UC San Diego

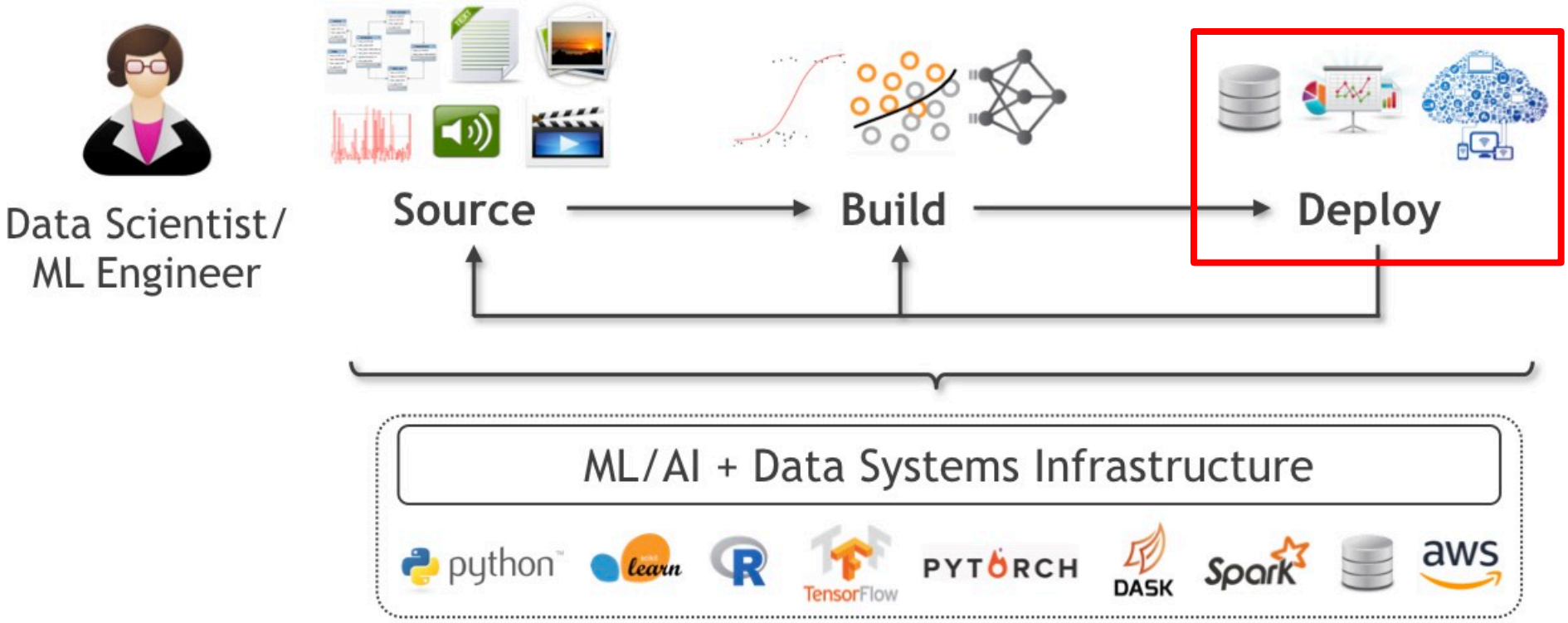
CSE 234
Data Systems for Machine Learning

Arun Kumar

Topic 3: ML Deployment, MLOps, and LLMOps

Chapter 8.5 of MLSys book

ML Deployment in the Lifecycle



Data acquisition
Data preparation

Feature Engineering
Training & Inference
Model Selection

Serving
Monitoring

ML Deployment in the Lifecycle

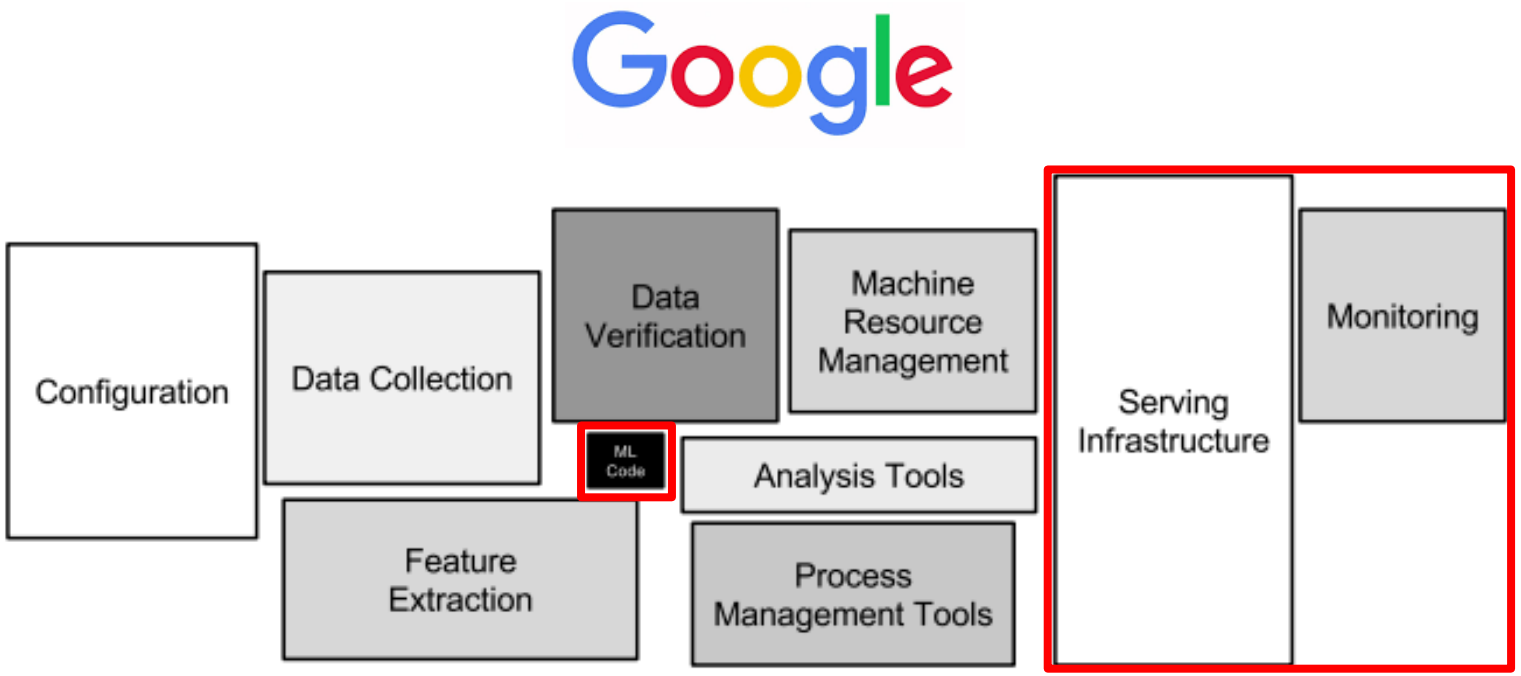


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Outline

- Offline ML Deployment
- MLOps:
  - Online Prediction Serving
  - Monitoring and Versioning
- Federated ML
- LLMOps

Offline ML Deployment

- **Given:** A trained prediction function f(); a *set* of (unlabeled) data examples
- **Goal:** Apply inference with f() to all examples *efficiently*
  - Key metrics: *Throughput*, cost, latency
- Historically, offline was the most common scenario
  - Still is at most enterprises, healthcare, academia
  - Typically once a day / week / month / quarter!
  - Aka model *scoring* in some settings

Offline ML Deployment: Systems

- Not particularly challenging in most applications
- All ML systems support offline batch inference by default

General ML Libraries:

Disk-based files:

Layered on RDBMS/Spark:

Cloud-native:



"AutoML" platforms:

GBDT Systems:

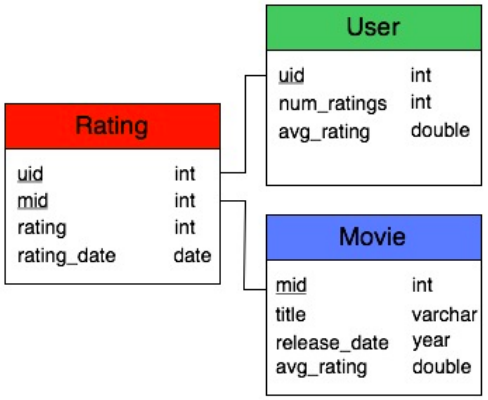
DL Systems:

LLM Systems:

Offline ML Deployment: Optimizations

Q: What systems-level optimizations are possible here?

