LLMOps on Azure

Day 1

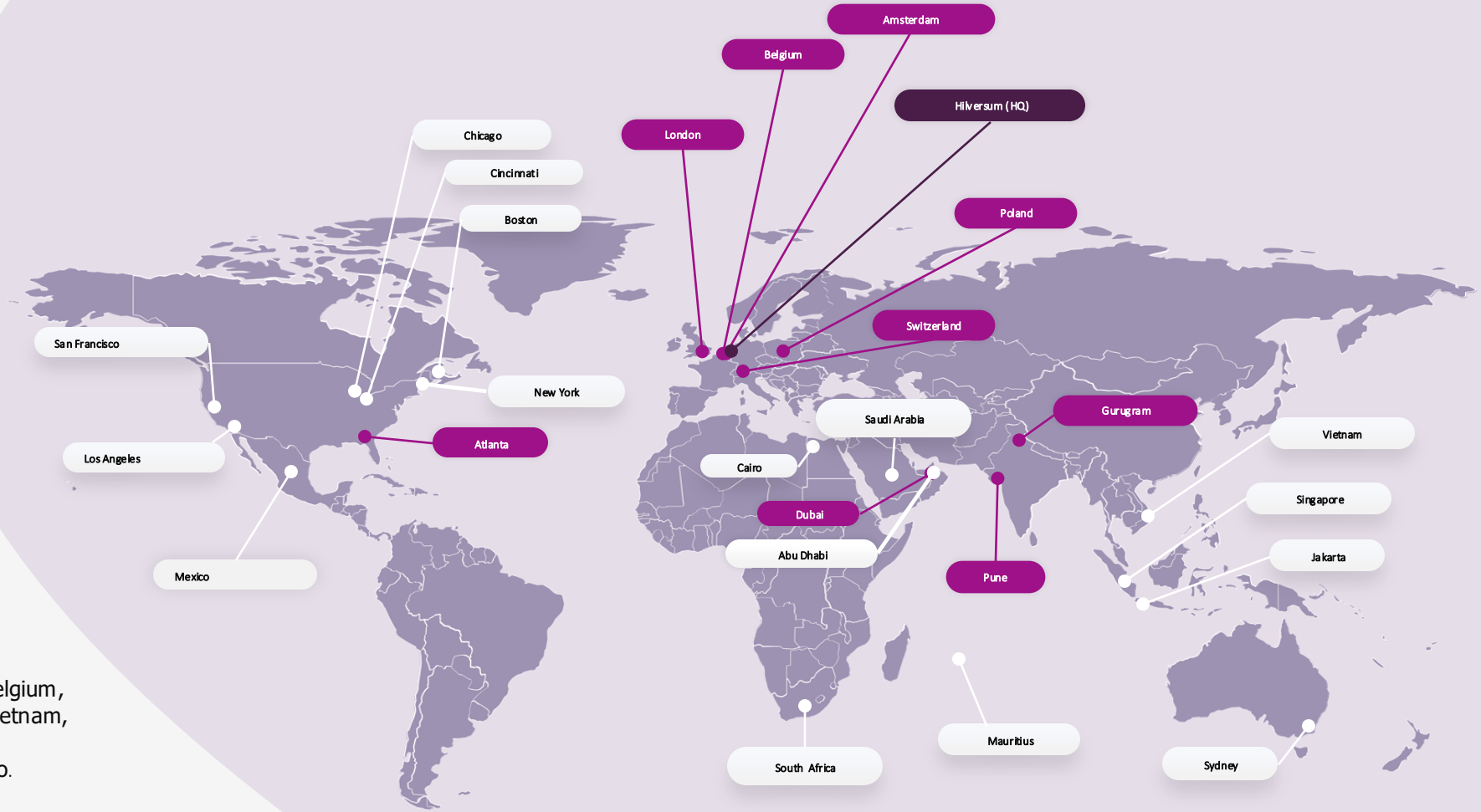
Xebia | Data

Xebia

Who are we?

Xebia

A **full-stack IT consultancy** with a global footprint



Xebia Facts

People

Jan 2024 5.000+ professionals

Offices

the Netherlands, Germany, UK, Belgium, Denmark, India, Poland, Dubai, Vietnam, US, Canada, Australia, Singapore, Switzerland and Mexico.

Full-stack digital consulting

Organized into service lines, each with a tremendous amount of knowledge and experience within a particular field, such as Agile, DevOps, **Data and AI**, Digital Strategy, Cloud, Software Technology, Security, Low-Code, and Microsoft Solutions.

Our Global Footprint

- Centre's
- Customer Presence

Xebia

About me

Prashant Srivastava

- 3 years at Xebia
- Machine Learning Engineer
- Cloud Architectures
- GenAI Applications
- Trainer

Short introductions

1. A bit about yourself…

   (What's your current position, your experience/background,…)
2. What is your experience with ML/ LLMs in Production?
3. (Maybe anything specific you'd like to learn?)

Some comments about this training…

1. We will cover a simple LLM Use Case

- won’t go too much detail into the LLM itself
- It’s more about LLMOps

2. It’s going to be quite hands-on

- we assume you have little prior MLOps knowledge

3. <u>This training is for you!</u>

- Let me know what you want to learn
- ask questions, give feedback, etc.

Schedule for today

Day 1

- Intro MLOps & LLMOps
- Use case & solution design
- Building modularized LLM app
- Lunch 🥗🌮☕
- Evaluation & experiment tracking
- Containerization and deployment
- Automation with CI/CD

Day 2

- Intro LLM system monitoring
- Tracking online metrics
- Logging traces
- Lunch 🥗🌮☕
- User feedback
- Iterating on the solution
- Wrap up

Repo: https://github.com/godatadriven/academy-llmops-in-azure

Xebia | Data

Why LLMOps?

Why ~~LLMOps~~ MLOps?

Daisy Data Scientist @ TurbineDynamics

Task: Help business improve generated power forecast.

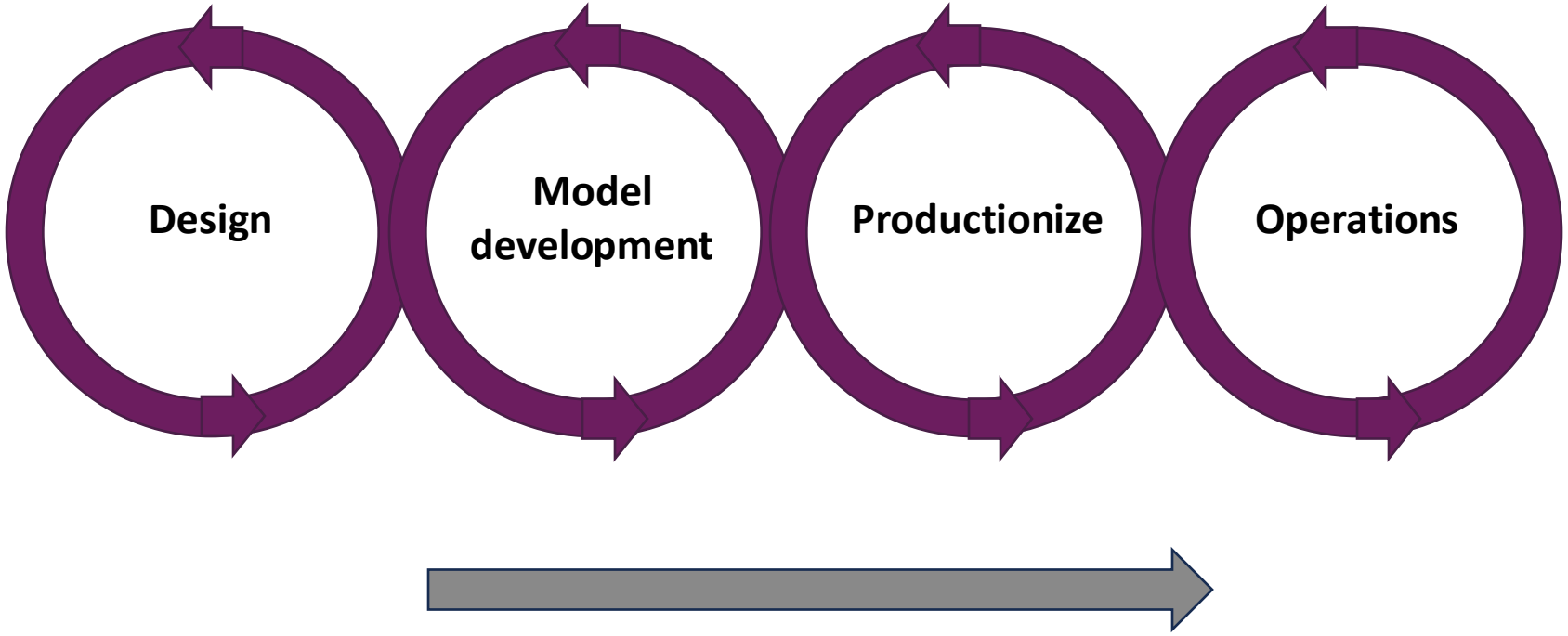
Ideate with business to define use case & value

Explore data to find possible relationships

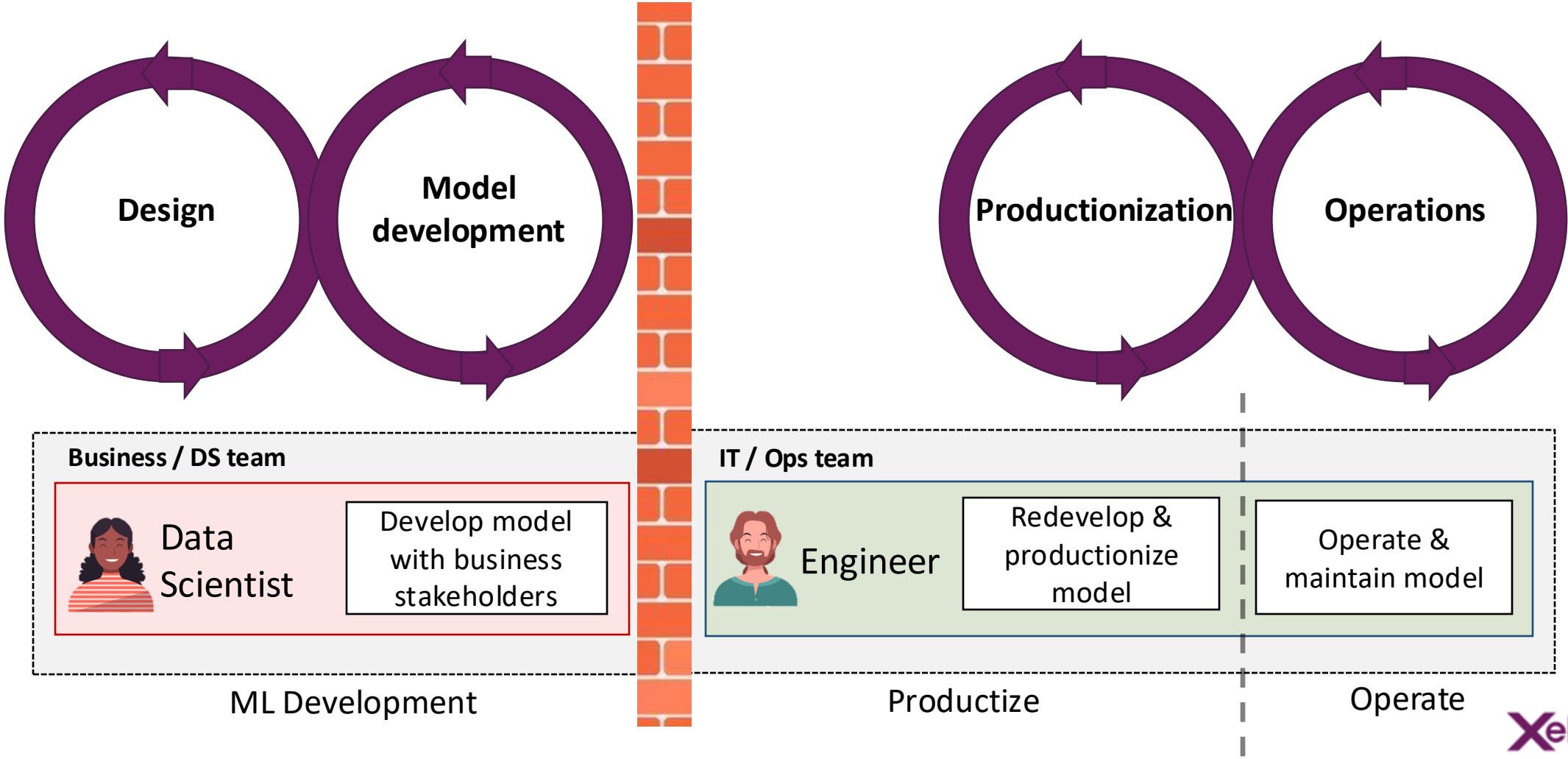
Create predictive model in a notebook

Now what?

Ideally: a project follows the “MLOps lifecycle”

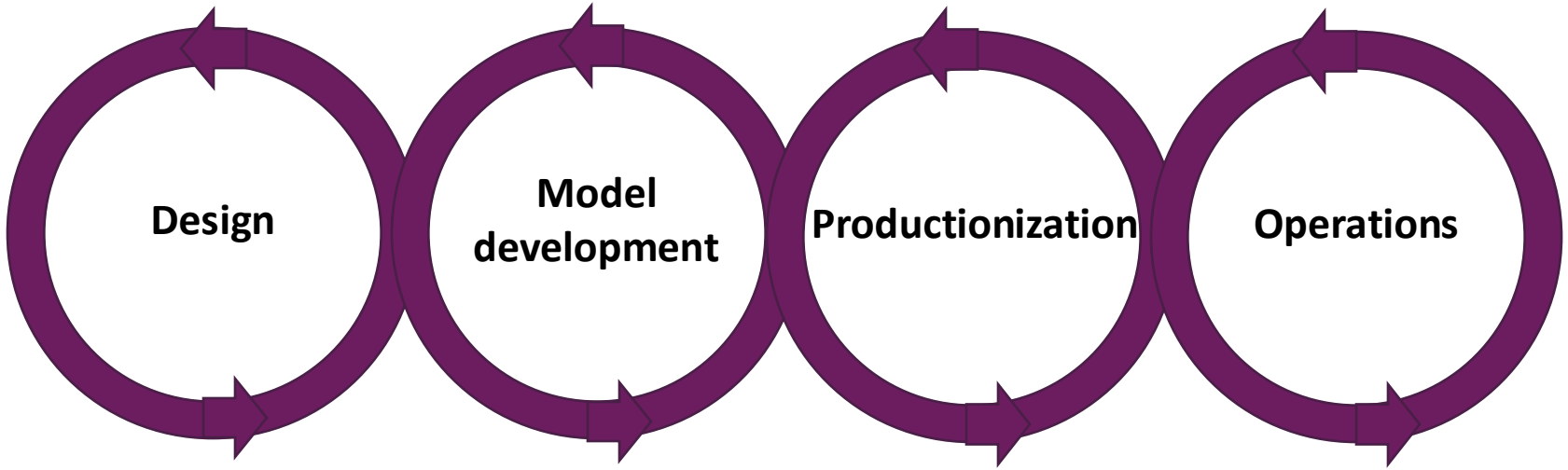


However, we often see a *handover*



Xebia | Data

Goal of MLOps
Close the gap between DS and Ops



End-to-end DS product team

Data Scientist

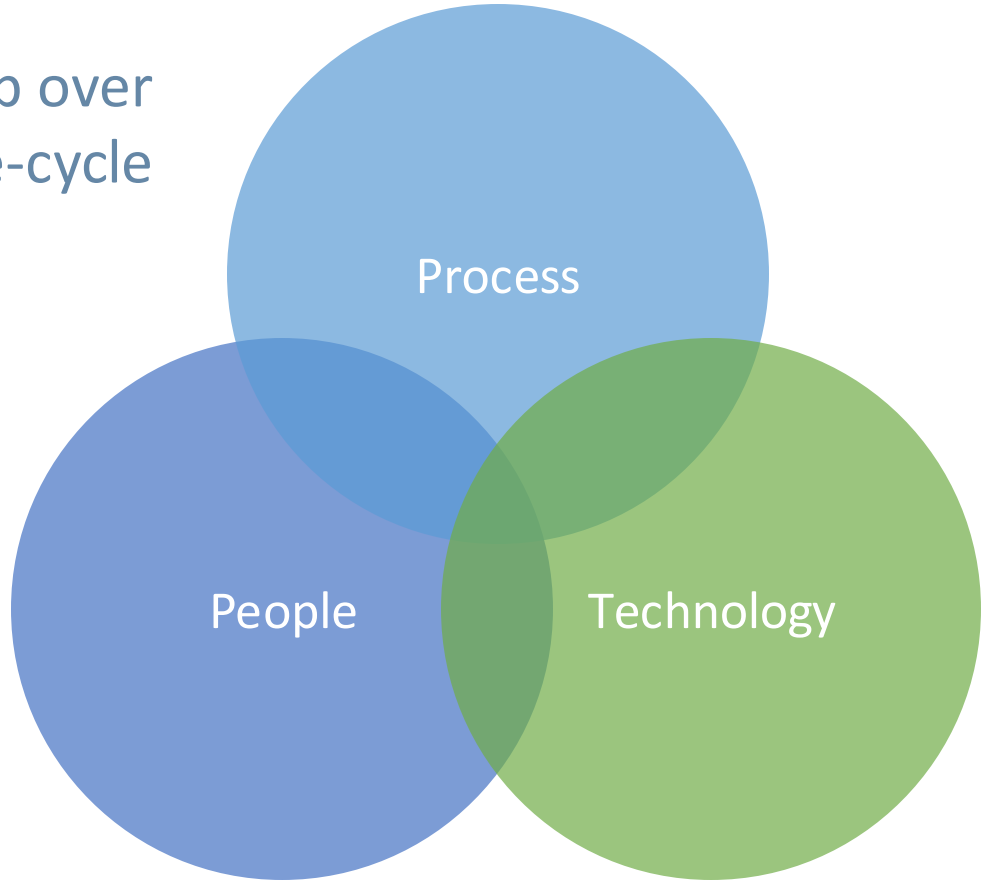
Develop model with business stakeholders	Create production ready model package	Monitor model issues

Engineer

Set up project (structure, CI pipelines, etc.)	Implement CD, orchestration, monitoring	Monitor pipeline issues

MLOps: Close the gap by combining the right **people, processes and technology**

Enable ownership over the entire ML life-cycle



Having the right roles & responsibilities

Support the product team with the right tools & platform

Xebia | Data

Why LLMOps?

Daisy Data Scientist @ LanguageDynamics

Task: help business **extract structured information from unstructured text**.

Ideate with business to define use case & value

Explore data and **tweak prompt** to get intended result

Create working solution in a notebook

Now what?

LLMOps aims to solve a similar problem as MLOps in a similar way!
That is, by combining the right **People**, **Processes**, and **Technology**

Xebia | Data

Differences MLOps and LLMOps

	MLOps	LLMOps
People	Data scientist (statistics, ML) Machine learning engineer Data engineer (structured data) Product owner …	Data scientist (NLP) Machine learning engineer Data engineer **(unstructured data)** Product owner ***Prompt engineer*** **(potentially)** …
Processes	Project templates (typical ML use cases) Experimentation (logging hyperparams and metrics) Pipelines (preprocessing and ML training) Deployment (usually API endpoints) Monitoring (typical ML metrics and infra) …	Project templates (typical LLM use cases, RAG, etc.) Experimentation **(logging prompts and metrics)** Pipelines (preprocessing and LLM batch jobs) Deployment **(usually web apps)** Monitoring (LLM metrics**, traces** and infra) …
Technology	Scikit-Learn Tensorflow/Keras/PyTorch/Jax MLflow FastAPI …	LangChain, LangGraph, PydanticAI LLM-specific APIs* Streamlit, Flask, FastHTML, … LangSmith …

*** Assuming we are using a provider's API**

Xebia | Data

Differences MLOps and LLMOps*

	MLOps	LLMOps
Compute	CPU / GPU	CPU (making use of external model API's)
Experimenting	Established metrics Logging model params and weights	Harder metrics definition Logging prompts
Pipelines	Training and evaluating Batch predict	Evaluating Batch predict
Deployment	As model endpoint	As web app or endpoint
Monitoring	Logs, metrics	Logs, metrics, traces

Use case

News Reader:
Extract structured info and predict stock price changes from news articles

- We work at a bank and are **interested in analyzing news articles for likely stock price changes**
- We have created a notebook with an **LLM-solution to extract relevant info from the articles**
- In the end, we want a **web app where users can upload their articles which are then automatically analyzed**

News article

```
title: "Company XYZ hit rock bottom"
summary: ...
is_about_business: TRUE
business_info: [
  {
    business: "XYZ"
    stock_price_change: "decrease"
    reason: "They hit rock bottom"
    relevant_substring: ...
  },
  ...
]
```

Structured output

Xebia | Data

Exercise:

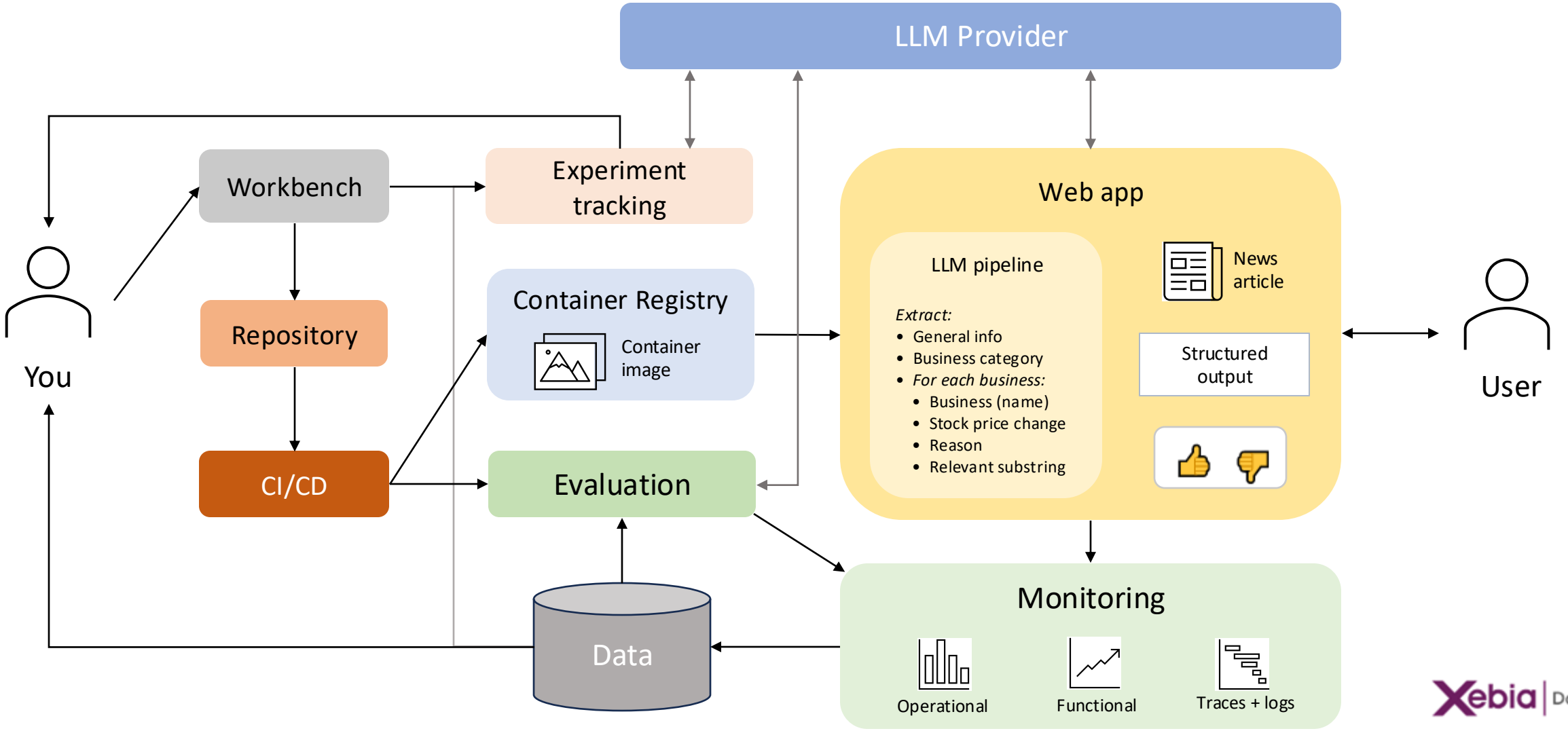
Develop a high-level solution design (architecture) for the News Reader use case

Requirements:

- Should provide a **web app** that accepts text and returns stock price change predictions
- The model is **a provider's LLM** (no self-training)
- The user should be **able to provide feedback** on the LLM outputs
- Ideally, you would want to include **monitoring & experimentation (evaluation)**
  - **For evaluation, you may want to store some annotated articles**
- For auditing, you would want to log not only the predictions,
  but also general info about the business, the article and a reason for the prediction
- Code should be maintained in a **code repository** (e.g. GitHub)
- The webapp is containerized, and we need to store our webapp versions and implement automatic testing/ quality checks etc. (**CI/CD**)

Xebia | Data

High-level solution diagram

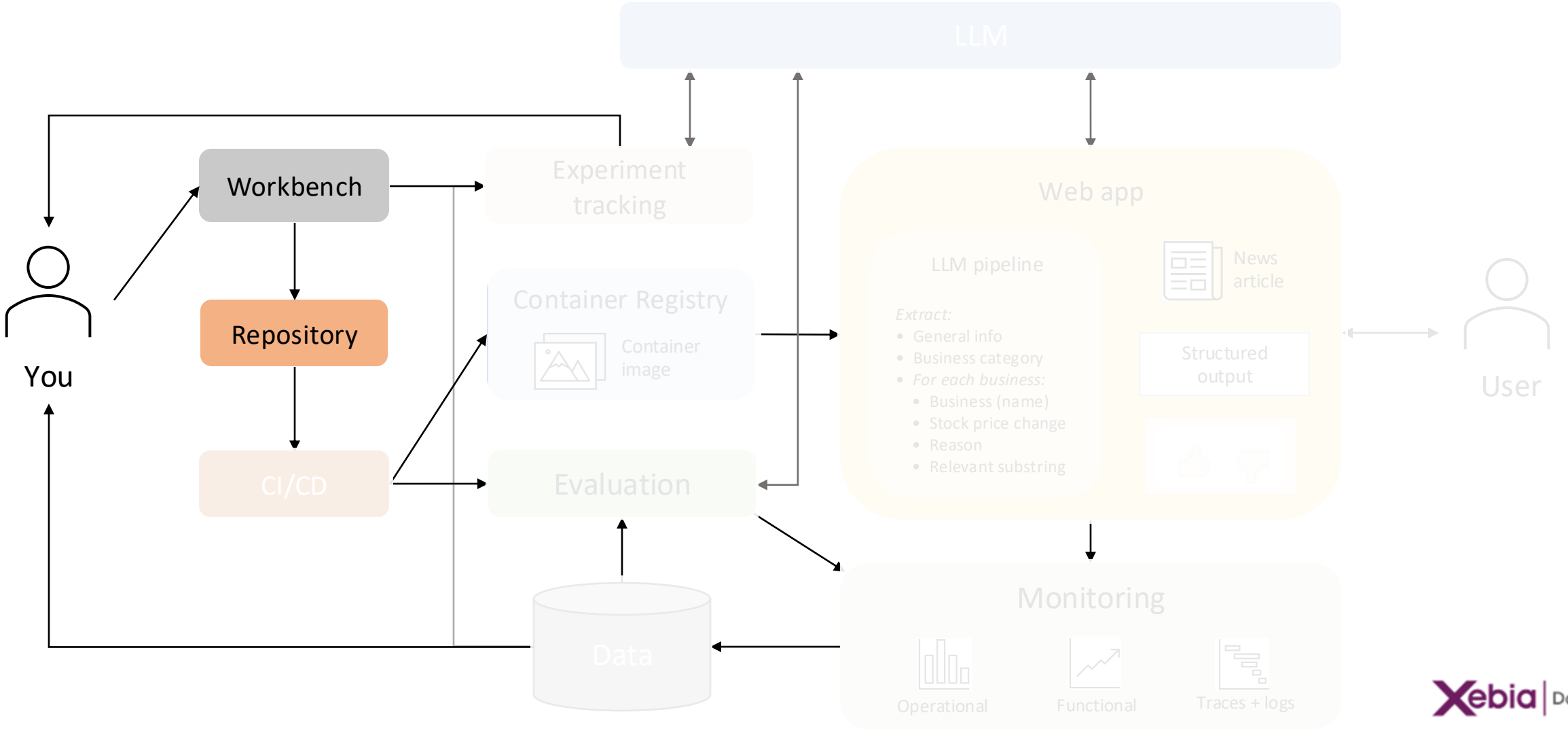


Xebia | Data

Break time!

