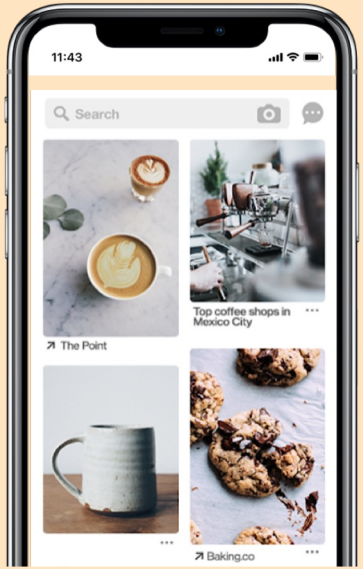
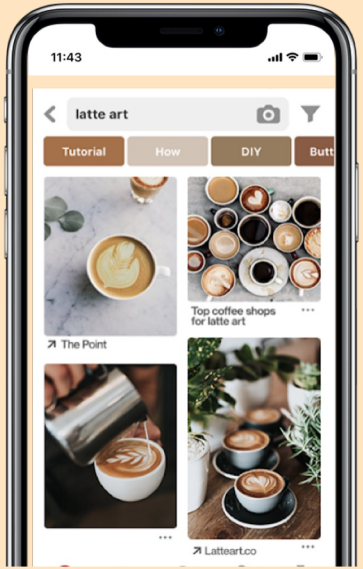
MLOps behind Ads Ranking @ Pinterest

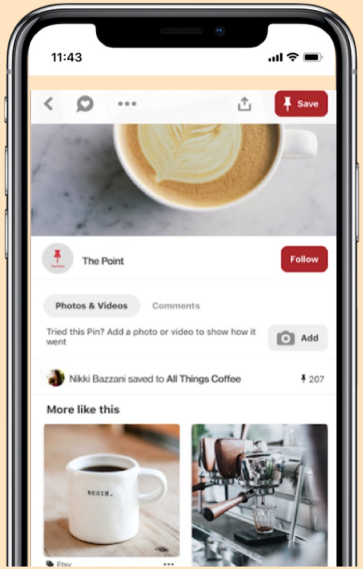
Aayush Mudgal
Modern DevOps Melbourne

October 9, 2023

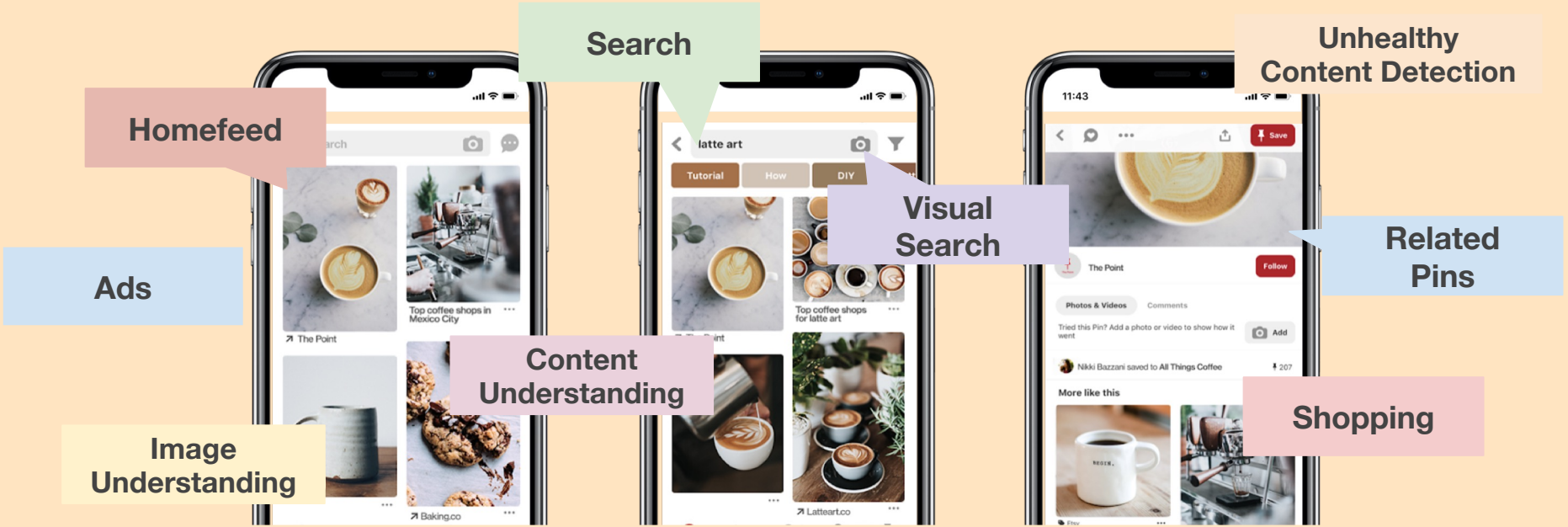
Bring everyone the inspiration to create the life they love



