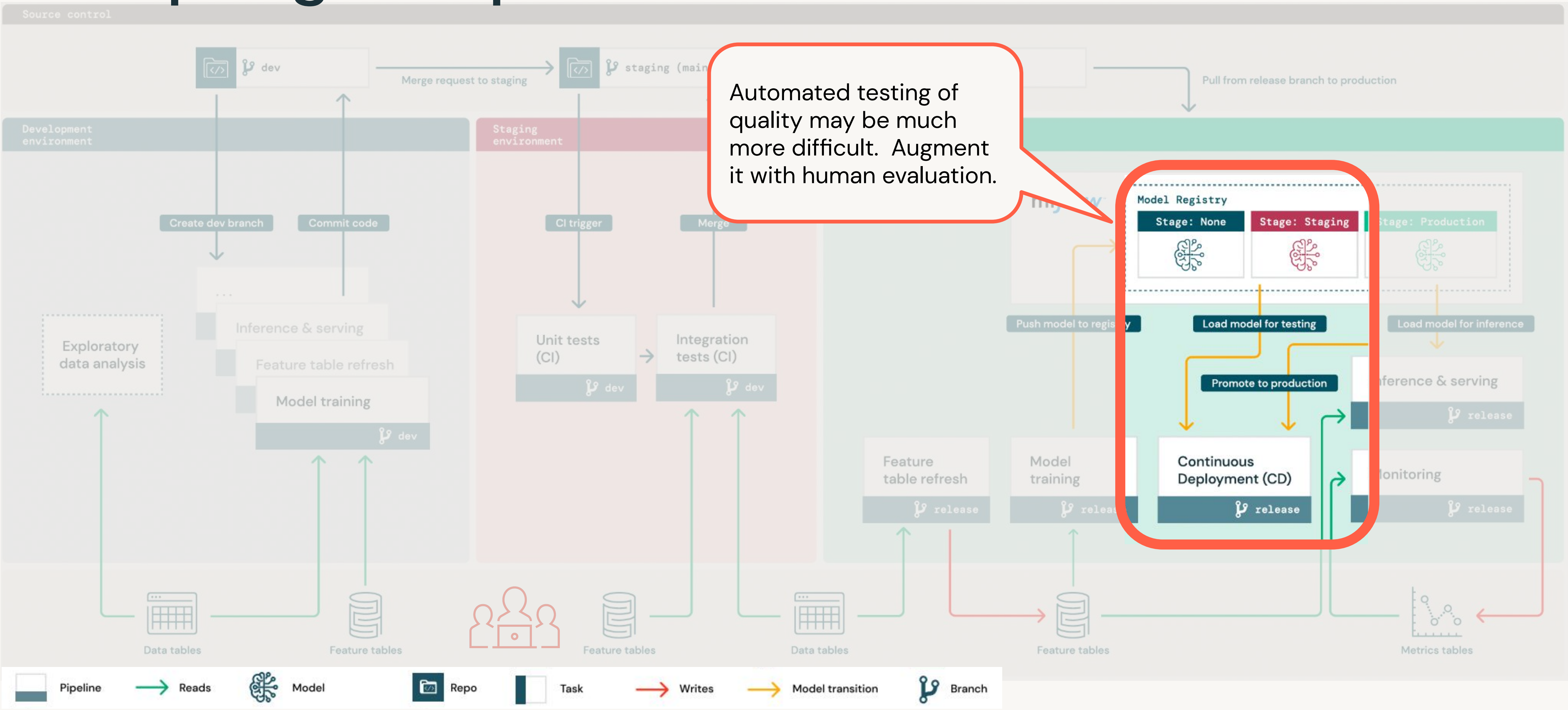
Adapting MLOps for LLMs



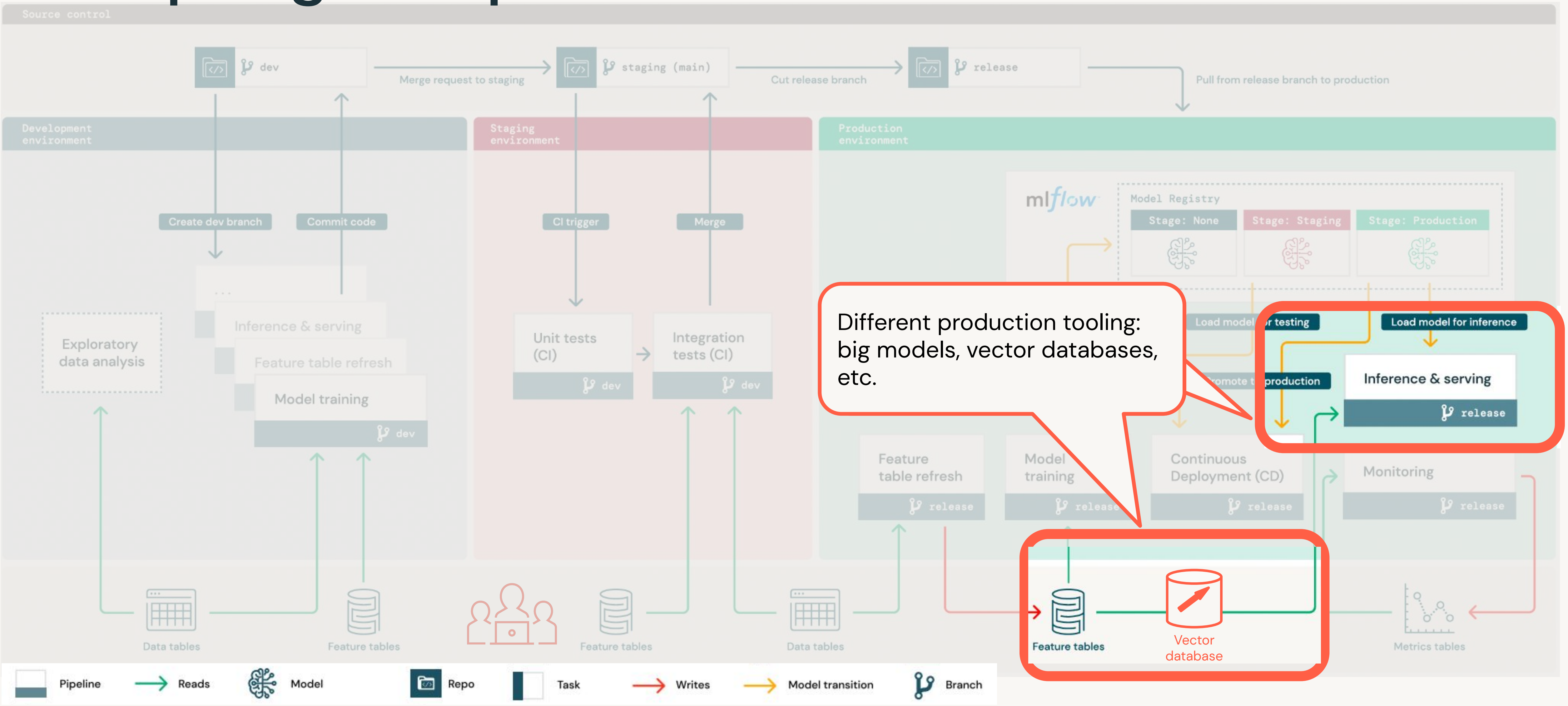
Adapting MLOps for LLMs



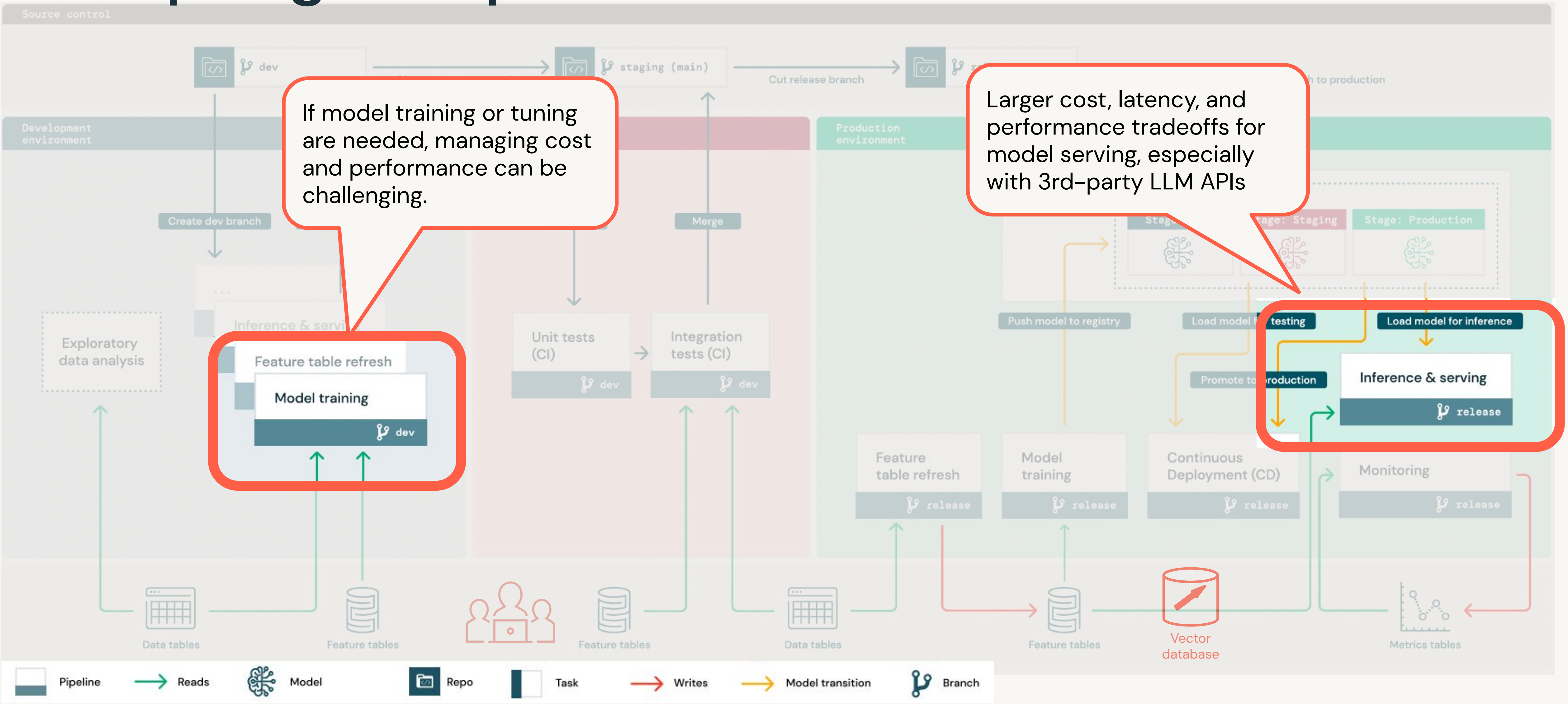
Adapting MLOps for LLMs



Adapting MLOps for LLMs