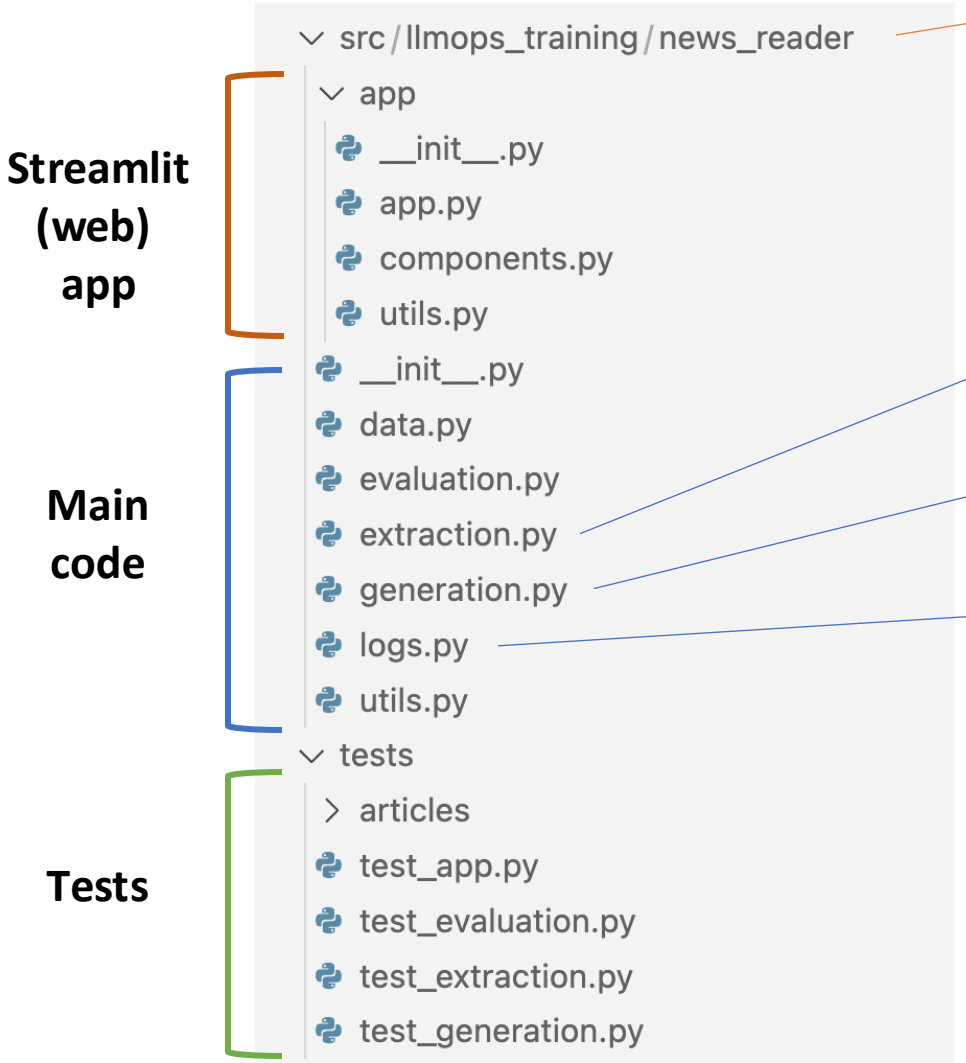
Getting started with the code

High-level solution diagram



Xebia | Data

Our code structure



Our package that we'll extend during the training

Main logic of our LLM solution

LLM API connections

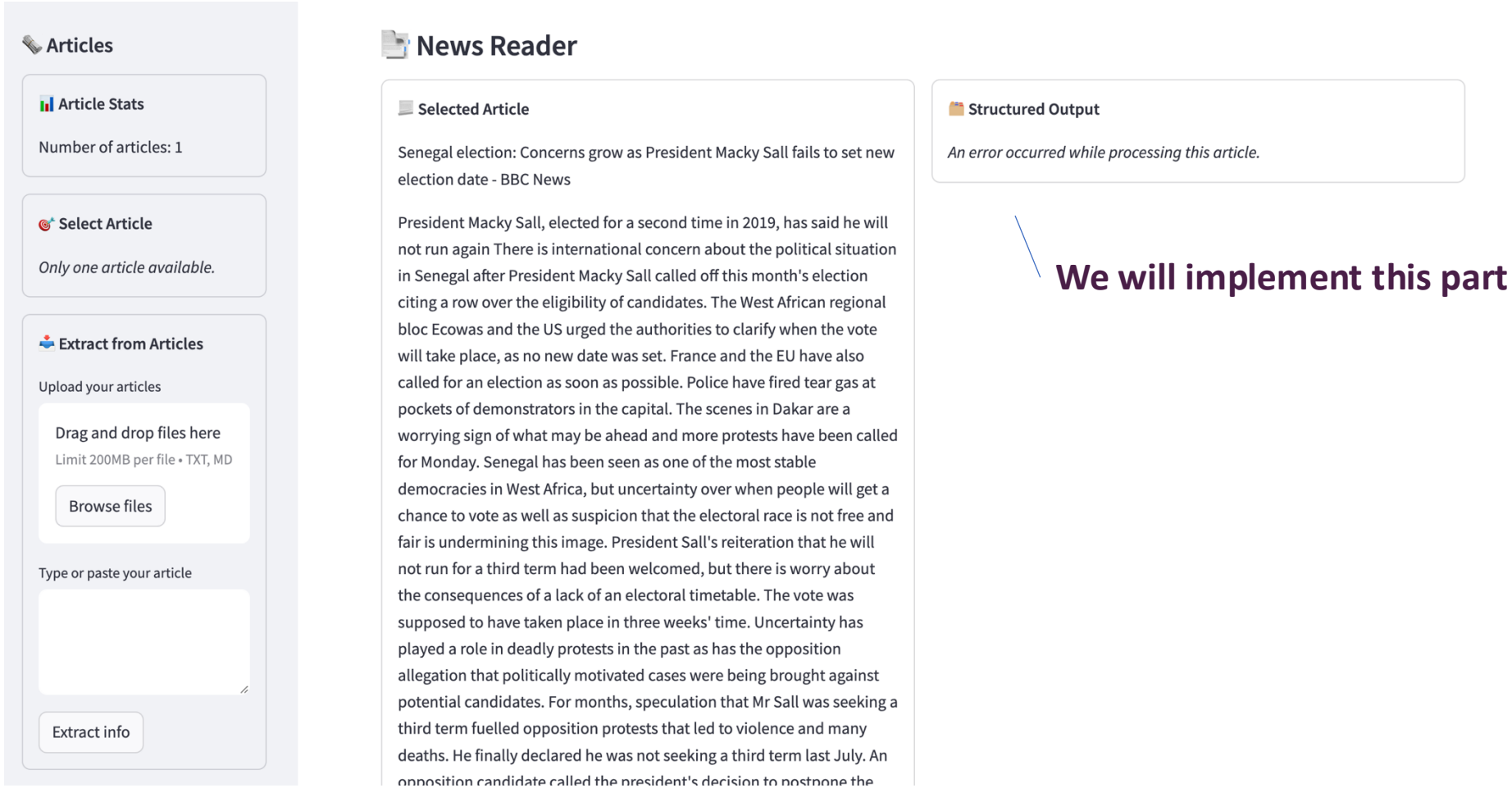
Configuring logger + tracer

Where you'll spend most time:

> exercises

Xebia | Data

How our web app looks like:



We will implement this part

Xebia | Data

Exercise:

Getting familiar with the use case

First, let's get you set up, so you can start coding!

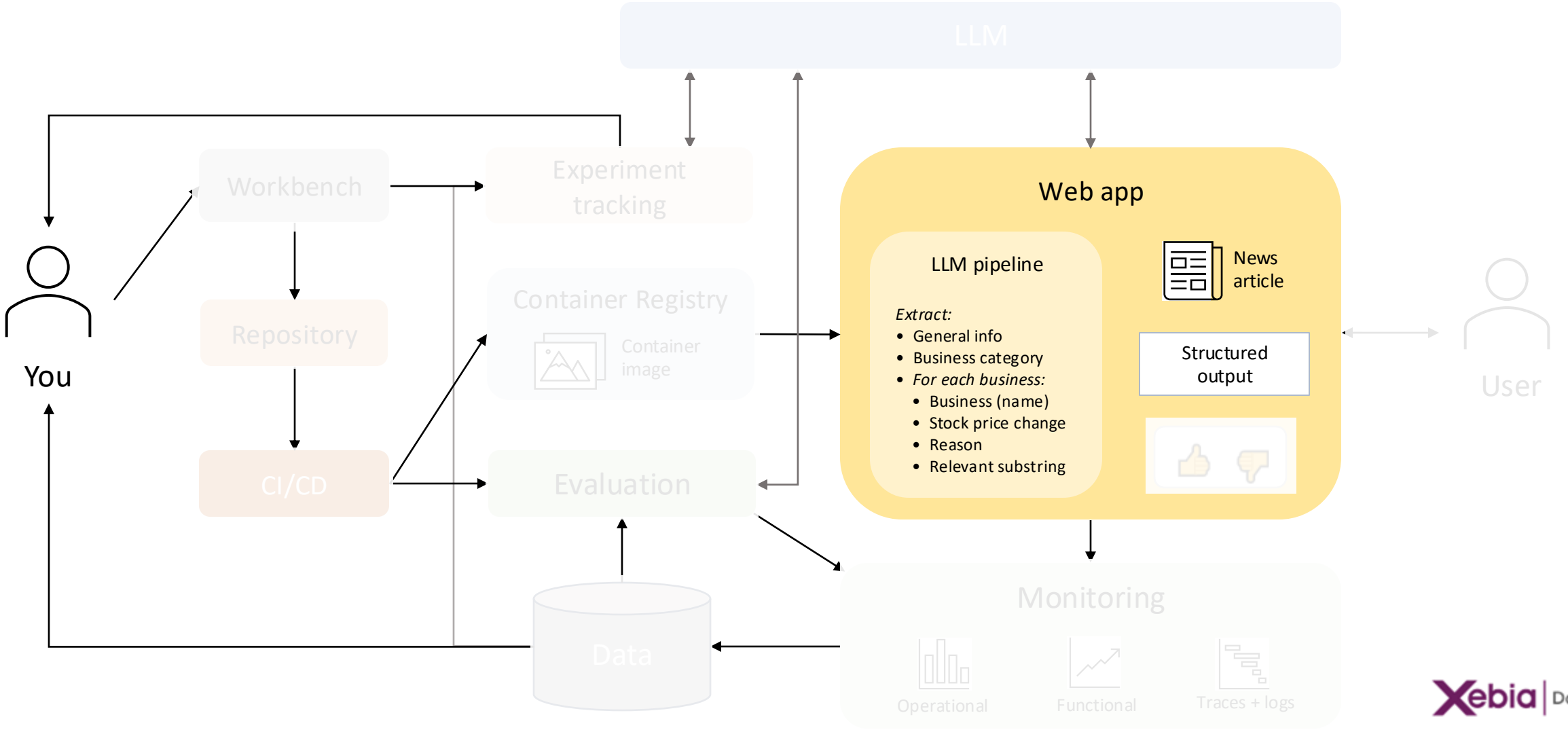
- Open the repository in GitHub: https://github.com/godatadriven/academy-llmops-in-azure
- Follow the instructions in the `readme.md` to set up your coding environment

Done setting up?

- Run through **exercises 1 - 3** in the `/exercises` directory

Modularizing the LLM pipeline

High-level solution diagram



Xebia | Data

“One prompt to rule them all” not always the best approach

```
prompt = f“””
You are a helpful assistant.

You extract structured info from news articles.

If a given article is business-related,
find all businesses mentioned in the article
and for each business in the article,
predict whether stock price will increase

Here’s a 100 articles:
{articles}
“””
```

? ? ?

LLM

Xebia | Data

Splitting complex tasks up into smaller tasks can have several benefits

- We can apply different models per task
- It allows the model to focus on smaller tasks at once, possibly improving performance
- It allows for better testing, debugging and maintaining
- In some cases, we can parallelize different tasks

However, there is a **balance** to be found!

- A single prompt can be **complex but quick** to execute
- Multiple smaller prompts can have above benefits, but may come **at the cost of speed**

Xebia | Data

How this could look like in our case

```
prompt1 = f“Extract general info from {article}”

prompt2 = f“Classify whether {article} is about business”

prompt3 = f“Extract involved businesses from {article}”

for business in businesses:

    prompt4 = f“Predict stock price change for {business} in {article}”
```

Exercise:
Modularizing the solution

Let's split up our LLM pipeline in smaller modules and integrate them in the app

- Run through **exercise 4** in the `/exercises` folder

Lunch break!

Schedule

Day 1

- Intro MLOps and LLMOps
- Use case & solution design
- Building modularized LLM app
- Lunch 🥗🌮☕
- **Evaluation and experiment tracking**
- **Containerization and deployment**
- **Automation with CI/CD**

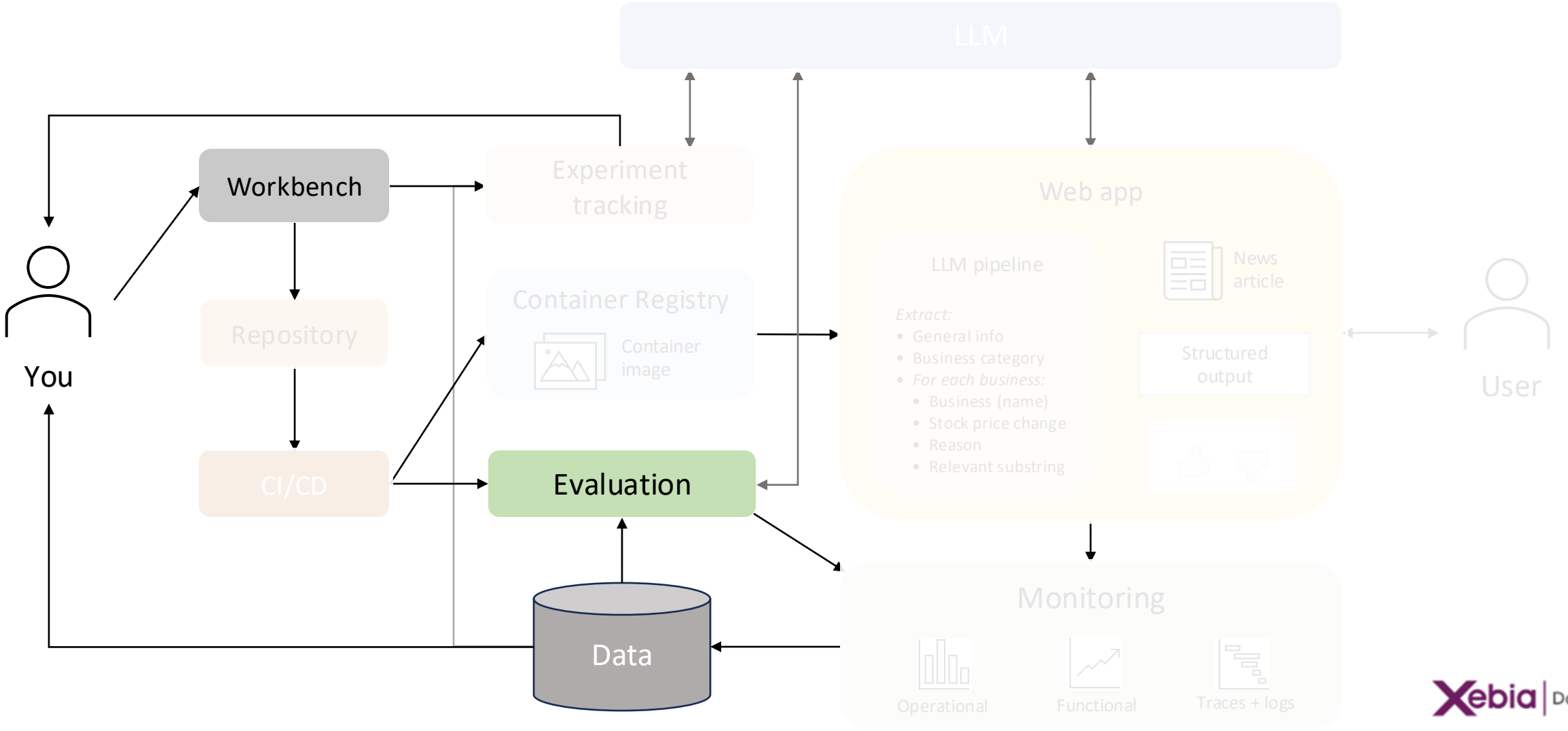
Day 2

- Intro LLM system monitoring
- Tracking online metrics
- Logging traces
- Lunch 🥗🌮☕
- User feedback
- Iterating on the solution
- Wrap up

Xebia | Data

Evaluating the LLM solution

High-level solution diagram



Xebia | Data

Different forms of evaluations

Offline:

- Eye-balling
- Smoke tests
- Using labeled data and predefined metrics
- Letting an LLM evaluate our outputs

Before we go live

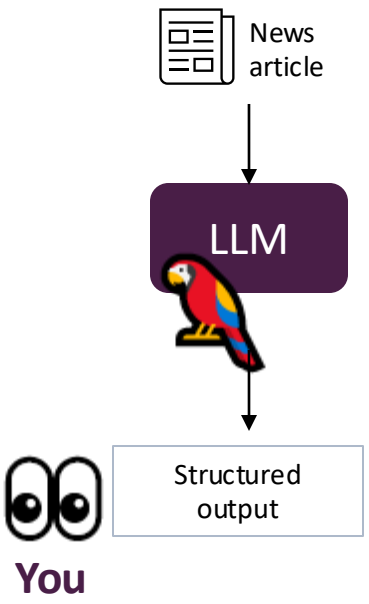
Online:

- Metrics collected from production
- Explicit user feedback
- A/B testing

When we are live

Offline evaluation for our use case

Eye-balling

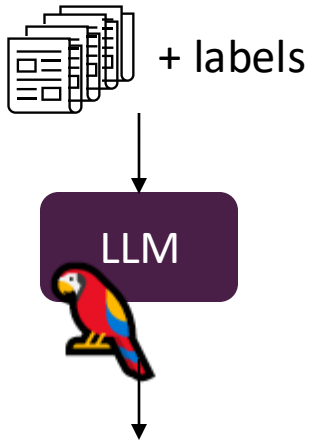


Smoke tests

```
assert summary_not_too_long
assert businesses_in_article
...
```

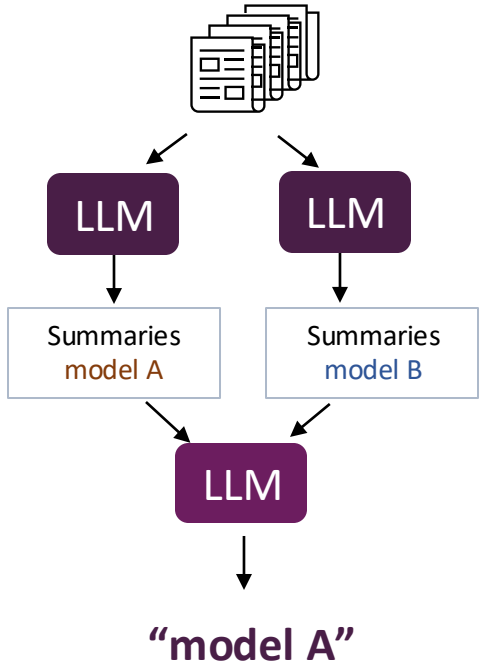
All tests passed!