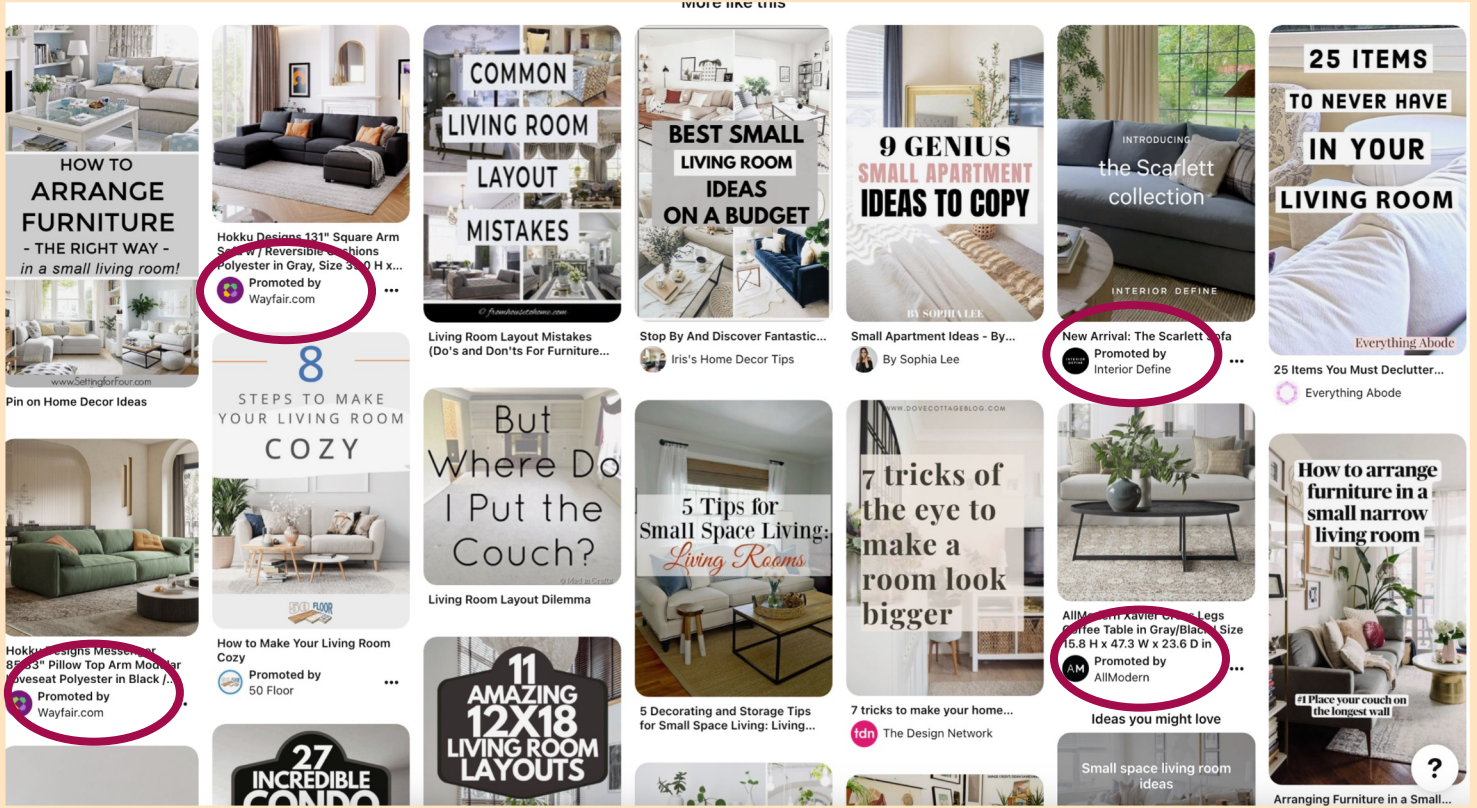




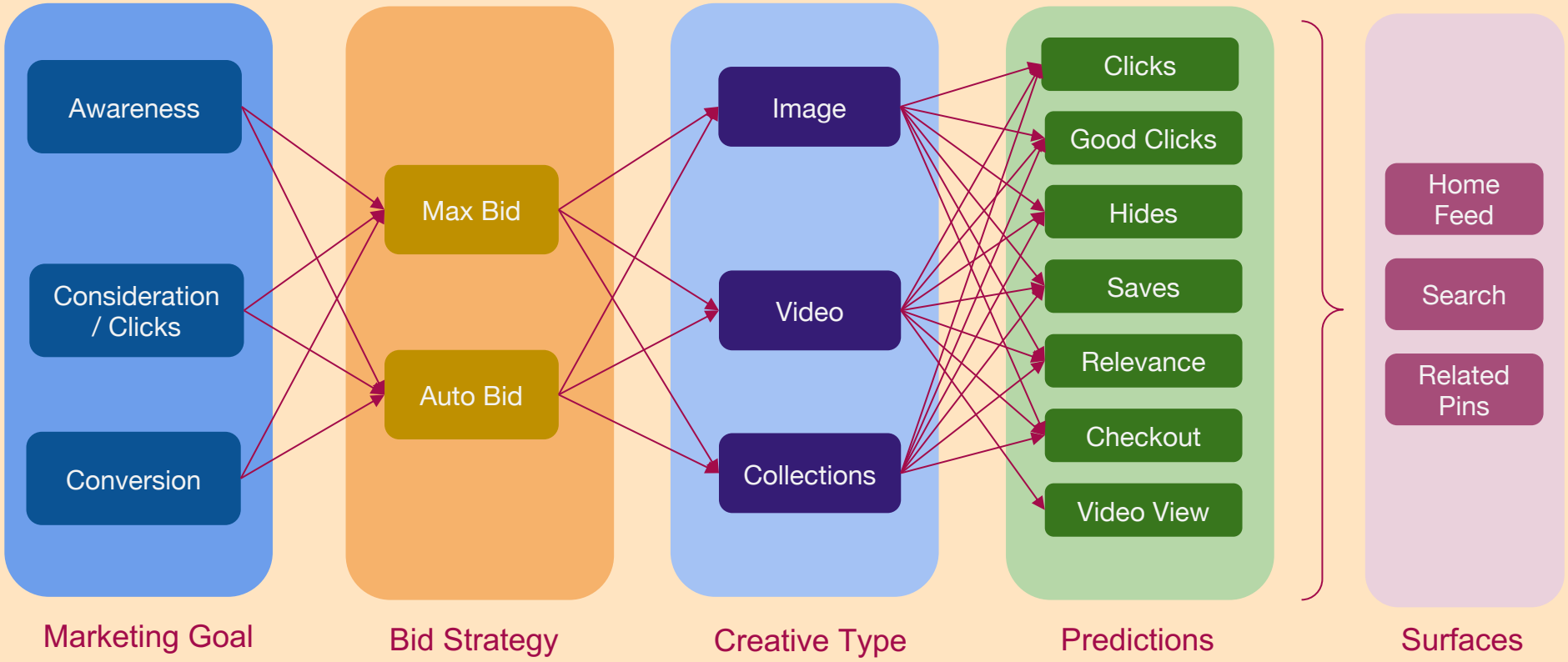
… powered by a variety of ML applications learning complex patterns from web scale data of +460M MAUs and billions of Pins