- **Data Parallelism:**
  - Inference is *embarrassingly parallel* across examples
- **Factorized ML (e.g., in Morpheus):**
  - Push ML computations down through joins
  - Pre-computes some FLOPS and reuses across examples



$$x_i = [x_{i,R}; x_{i,U}; x_{i,M}]$$

Example: GLM inference:

$$w^T x_i = w_R^T x_{i,R} + w_U^T x_{i,U} + w_M^T x_{i,M}$$

Offline ML Deployment: Optimizations

***Q:** What systems-level optimizations are possible here?*

- **More general pre-computation / caching / batching:**
  - Factorized ML is a specific form of sharing/caching
  - Other forms of "multi-query optimization" possible

Example: Batched inference for separate GLMs:

$$X_{n\times d}(w_1)_{d\times 1} \quad Xw_2 \quad Xw_3 \longrightarrow X[w_1; w_2; w_3]_{d\times 3}$$

Reduces memory stalls for X; raises hardware efficiency

Peer Instruction Activity

(Switch slides)

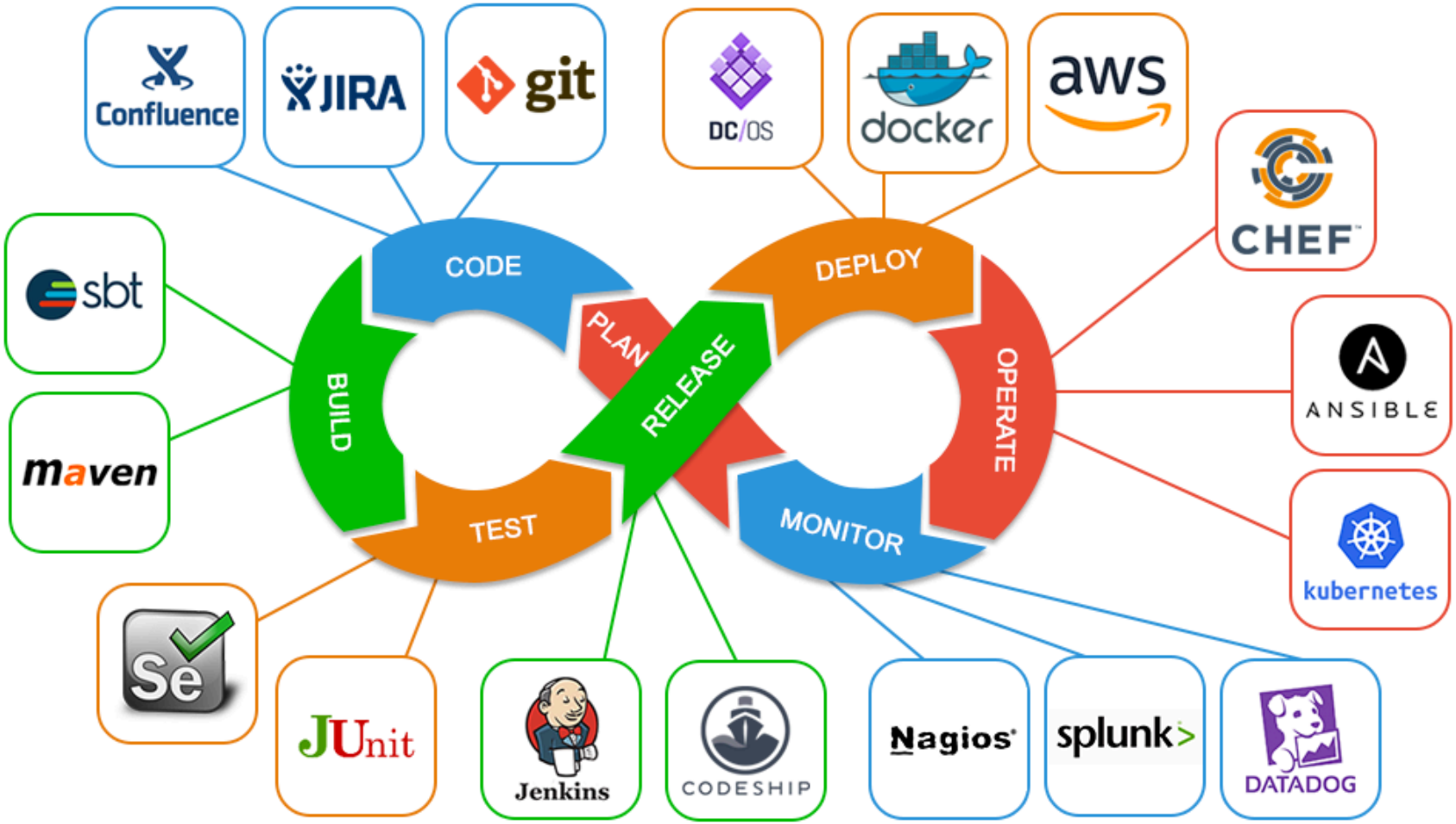
Outline

- Offline ML Deployment
- MLOps:
  - Online Prediction Serving
  - Monitoring and Versioning
- Federated ML
- LLMOps

Background: DevOps

- Software Development + IT Operations (DevOps) is a long standing subarea of *software engineering*
- No uniform definition but loosely, the science + engineering of administering software in production
  - Fuses many historically separate job roles
- Cloud and “Agile” s/w eng. have revolutionized DevOps

Background: DevOps



https://medium.com/swlh/how-to-become-an-devops-engineer-in-2020-80b8740d5a52

Key Parts of DevOps Stack/Practice

Logging & Monitoring

Building & Testing

Continuous Integration (CI) & Continuous Delivery (CD)

Version Control

Infrastructure-as-Code (IaC), including Config. & Policy

Microservices / Containerization & Orchestration

Content Credit: Manasi Vartak, Verta.AI
https://aws.amazon.com/devops/what-is-devops/

The Rise of MLOps

- MLOps = DevOps for ML-infused software
  - Much harder than for deterministic software!
- Things that matter beyond just ML model codes:
  - Training and validation datasets
  - Data cleaning/prep/featurization codes/scripts
  - Hyperparameters, other training configs
  - Post-inference rules/configs/ensembling
  - Software versions/configs?
  - Training hardware/configs?

Content Credit: Manasi Vartak, Verta.AI

The Rise of MLOps

- Need to change DevOps for ML program semantics
- **Online Prediction Serving**
- **Logging & Monitoring:**
  - Prediction failures; concept drift; feature inflow changes
- **Version Control:**
  - Anything can change: ML code, data, configs, etc.
- **Build & Test; CI & CD:**
  - Rigorous train-val-test splits; beware insidious overfitting
- New space with a lot of R&D; no consensus on standards

Content Credit: Manasi Vartak, Verta.AI

The “3 Vs of MLOps”

Operationalizing Machine Learning: An Interview Study

Shreya Shankar*, Rolando Garcia*, Joseph M. Hellerstein, Aditya G. Parameswaran
University of California, Berkeley

- **Velocity:**
  - Need for rapid experimentation, prototyping, and deployment with minimal friction
- **Validation:**
  - Need for checks on quality and integrity of data, features, models, predictions
- **Versioning:**
  - Need to keep track of deployed models and features to ensure provenance and fallback options

https://arxiv.org/pdf/2209.09125.pdf

The “3 Vs of MLOps”

- Interplay/tussles between the 3 Vs shapes decisions on tools, processes, and people management in MLOps
- **Examples:**
  - Should Jupyter notebooks be deployed to production? Velocity vs. Validation
  - Are feature stores needed? Velocity vs. Versioning
  - Relabel/augment val. data? Validation vs. Versioning