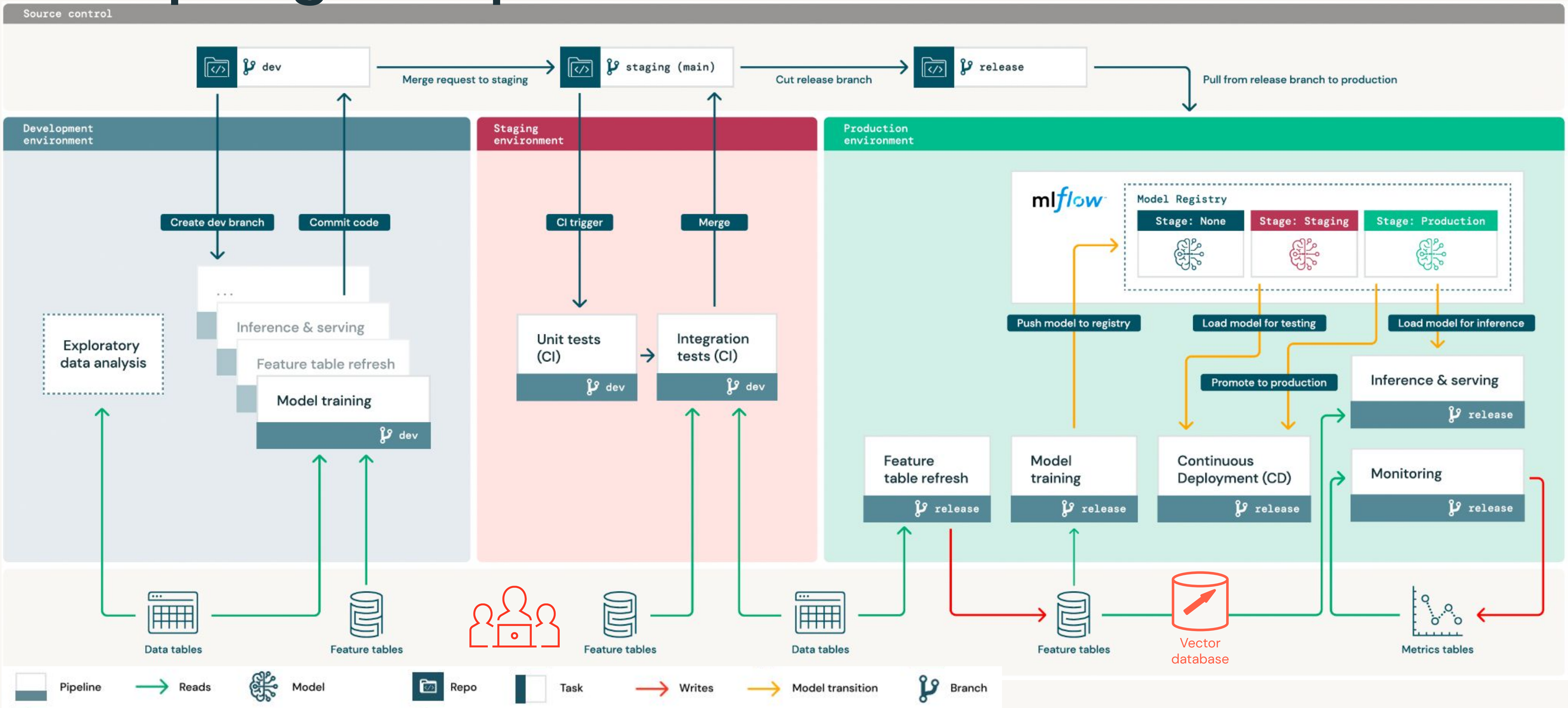


Adapting MLOps for LLMs



Adapting MLOps for LLMs

Some things change—but even more remain similar.



LLMOps details:

“Plan for key concerns which you may encounter with operating LLMs”

Key concerns

- Prompt engineering
- Packaging models or pipelines for deployment
- Scaling out
- Managing cost/performance tradeoffs
- Human feedback, testing, and monitoring
- Deploying models vs. deploying code
- Service infrastructure: vector databases and complex models

Prompt engineering

1. Track

Track queries and responses, compare, and iterate on prompts.

Example tools:
MLflow

2. Template

Standardize prompt formats using tools for building templates.