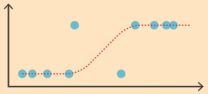


Ads

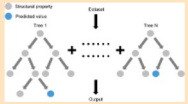


Ads Product in a nutshell
Managing Complexity



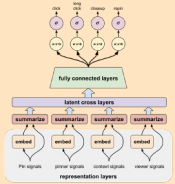


First ML model

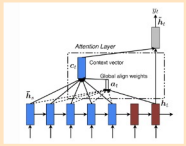


GBDT Model + LR

Neural Network Model (Deep and Wide)*

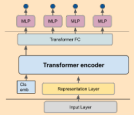


Deep Multitask NN model

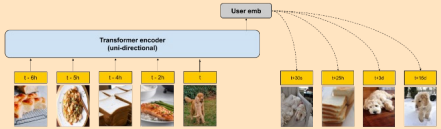


Attention Sequence

Native Tensorflow Serving



Transformer Sequence

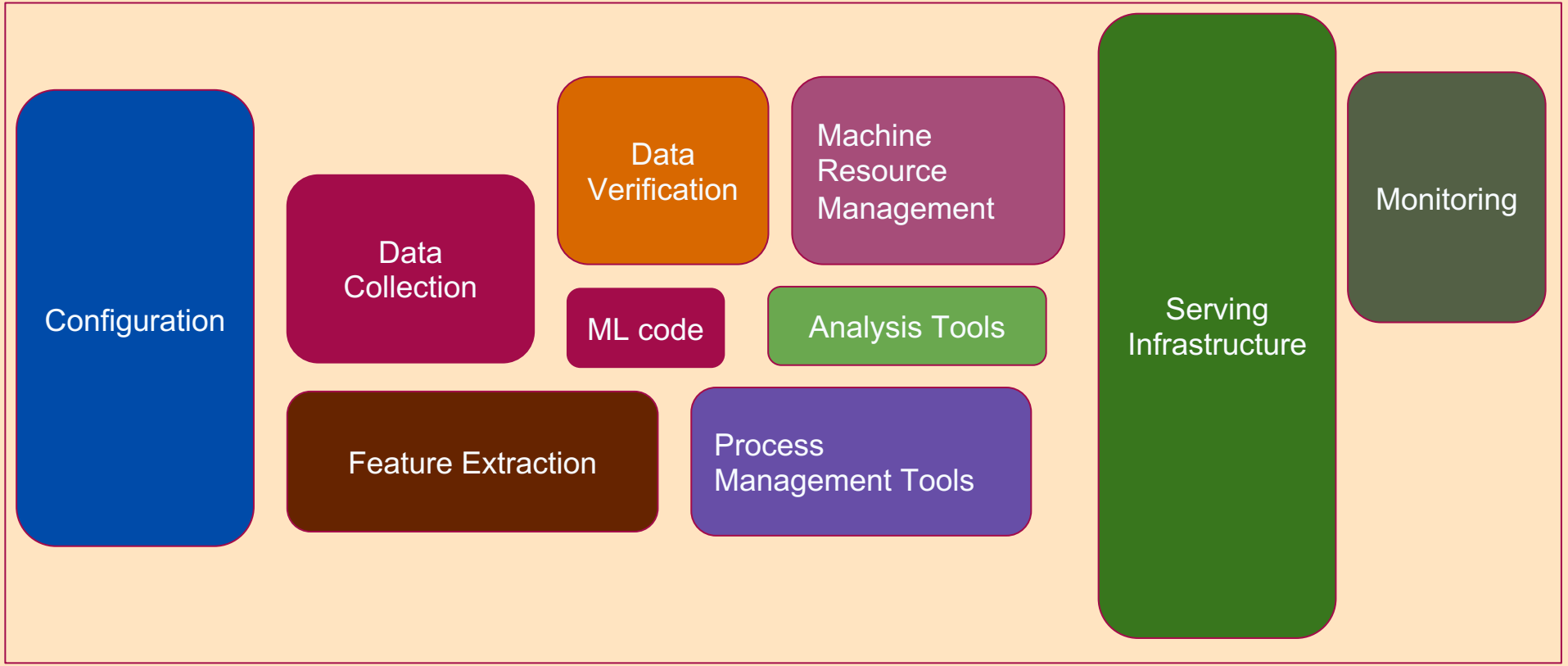


PinnerFormer

MLEnv & Pytorch

2014 2017 2018 2020 2021 2022 2022

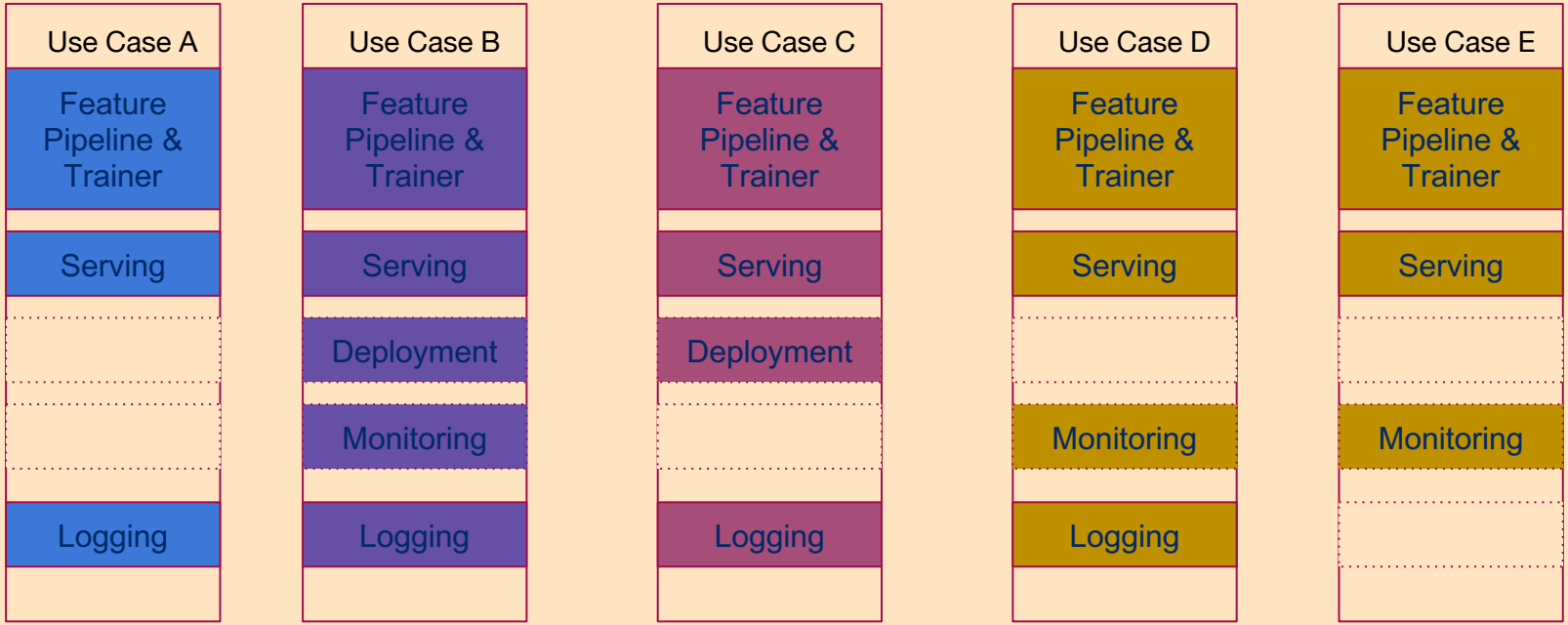
Empowering web-scale ML is complicated…
It has a LOT of steps



ML @ Pinterest

- **Large Scale:**
  - 460M+ MAUs, billions of Pins, billions of daily events
- **Variety of Use-cases:**
  - Recommendation & Ads (Retrieval, Ranking, Blending)
  - Representation Learning
  - Content Quality
  - User Understanding
  - …
- **Multiple Product Surfaces:**
  - Home-feed, Related Pins, Search, Visual Search, Shopping, Boards, Notifications, …
- **Centralized ML Platform**
  - Improve foundational components with **new capabilities** (Feature Store, Inference Service, Training Platform)
  - **100s of ML engineers**
  - Improve **ML developer velocity** (Calendar days, engineer days)