https://arxiv.org/pdf/2209.09125.pdf

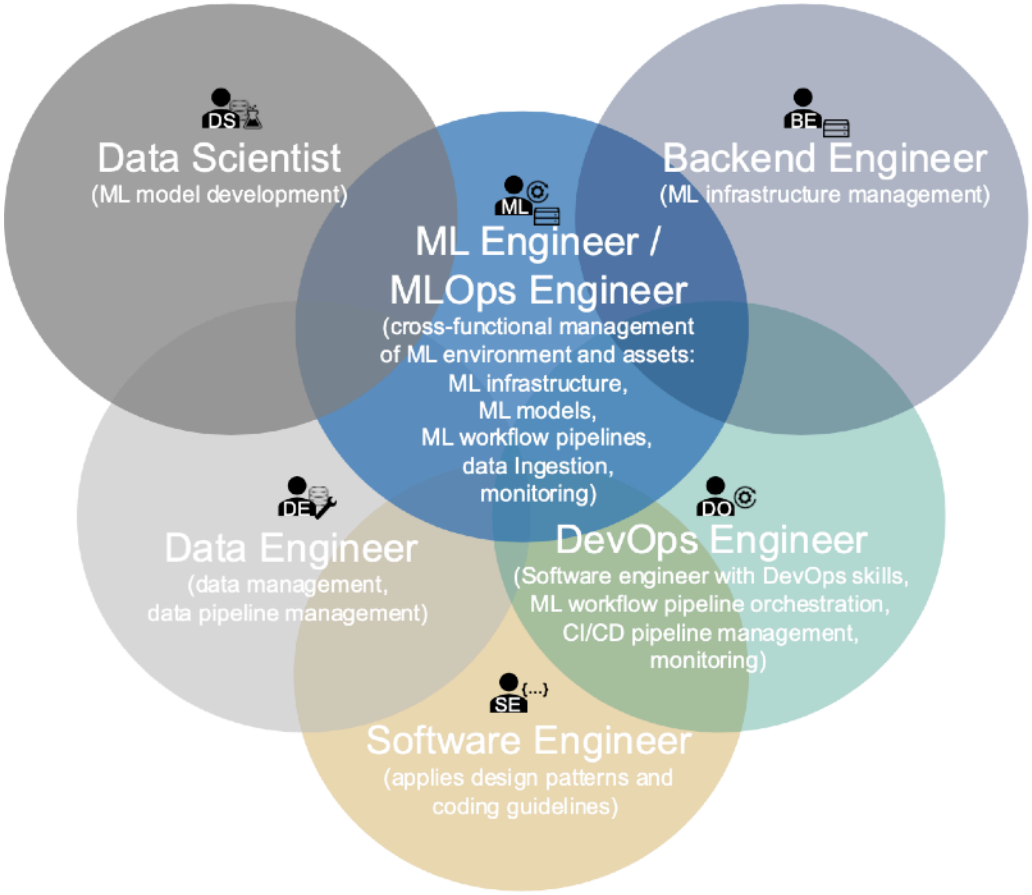
Birds-eye View of MLOps

Machine Learning Operations (MLOps): Overview, Definition, and Architecture

Dominik Kreuzberger
KIT
Germany

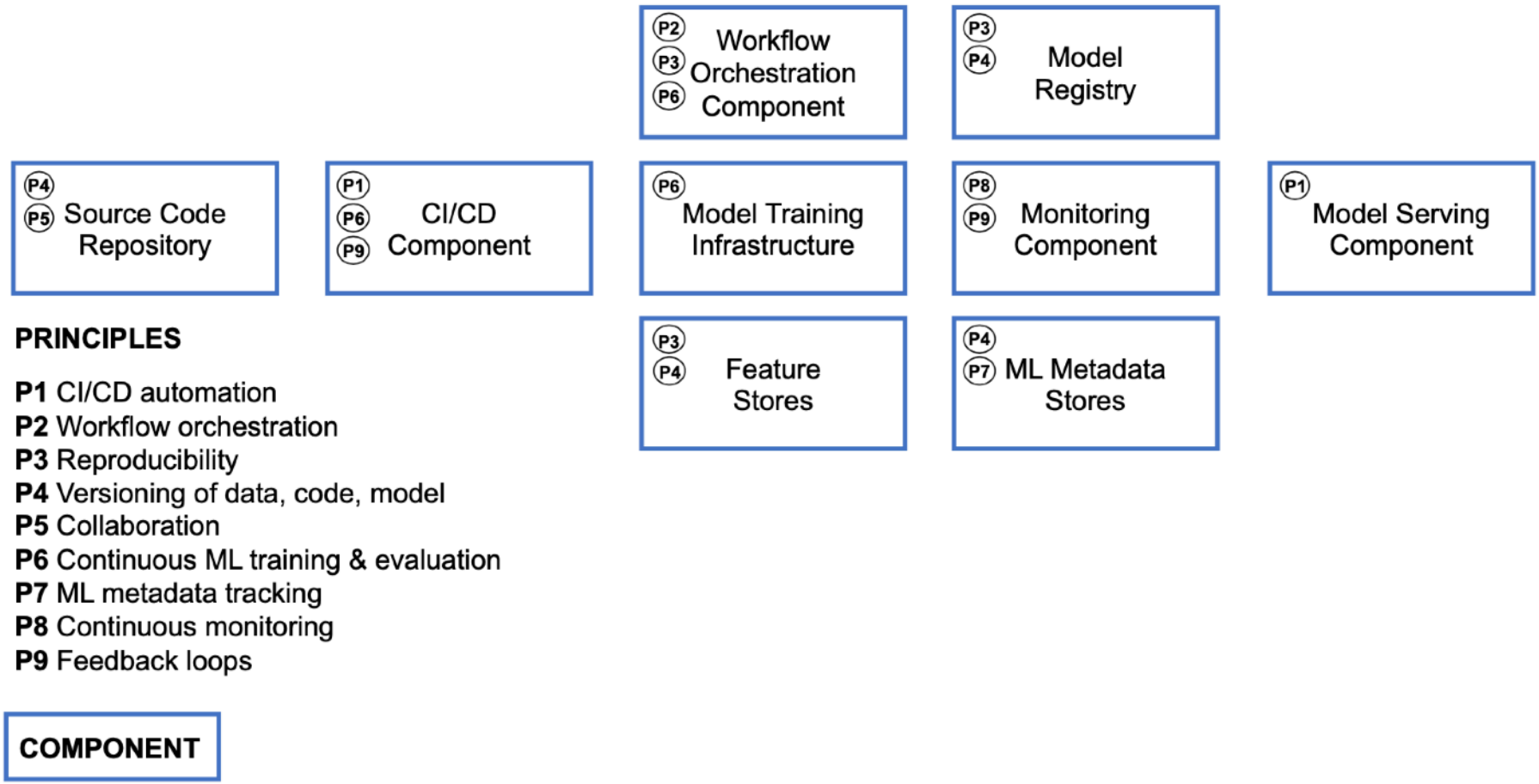
Niklas Kühl
KIT
Germany

Sebastian Hirschl
IBM†
Germany



https://arxiv.org/pdf/2205.02302.pdf

Birds-eye View of MLOps



https://arxiv.org/pdf/2205.02302.pdf

Outline

- Offline ML Deployment
- MLOps:
  - Online Prediction Serving
    - Monitoring and Versioning
- Federated ML
- LLMOps

Online Prediction Serving

- Standard setting for Web and IoT deployments of ML
  - Typically need to be *realtime*; < 100s of milliseconds!
  - AKA *model serving* or *ML serving*
- **Given:** A trained prediction function f() + a *stream* of unlabeled data example(s)
- **Goal:** Apply f() to all/each example *efficiently*
  - Key metrics: Latency, memory footprint, cost, throughput

Online Prediction Serving

- Surprisingly challenging to do well in ML systems practice!
  - Active area of R&D; many startups
- **Key Challenges:**
  - **Heterogeneity** of environments: webpages, cloud-based apps, mobile apps, vehicles, IoT, etc.
  - **Unpredictability** of load: need to elastically upscale or downscale resources
  - **Function's complexity**: model, featurization and data prep code, output thresholds, etc.
    - May straddle libraries and even PLs!
    - Hard to optimize end to end in general

The Rise of Serverless Infra.

- Prediction serving is a “killer app” for Function-as-a-Service (FaaS), AKA serverless cloud infra.
  - Extreme pay-as-you-go; can rent at millisecond level!

- Still, many open efficiency issues for ML deployment:
  - Reduce memory footprints, input access restrictions, logging / output persistence restrictions, latency

Online Prediction Serving: Systems

- Numerous serving systems have sprung up

General-purpose (supports multiple ML tools):

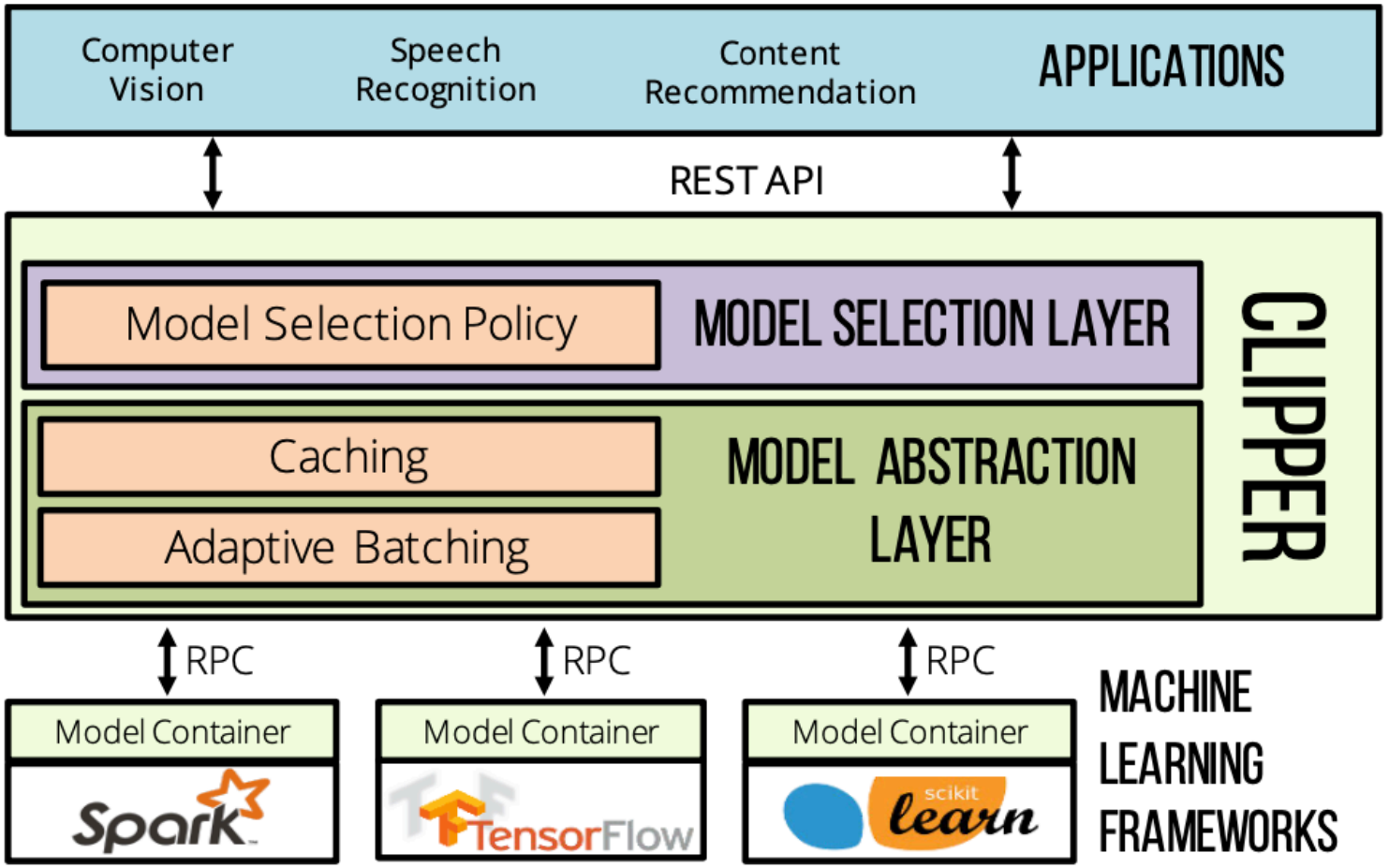
ML System-specific:

TF Serving

TorchServe

Clipper

- A pioneering general-purpose ML serving system

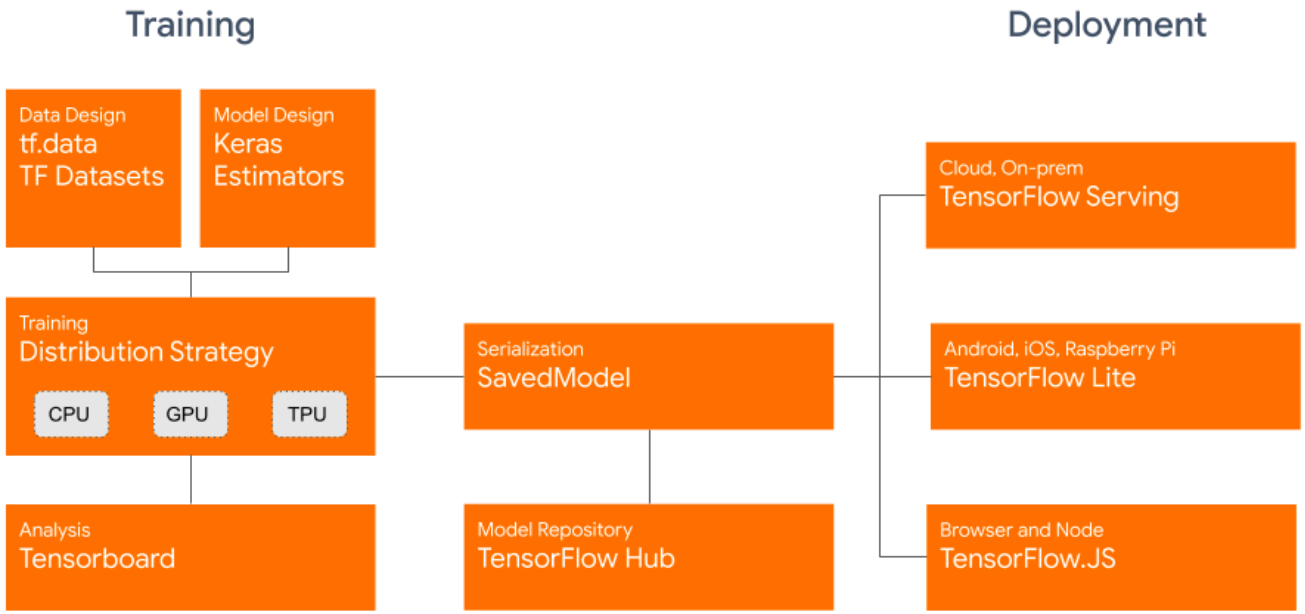


Clipper: Principles and Techniques

- **Generality and modularity:**
  - One of the first to use *containers* for prediction serving
  - Supports multiple ML tools in unified layered API
- **Efficiency:**
  - Some basic optimizations: *batching* to raise throughput; *caching* of frequently access models/vectors
- **Multi-model deployment and flexibility:**
  - A heuristic “model selection” layer to dynamically pick among multiple deployed models; ensembling

TensorFlow Serving

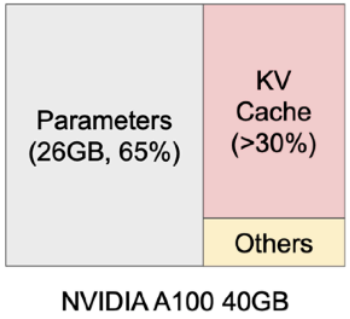
- TF Serving is a mature ML serving system, also pioneering
  - Optimized for TF model formats; also supports batching
  - Dynamic reloading of weights; multiple data sources

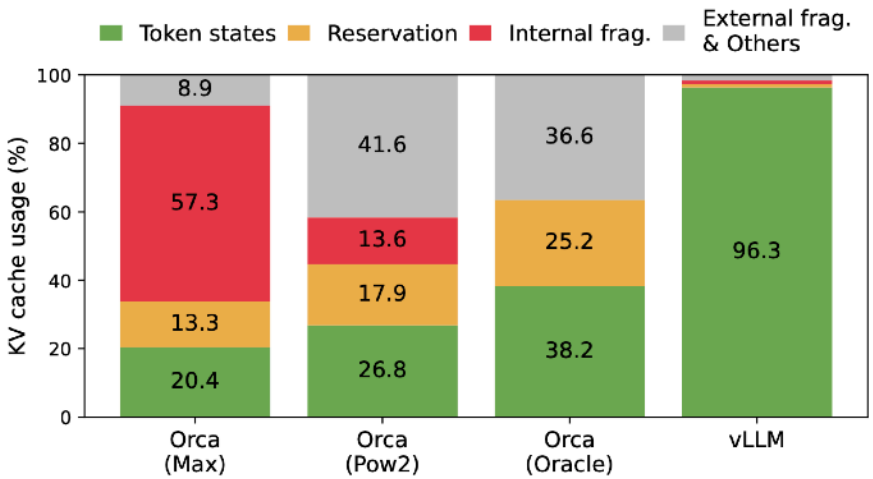


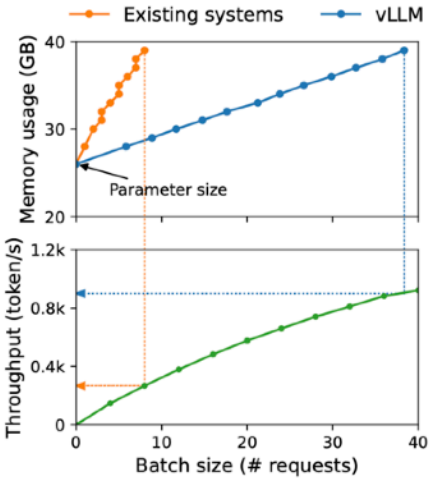
- TF Lite and TF.JS optimized for more niche backends/ runtime environments

VLLM: Overview

- **Goal:** Improve throughput of serving LLMs on GPUs
- **Observation:** Memory fragmentation due to dynamic memory footprint of attention ops' KV tensors wastes GPU memory
- **Key Idea:**
  - Level of indirection akin to paging and virtual memory in OS
  - Group attention ops' KV tensors into a block per set of tokens
  - Blocks need not be laid out contiguously in GPU memory







VLLM: Techniques and Impact

- (Switch to Hao's slide deck)

Your Reviews on VLLM Paper

- (Walked through in class)

Comparing ML Serving Systems

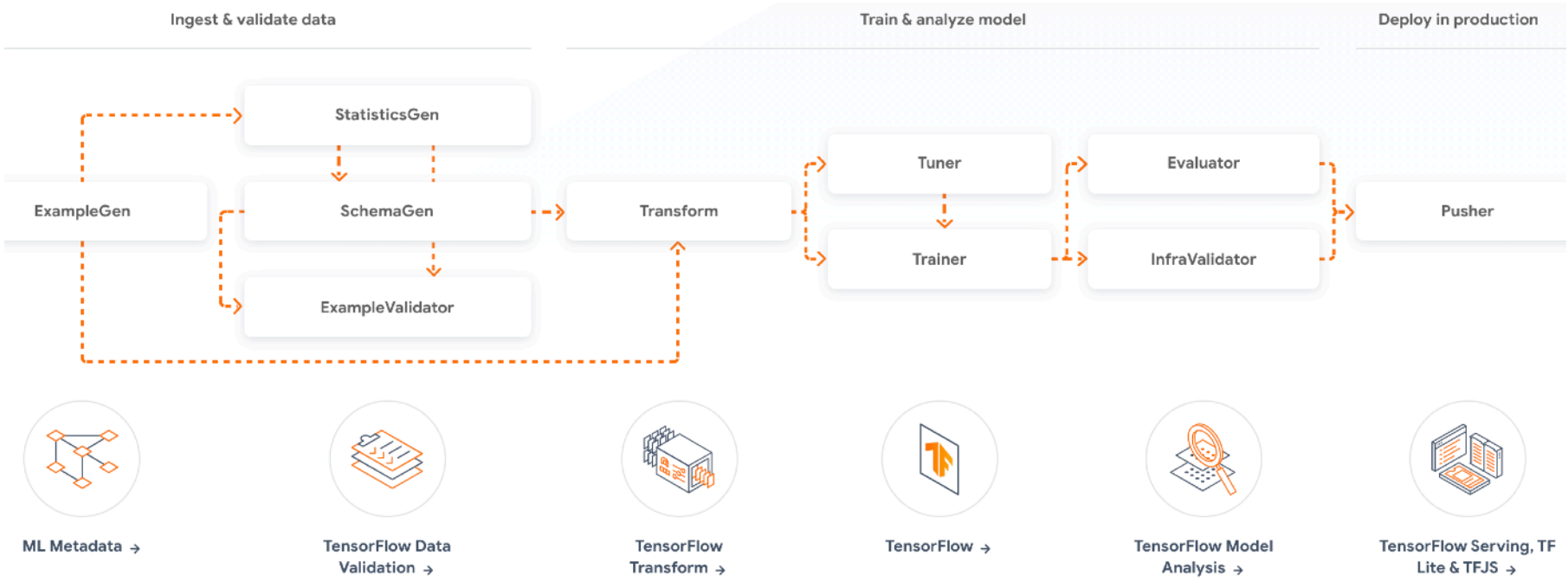
- Benefits of **general-purpose** vs. ML system-specific:
  - Tool heterogeneity is a reality for many orgs
  - More nimble to customize accuracy post-deployment with different kinds of models/tools
  - Flexibility to swap ML tools; no “tool lock-in”
- Benefits of **ML system-specific** vs. general-purpose:
  - Generality may not be needed inside org. (e.g., Google); lower complexity of MLOps
  - Likely more amenable to code/pipeline optimizations
  - Likely better hardware utilization, lower cloud costs

Outline

- Offline ML Deployment
- MLOps:
  - Online Prediction Serving
  - Monitoring and Versioning
- Federated ML
- LLMOps