Example tools:
LangChain,
LlamaIndex

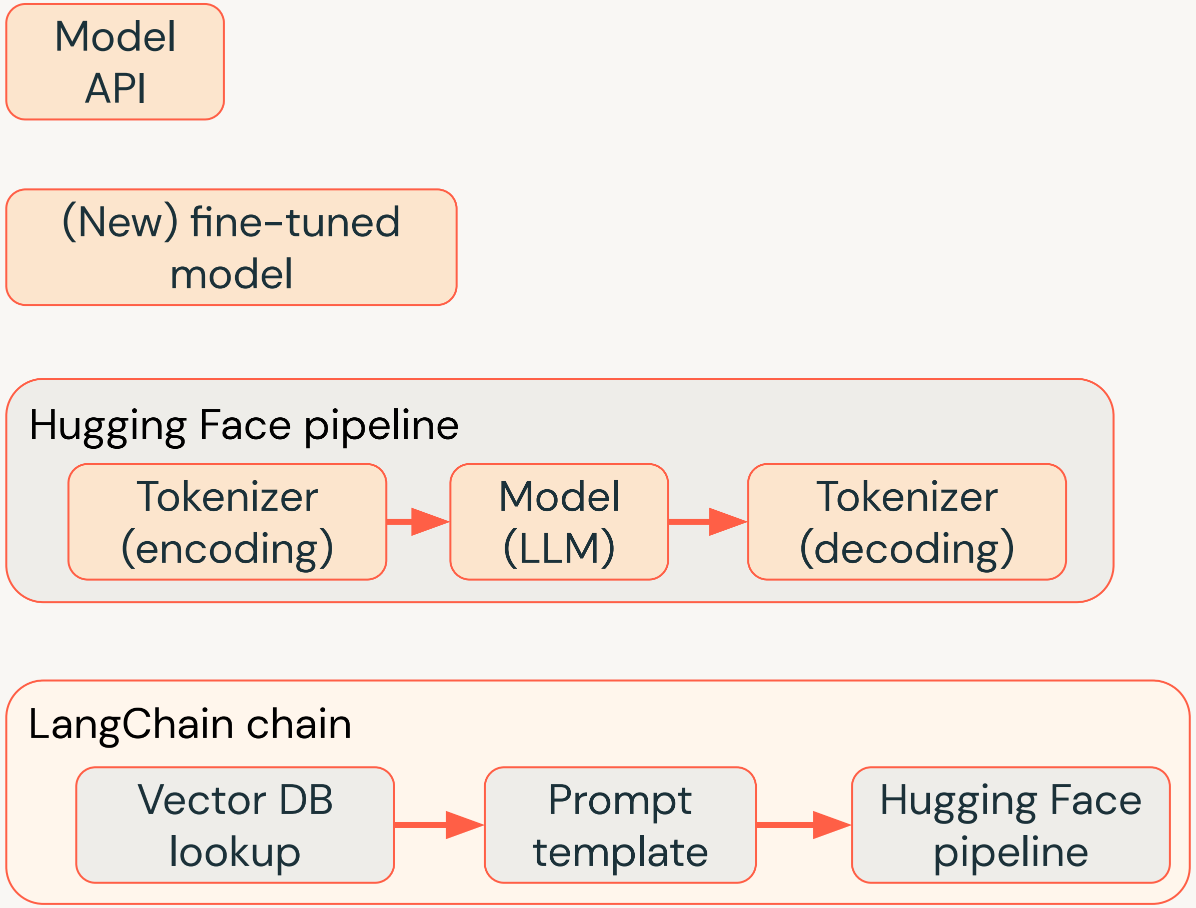
3. Automate

Replace manual prompt engineering with automated tuning.

Example tools:
DSP (Demonstrate-Search-Predict Framework)

Packaging models or pipelines for deployment

Standardizing deployment for many types of models and pipelines



Packaging models or pipelines for deployment

Standardizing deployment for many types of models and pipelines

Model API

```
mlflow.openai.log_model(model="gpt-3.5-turbo",
                        task=openai.ChatCompletion, …)
```

(New) fine-tuned model

```
mlflow.pytorch.log_model(
    pytorch_model=my_finetuned_model, …)
```

Hugging Face pipeline

Tokenizer (encoding) → Model (LLM) → Tokenizer (decoding)

```
mlflow.transformers.log_model(
    transformers_model=dolly
    artifact_path="dolly3b", …)
```