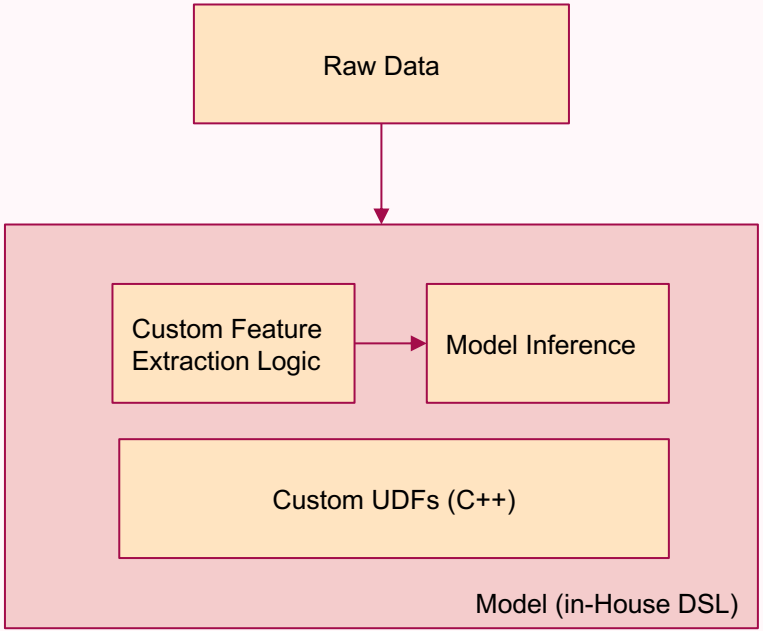
Reinventing the wheel



Feature Format (Previously)

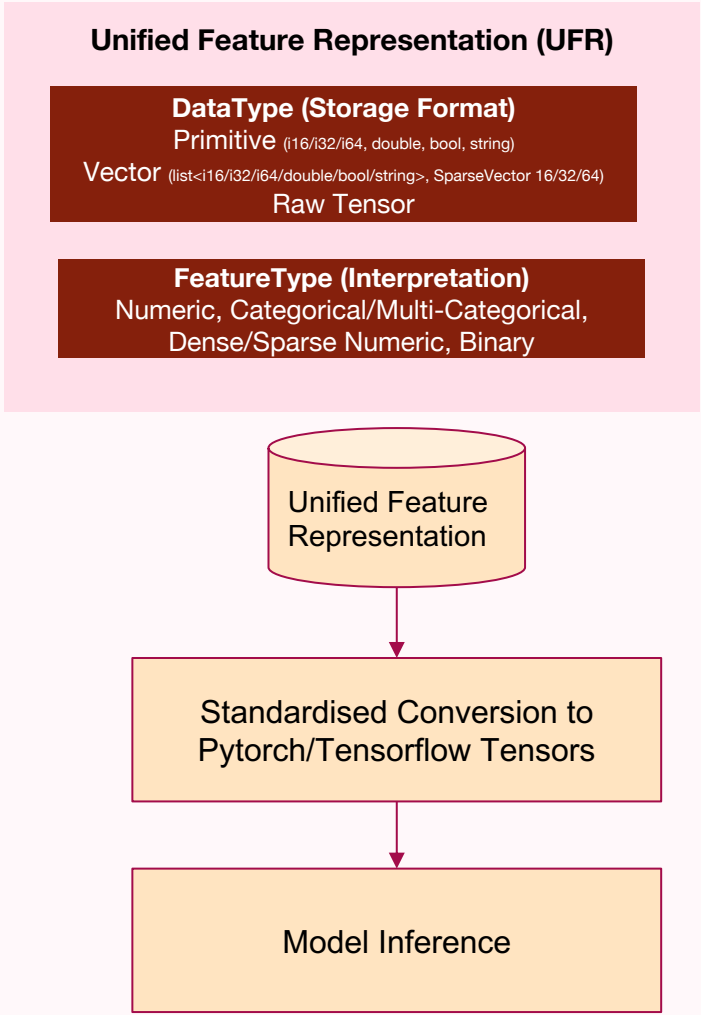
- Nested Feature Definitions
- Team-Specific Raw Data Structures, Thrift and FlatBuffers
- Slow Dev Velocity

```
struct Features {
    1: map<datatype, IOFBufPTr> raw_features
}
e.g.
struct UserViewTypeEnagement {
      1: map<string, double> clicks
}
```



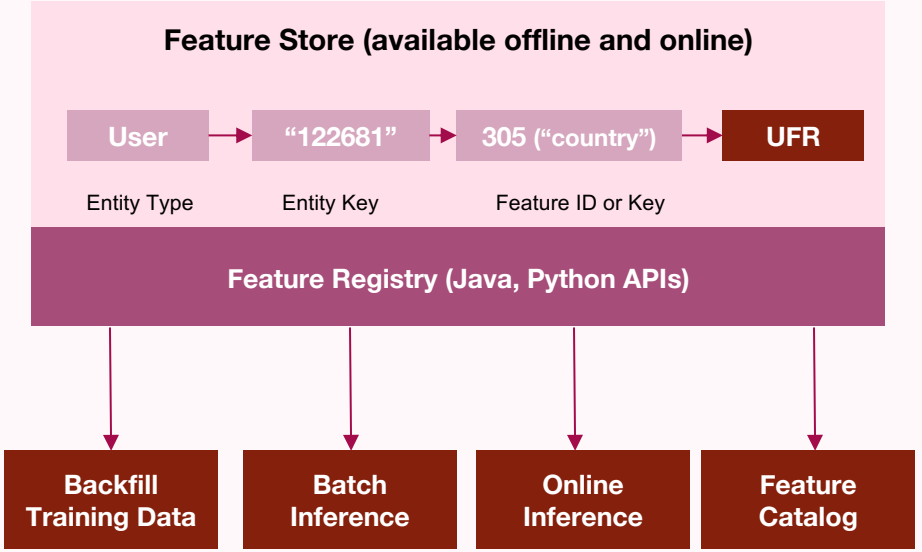
Feature Format (Now)

- Flattened Feature Definitions
- Easily Shareable Across Use-Cases
- Simplifies Models



Shared Feature Store (Now)

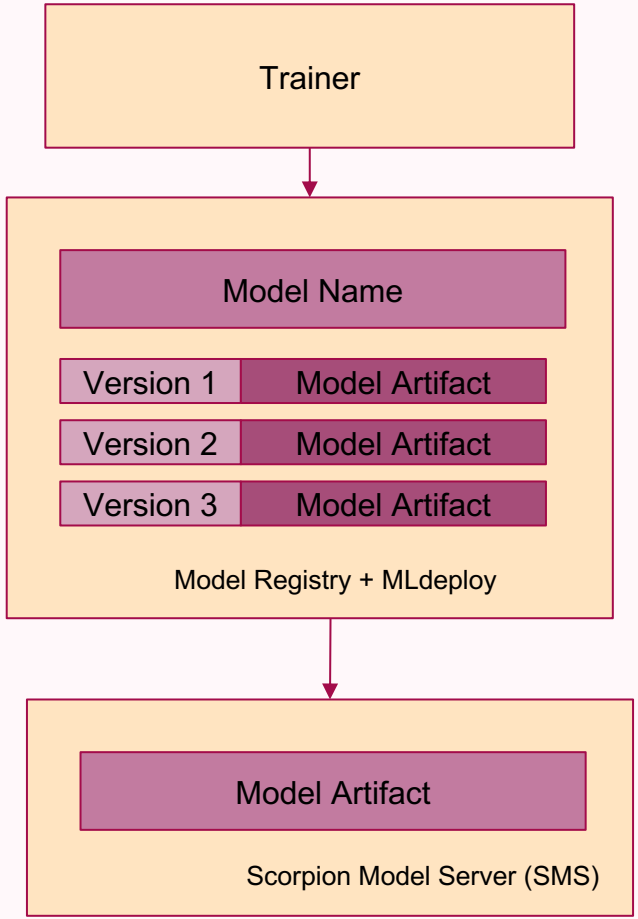
- Unified Feature Store: Features shared easily across all modeling use-cases
- Feature Backfilling Capabilities
- Feature Coverage and Alerting (offline and online)
- Self-Serve UI: To track feature usage across the system