Example for ML Monitoring: TFX

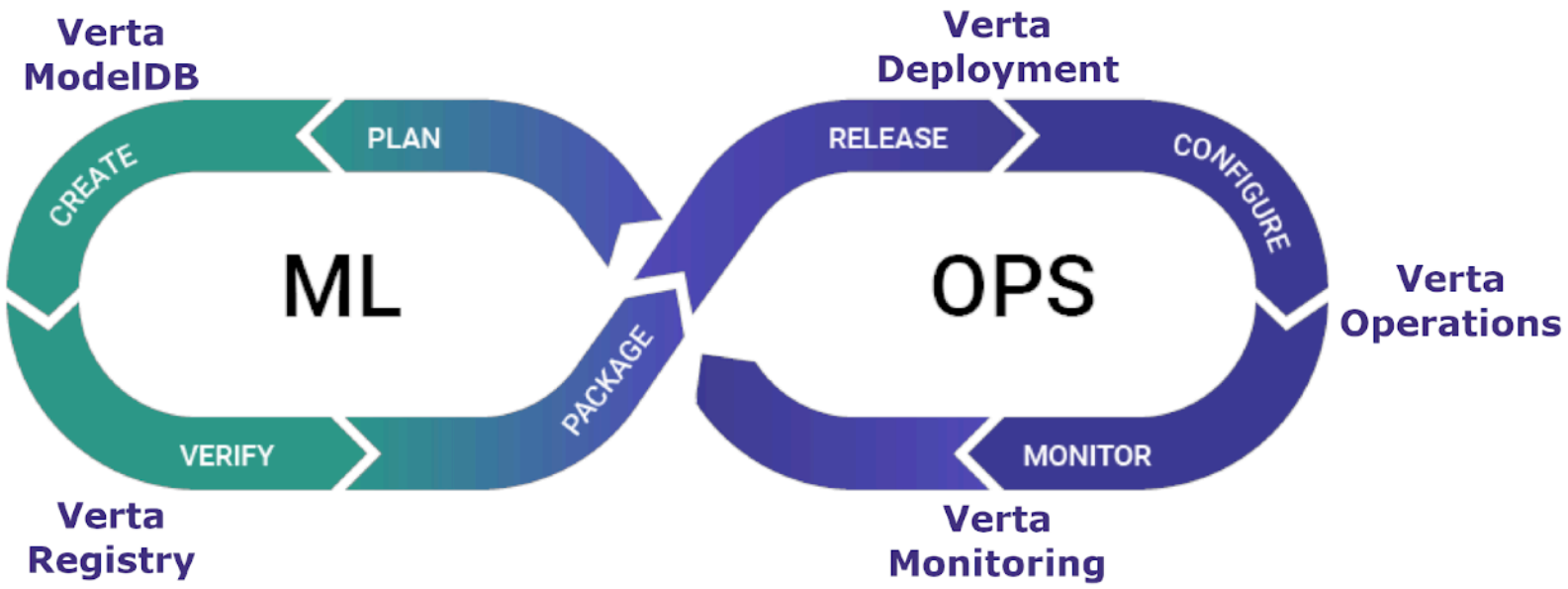
- TFX's "Model Analysis" lets user specify metrics, track over time automatically, alert on-call
- Can specify metrics for feature-based data "slices" too



https://www.tensorflow.org/tfx/guide/tfma

Example for ML Versioning: Verta

- Started with ModelDB for storing and tracking ML artifacts
  - ML code; data; configuration; environment
- APIs as hooks into ML dev code; SDK and web app./GUI
- Registry for versions and workflows



https://blog.verta.ai/blog/the-third-wave-of-operationalization-is-here-mlops

Open Research Questions in MLOps

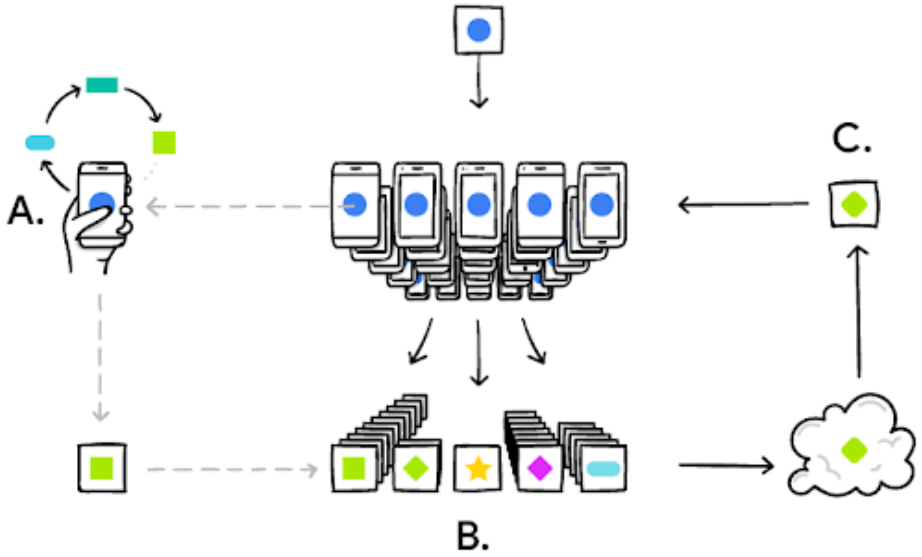
- Efficient and consistent version control for ML datasets and featurization codes
- Automate prediction failure detection and recovery
- Detect concept drift in an actionable manner; prescribe fixes
- Velocity and complexity of streaming ML applications
- CI & CD for model ensembles without insidious overfitting
- Automated end-to-end optimizations
- …

Outline

- Offline ML Deployment
- MLOps:
  - Online Prediction Serving
  - Monitoring and Versioning
- Federated ML
- LLMOps

Federated ML

- Pioneered by Google for ML/AI applications on smartphones
- Key benefit is more **user privacy**:
  - User's (labeled) data does not leave their device
  - Decentralizes ML model training/finetuning to user data



https://ai.googleblog.com/2017/04/federated-learning-collaborative.html
https://mlsys.org/Conferences/2019/doc/2019/193.pdf

Federated ML

- **Key challenge:** Decentralize SGD to intermittent updates
- They proposed a simple “federated averaging” algorithm

$$\min_{w \in \mathbb{R}^d} f(w) \qquad \text{where} \qquad f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w).$$

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w).$$

- User-partitioned updates breaks IID assumption; skews arise
- Turns out SGD is still pretty robust (recall async. PS); open theoretical questions still being studied

https://arxiv.org/abs/1602.05629

Federated ML

- **Privacy/security-focused improvements:**
  - New SGD variants; integration with differential privacy
  - Cryptography to anonymize update aggregations
- Apart from strong user privacy, *communication and energy efficiency* also major concerns on battery-powered devices
- **Systems+ML heuristic optimizations:**
  - Compression and quantization to save upload bandwidth
  - Communicate only high quality model updates
  - Novel federation-aware ML algorithmics

https://arxiv.org/abs/1602.05629
https://arxiv.org/pdf/1610.02527.pdf
https://eprint.iacr.org/2017/281.pdf

Federated ML

- Federated ML protocol has become quite sophisticated to ensure better stability/reliability, accuracy, and manageability

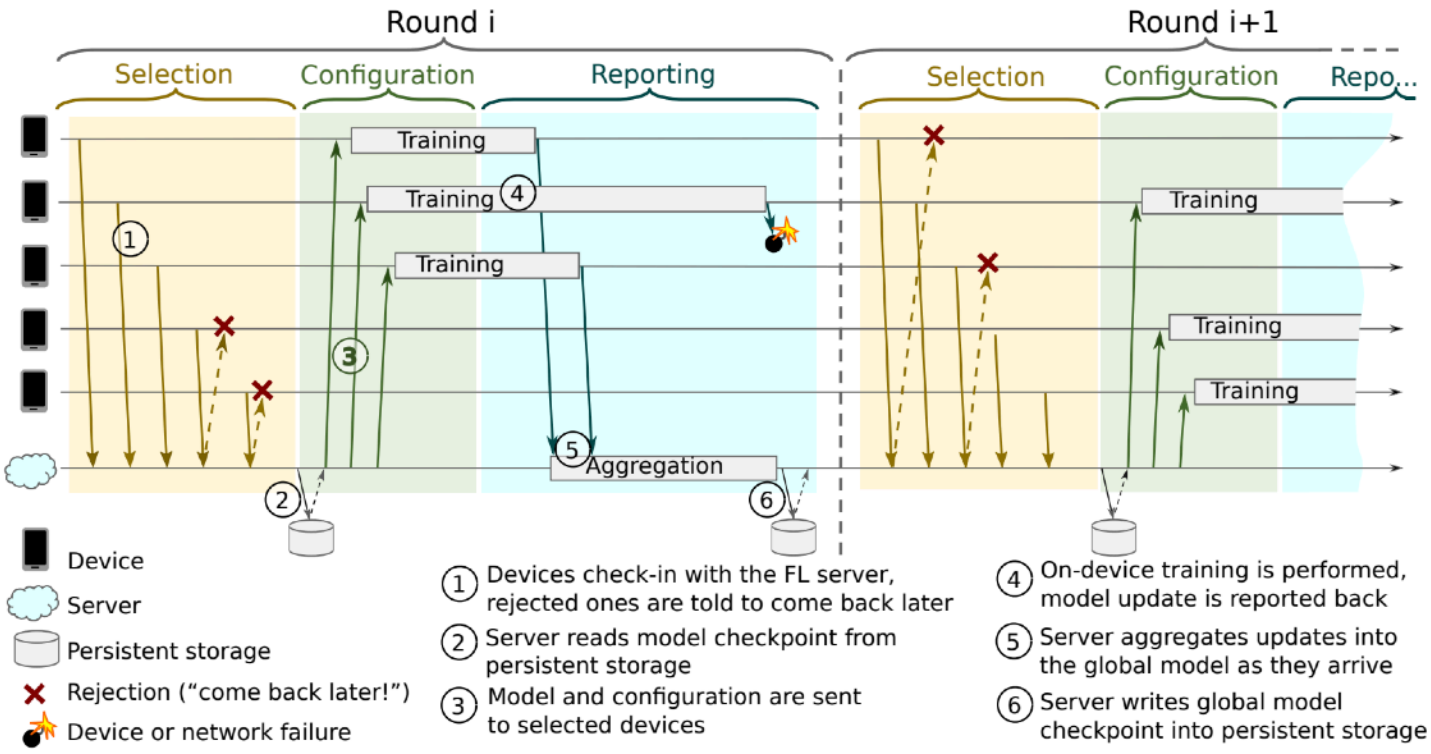
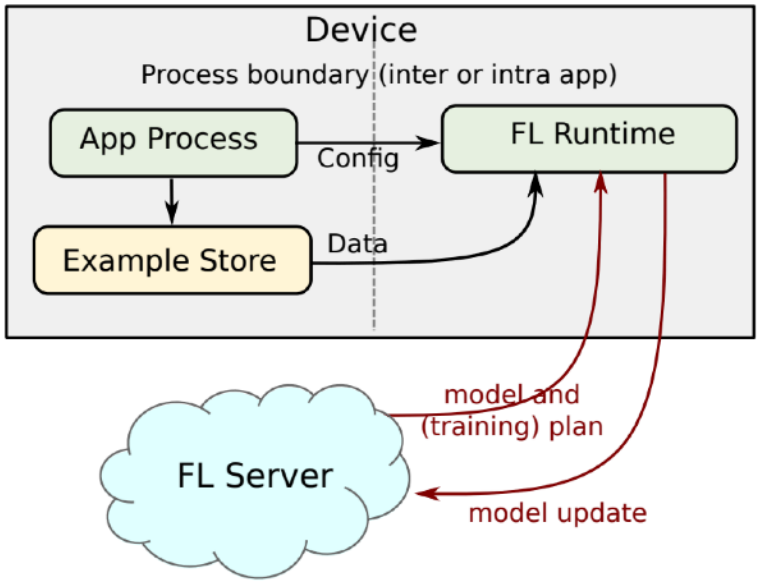