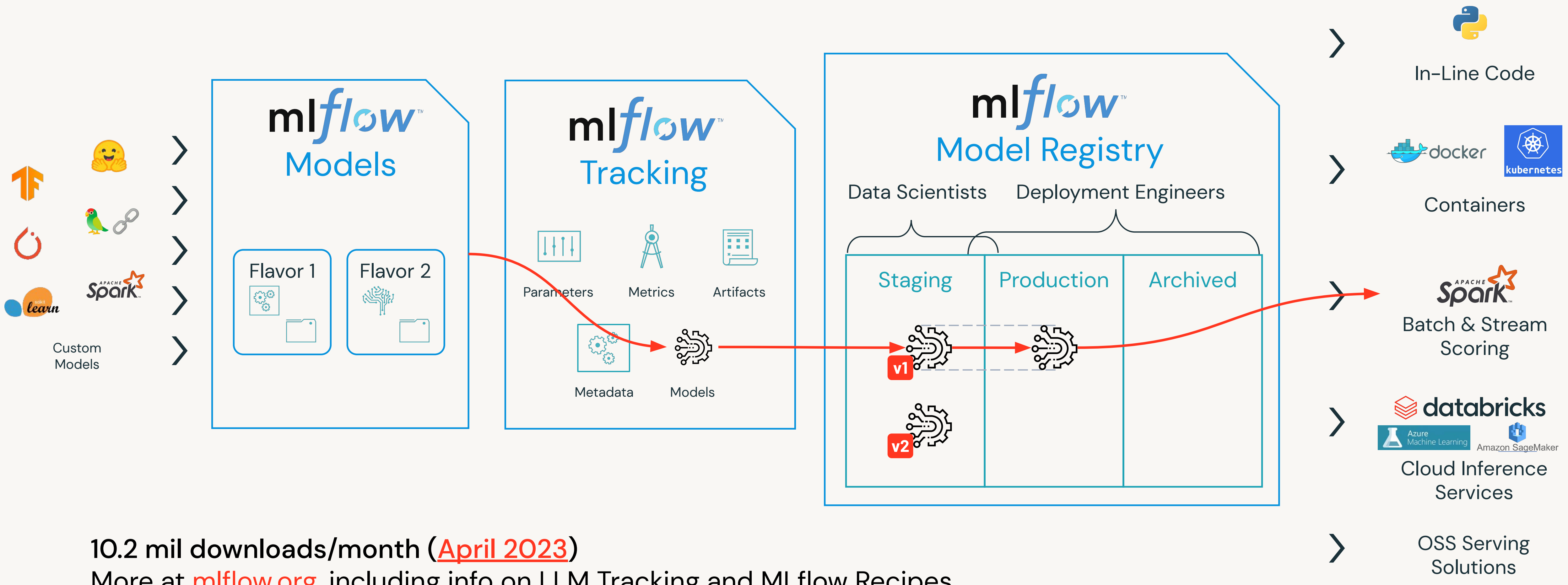
LangChain chain

Vector DB lookup → Prompt template → Hugging Face pipeline

```
mlflow.langchain.log_model(lc_model=llm_chain, …)
```

mlflow™

An open source platform for the machine learning lifecycle



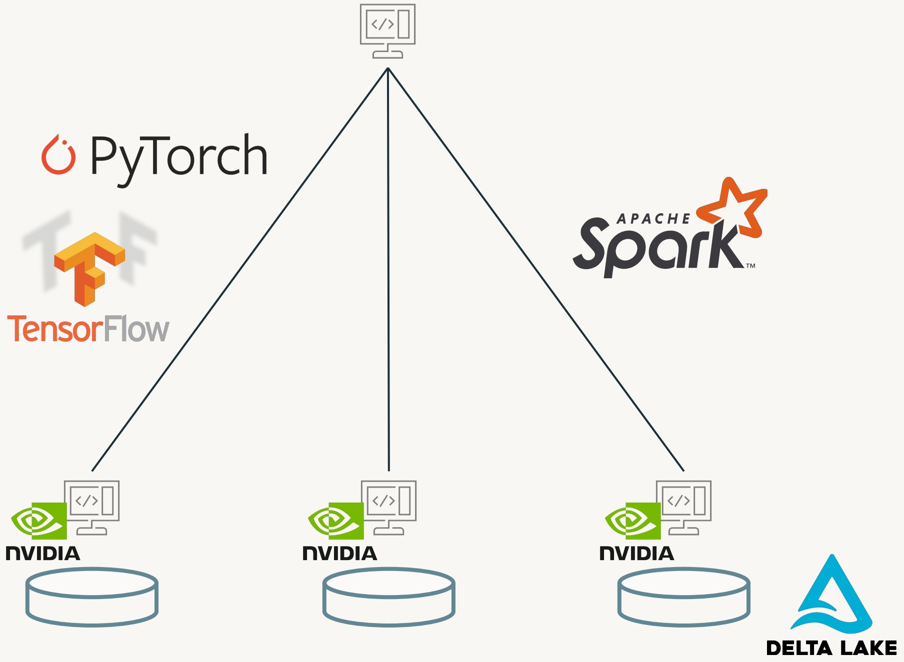
10.2 mil downloads/month (April 2023)
More at mlflow.org, including info on LLM Tracking and MLflow Recipes.