Standardized Deployment through MLFlow

mlflow

- Version Controlled Models
- Tracks Training Parameters and Evaluation Metrics
- Reproducible Models
- Fast Code-Free Deployment
- UI Based Deploy and Rollback



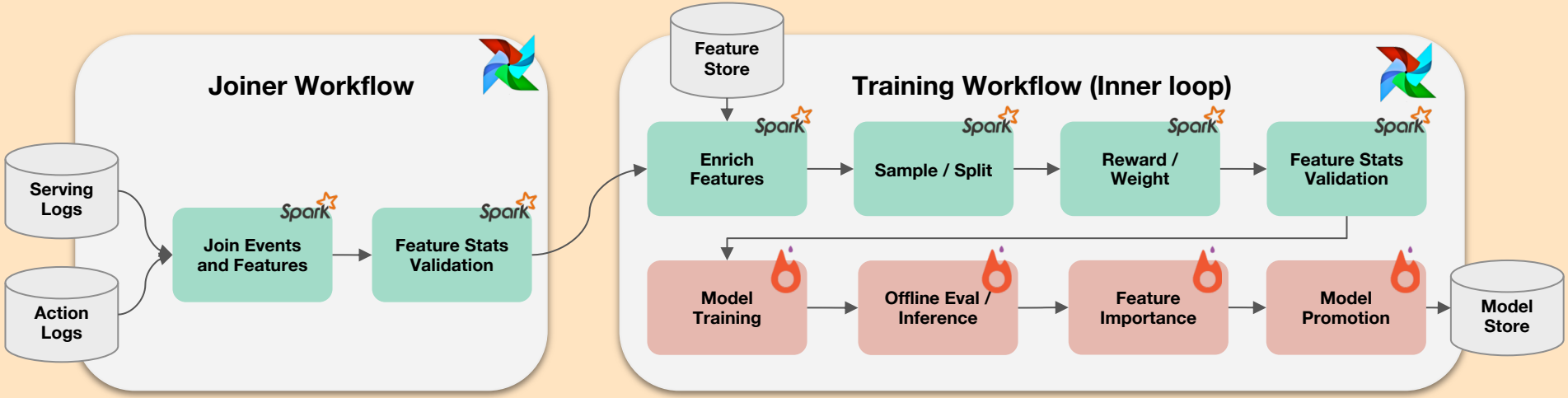
Model Insights and Analysis

Real-Time Feature Distribution and Coverage

Feature Importance Analysis
Local (Single Prediction) and Global (model wide)

Model Rollout Monitoring

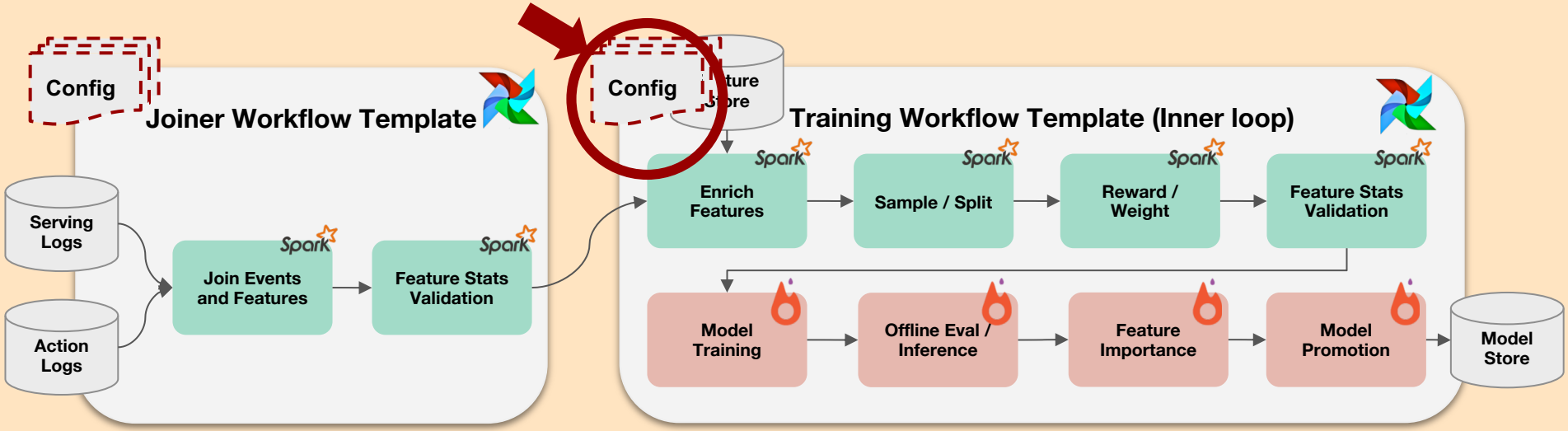
Typical ML Training Workflow



Templatized ML Workflows enable fast experimentation

- Rule of Three applies to ML Ops!
  - Refactoring =>Templatizing workflows.

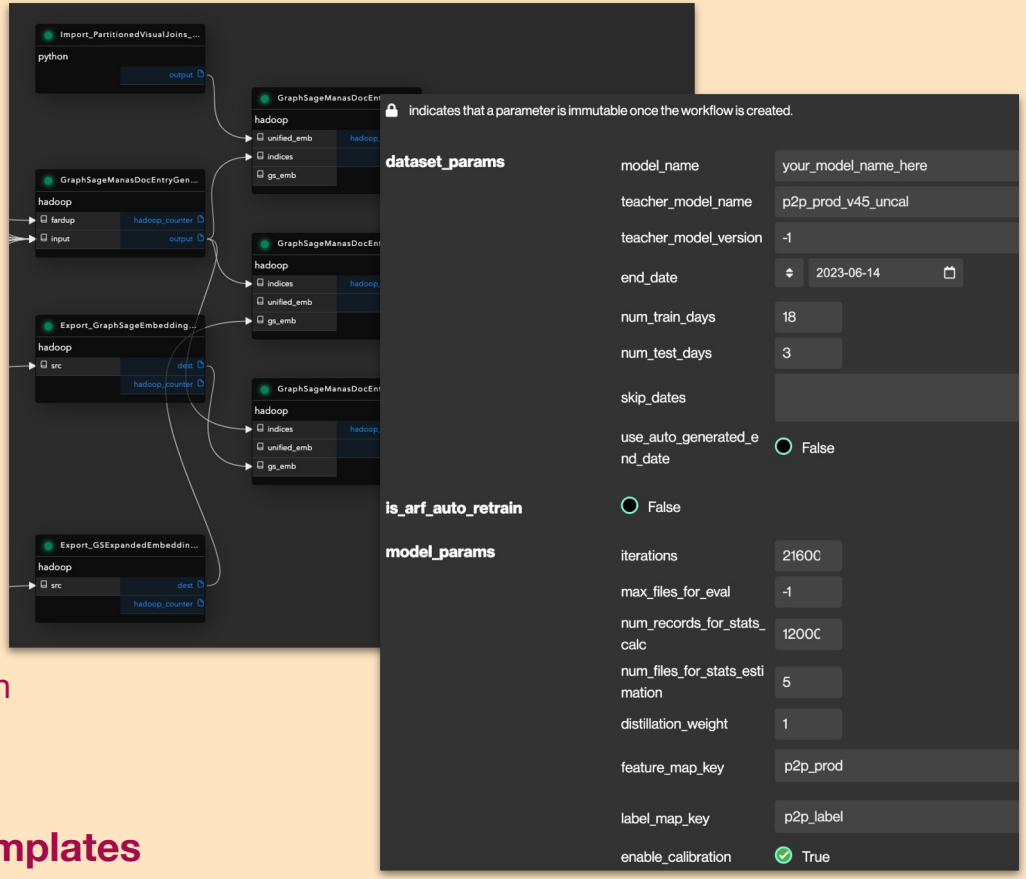
- Workflow templates significantly improve ML developer velocity and more…
  - Dataset Management: Caching, GC, discoverability, deduplication, lineage tracking, etc.
  - Reproducibility: Model = f(config, template)
  - Declarative Configs: experimentation plan