Labeled data




```
{
  “summary_rouge”: 0.6,
  “business_recall”: 0.8,
  “title_accuracy”: 0.7,
}
```

(bonus) LLM as judge*



* Only use with caution! Good alignment with human judgement essential. See this blogpost

“You should track the correlation between model-based and human evaluation to decide how much you can rely on automatic evaluation”

“I sent my colleague Phillip the following spreadsheet every few days to grade for a different use-case”

C	E	F	G	H	I	J
model response	**model critque**	**model outcome**	**phillip critique**	**phillip outcome**	**phillip revised response**	**agreement**
{"calculations":[{"column":"duration_ms","op":"MAX"}],"filters":[{"column":"trace.parent_id","op":"does-not-exist","join_column":""}],"orders":[{"column":"duration_ms","op":"MAX","order":"descending"}],"limit":1,"time_range":7200}	The response is nearly correct, as it is looking for the slowest trace by using MAX(duration_ms) and ordering by duration_ms in descending order, which is appropriate for finding the 'slowest' trace. Additionally, filtering with trace.parent_id does-not-exist correctly identifies root spans. However, the query should be grouping by trace.trace_id to ensure that we identify distinct traces, not just the longest individual span. Without the correct grouping, the analysis does not guarantee that the result is a full trace, but merely the longest span. Also, specifying a limit of 1 is good as it will return the single slowest trace, as requested.	bad	The response is nearly correct, as looking for the slowest trace by u MAX(duration_ms) and ordering duration_ms in descending order which is appropriate for finding t 'slowest' trace. Additionally, filter with trace.parent_id does-not-ex correctly identifies root spans. However, the query should be grouping by trace.trace_id to act show the slowest trace. Without grouping, the query only shows the MAX(duration_ms) measurement over time, irrespective of which trace is responsible for that measurement.	bad	:""}],"orders":[{"column":"duration_ms","op":"MAX","order":"descending"}],"limit":1,"time_range":7200}	TRUE

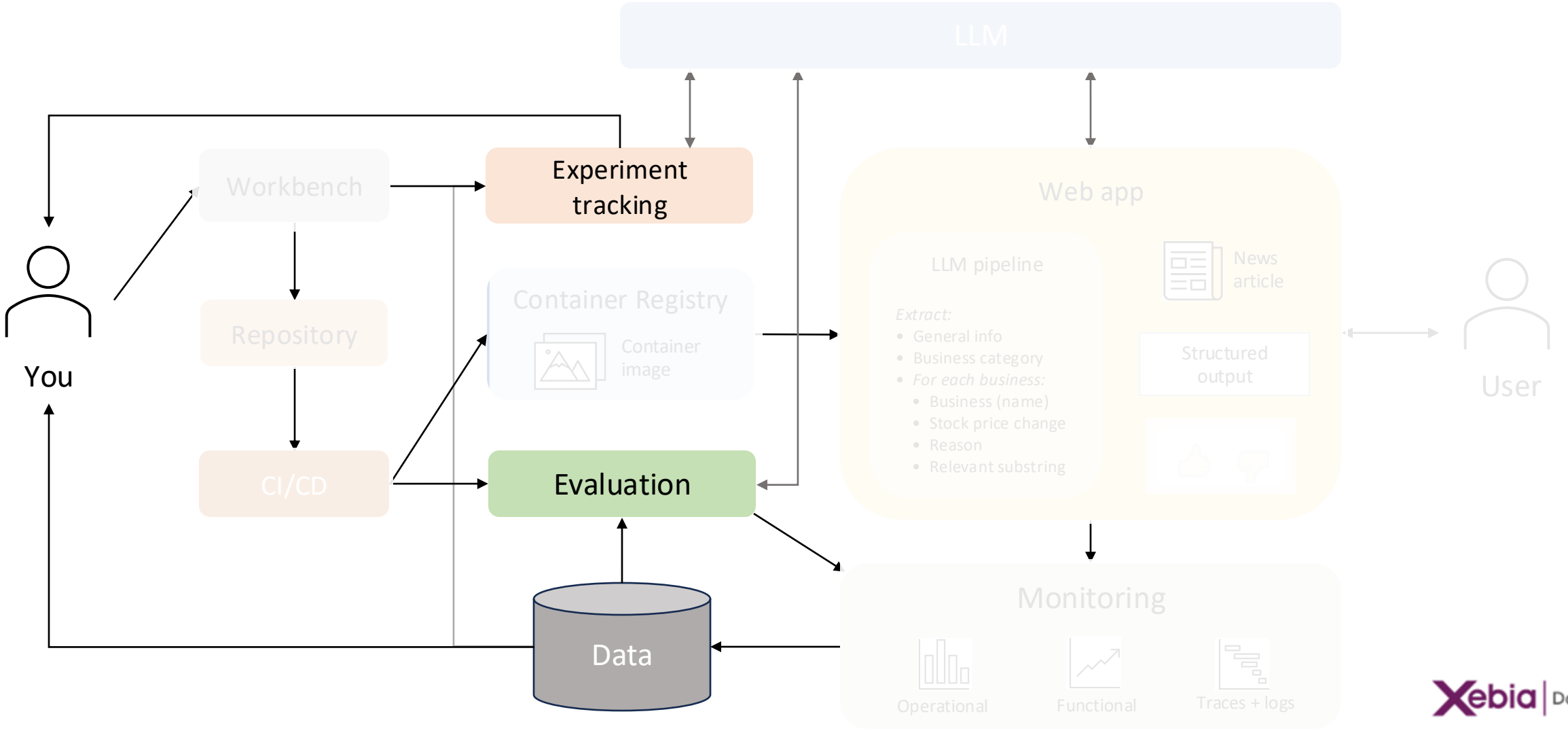
	A	B
1	Iteration	Agreement %
2	1	68.00%
3	2	82.61%
4	3	94.00%

* Only use LLM as a judge with caution! Good alignment with human judgement essential. See [this blogpost]

Xebia | Data

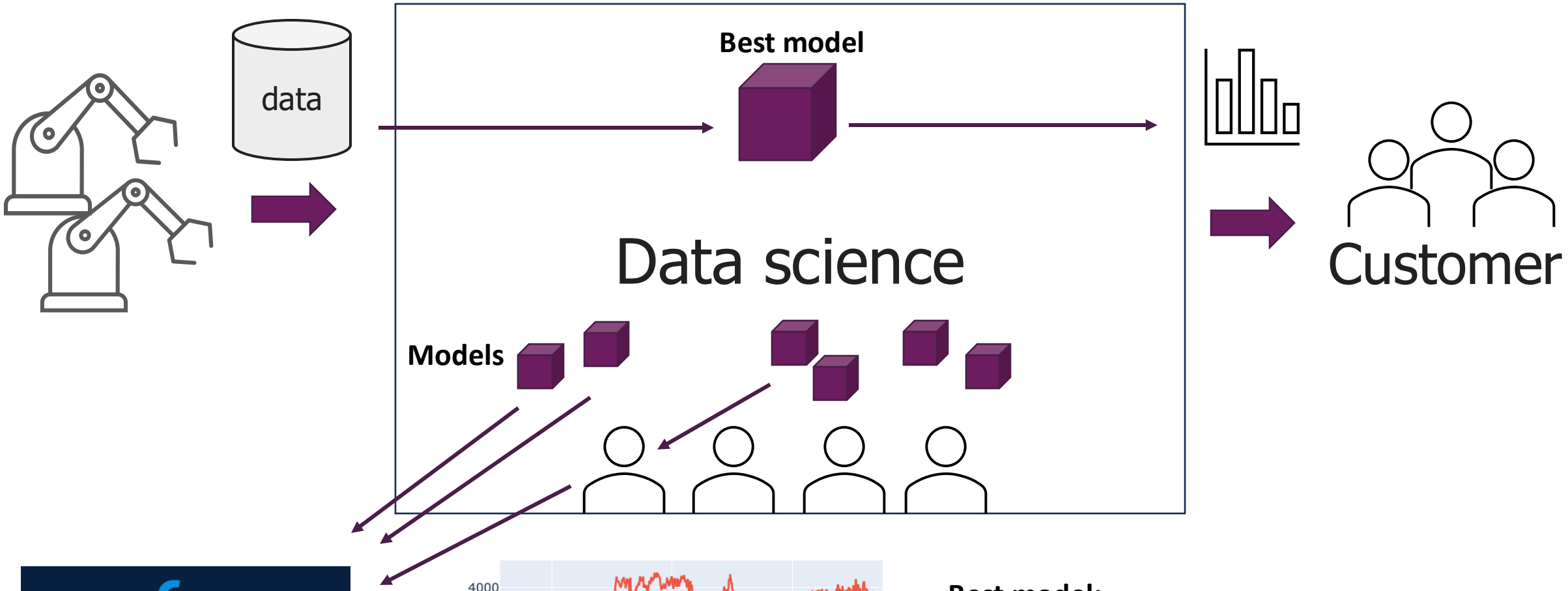
Experiment tracking

High-level solution diagram



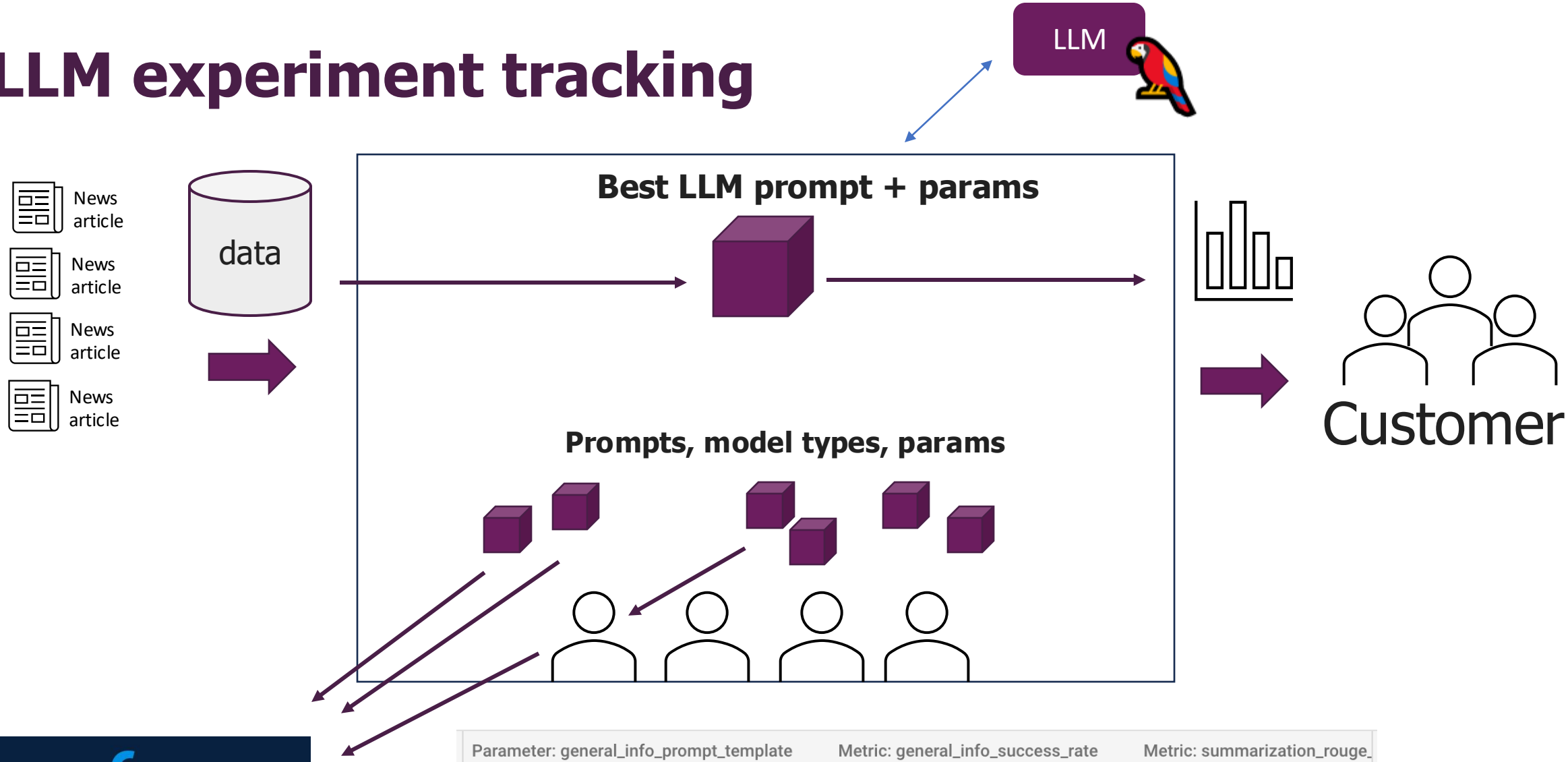
Xebia | Data

Recap ML experiment tracking



Best model:
R2 score = 0.9

LLM experiment tracking



Parameter: general_info_prompt_template	Metric: general_info_success_rate	Metric: summarization_rouge_
Extract structured information from the following article: {article}	0.9	0.2529504825

(or LLM-specific tools like LangFuse, LangSmith,...)

Exercise:
Evaluating the solution and tracking experiments

Let's add some assertions and allow for evaluations based on a labeled dataset

- Run through **exercise 5** in the `/exercises` folder

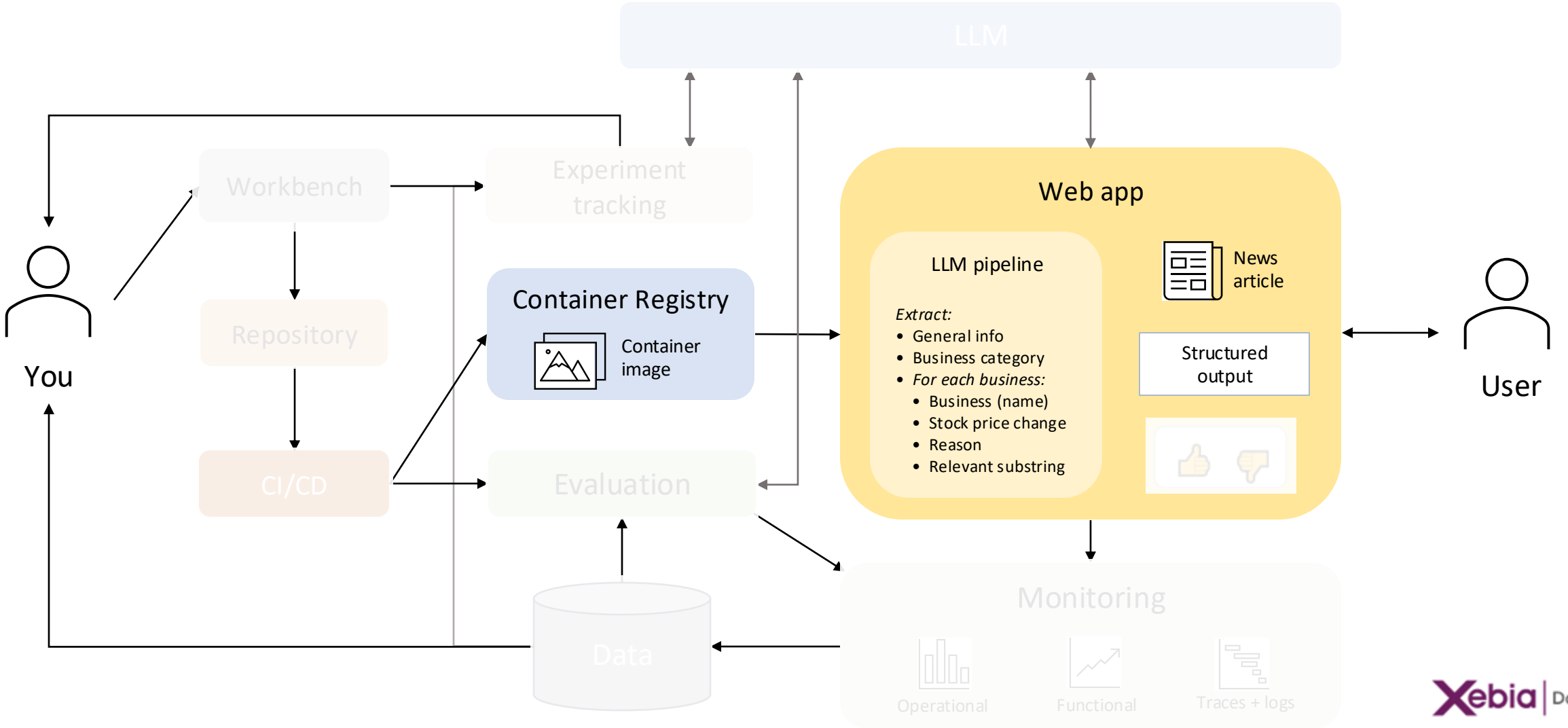
Now, we can try to improve those metrics, but it's important to keep track of our runs!

- Run through **exercise 6** in the `/exercises` folder

Break time!

Deployment

High-level solution diagram

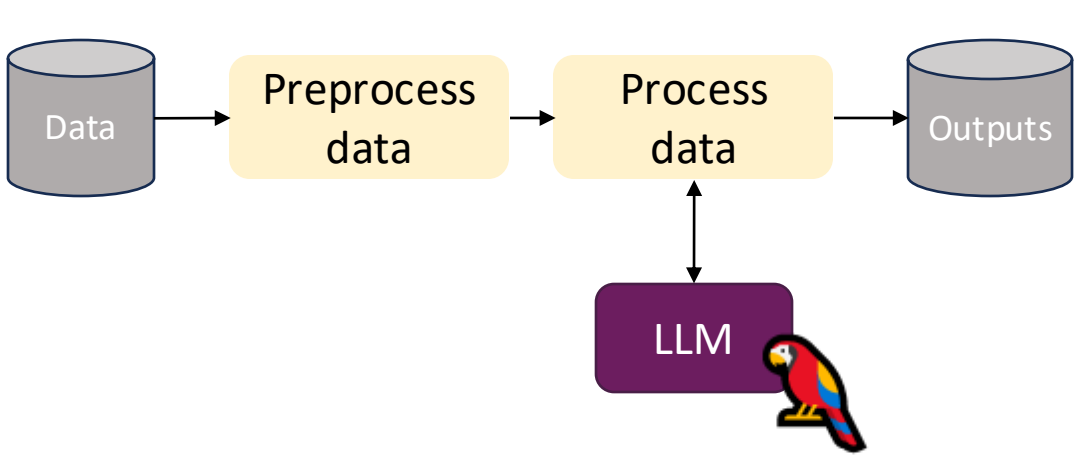


Xebia | Data

Serving patterns

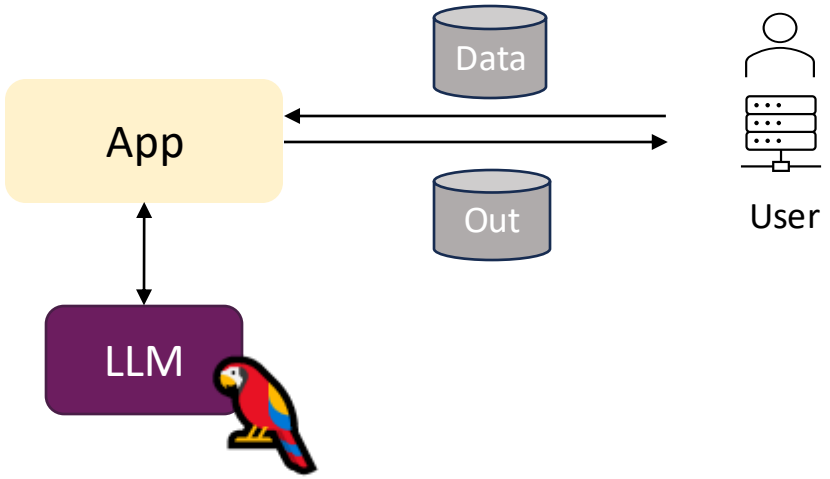
Batch:

- When predictions/generations can wait

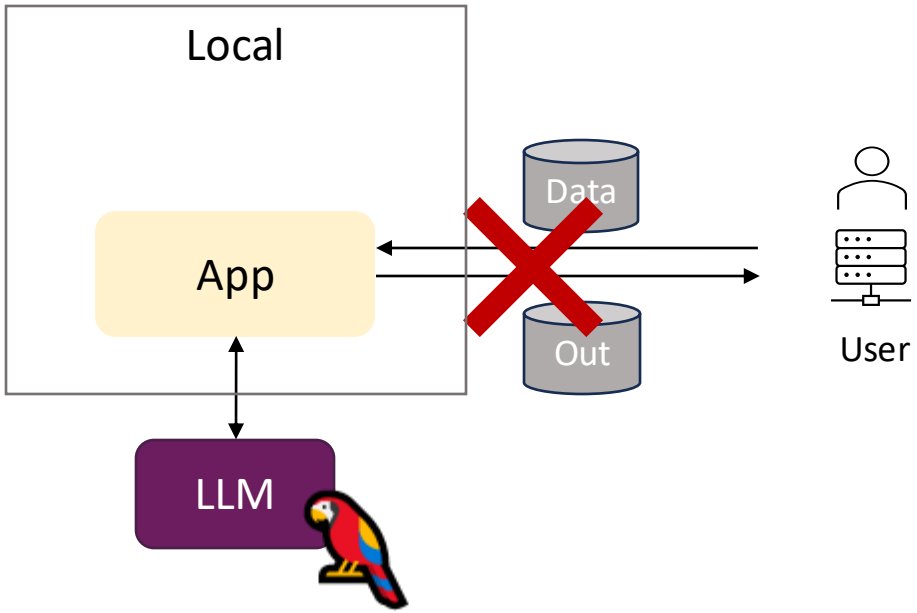


On-demand:

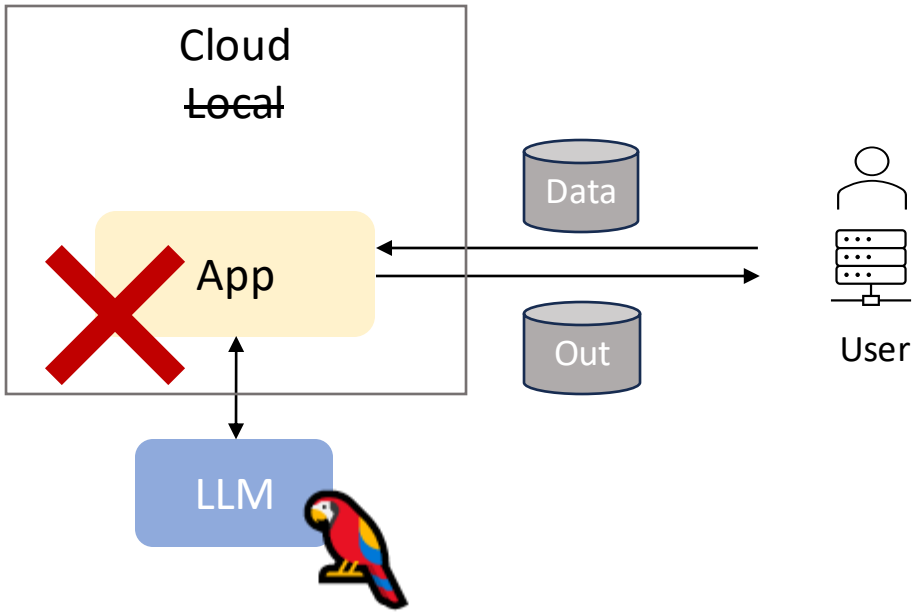
- When predictions/generation need to be served immediately



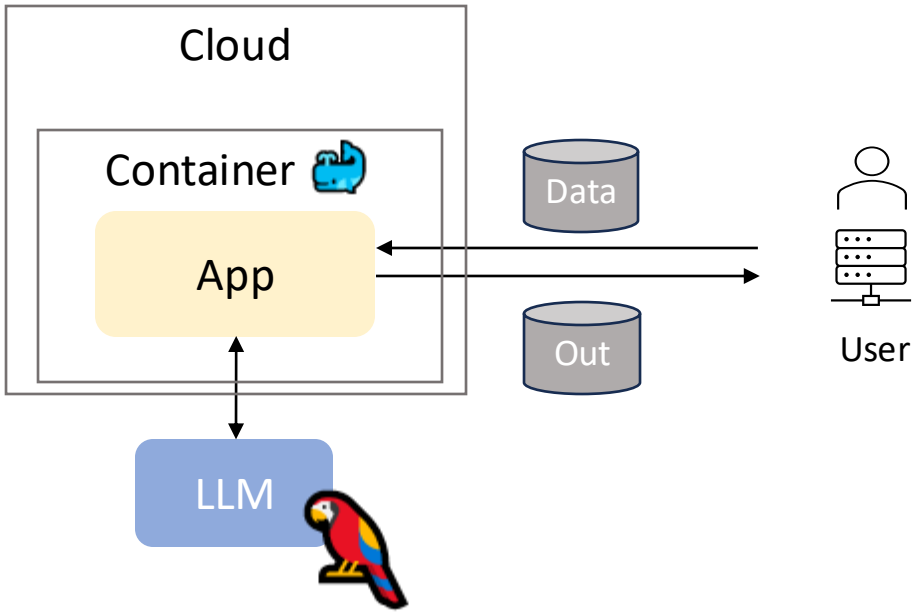
Users cannot access our app yet!



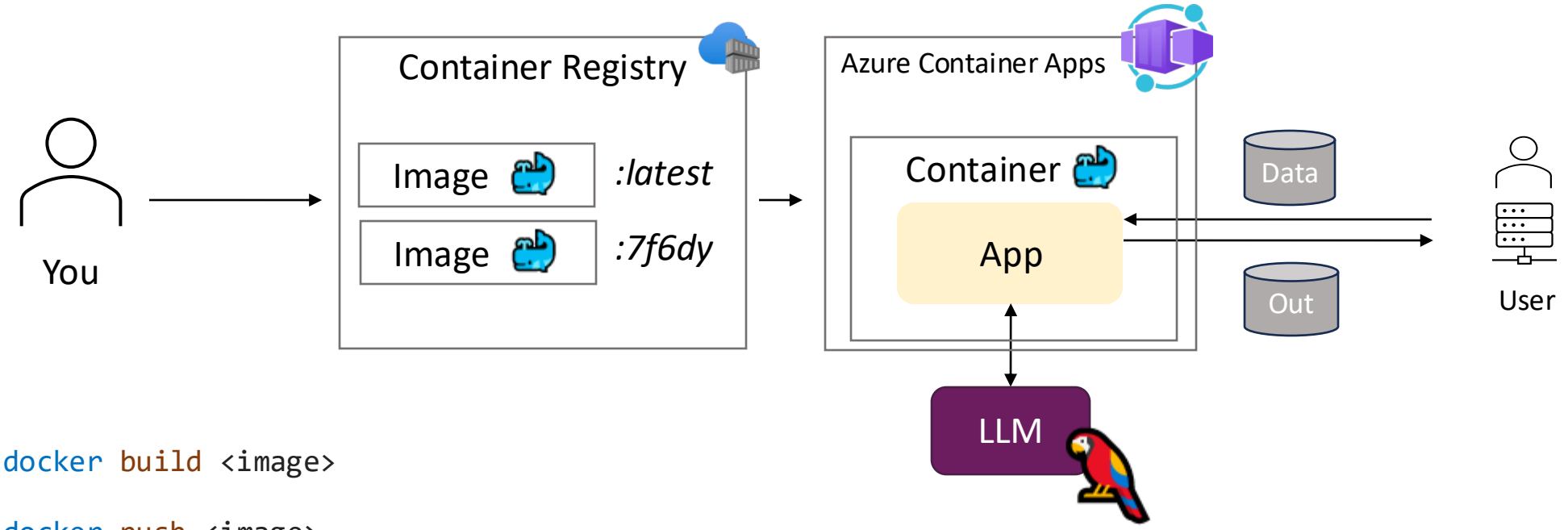
Let’s put it in the cloud!