Scaling out

Distribute computation for larger data and models

Fine-tuning and training

- Distributed Tensorflow
- Distributed PyTorch
- DeepSpeed
- Optionally run on Apache Spark, Ray, etc.



Serving and inference

- Real-time: scale out end points
- Streaming and batch: Scale out pipelines, e.g. Spark + Delta Lake

Managing cost/performance tradeoffs

Metrics to optimize

- Cost of queries and training
- Time for development
- ROI of the LLM-powered product
- Accuracy/metrics of model
- Query latency

Tips for optimizing

- Go simple to complex: Existing models → Prompt engineering → Fine-tuning
- Scope out costs.
- Reduce costs by tweaking models, queries, and configurations.
- Get human feedback.
- Don't over-optimize!

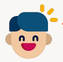
Human feedback, testing, and monitoring

Human feedback is critical, so plan for it!

- Build human feedback into your application from the beginning.
- Operationally, human feedback should be treated like any other data: feed it into your Lakehouse to make it available for analysis and tuning.

Select the best image to download it.

Sources of implicit user feedback.



Q: Hey tech support bot, how can I upload a file to the app?

A: Go to the user home screen, and click the image of a document in the sidebar.
Sources:
- Docs: File management
- Docs: User home screen

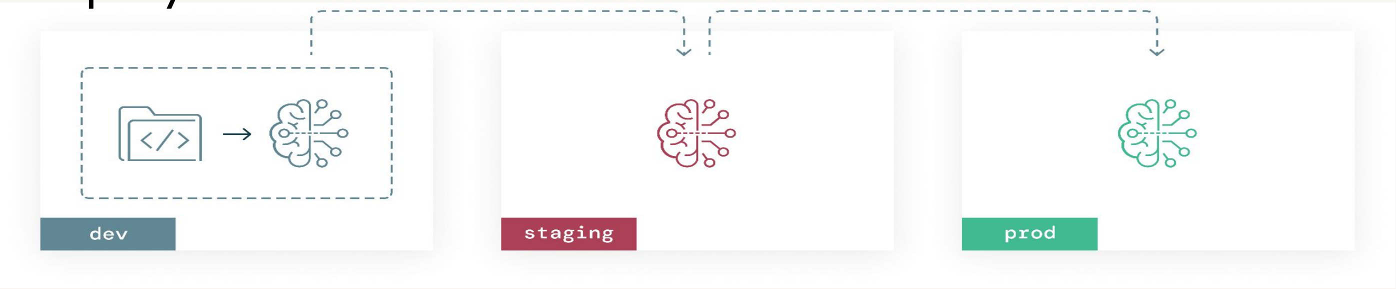
Click here to chat with a human.

Deploying models vs. deploying code

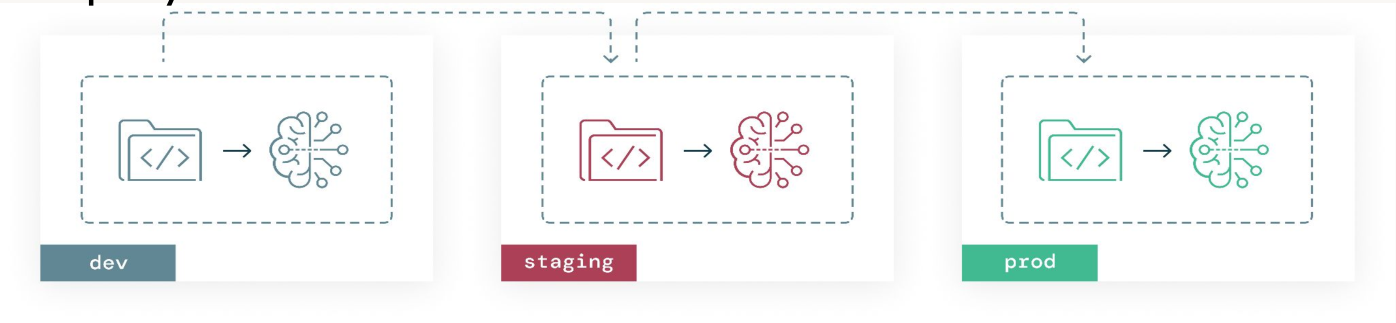
What asset(s) move from dev to prod?