Figure 1: Federated Learning Protocol

https://mlsys.org/Conferences/2019/doc/2019/193.pdf

Federated ML

- Google has neatly abstracted the client-side (embedded in mobile app.) and server-side functionality with actor design



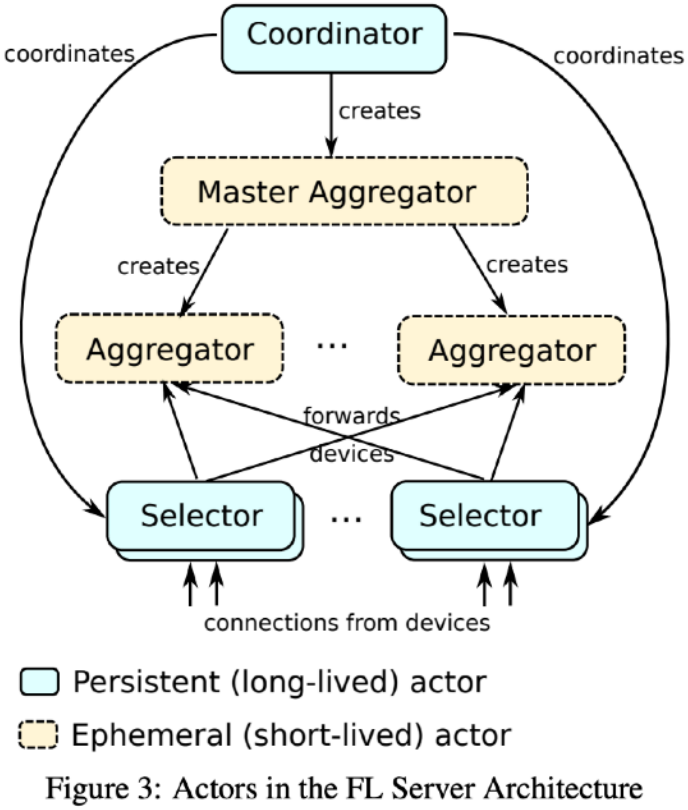
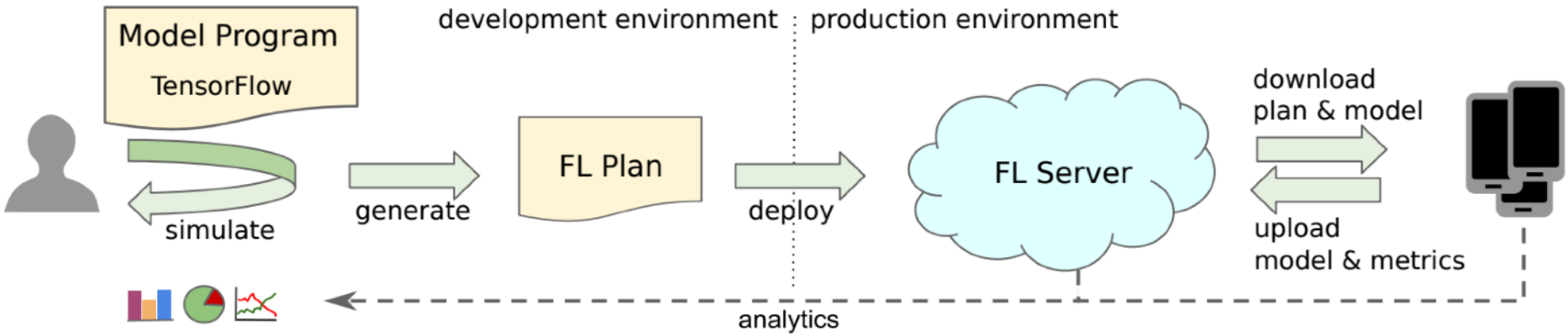


Figure 3: Actors in the FL Server Architecture

https://mlsys.org/Conferences/2019/doc/2019/193.pdf

Federated ML

- Notion of “FL Plan” and simulation-based tooling for data scientists to tailor ML for this deployment regime
  - (Users’) Training data is out of reach!
  - Model is updated asynchronously automatically
  - Debugging and versioning become even more difficult



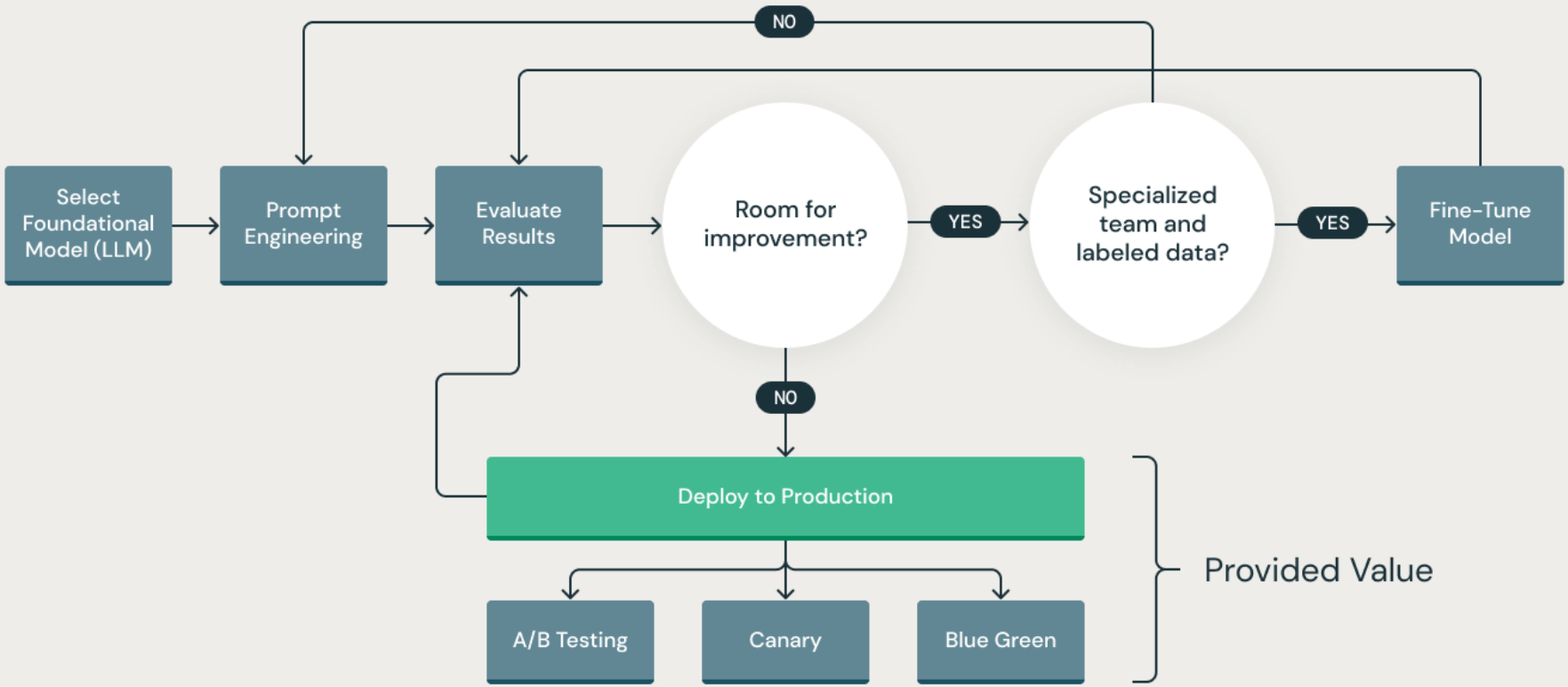
https://mlsys.org/Conferences/2019/doc/2019/193.pdf

Outline

- Offline ML Deployment
- MLOps:
  - Online Prediction Serving
  - Monitoring and Versioning
- Federated ML
- LLMOps