Containerization needed before we can run our code in the cloud



We'll use Azure Container Registry and Azure Container Apps




```
$ docker build <image>

$ docker push <image>

$ az containerapp up --image <image>
```

Xebia | Data

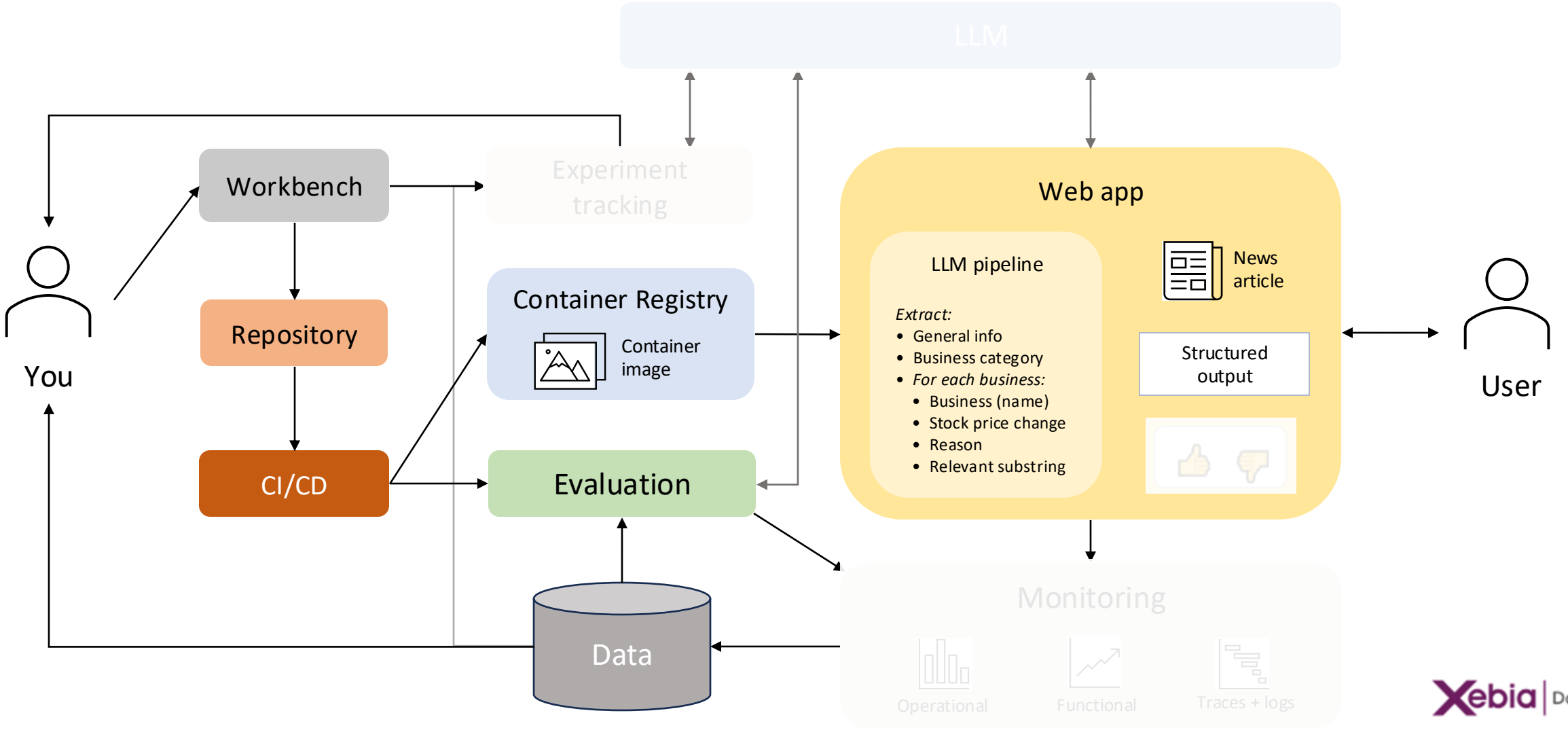
Exercise:
Deploying our solution

Let's containerize and deploy our app! 🚀

- Run through **exercises 7 + 8** in the `/exercises` folder

Automation with CI/CD

High-level solution diagram



Xebia | Data

What is CI/CD?

- CI/CD is a methodology of **frequently delivering** software **by automating** many stages of development
- Consists of two practices:
  - **Continuous Integration**: continuously building, testing and merging code into a single code base
  - **Continuous Deployment**: continuously deploying new releases/updates of your application to end users

Examples of CI steps

- Checking code quality (e.g. using pylint)
- Checking code formatting (e.g. using black)
- Running unit tests (e.g. using pytest)
- Building/testing a Docker image
- …

Examples of CD steps

- Building and pushing a Python package
- Building and pushing a Docker image
- Deploying an API to a (dev/test/prod) environment
- …

How to implement CI/CD – YAML!

```yaml
name: CI

on:
  pull_request:
    branches:
      - main

jobs:
  run-checks:
    steps:
      - name: ...

      - name: Run pytest
        run: pytest

  another-job:
    steps:
      ...

```

```yaml
name: CD

on:
  push:
    branches:
      - main

jobs:
  build-push-docker:
    runs-on: ubuntu-latest

    steps:
      - name: Login to Google Artifact Registry
        uses: ...

      - name: Build Docker image
        run: docker build ...

      - name: ...

      - name: Deploy to Cloud Run
        run: gcloud run deploy ...

```

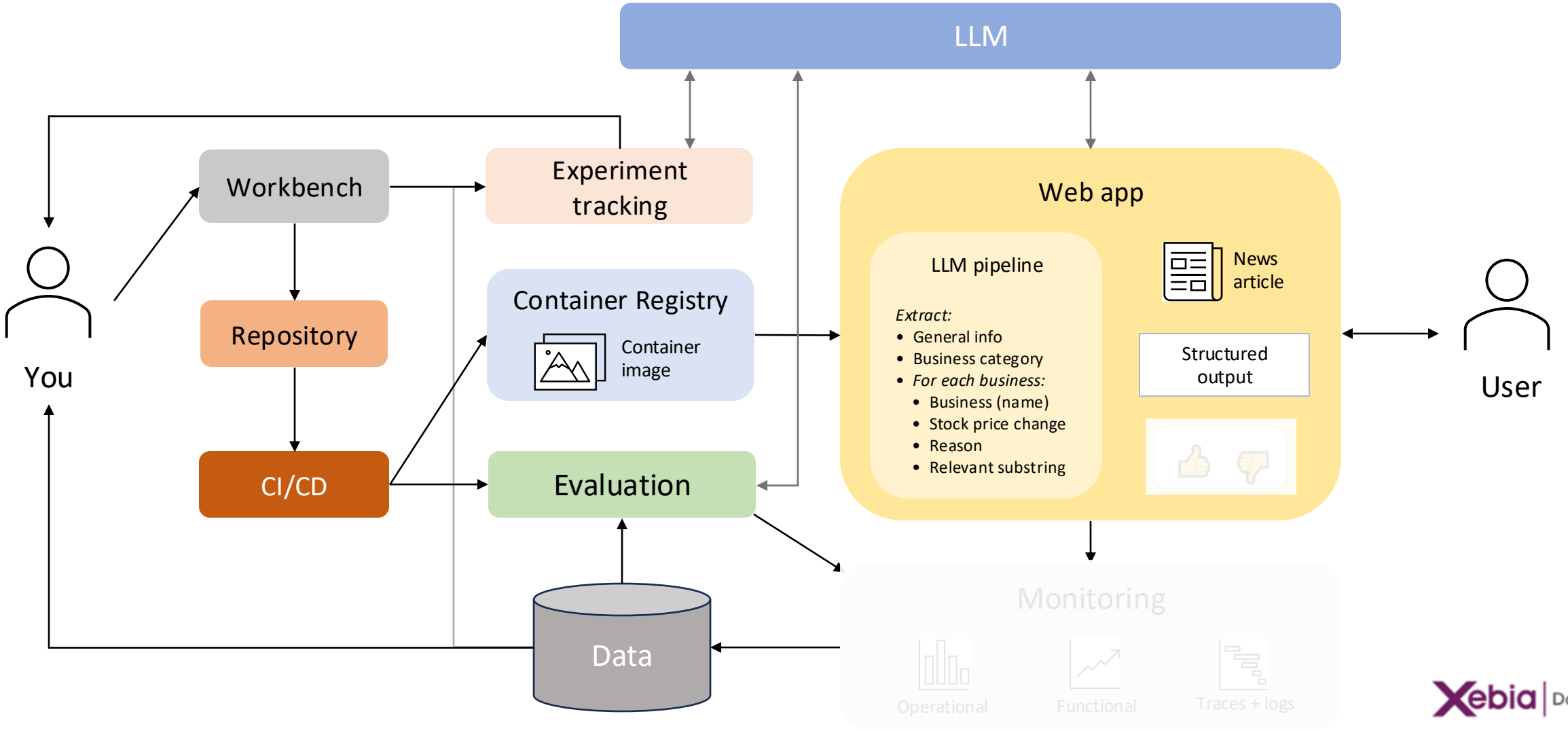
Xebia | Data

Exercise:
Automation with CI/CD

We're running too much still manually, let's automate (almost) everything!

- Run through **exercise 9** in the `/exercises` folder

High-level solution diagram



Xebia | Data

LLMOps on Azure

Day 2

Schedule

Day 1

- Intro MLOps and LLMOps
- Use case & solution design
- Building modularized LLM app
- Lunch 
- Evaluation and experiment tracking
- Containerization and deployment
- Automation with CI/CD

Day 2

- **Intro LLM system monitoring**
- **Tracking online metrics**
- **Logging traces**
- Lunch 
- User feedback
- Iterating on the solution
- Wrap up

Repo: https://github.com/godatadriven/academy-llmops-in-azure

Xebia | Data

Recap day 1

Differences MLOps and LLMOps

	MLOps	LLMOps
People	Data scientist (statistics, ML) Machine learning engineer Data engineer (structured data) Product owner ...	Data scientist (NLP) Machine learning engineer Data engineer **(unstructured data)** Product owner ***Prompt engineer* (?)** ...
Processes	Project templates (typical ML use cases) Experimentation (logging hyperparams and metrics) Pipelines (preprocessing and ML training) Deployment (model endpoint) Monitoring (typical ML metrics and infra) ...	Project templates (typical LLM use cases, RAG, etc.) Experimentation **(logging prompts and metrics)** Pipelines (preprocessing and LLM batch jobs) Deployment **(web app)** Monitoring (LLM metrics**, traces** and infra) ...
Technology	Scikit-Learn Tensorflow/Keras/PyTorch/Jax MLflow FastAPI ...	LangChain PydanticAI LLM-specific API* Streamlit LangSmith ...

*** Assuming we are not training our own LLM, but use a provider's LLM**

News Reader:
Extract structured info and predict stock price changes from news articles

- We work at a bank and are **interested in analyzing news articles for likely stock price changes**
- We have created a notebook with an **LLM-solution to extract relevant info from the articles**
- In the end, we want a **web app where users can upload their articles which are then automatically analyzed**

News article

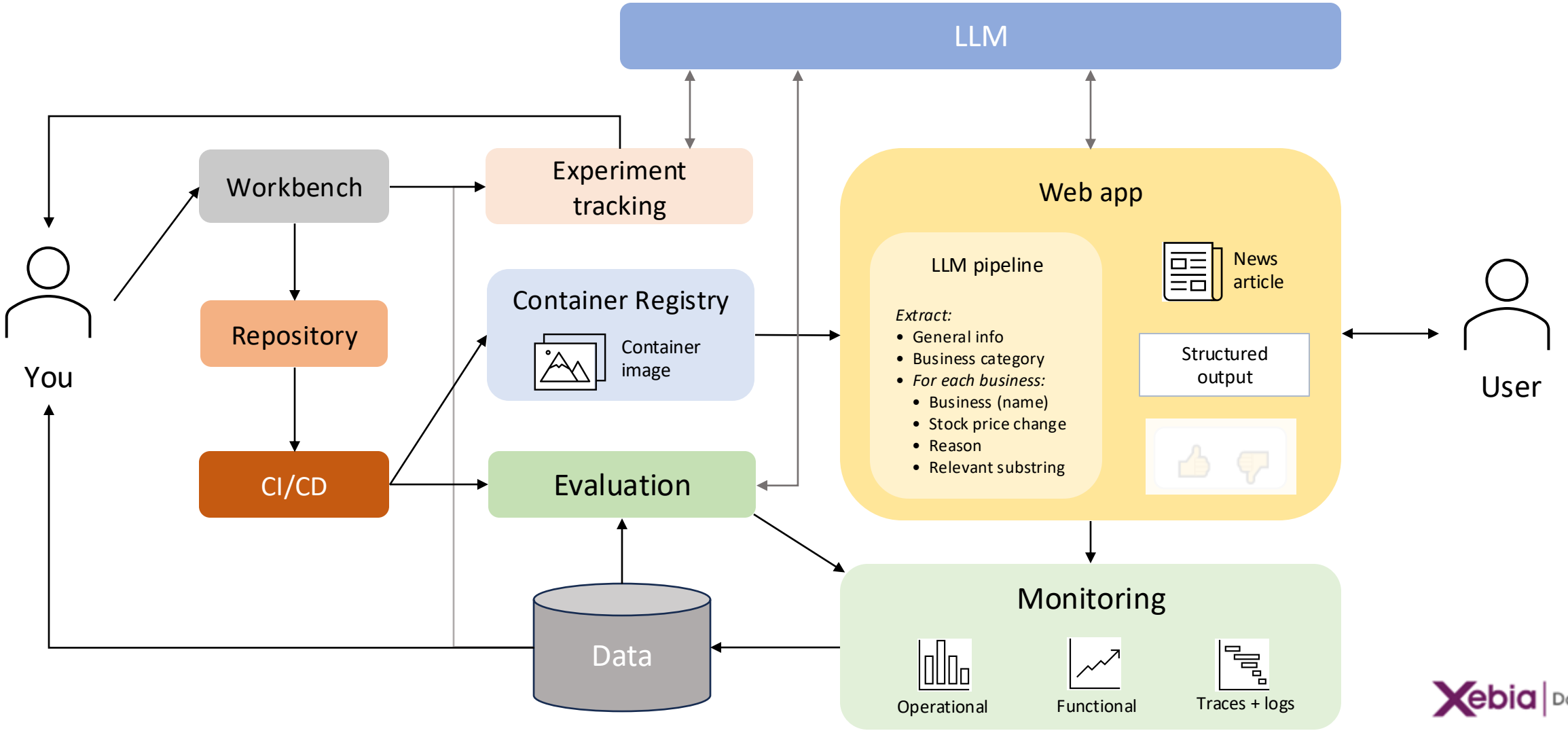
LLM

```
title: "Company XYZ hit rock bottom"
summary: ...
is_about_business: TRUE
business_info: [
  {
    business: "XYZ"
    stock_price_change: "decrease"
    reason: "They hit rock bottom"
    relevant_substring: ...
  },
  ...
]
```

Structured output

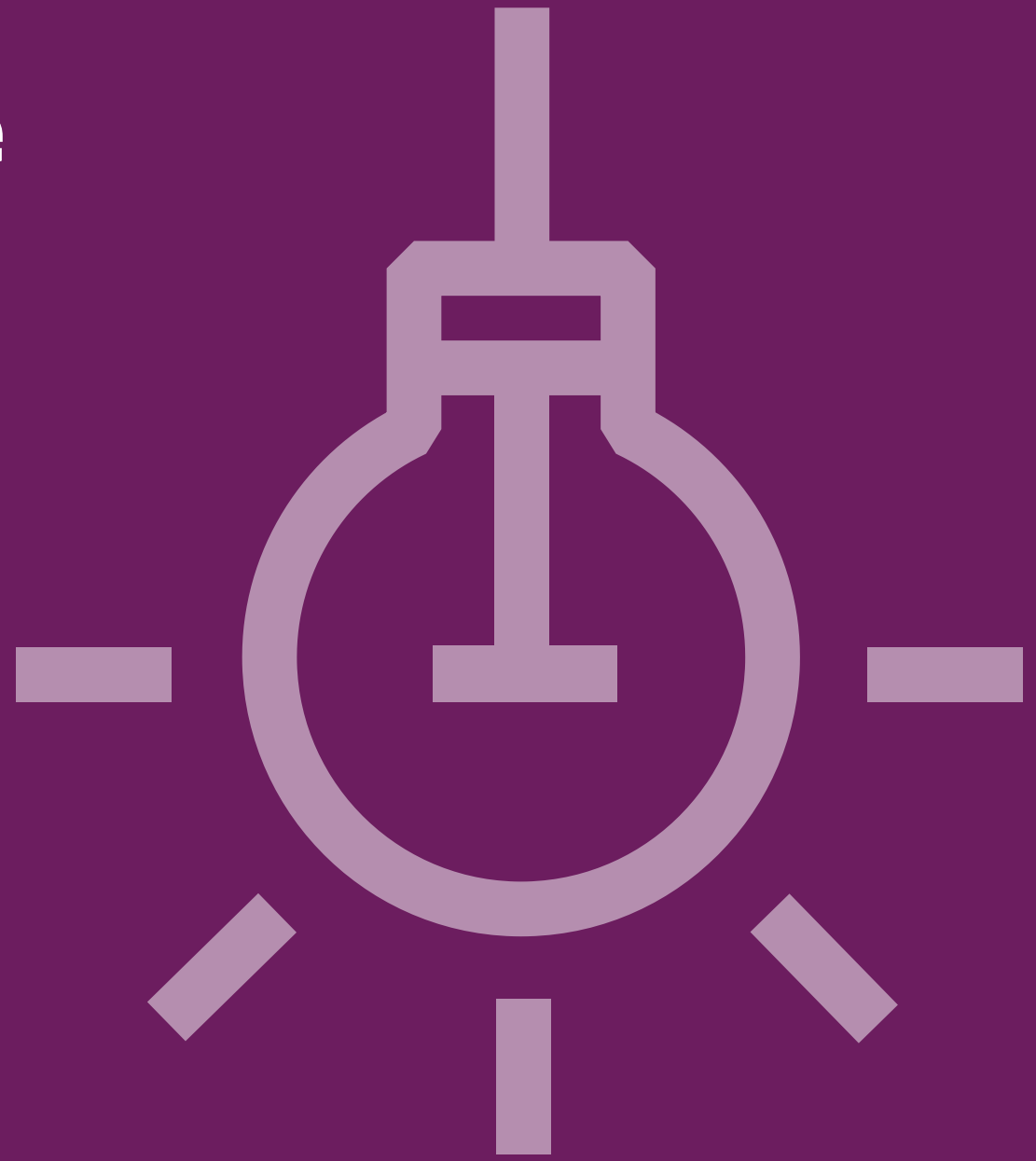
Xebia | Data

High-level solution diagram

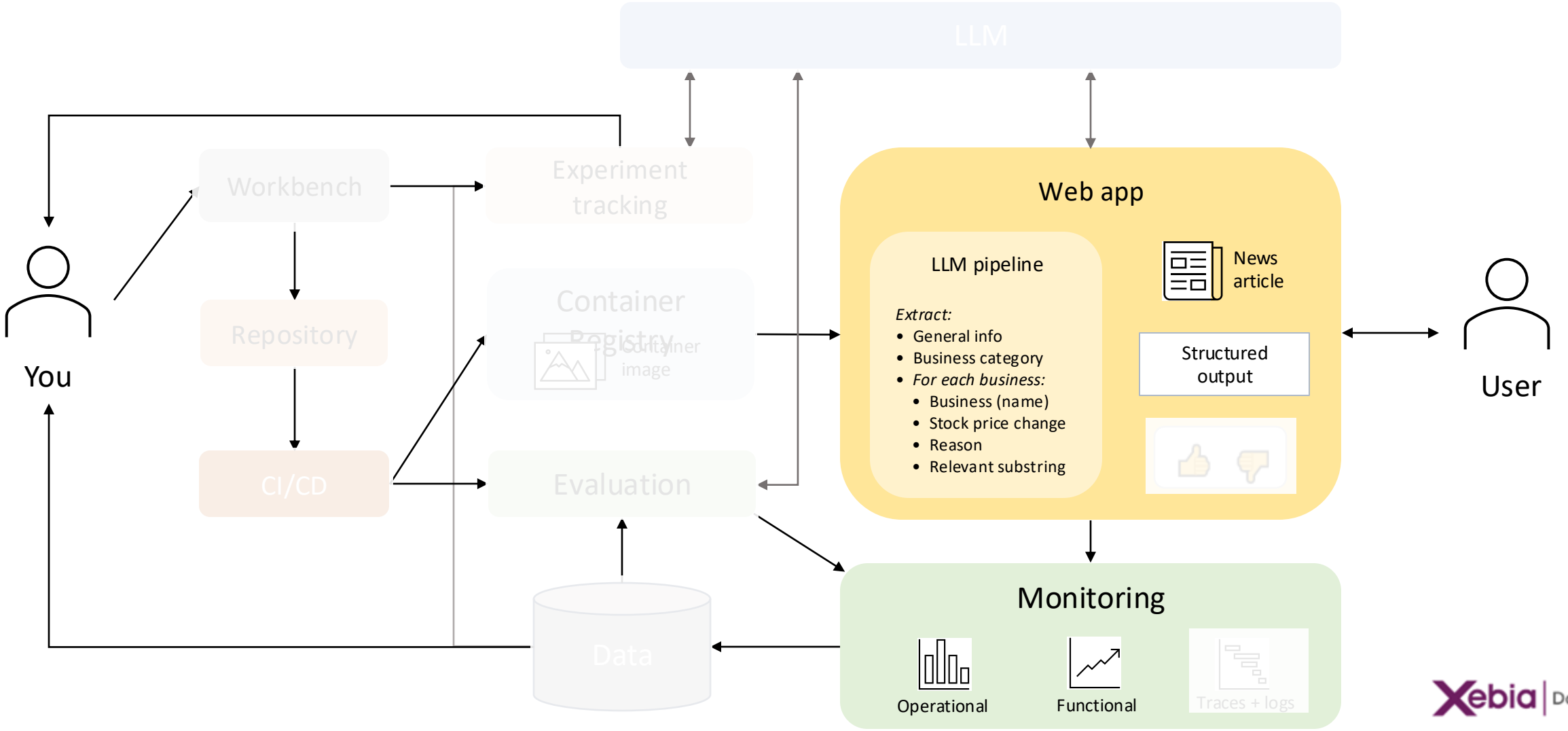


Xebia | Data

Monitoring performance



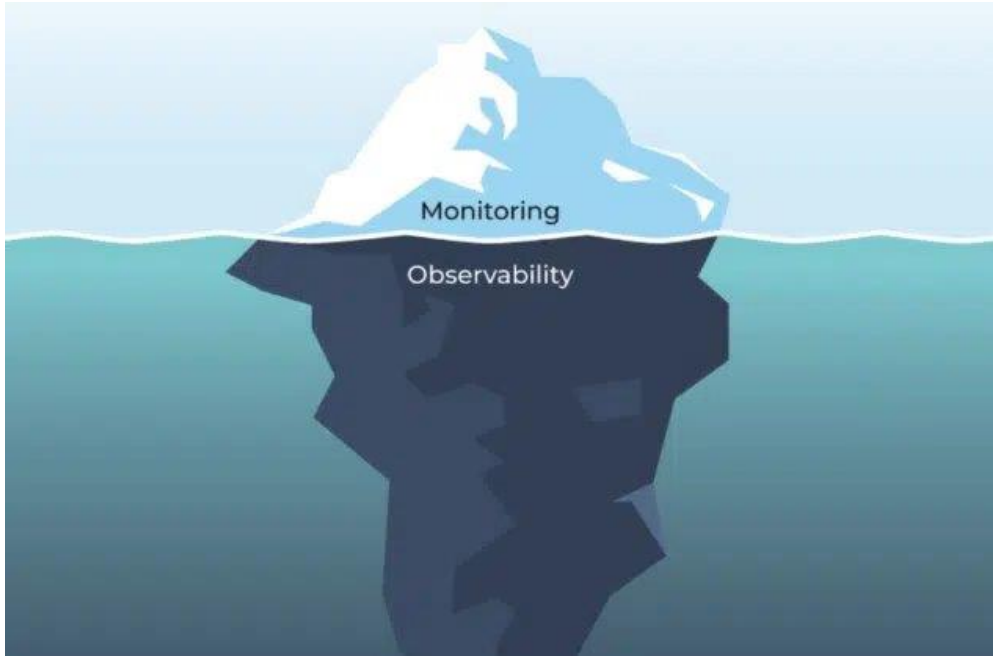
High-level solution diagram



Xebia | Data

Why monitor?