Prompt engineering and pipeline tuning	Deploy pipelines as "models"
Fine-tuning or training models	Deploy code or models; depends on problem size. Train novel model ⇒ $1M+ Fine-tune model ⇒ $100
Both	Consider service architecture

Deploy models



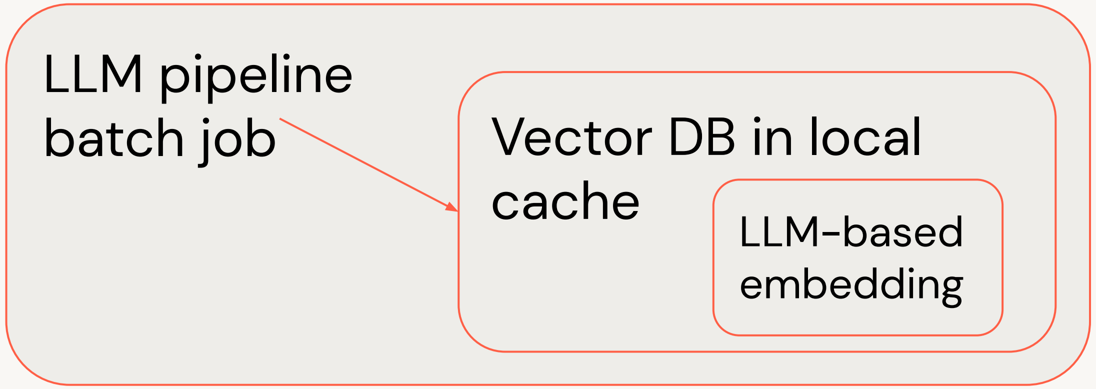
Deploy code

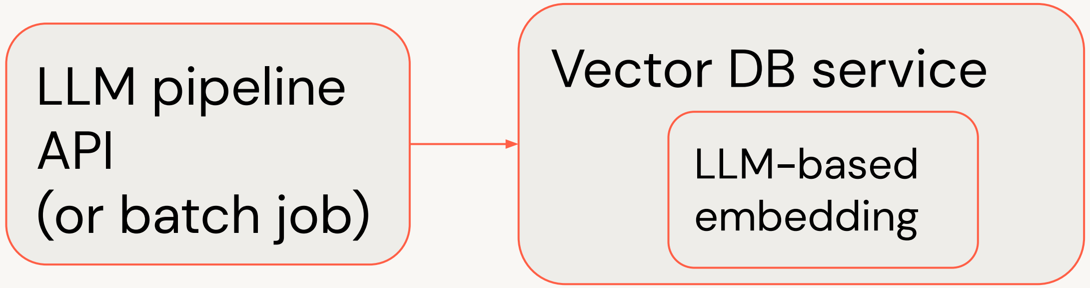


Source: The Big Book of MLOps

Service architecture

Vector databases





Complex models behind APIs

- Models have complex behavior and can be stochastic.
- How can you make these APIs stable and compatible?

What behavior would you expect?

- Same query, same model version
- Same query, updated model

Module Summary

LLMOps - What have we learned?

- LLMOps *processes and automation* help to ensure stable performance and long-term efficiency.
- LLMs put new requirements on MLOps platforms — but many parts of Ops remain the same as with traditional ML.
- Tackle challenges in each step of the LLMOps process as needed.

Time for some code!