EzFlow - ML Workflow templatization system

- Workflow Templating as first-class feature

```
@WorkflowTemplate(
    date_range=DynamicDateRange,
    log_joiner_param=P2PLogJoinerParam,
    executor_name=StringChoiceField(
        default_value=HadoopOpEnvironment.MONARCH_PROD_002,
        choices=[
            HadoopOpEnvironment.MONARCH_ADHOC,
            HadoopOpEnvironment.MONARCH_PROD_001,
            HadoopOpEnvironment.MONARCH_PROD_002,
            HadoopOpEnvironment.MONARCH_PROD_003,
        ],
        mutable=True,
    ),
    joiner_lifespan=IntField(
        default_value=scorpion_consts.LIFESPAN_60_DAYS,
        mutable=True,
        validation_udf=is_positive,
        tooltip="life span for joiner output",
    ),
)
def daily_p2p_joiner_flow(flow, date_range, log_joiner_param, exe
    # User defined job chaining logic...
    return flow
```

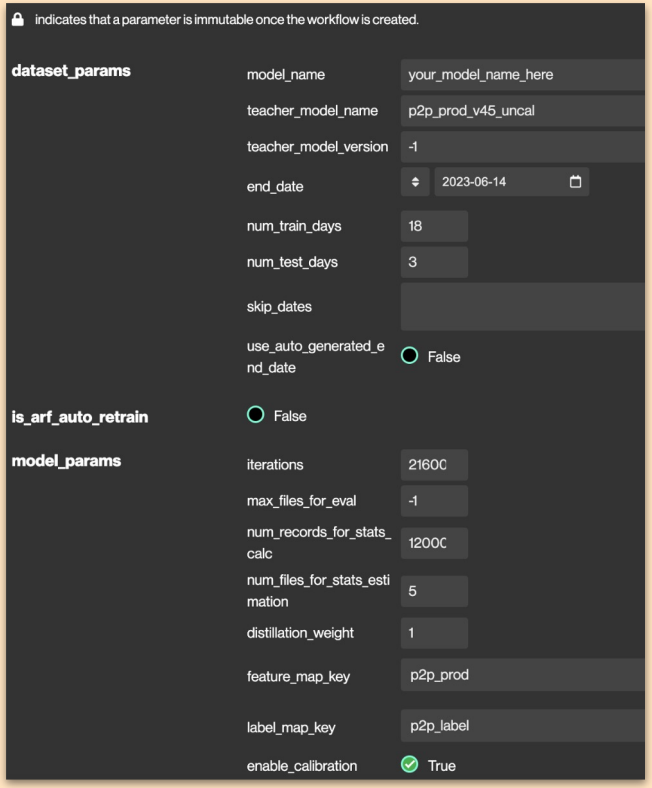


- Intermediate **data lineage tracking**
  - Runtime Synchronization, Deduplication
  - Automated dataset management
  - Reproducibility

- Enables **domain-specific workflow templates**

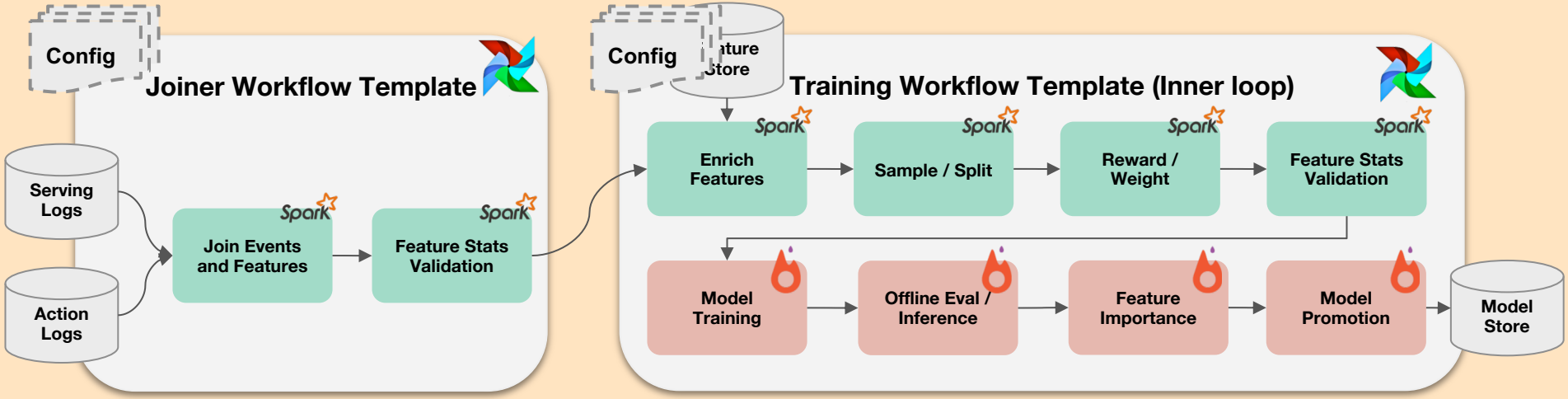
Iterating on new ideas with Workflow Templates

- Case:
  - Train a new student model using a different teacher model
- Steps (~hours) :
  - [Optional] Make model training script changes
  - Find the template you need
  - Clone configs from your "control" model (model you need to beat)
  - Change a few things, and schedule the workflow
  - Workflow runs periodically, refreshes your model
  - Deploy, run online A/B experiment.
- A few minutes / hours of developer time to push new model



Templatized ML Workflows enable fast experimentation

- 100s Pbs of training data
- ~1500 workflow runs / day
- ~3000 training jobs / day
- Faster dev velocity => More models trained => More improvements



Iterating on new ideas with Workflow Templates

- **Case:**
  - Try a new weighting algorithm for each event that takes downstream actions into account
- **Steps (~weeks):**
  - Write Job
    - Write a new **Spark job in scala** (e.g. need heavy row-wise operations)
    - Kick it off until it works
    - Write Unittest
    - Integration Test, data validation
  - Integrate with workflow template
    - Add a new **airflow operator,** and add if/else logic to the **workflow template**
    - Unittest workflow template
    - Integration Test
    - Land
  - Go to **meetings** between steps…
  - Kick off workflows using the templates (Same as before)

Hidden cost of pipelines - Scale first, learn last

- What's happening here? - Classic challenge with templating
  - ML changes really fast, and templates inherently reflect our understanding of the past
  - Reproducibility comes at the cost of dev-velocity!
    - 100X variance in developer velocity (Minutes/Hours -> Weeks)
  - **Scale First, Learn Last**

- Where were the bottlenecks?
  - Too many languages (Python, Scala, Java, Cpp, SQL, ..)
  - Too many runtimes / Frameworks (Airflow, Spark, PyTorch, In-house libs, …)
  - Code reviews
  - Build system
  - Meeting
  - Development is not interactive!