Goal: Quantify performance and flag potential issues in the system



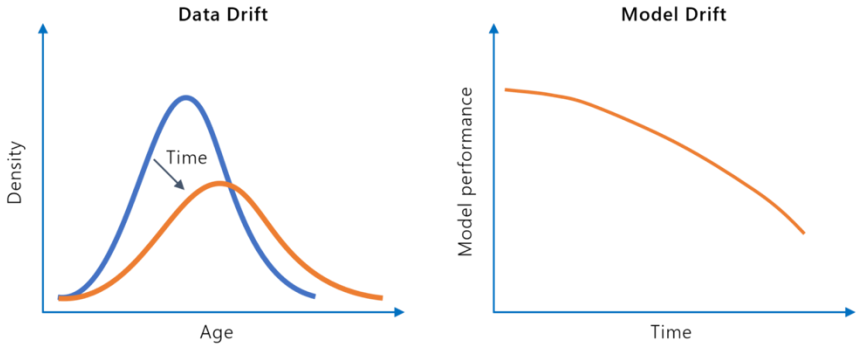
What to monitor?

Operational (infra-structure level)

- Number of requests from users
- Memory usage by the service
- Failed requests
- …

Functional (model/application level)

- Number of tokens to- and from LLM
- Number of requests to LLM
- How often output is saved by the user
- …



Xebia | Data

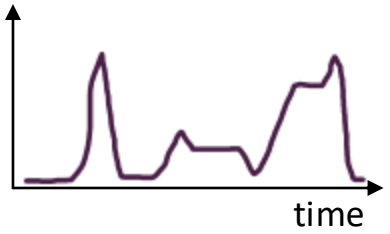
Logs, metrics and traces

Logs

- Give insight in what is going on inside application
- Can be structured (e.g. JSON) or natural language

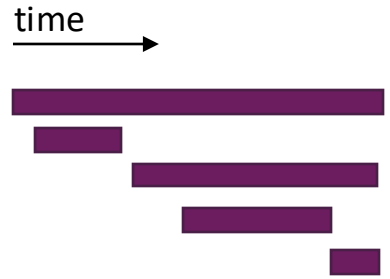
Metrics

- Flag potential issues (with alerts) for investigation
- Operational: built-in, e.g. provided by Azure Services
- Functional: collected from our own (structured) logs



Traces

- Follows the journey of a user/event through the entire stack
- Allows you to find bottlenecks or identify causes of issues

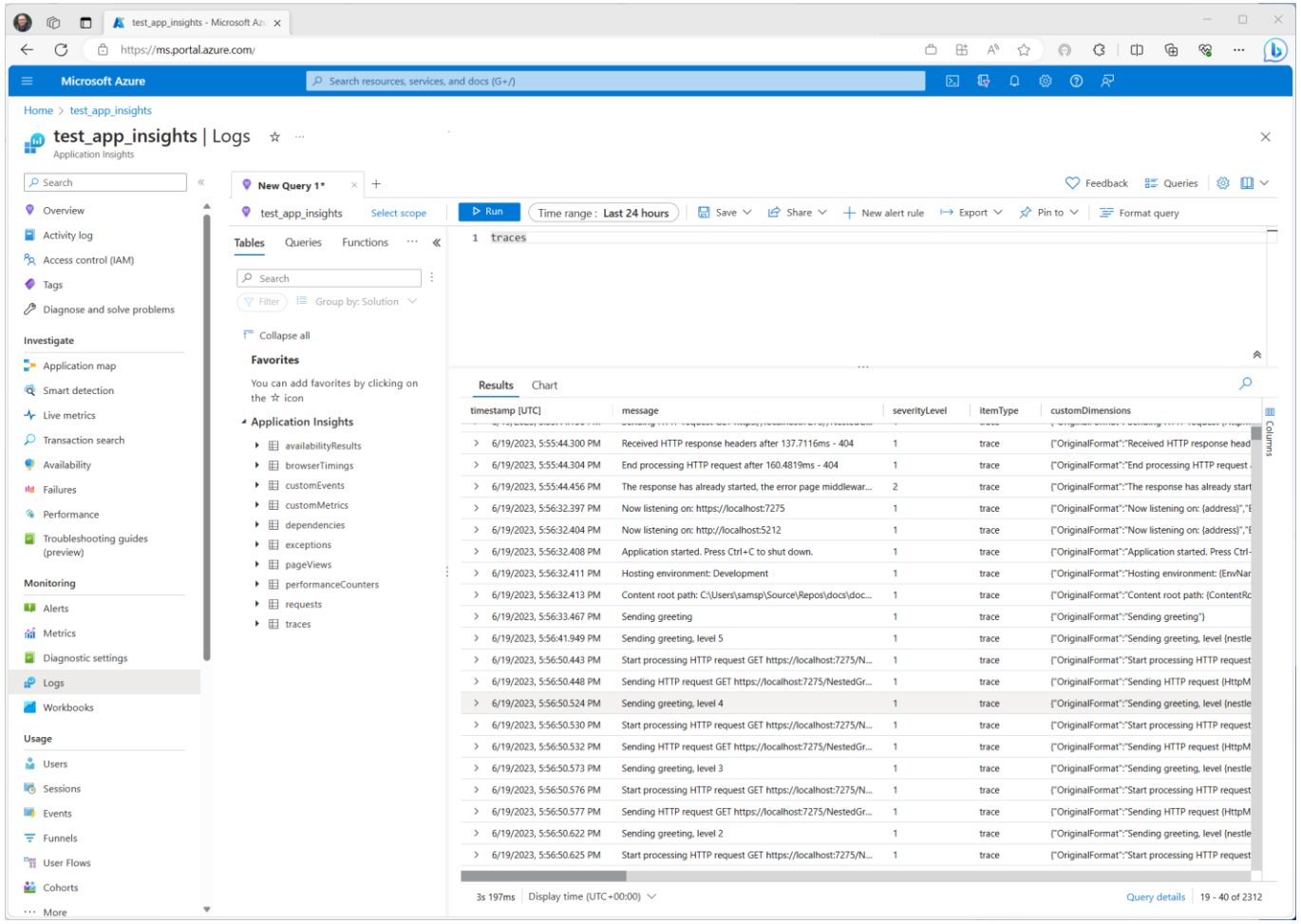


Xebia | Data

Structured logging

```
print(“This is not a log statement”)

>> This is not a log statement


logger = logging.get_logger()

logger.info(“This is a regular log statement”)

>> INFO: This is a regular log statement


logger = structlog.get_logger()

... # some config
logger.info(“Structured log”, key1=42, key2=“value”)

>> {“event”: “Structured log”, ”key1”: 42, “key2”: “value”}
```

Azure Logging



https://learn.microsoft.com/en-us/dotnet/core/diagnostics/observability-applicationinsights

Xebia | Data

Exercise:
Monitoring operational and functional metrics

Let's define our own custom log-based metrics and monitor our solution in the cloud!

- Run through **exercises 10 + 11** in the `/exercises` folder

Break time!

Intermezzo – Tools available

Tools

LangSmith

https://github.com/langchain-ai/langsmith-sdk

- Structured in projects
- Within a project:
  - **Traces**: end-to-end invocations of your application, with input prompts and output responses and metadata
  - **LLM calls**: isolate only the LLM calls so you can isolate prompts and generations
  - **Monitoring**: track different statistics over time: Trace count, LLM call count, success rates, etc.
  - Can jump into any **LLM call** in a playground, changing the settings and testing the completions
  - Allows for **filtering** on attributes, e.g. feedback
  - Allows you to store and update **datasets** for testing and evaluation

Langfuse

https://github.com/langfuse/langfuse

Develop

- **Observability:** Instrument your app and start ingesting traces to Langfuse (Quickstart, Integrations Tracing)
- **Langfuse UI:** Inspect and debug complex logs (Demo, Tracing)
- **Prompt Management:** Manage, version and deploy prompts from within Langfuse (Prompt Management)
- **Prompt Engineering:** Test and iterate on your prompts with the LLM Playground

Monitor

- **Analytics:** Track metrics (cost, latency, quality) and gain insights from dashboards & data exports (Analytics)
- **Evals:** Collect and calculate scores for your LLM completions (Scores & Evaluations)
  - Run model-based evaluations (Model-based evaluations) within Langfuse
  - Collect user feedback (User Feedback)
  - Manually score observations in Langfuse (Manual Scores)

Test

- **Experiments:** Track and test app behaviour before deploying a new version
  - Datasets let you test expected in and output pairs and benchmark performance before deploying (Datasets)
  - Track versions and releases in your application (Experimentation, Prompt Management)

Phoenix

https://github.com/Arize-ai/phoenix

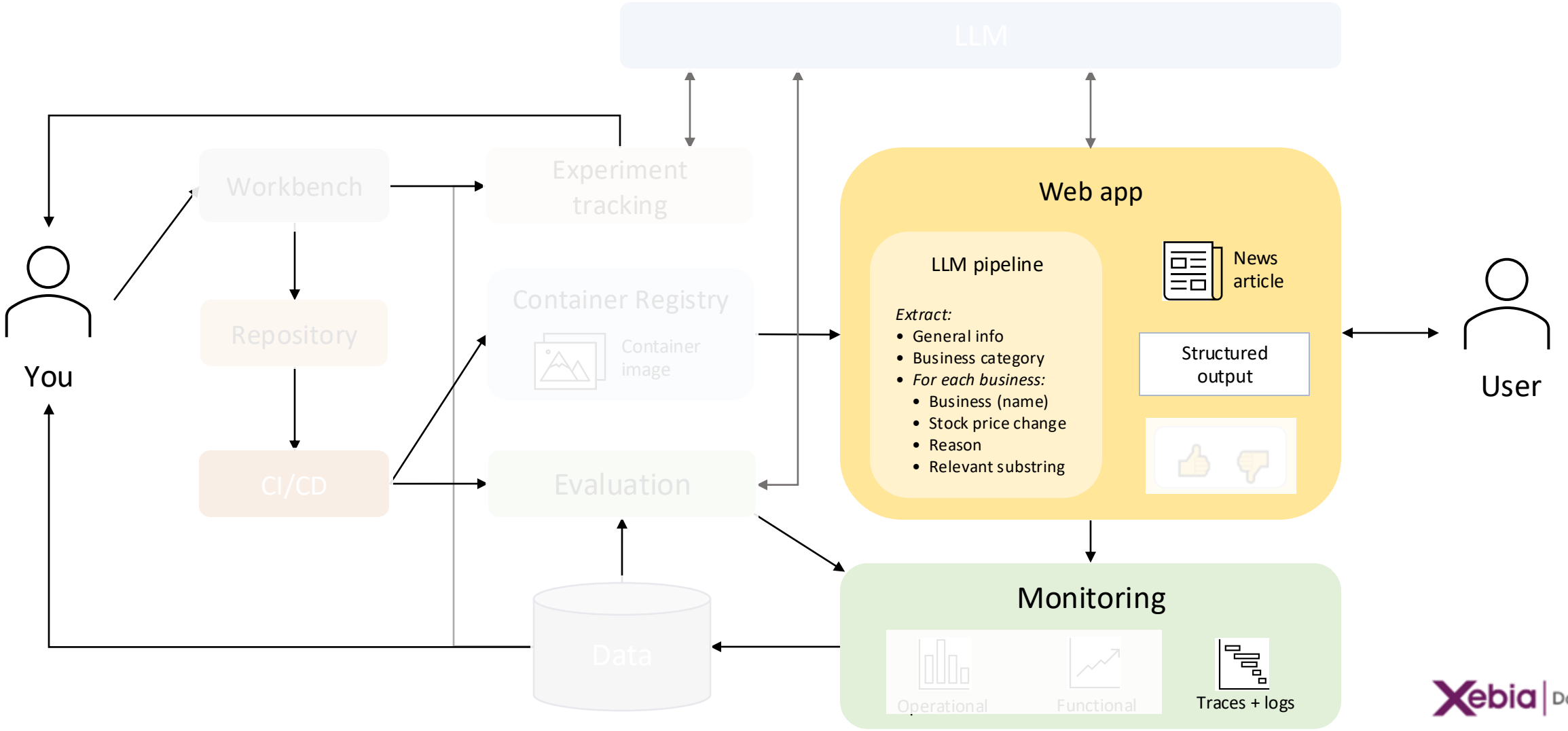
- ***Tracing*** - Trace your LLM application's runtime using OpenTelemetry-based instrumentation.
- ***Evaluation*** - Leverage LLMs to benchmark your application's performance using response and retrieval evals.
- ***Datasets*** - Create versioned datasets of examples for experimentation, evaluation, and fine-tuning.
- ***Experiments*** - Track and evaluate changes to prompts, LLMs, and retrieval.
- ***Inference Analysis*** - Visualize inferences and embeddings using dimensionality reduction and clustering to identify drift and performance degradation.

Common features

- **Monitoring**: tracking LLM calls, tokens, etc.
- **Tracing**: insight in end-to-end flow of LLM system
- **Logging**/debugging single LLM calls: to isolate good or bad examples
- **Dataset management**: quickly add new labeled samples (based on logs)
- **Prompt management:** store and version prompts
- **Evaluations:** run LLM on dataset to collect predefined metrics
- **Experimentation:** keep track of parameters, prompt, models used
- **Iteration:** allow for quick iteration with playground and log insights

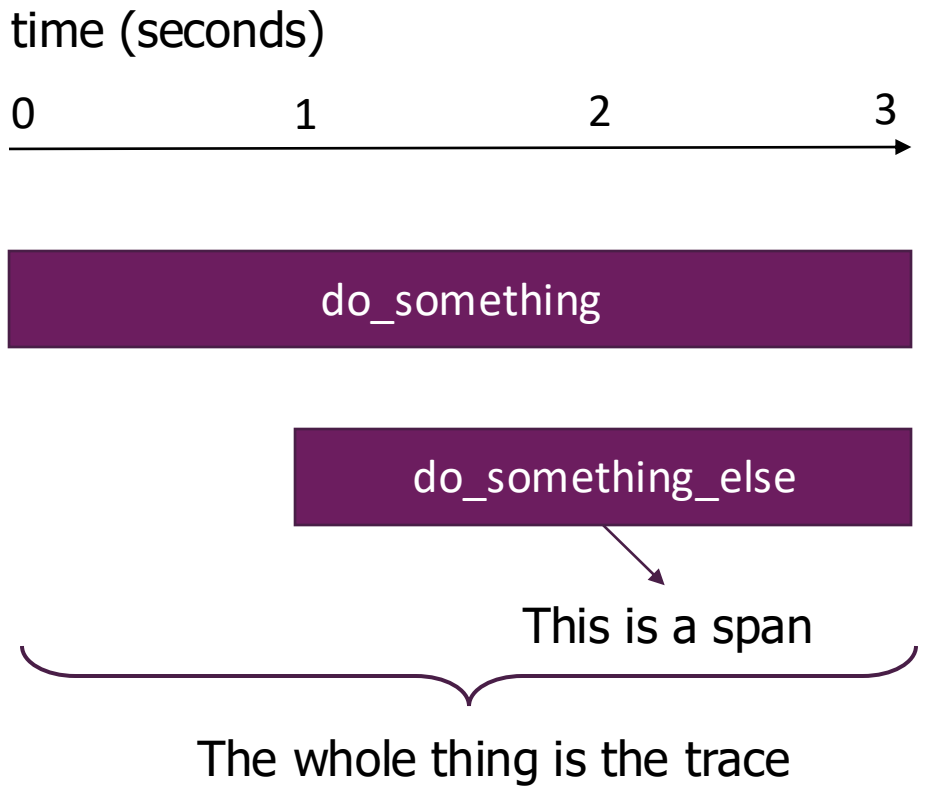
Tracing

High-level solution diagram



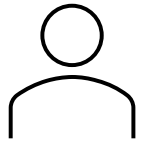
What is a trace?

```python
@tracer.start_as_current_span("do_something_else")
def do_something_else():
    time.sleep(2)

@tracer.start_as_current_span("do_something")
def do_something():
    time.sleep(1)
    do_something_else()

do_something()
```

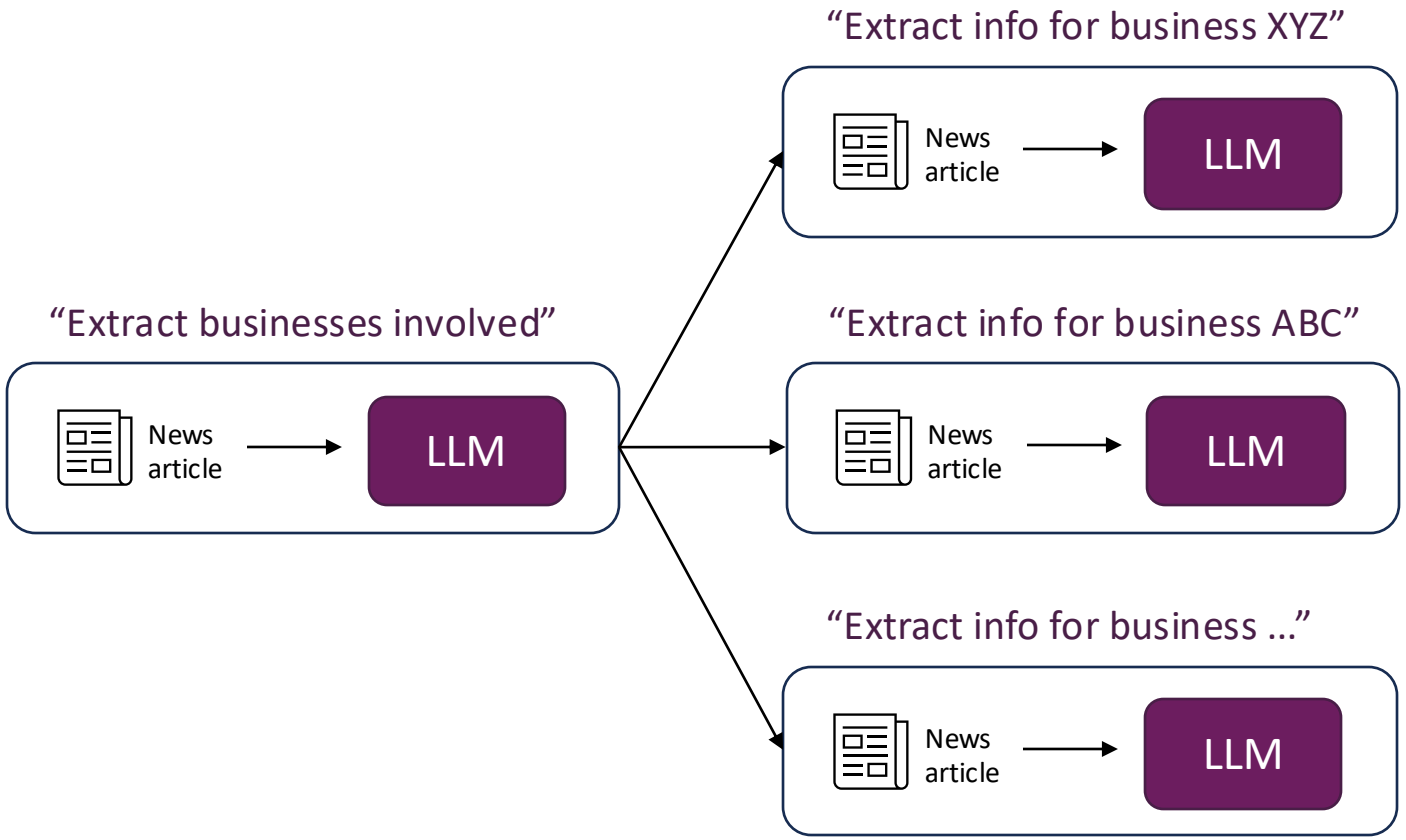


Why tracing?

*"This output is bad!
XYZ is not a business"*



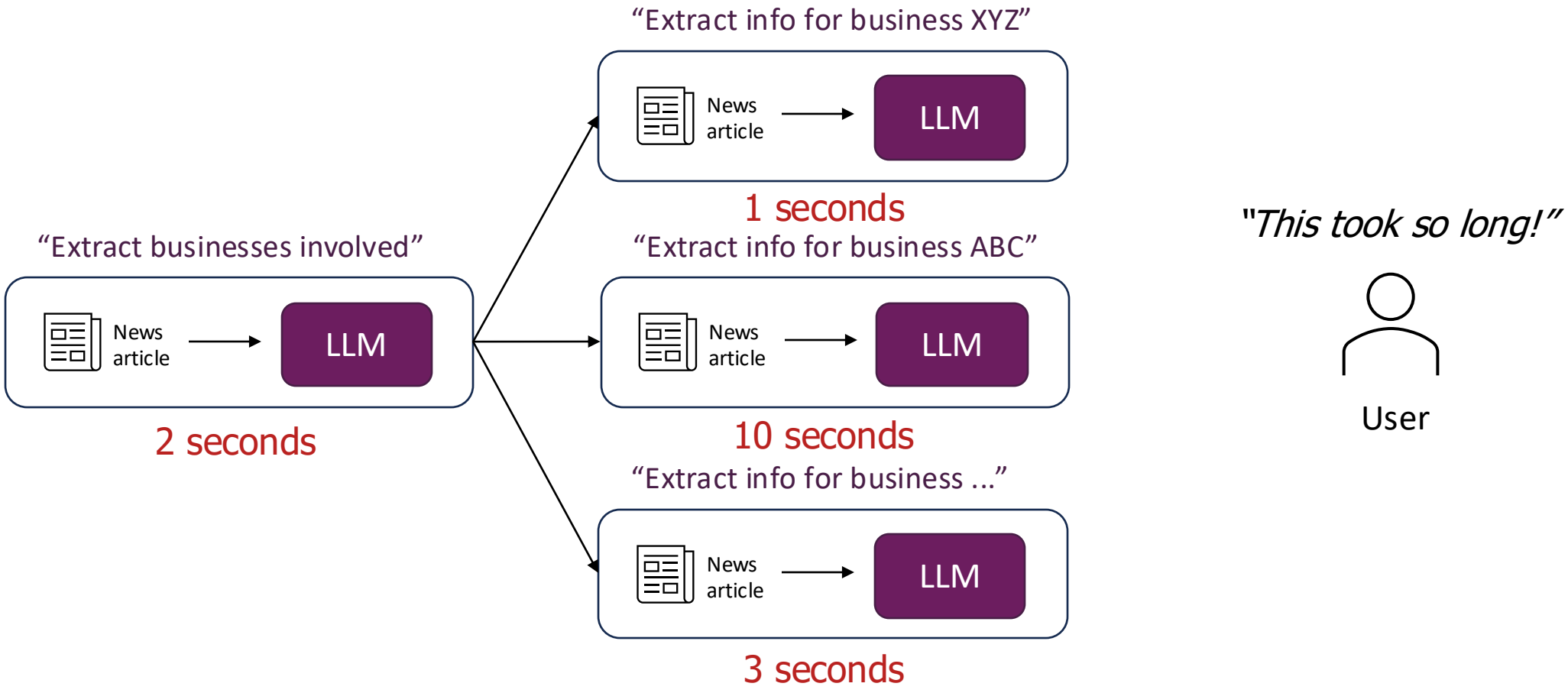
User



💡 We need insight into the entire trace to investigate **where mistakes originate from**

Xebia | Data

Why tracing?