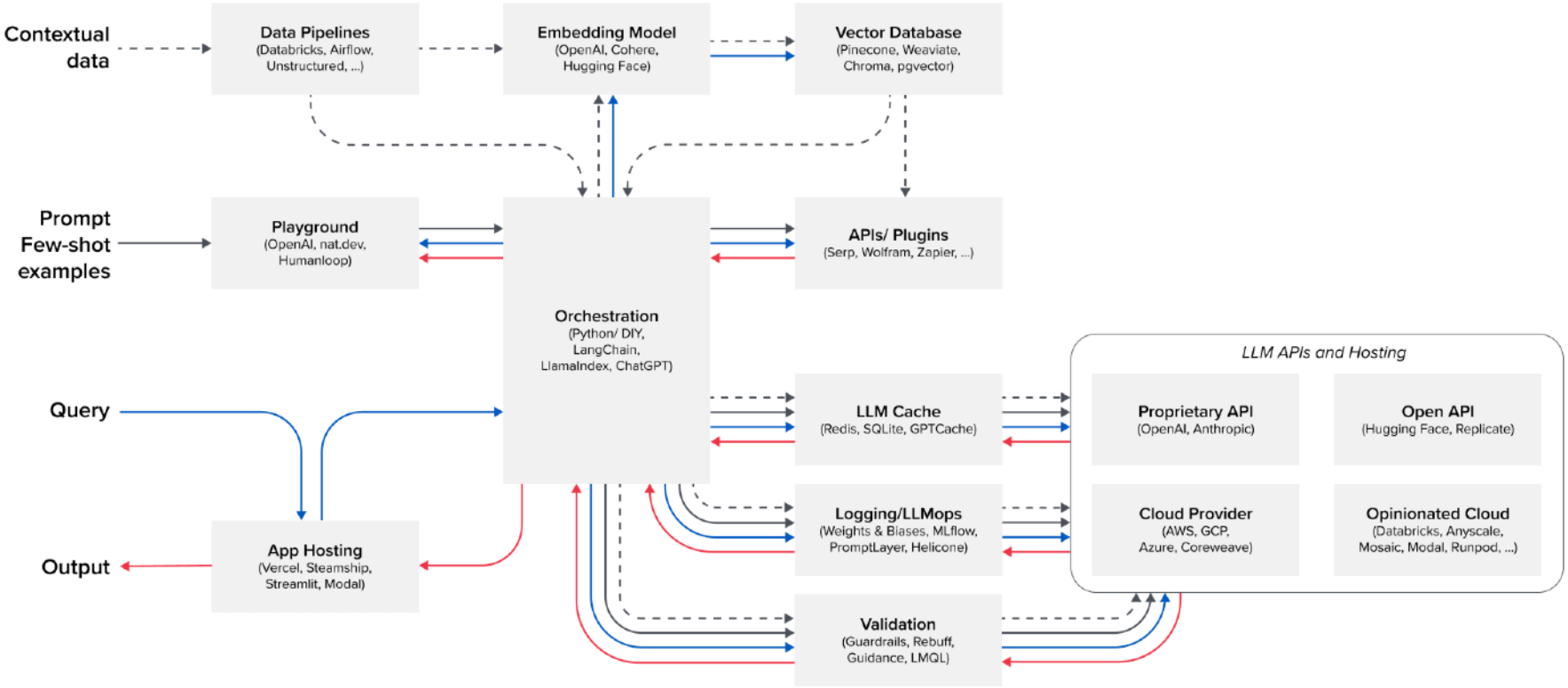
LLMOps: Birds-Eye View

Example of development-to-production workflow for LLMs



https://www.databricks.com/glossary/llmops

LLMOps: Emerging Stack



a16z Enterprise

https://a16z.com/emerging-architectures-for-llm-applications/

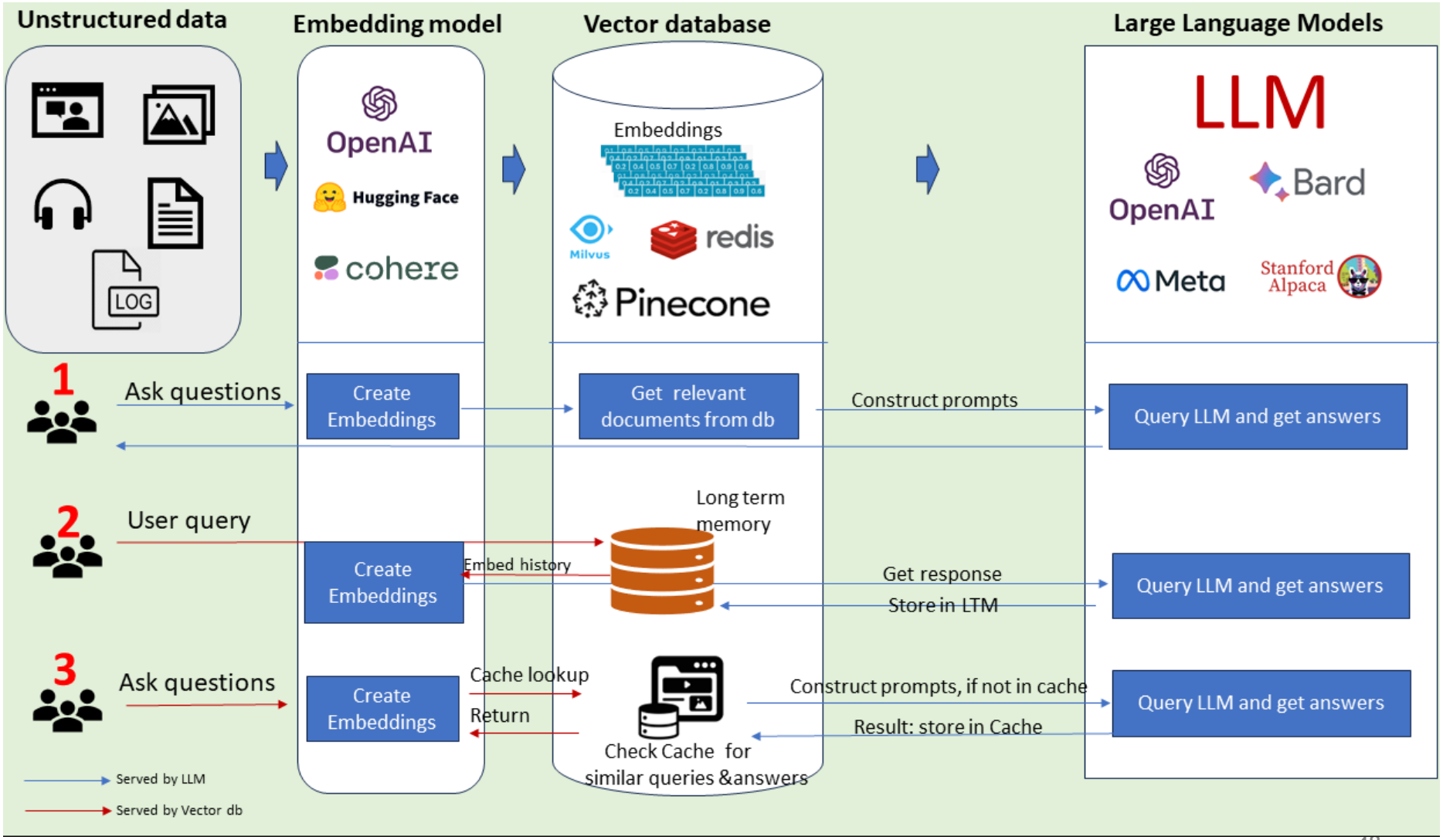
LLMOps: Principles and Practices

- Three main groups of new technical concerns in LLMOps:
- **Managing Data Ingestion**
  - Chunking of text, embedding creation/indexing/maintenance, Retrieval-Augmented Generation (RAG)
- **Managing LLM API Usage**
  - Prompt engineering / management, application abstractions, caching, logging, validation/“guardrails”
- **Customizing the LLM**
  - Finetuning, transfer learning, routing layers

LLMOps: Managing Data Ingestion

- LLM applications need to handle large multimodal corpora: text, PDFs, JSON, images, etc.
- 3 key sub-parts on handling such data:
  - Chunking: Partition docs, pages, etc. into bite-sized pieces
  - Embedding: Generate embeddings for chunks (using LLMs)
  - Vector DBMS: Store; index; retrieve for queries; maintain

LLMOps: Managing Data Ingestion



https://www.linkedin.com/pulse/3-ways-vector-databases-take-your-llm-use-cases-next-level-mishra

LLMOps: Prompt Engineering

***Q:** What is a "prompt"?*

- A prompt is an input to an LLM API with some of these elements:
- **Instruction:** A specific task or instruction for the model to do
- **Context:** External information or additional context that can steer the model to better responses
- **Input Data:** The input or question we need a response for
- **Output Indicator:** The type or format of the output

Prompt:

```
Classify the text into neutral, negative or positive.
Text: I think the vacation is okay.
Sentiment: neutral
Text: I think the food was okay.
Sentiment:
```