- Renaissance ML engineers move fast. Others move very slow. Bad for the business.

Model Iteration

- Model architecture became standardized with common building blocks e.g. transformers.
- Modern GPUs and training frameworks has made the training time much shorter and only account for a small portion of the total developer time

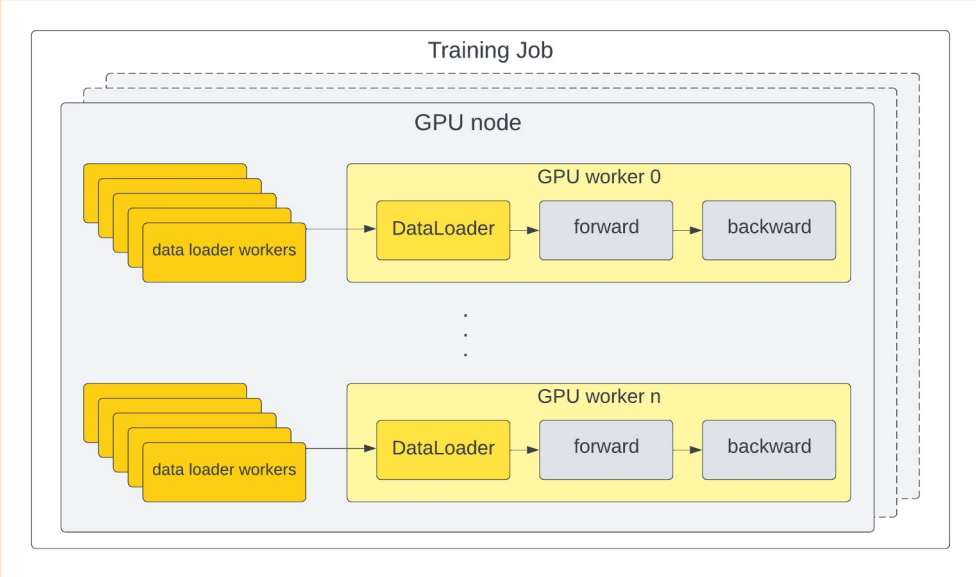
ML Dataset Iteration

- Achieve great improvement with the right dataset preparation.
  - Sampling strategies
  - Labelling
  - Weighting
  - Batching inference for transfer learning and distillation
  - Feature backfilling
- Requires lots of iterations to find the right combinations
- Dataset Iteration at this scale is **Slow**!
  - Opportunities for optimization!

Last-Mile Processing in Trainer

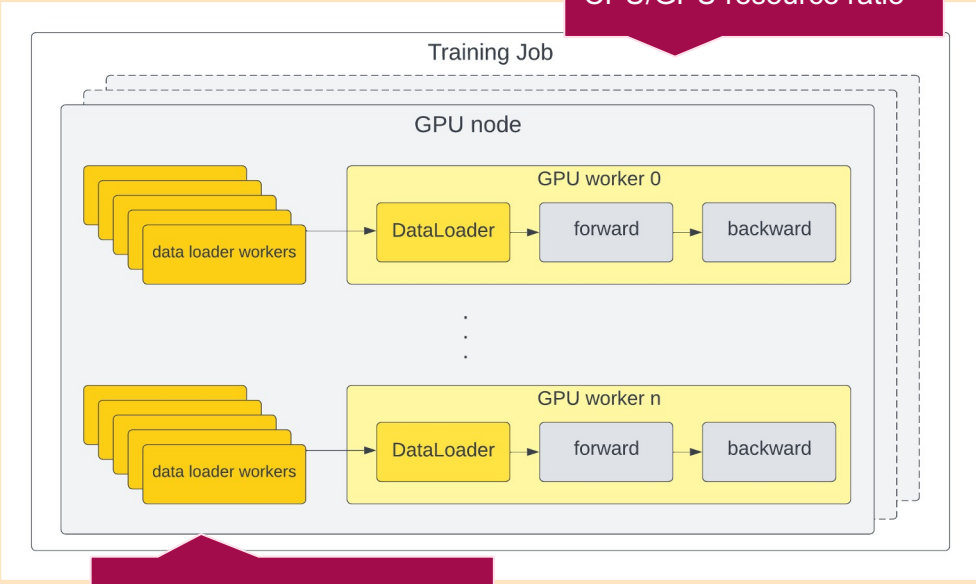
Engineers start to move data processing in to the trainer to address the challenges in the workflow pattern

- Easy to iterate, All done in one framework
- Low Engineer-days cost: Streaming Pipeline: feedback are immediate
- Low Calendar-days cost