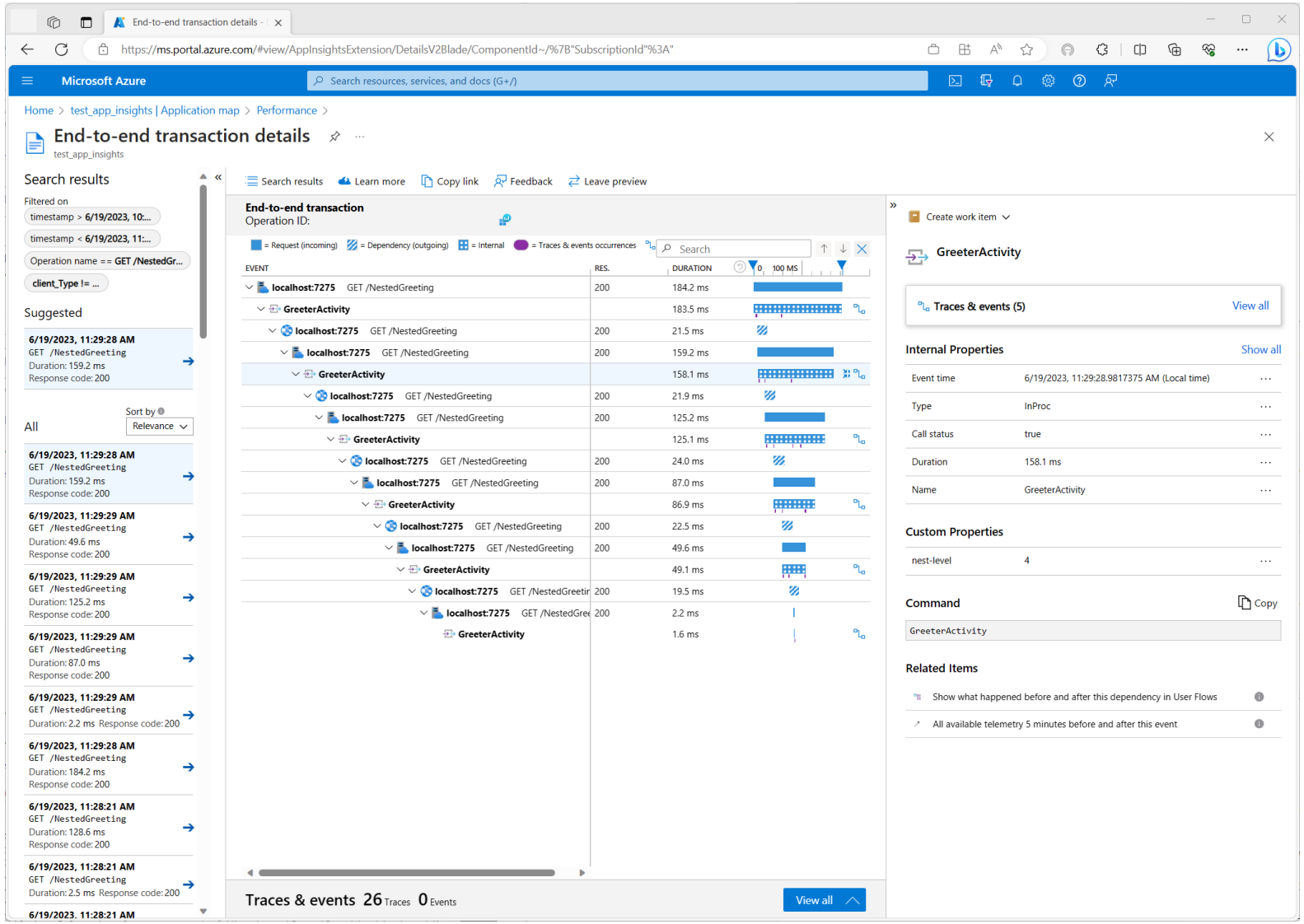
💡 We need insight into the entire trace to investigate **where the bottlenecks are.**

Xebia | Data

OpenTelemetry

- Open-source observability framework
- Common framework for tracing
- Integrates nicely with other tools, e.g. Phoenix, Azure Monitor

Tracing



https://learn.microsoft.com/en-us/dotnet/core/diagnostics/observability-applicationinsights

Xebia | Data

Exercise:
Logging traces to the cloud

Let's extend our monitoring solution by logging traces for the steps in our LLM pipeline

- Run through **exercise 12** in the `/exercises` folder

Lunch break!

Schedule

Day 1

- Intro MLOps and LLMOps
- Use case & solution design
- Building modularized LLM app
- Lunch 
- Evaluation and experiment tracking
- Containerization and deployment
- Automation with CI/CD

Day 2

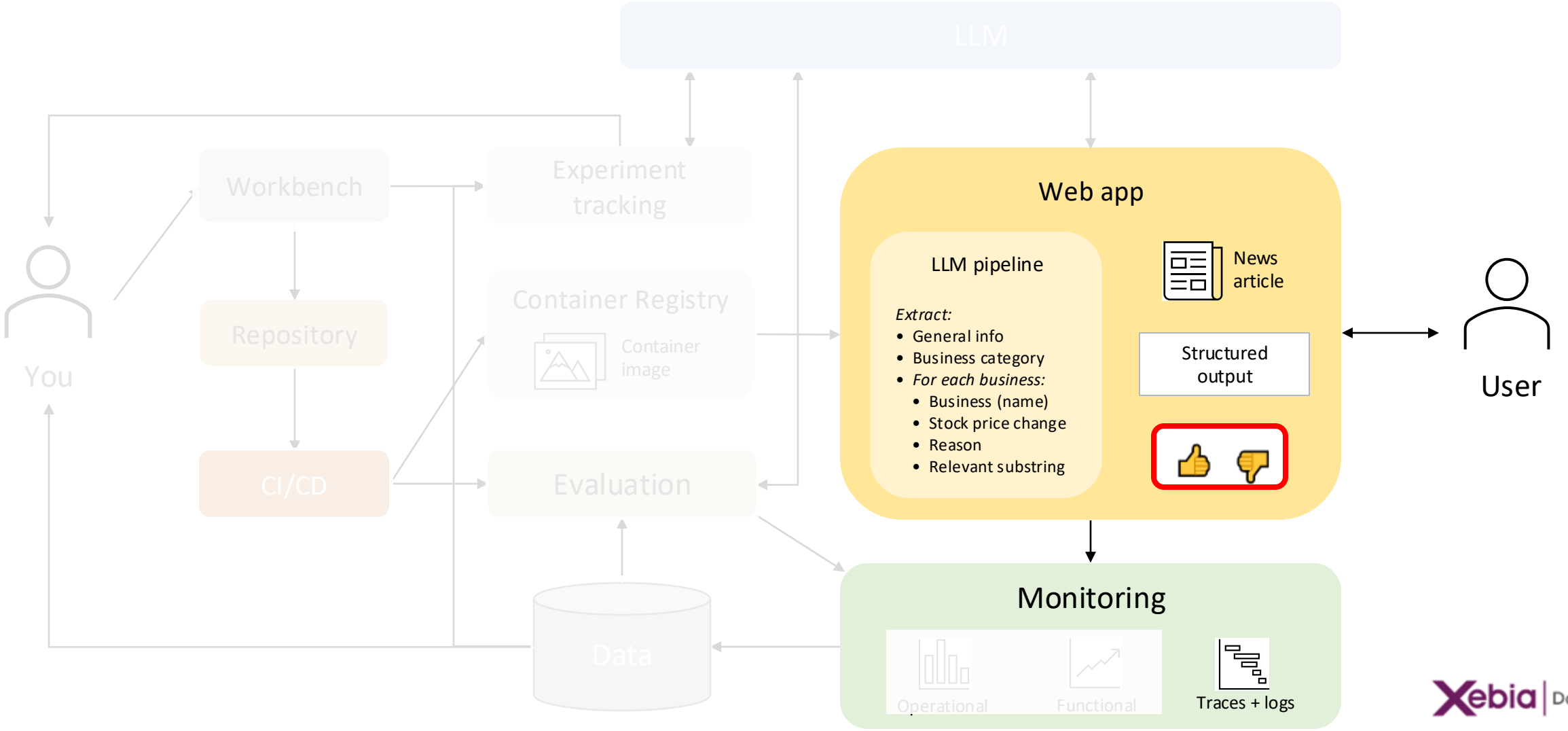
- Intro LLM system monitoring
- Tracking online metrics
- Logging traces
- Lunch 
- User feedback
- Iterating on the solution
- Wrap up

Repo: https://github.com/godatadriven/academy-llmops-in-azure

Xebia | Data

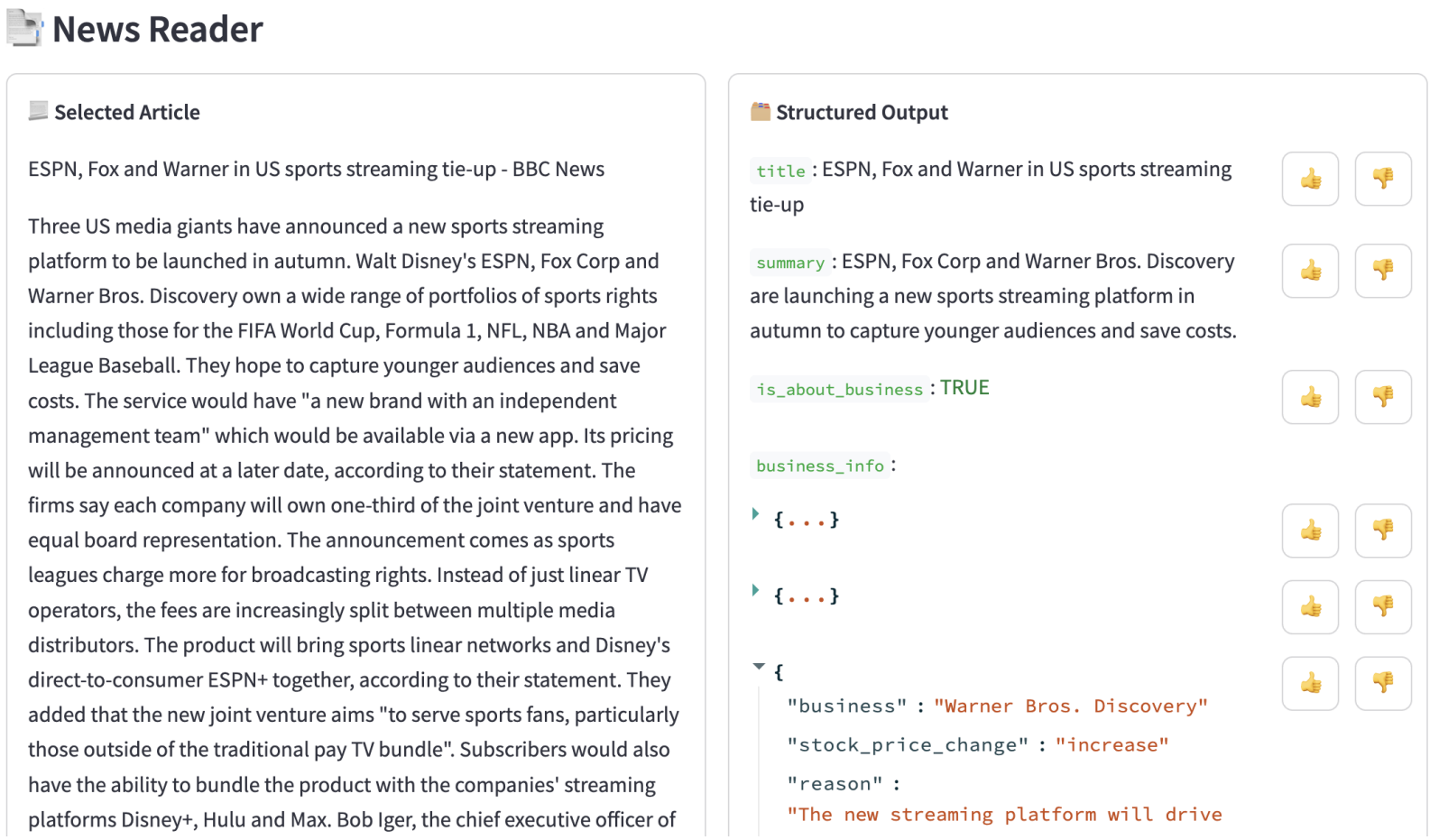
Collecting user feedback

High-level solution diagram

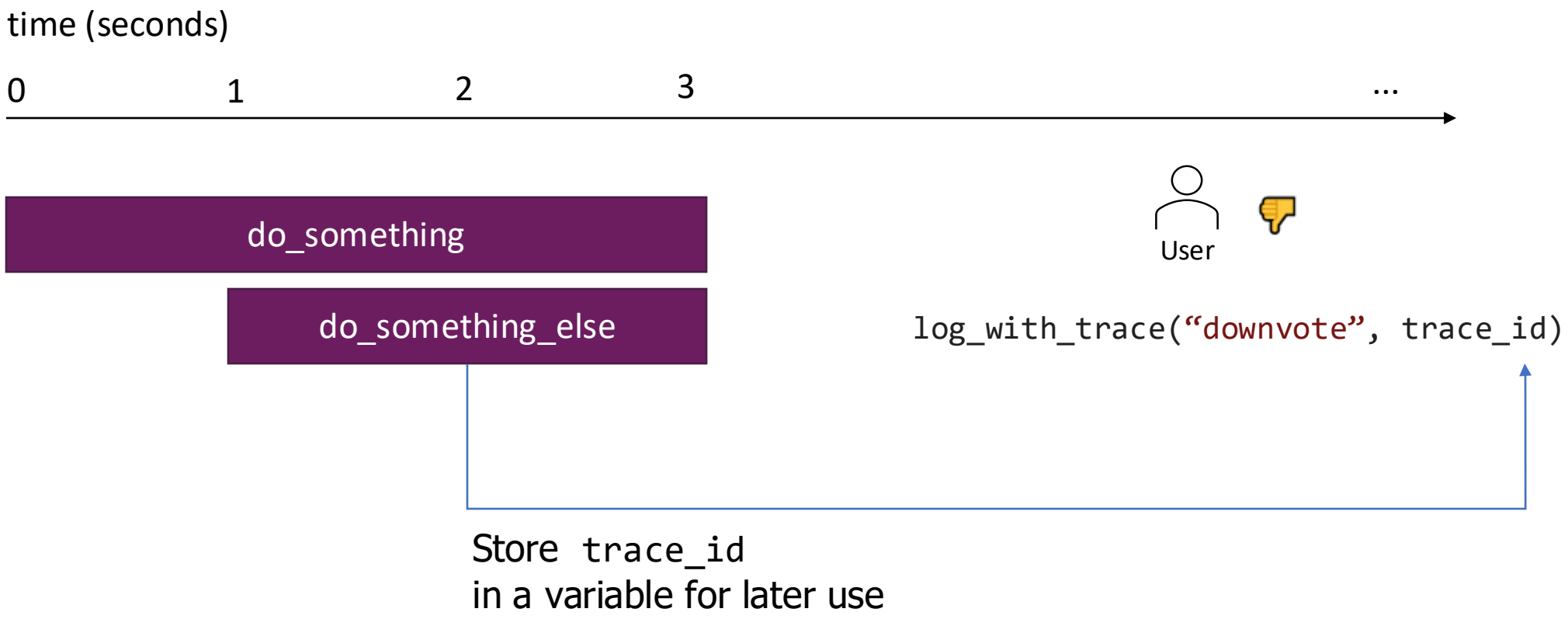


Xebia | Data

User feedback provides an invaluable source of information. How to collect it?



User feedback provides an invaluable source of information. How to associate it to a trace?



Xebia | Data

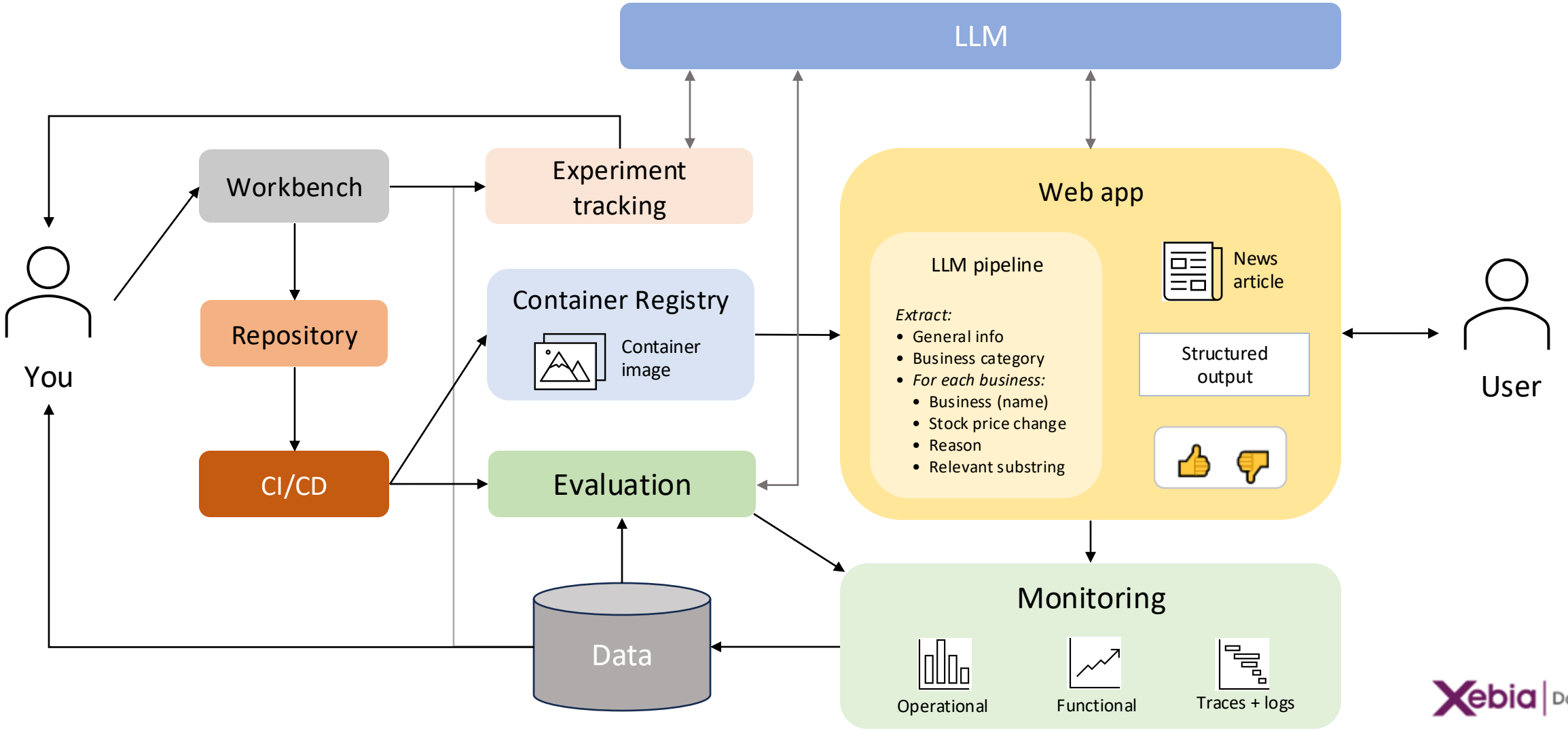
Exercise:
Collect user feedback for LLM outputs

Let's extend the app with buttons to collect and log user feedback to the cloud!

- Run through **exercise 13** in the `/exercises` folder

Iterating on the system

High-level solution diagram



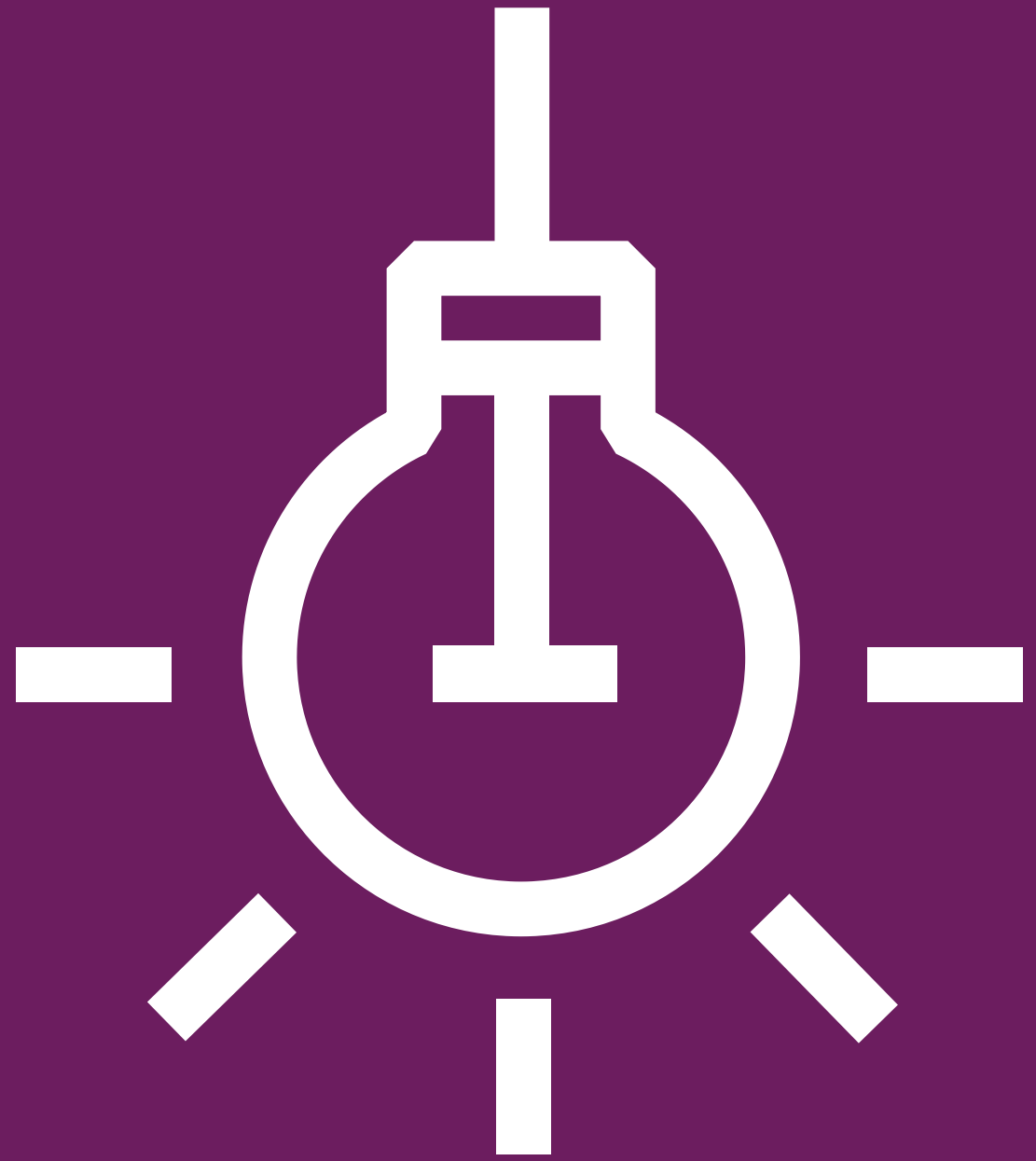
Xebia | Data

Exercise:
Iterate on the solution

Choose a direction to iterate on, work together (and present results at the end) !

- For inspiration, see **exercise 14** in the `/exercises` folder

Agent Orchestration

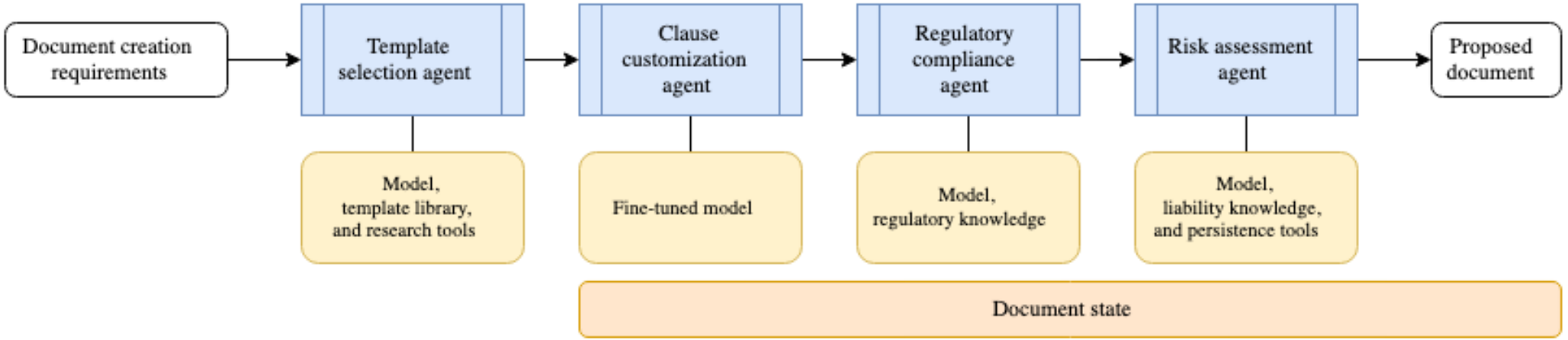


Agent Orchestration

- Coordinate specialized agents to solve complex, multidisciplinary tasks.
- Enable scalability and parallelism by separating concerns into independent units.
- Improve maintainability, testing, and iteration through isolated responsibilities.
- Optimize latency and cost by choosing the right pattern (sequential vs concurrent).
- Provide clear boundaries for observability, security, and governance.