Output:

```
neutral
```


https://www.promptingguide.ai/introduction/elements

LLMOps: Prompt Engineering

Q: *What is a “prompt engineering”?*

LLMOps: Prompt Engineering

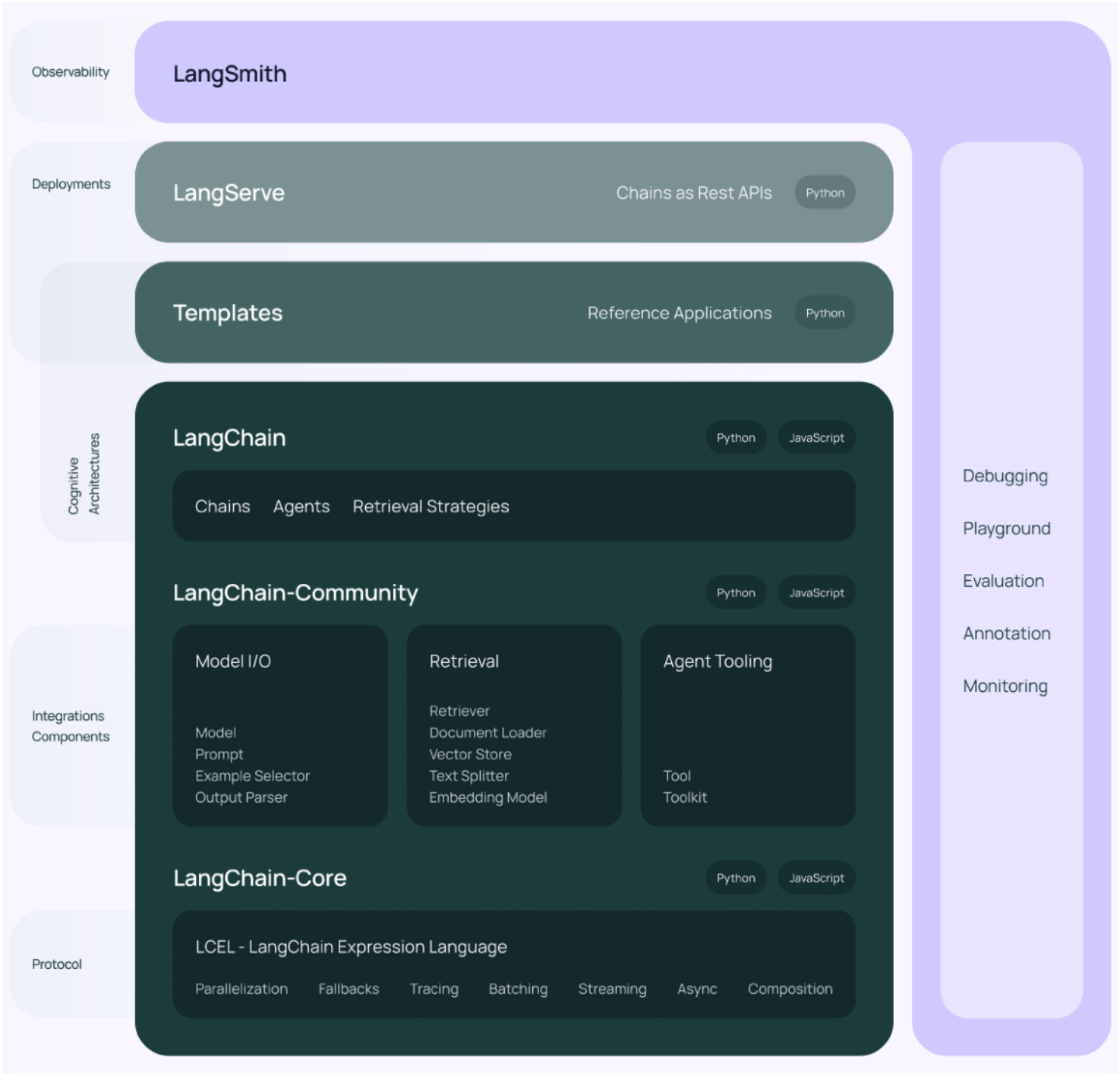
***Q:** What is a "prompt engineering"?*

- A set of "best practices" (witchcraft?) and "guidelines" (spellbook?) to craft prompts (spells?) for more effective use of LLMs
- Tricks and techniques abound, e.g,. "chain of thought", "self-consistency", "few-shot", etc.; take a generative NLP course
- Some useful practical references:

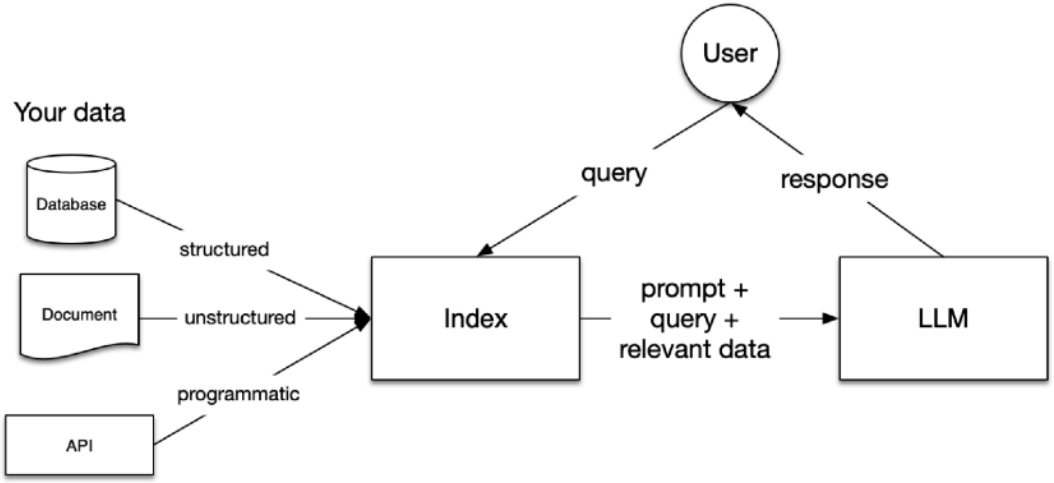
https://www.promptingguide.ai/techniques
https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results
https://aws.amazon.com/what-is/prompt-engineering/

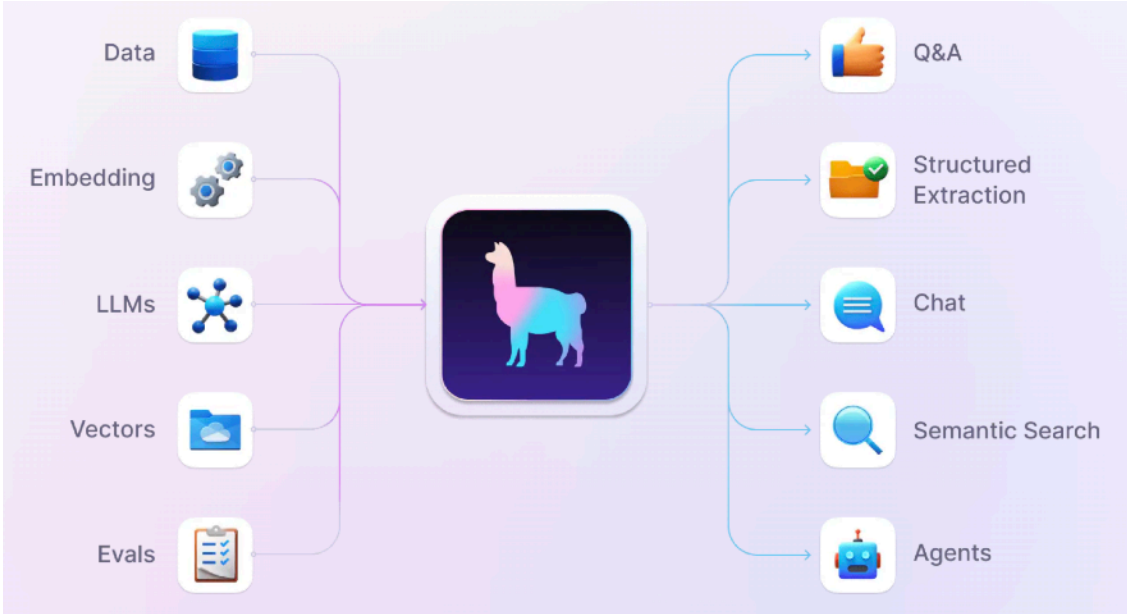
Emerging LLMOps-Specific Tools

- Prompt management and data embedding management are increasingly critical for LLM applications
- Tools for LLM appl. dev in flux; 2 recent popular examples:
- **LangChain:**
  - Enables more structured compositions of LLM API invocations called “chains” (akin to “transactions” in RDBMS applications)
  - Stitch together more holistic “agents” for an application
- **LlamaIndex:**
  - Mainly an aid for Retrieval-Augmented Generation and similarity search
  - APIs to parse, chunk, store, index, and query pieces


Observability
LangSmith
Deployments
LangServe
Chains as Rest APIs
Python
Templates
Reference Applications
Python
Cognitive Architectures
LangChain
Python
JavaScript
Chains
Agents
Retrieval Strategies
LangChain-Community
Python
JavaScript
Integrations Components
Model I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tool
Toolkit
LangChain-Core
Python
JavaScript
Protocol
LCEL - LangChain Expression Language
Parallelization
Fallbacks
Tracing
Batching
Streaming
Async
Composition
Debugging
Playground
Evaluation
Annotation
Monitoring

RAG and LlamaIndex





Regular MLOps vs. LLMOps

- **Center of the universe?** In-house trained model artifact vs. API-based access to a pre-trained LLM
- **Query input?** Feature vector vs. rich customizable prompts
- **Typical prediction targets?** Discriminative/inferential targets vs. generative content
- **Tuning?** Model selection aspects (hyper-par. tuning, arch. tuning, feature eng., etc.) vs. in-context learning / prompt tuning
- **Auxiliary data stores?** Feature stores/ML platforms vs. Vector DBMSs
- **Cost/latency optimizations?** In-house model/pipeline/resource optimizations vs. Fixed LLM operating points or re-routing

Review Questions

1. Briefly explain 2 reasons why online prediction serving is typically more challenging in practice than offline deployment.
2. Briefly describe 1 systems optimizations performed by both Clipper and VLLM for prediction serving.
3. Briefly discuss 1 systems-level optimization amenable to both offline ML deployment and online prediction serving.
4. Name 3 things that must be versioned for rigorous version control in MLOps.
5. Briefly explain 2 reasons why ML monitoring is needed.
6. Briefly explain 2 reasons why federated ML is more challenging for data scientists to reason about.
7. Briefly explain 1 way LLMOps deviates from regular MLOps.
8. Briefly explain 1 new technical concern in LLMOps that required new tool development.