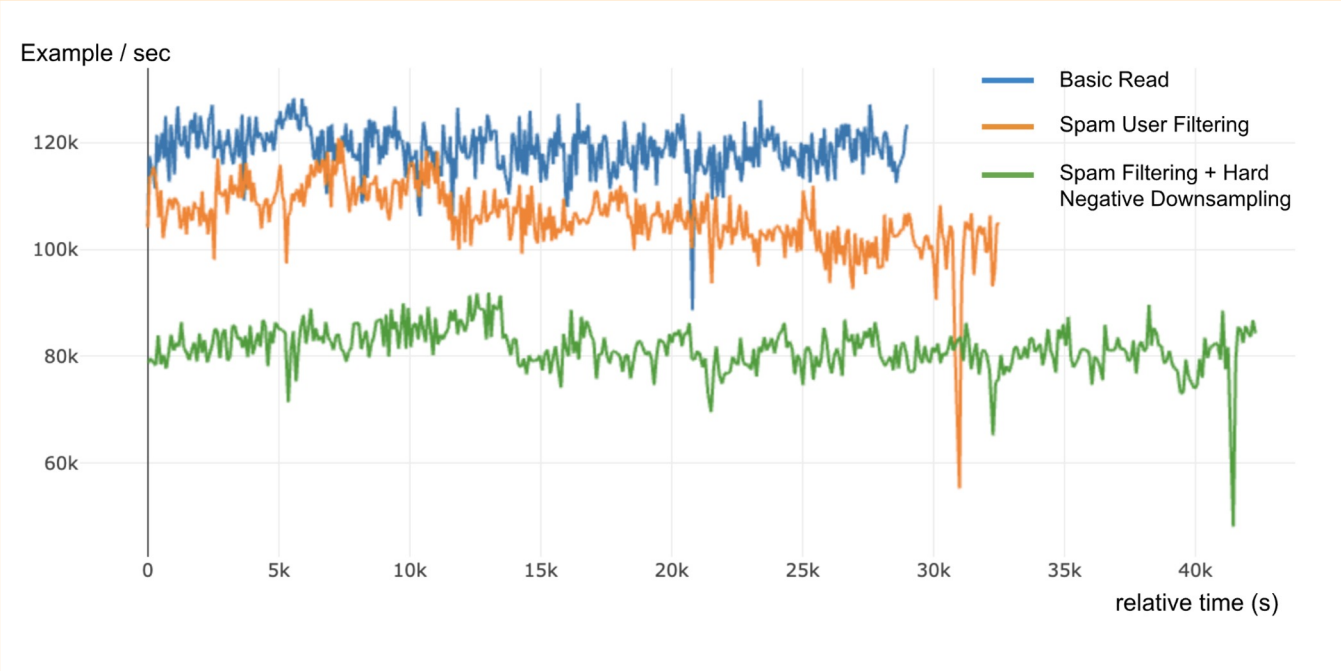
Challenges

- Data processes cannot scale beyond local machine.
- When adding more data processing workload, cpu utilization grows faster than gpu utilization causing GPU to be underutilized.



Challenges

Adding more data processing workload on the trainer inevitably reduce the overall training throughput.



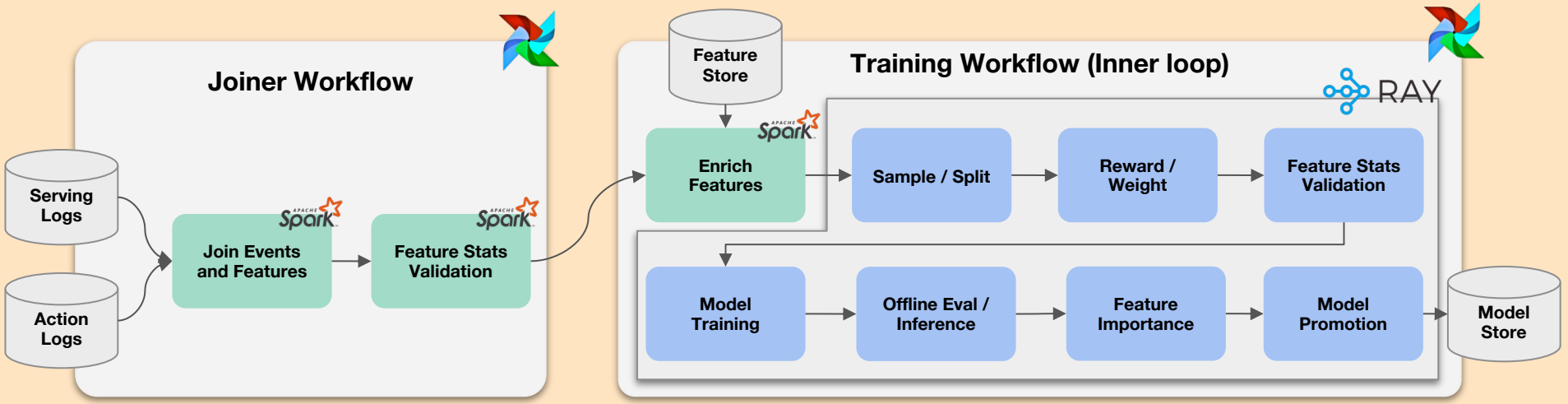
Three Key Requirements

- **Distributed Processing**:
  - Able to efficiently parallelize large scale data processing across multiple nodes
- **Heterogeneous Resource Management**:
  - Managing both GPU and CPU, ensuring workloads are scheduled on the most efficient hardware
- **High Dev Velocity**:
  - Everything should be in a single framework.

Ray fulfills all these requirements. In addition, it presents a unique opportunity to provide engineers **a unified AI Runtime for all the MLOps components**

Last-Mile Processing with Ray

Overview of our training pipeline with Ray:

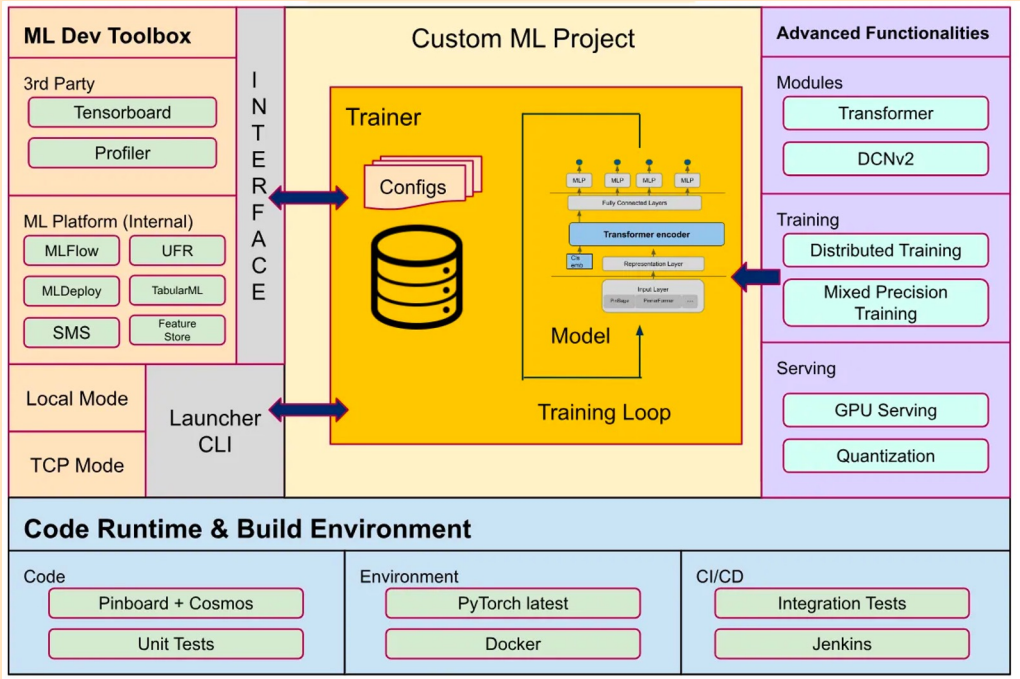


How ML is done in Pinterest Today!

In Pinterest , we built **an unified, Pytorch-based** ML framework that provides:

- **CICD & Docker image as a service**
- **Standardized MLOps** integration
- **Reusable building blocks** shared across multiple teams, use cases. (e.g. Dataset loader, Torch modules, training loop)
- **95% of Training Jobs** at Pinterest are built on top of this framework.

See our blog post for detail!



Acknowledgement

Thanks for XFN collaboration with entire Ads Quality, Machine Learning Platform, Core Engineering Team, Ads Infra, Advanced Technology Group, Content and User Engineering teams