Sequential Agent

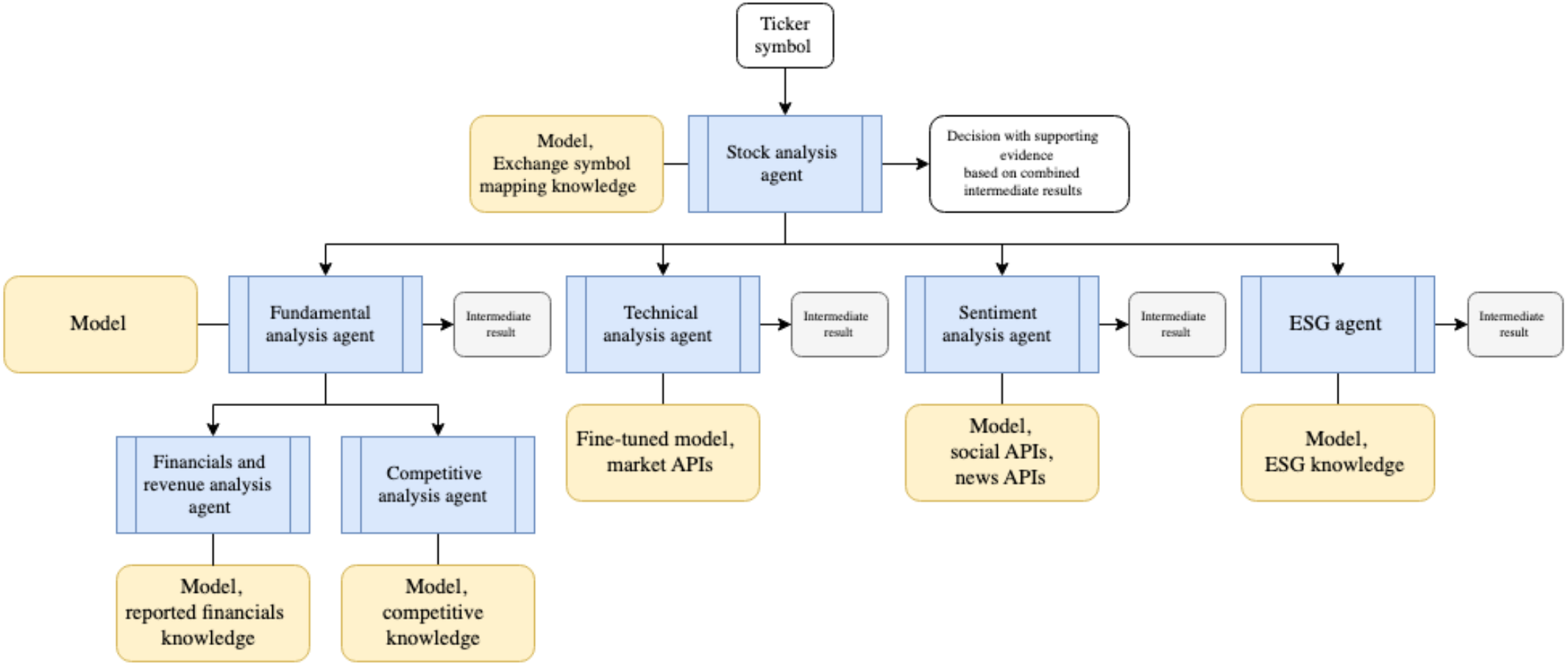
- A deterministic pipeline where each agent consumes previous agent output.
- template selection > clause customization > compliance check > risk assessment.



- Avoid when paralellization is required.

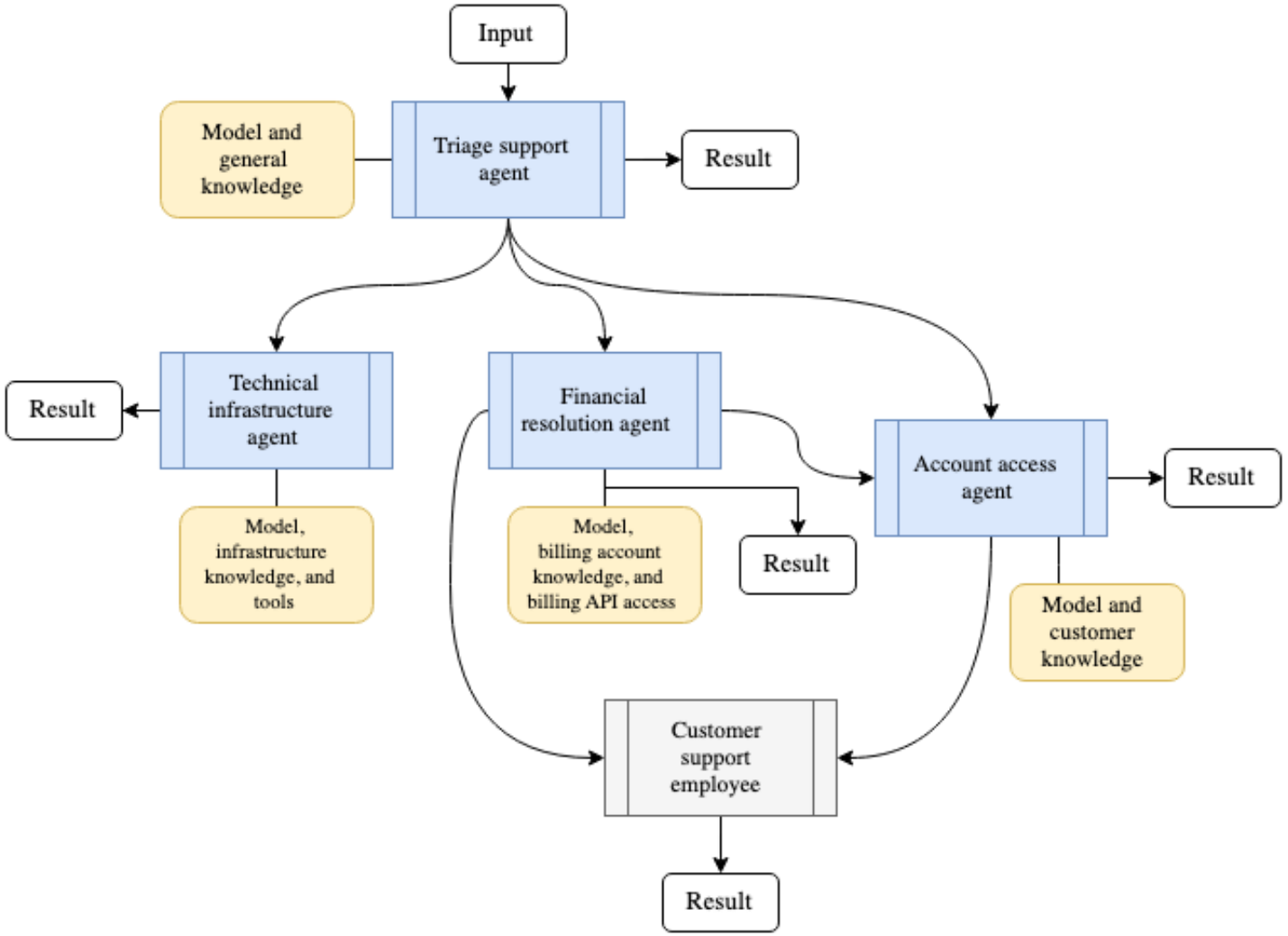
Concurrent Orchestration

- Can run in parallel, either by using a fixed set of agents or by dynamically choosing AI agents based on specific task requirements.
- multi-agent decision-making, low latency.



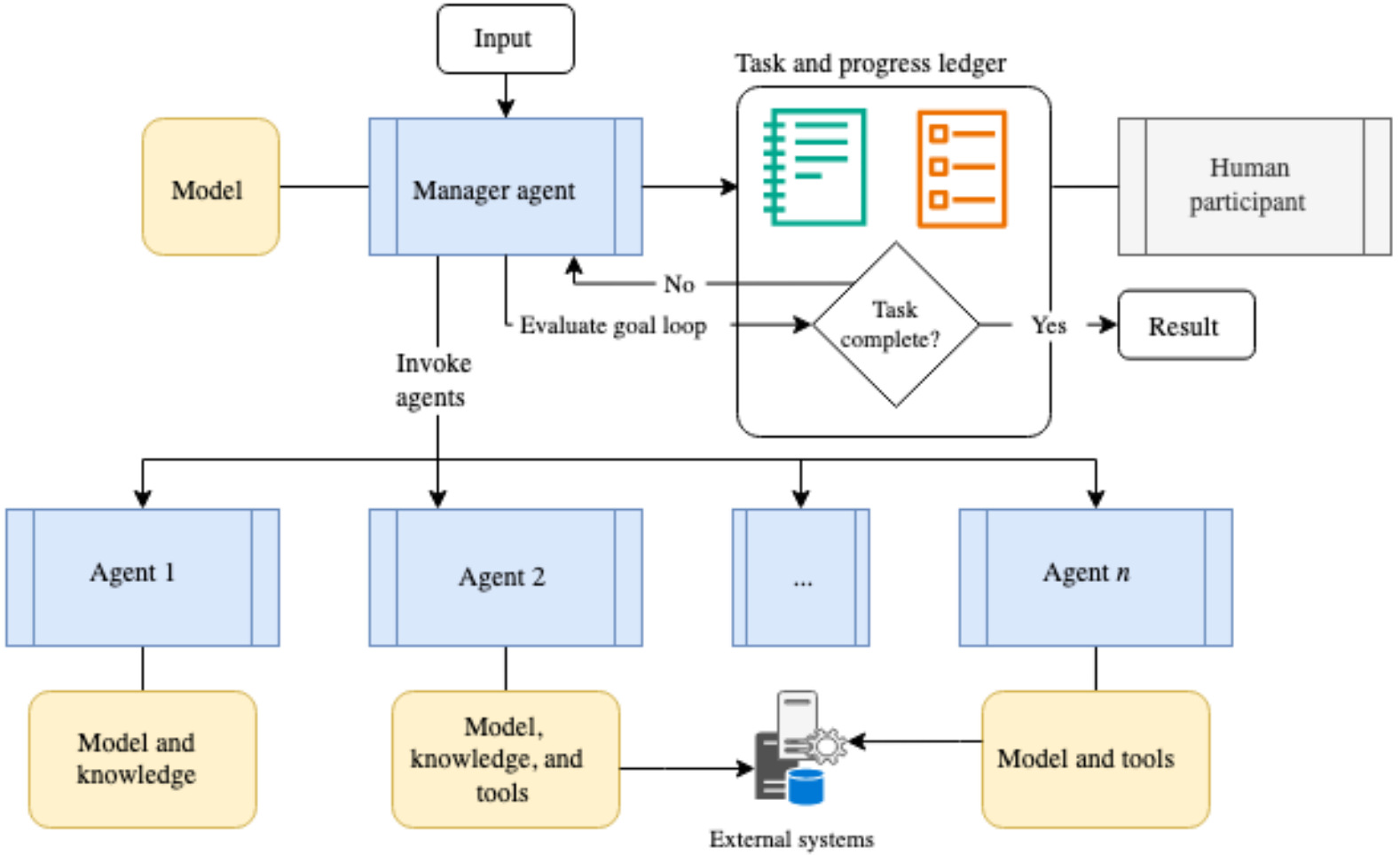
Handoff Orchestration

- dynamic routing/handoff; one agent passes task to another based on capability or context.
- triage workflows where the optimal handler isn't known upfront.



Magnetic Orchestration

- open-ended and complex problems that don't have a predetermined plan of approach



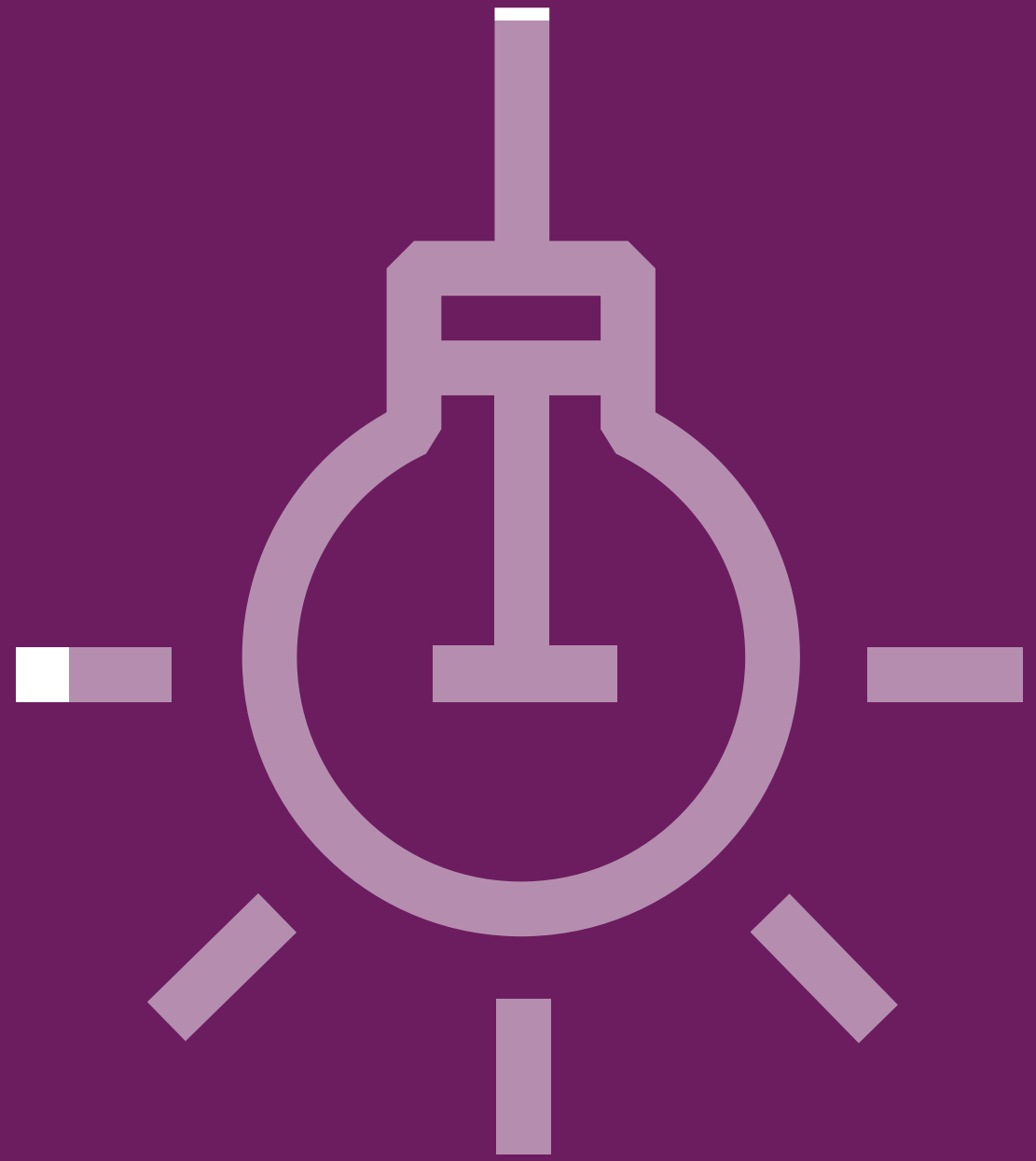
Async Agents

- Why async?
  - Speed: parallelize independent LLM calls to reduce total wall time.
  - Scalability: better utilization of I/O-bound model calls; useful for concurrent pattern or extracting many subitems.
- When to use:
  - Business-info extraction for many entities in same article (concurrent per-entity calls).
  - Evaluation across many articles (bulk eval).
- Caveats:
  - Respect model rate limits and quotas; add concurrency limits/backpressure.
  - Add timeouts, retries with exponential backoff, circuit breakers.
  - Preserve traces/observability per logical task.

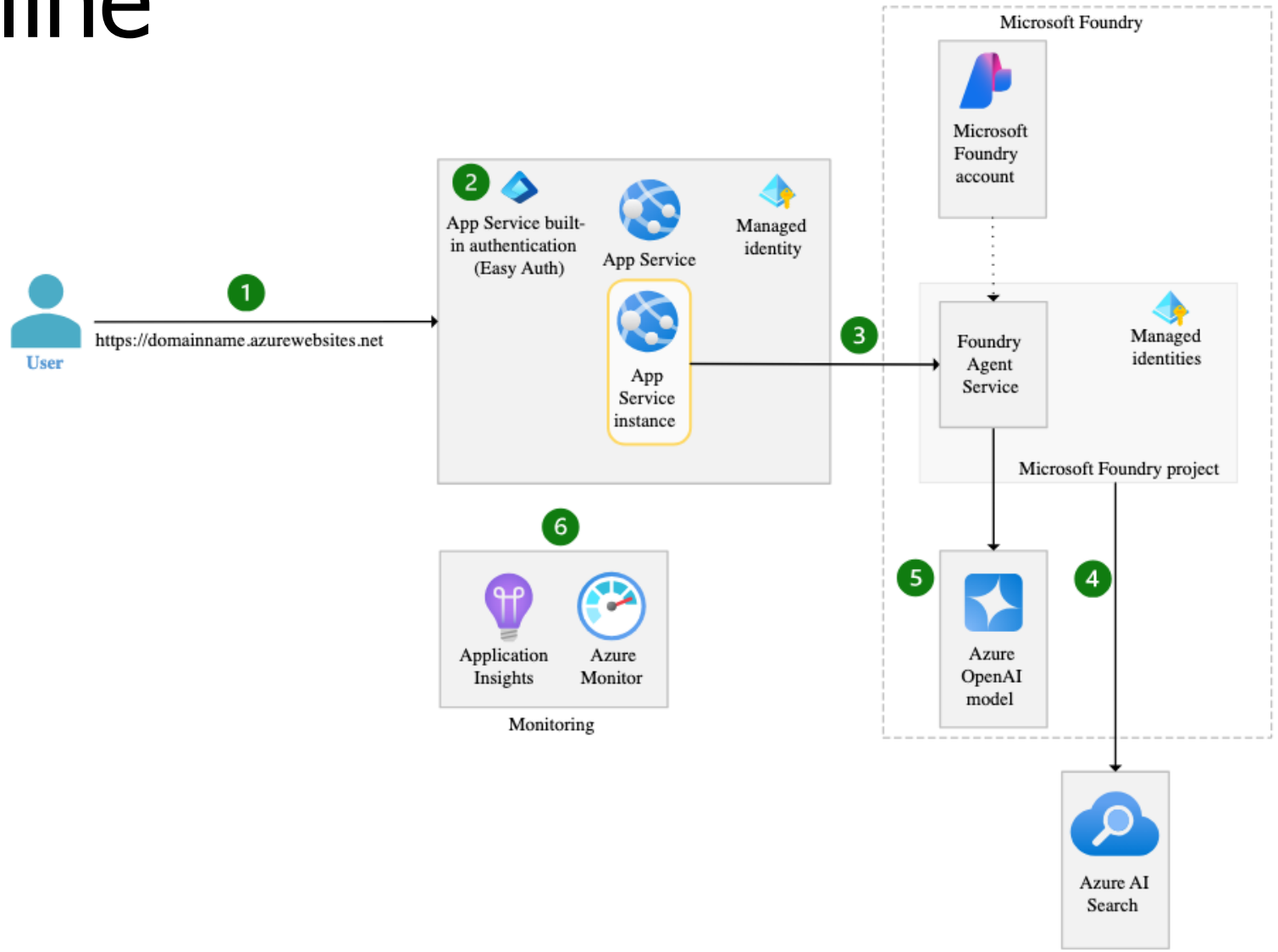
Implementation

- Routing & decision logic: deterministic vs. dynamic.
- Context handling: pass full vs. summarized context between agents.
- Retry and timeouts: per-agent timeouts + backoff.
- Observability: tracing, structured logs, per-agent metrics, distributed traces.
- Security: auth between agents, least privilege for tools, PII redaction.
- Testing: unit-test agents independently; integration tests for combined flows.
- Resource management: rate limits, compute quotas, batching and cost control.

Architecture Patterns



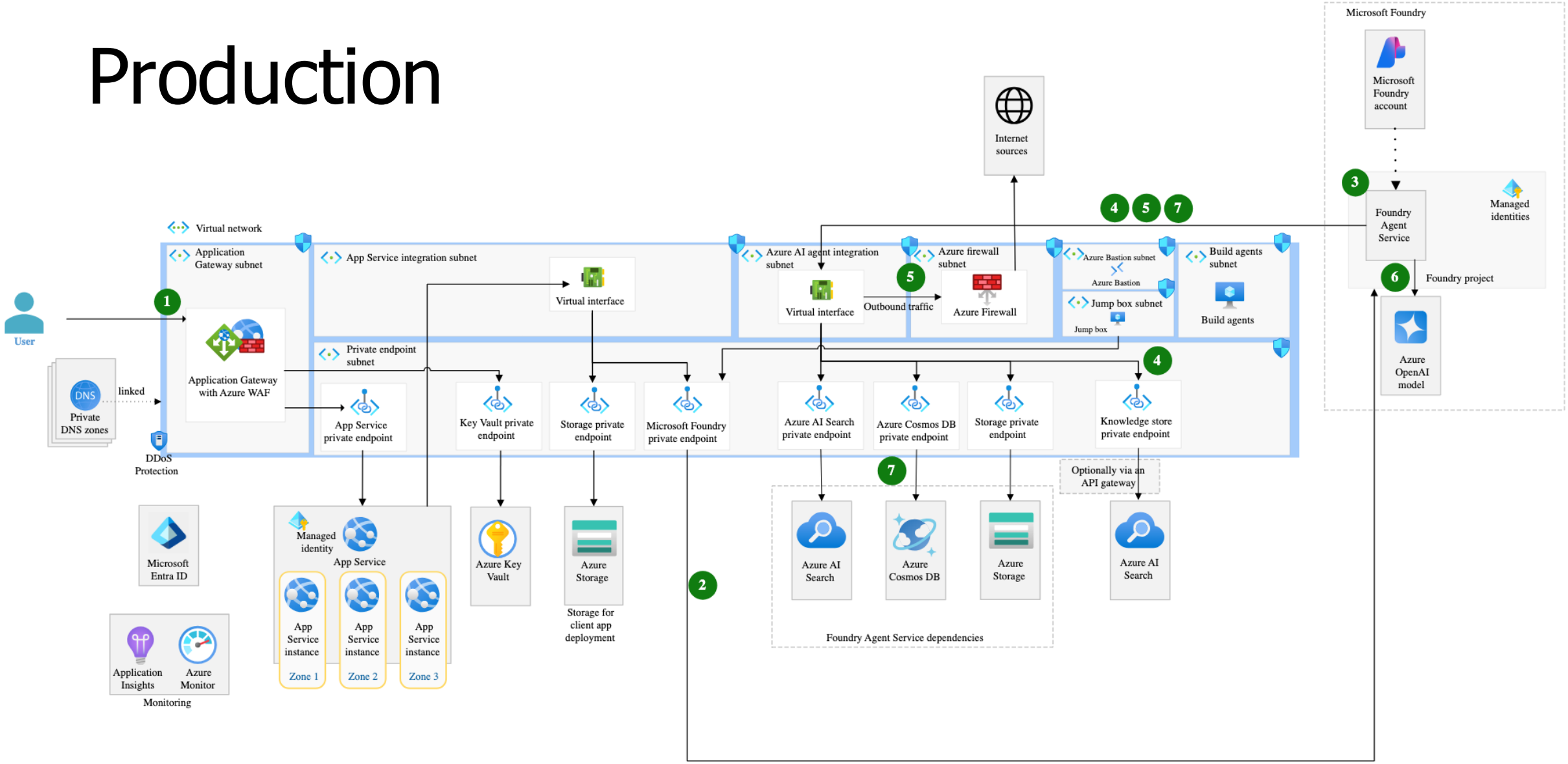
Baseline


Microsoft Foundry
Microsoft Foundry account
2
App Service built-in authentication (Easy Auth)
App Service
Managed identity
1
User
https://domainname.azurewebsites.net
App Service instance
3
Foundry Agent Service
Managed identities
Microsoft Foundry project
6
5
4
Application Insights
Azure Monitor
Monitoring
Azure OpenAI model
Azure AI Search

Production

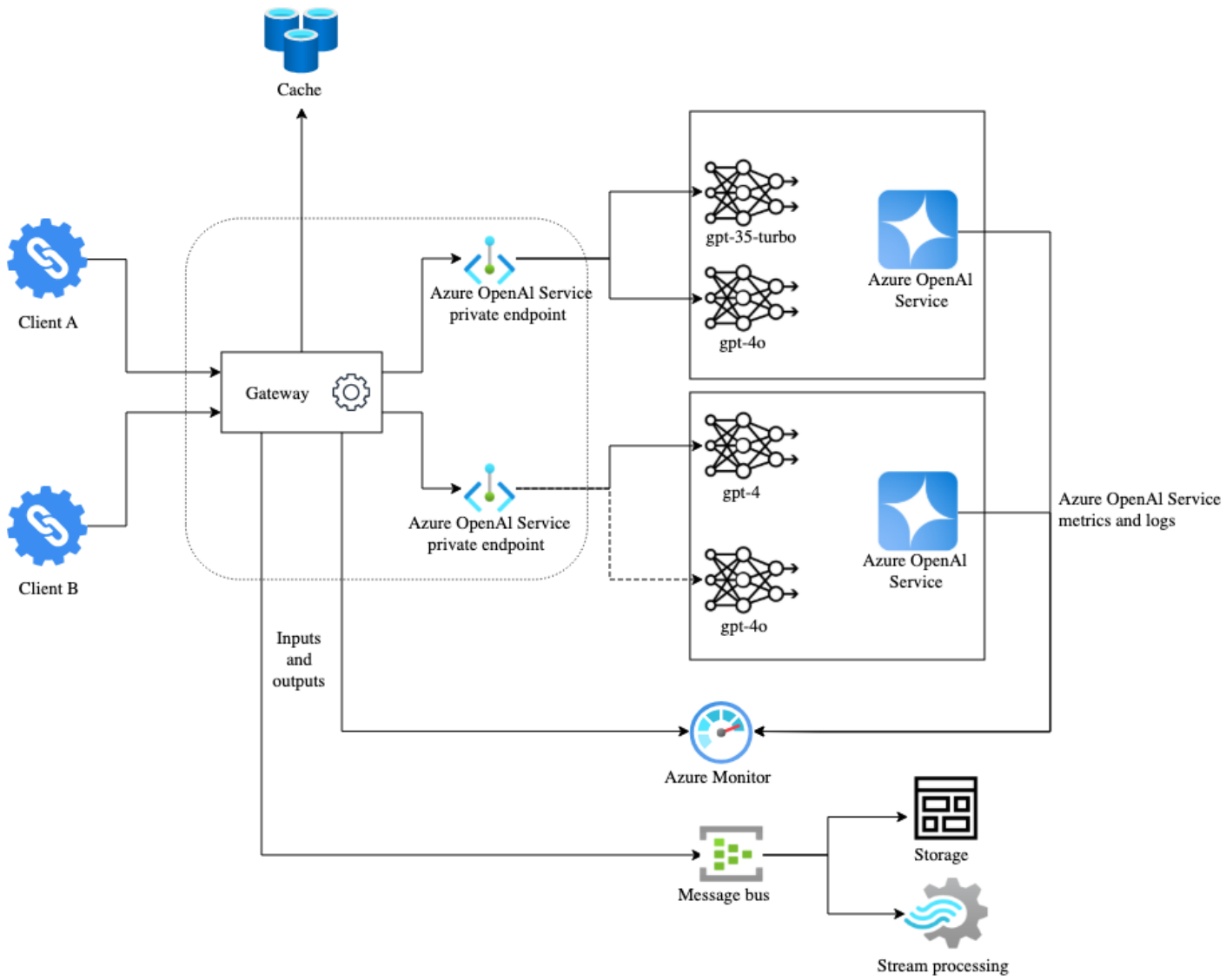
- A single, secure entry point for all chat UI traffic, which minimizes the attack surface
- Filtered ingress and egress network traffic by using a combination of NSGs, a web application firewall, user-defined routes (UDRs), and Azure Firewall rules
- End-to-end encryption of data in transit by using TLS
- Network privacy by using Private Link for all Azure PaaS service connections
- Zero Trust Policy for all resources

Production


Microsoft Foundry
Microsoft Foundry account
Foundry Agent Service
Managed identities
Foundry project
Azure OpenAI model
Internet sources
Virtual network
Application Gateway subnet
App Service integration subnet
Azure AI agent integration subnet
Azure firewall subnet
Azure Bastion subnet
Azure Bastion
Jump box subnet
Jump box
Build agents subnet
Build agents
Private endpoint subnet
Virtual interface
Outbound traffic
Azure Firewall
User
Private DNS zones
linked
DDoS Protection
Application Gateway with Azure WAF
App Service private endpoint
Key Vault private endpoint
Storage private endpoint
Microsoft Foundry private endpoint
Azure AI Search private endpoint
Azure Cosmos DB private endpoint
Storage private endpoint
Knowledge store private endpoint
Optionally via an API gateway
Microsoft Entra ID
Managed identity
App Service
App Service instance
Zone 1
Zone 2
Zone 3
Azure Key Vault
Azure Storage
Storage for client app deployment
Azure AI Search
Azure Cosmos DB
Azure Storage
Foundry Agent Service dependencies
Azure AI Search
Application Insights
Azure Monitor
Monitoring

API Gateway

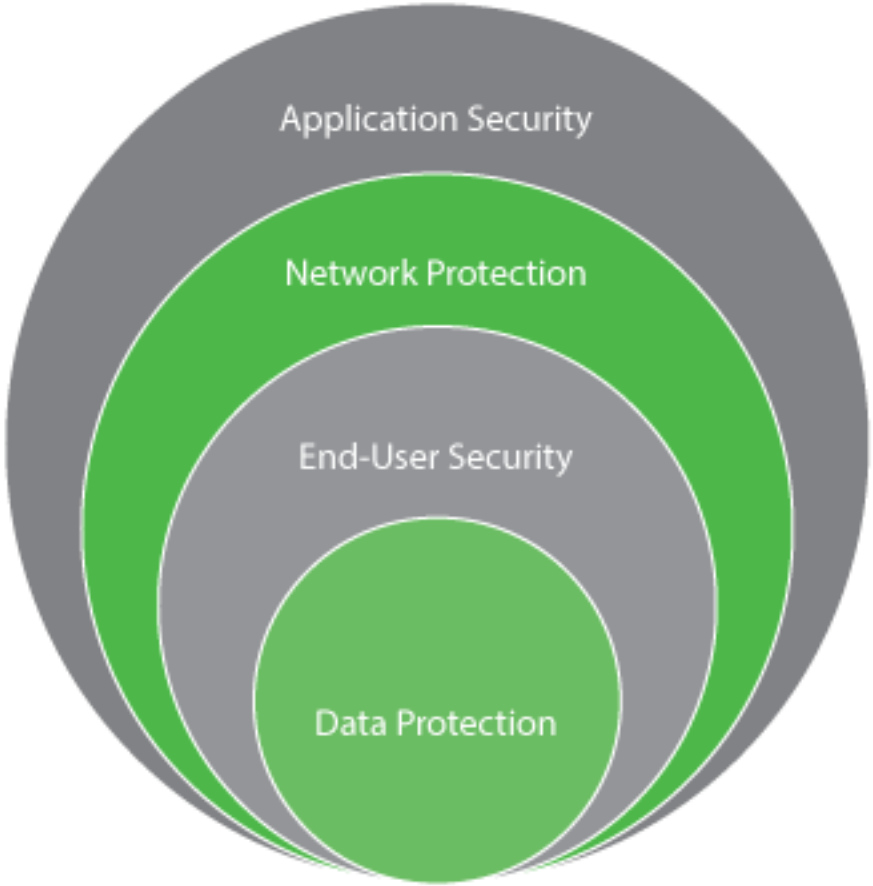
- Capability to log both the client's request and the final response that it received
- For real-time monitoring, the logs can be published to a message bus.
- Beneficial when you have multiple clients or multiple models that you need to monitor



Networking and Security

Key Risks

- Data leakage (inputs/outputs sent to external models)
- Access to resources
- Prompt injection, model poisoning, hallucination risks
- Supply-chain compromise (dependencies, images, models)
- Runaway Costs



Cybersecurity Onion

IAM

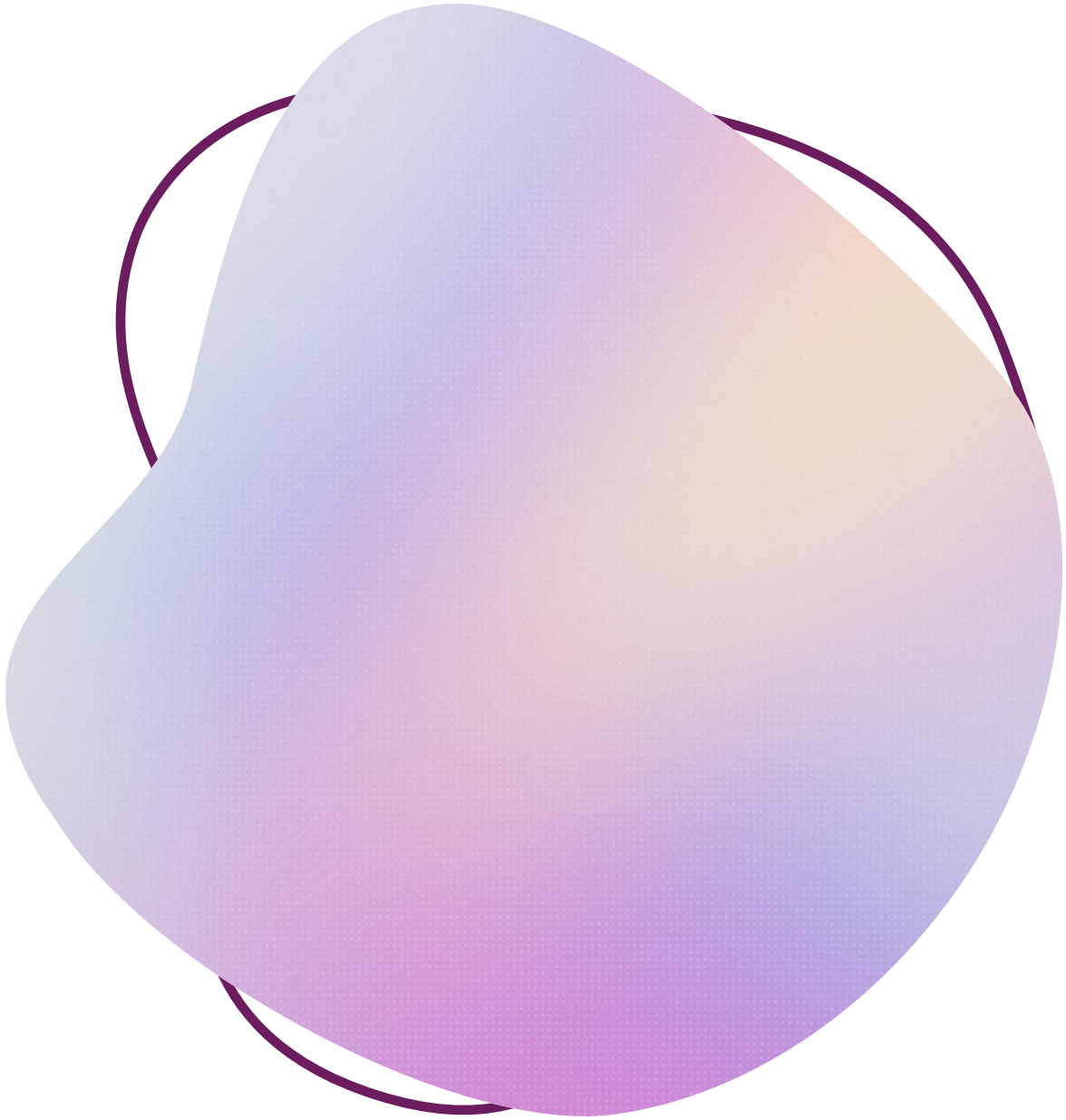
- Central identity (Entra ID / OIDC), RBAC and least-privilege for services and humans
- Managed identities for service-to-service auth (no secrets in code)
- Conditional Access & MFA for operator consoles and CI/CD access
- Federate on-prem identities with Entra ID (Azure AD Connect or Workload Identity Federation)

Data Protection

- Encrypt data in transit and at rest.
- Classify, redact or tokenise PII before external calls
- Differential privacy or synthetic data for training/fine-tuning
- Data residency within regions and private networks

Networking

- Define trust boundaries — map data flows for ingress, storage, inference, egress and logs.
- Use VNet integration for inference, storage, and control-plane services to avoid public exposure.
- Private Endpoints / Private Link for managed services (OpenAI, Key Vault, Storage, Search)
- Protect ingress with App Gateway + WAF (or similar) and avoid direct public app endpoints for sensitive UIs/APIs.
- Upload critical logs to SIEM tools like Sentinel/Defender.



Introduction to Fine-tuning

Xebia | Data

Connections – What do you already know?

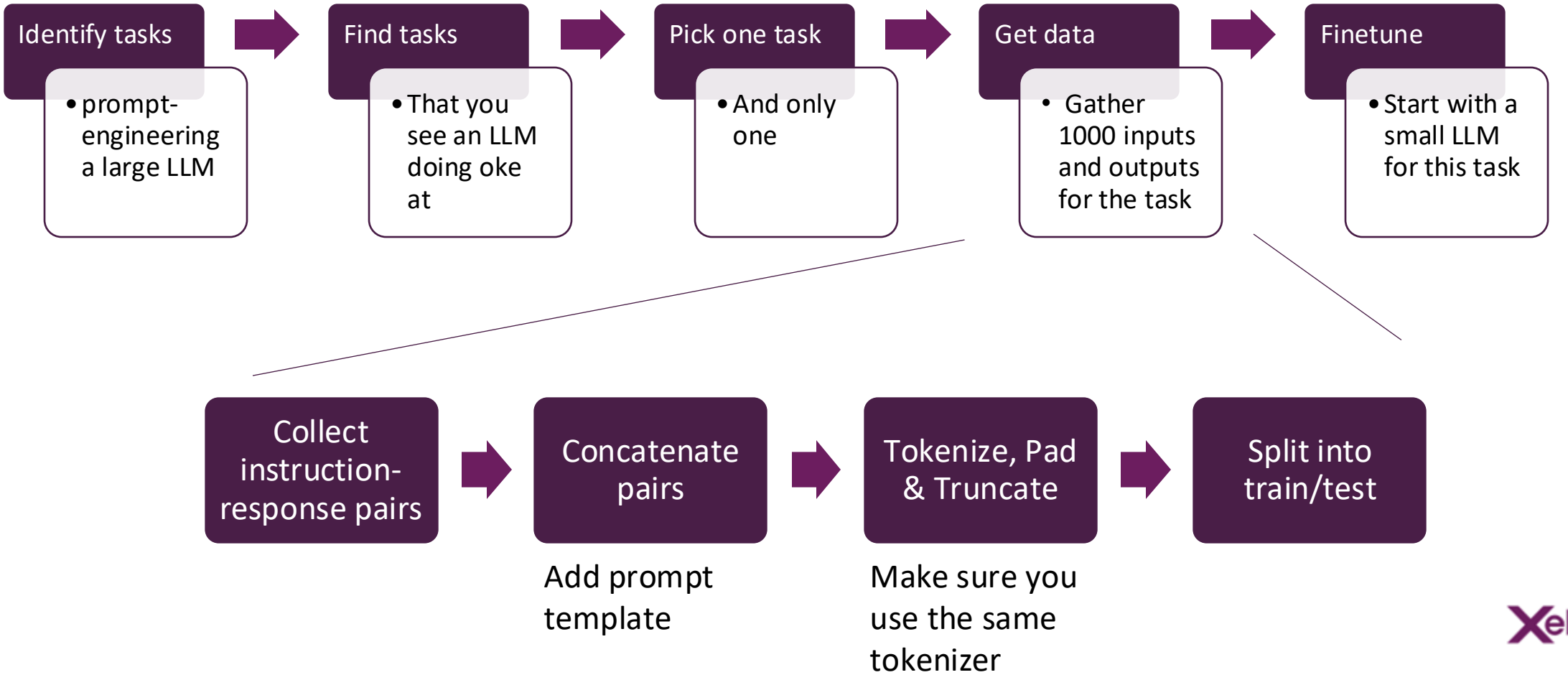
- **Who has fine-tuned an LLM?**
  - QLoRA
  - RLHF
- **Who is familiar with training a neural network?**
- PyTorch, JAX, TensorFlow, Keras, FastAI
- Epoch, batch, learning rate, back-prop?

- **What questions do you already have?**

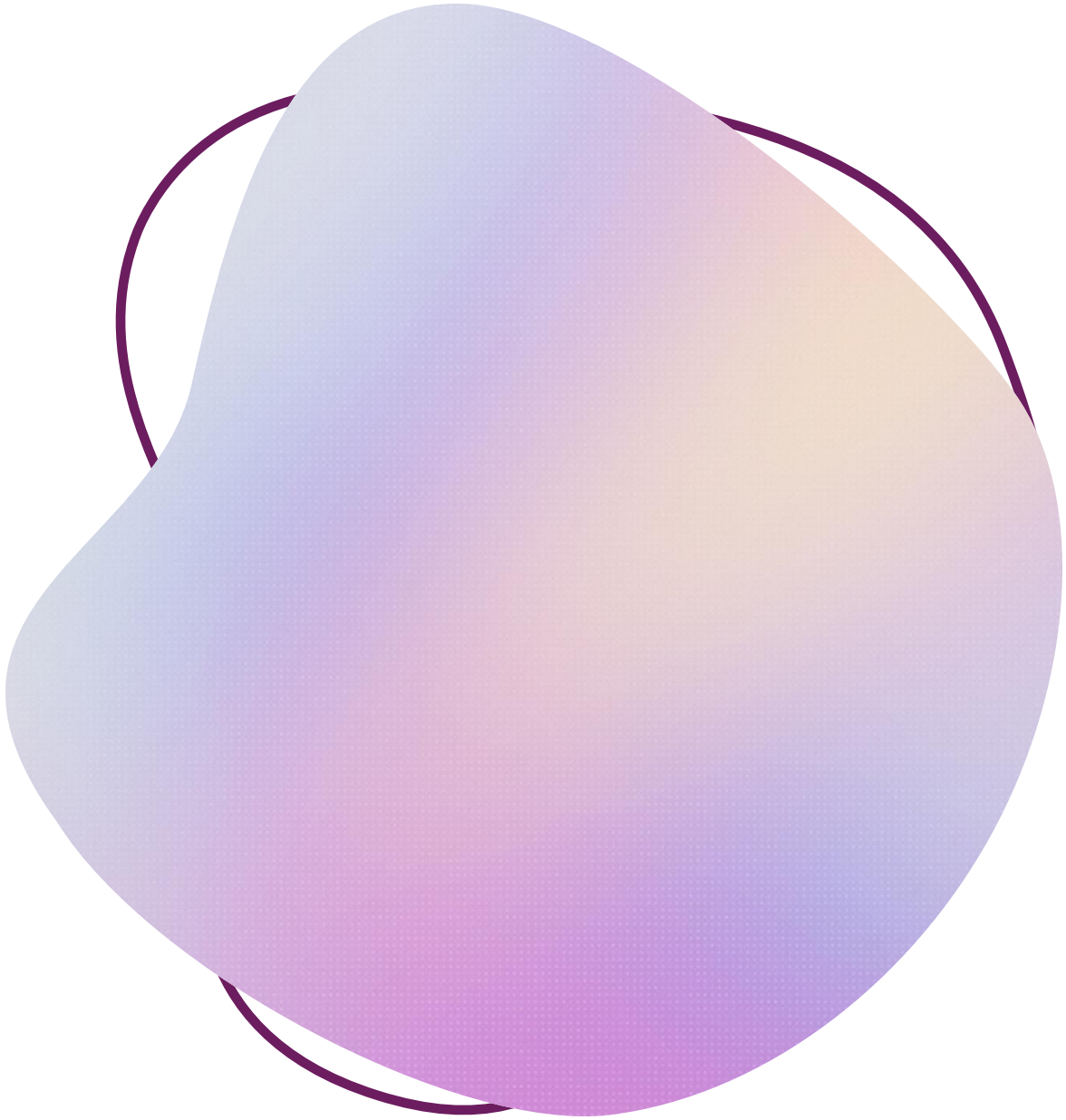
Connections – Why do you need to learn this?

- For most use-cases: fine-tuning is NOT needed and should be considered a last resort.
- For example, domain knowledge can be sent to the model using RAG.
- However, in certain cases, models might require additional training or specialized datasets to handle these domains effectively.

Process for Finetuning



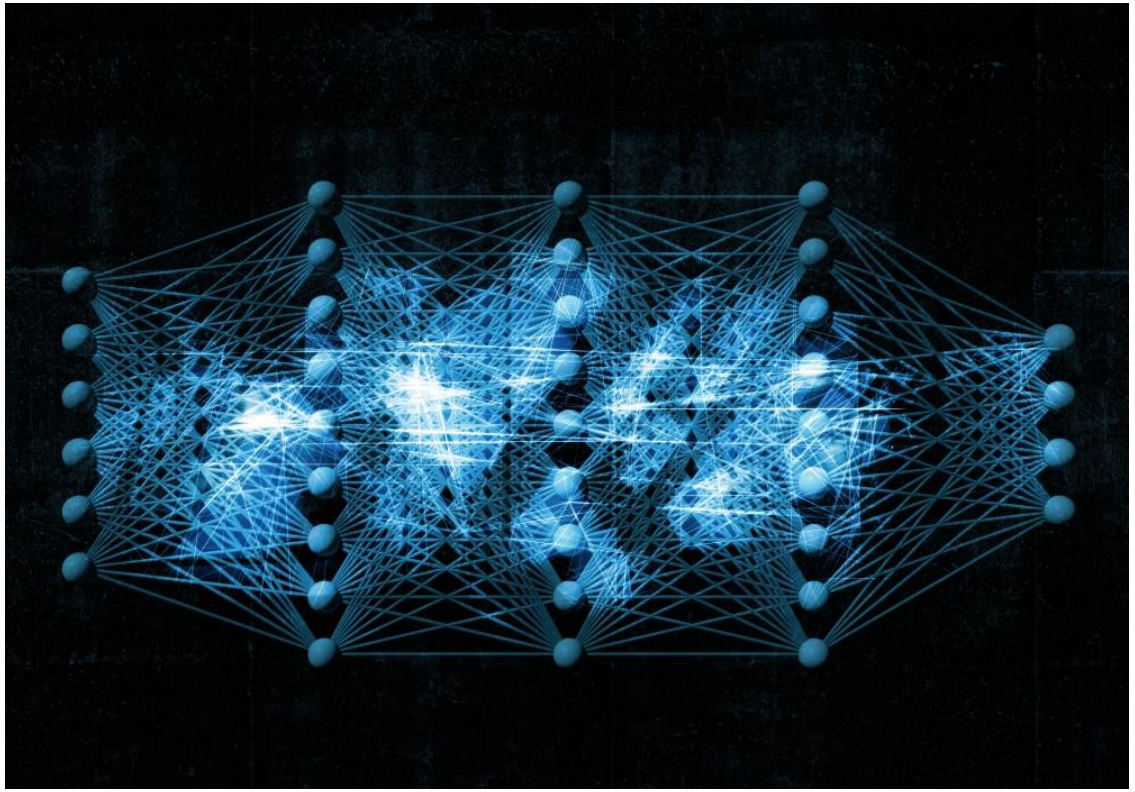
Xebia | Data



Parameter Efficient Fine-tuning (PEFT)

Full parameter fine-tuning

Full-parameter fine-tuning involves updating all the parameters of a pre-trained model on a new task-specific dataset.



https://www.mygreatlearning.com/blog/things-to-know-before-deep-diving-into-neural-networks/

Full parameter fine-tuning Issues

Resource-Intensive: Requires extensive computational power (GPU/TPU) and large storage due to model size

Slow Training: Retraining the entire model takes significant time.

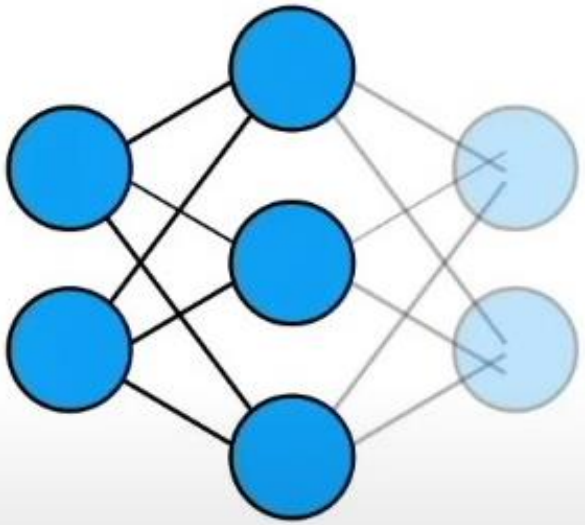
Catastrophic Forgetting: Fine-tuning can cause the model to forget previously learned information.

Model Name	n_{params}
GPT-3 Small	125M
GPT-3 Medium	350M
GPT-3 Large	760M
GPT-3 XL	1.3B
GPT-3 2.7B	2.7B
GPT-3 6.7B	6.7B
GPT-3 13B	13.0B
GPT-3 175B or "GPT-3"	175.0B

Xebia | Data

https://arxiv.org/pdf/2005.14165

Full Parameter Fine-tuning



$x \rightarrow h(x)$

$$h(x) = W_0 x$$

$$W_0 \in R^{d \times k}$$
$$x \in R^{k \times 1}$$
$$h(x) \in R^{d \times 1}$$

W_0 x $=$ $h(x)$

Trainable

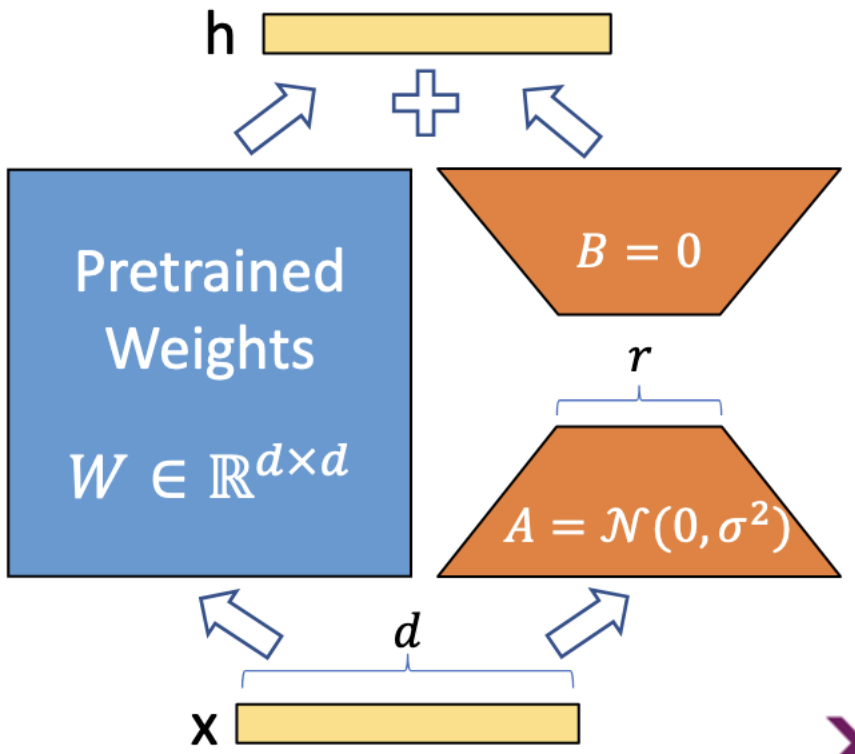
$$d = 1{,}000$$
$$k = 1{,}000$$
$$\Longrightarrow$$
$$d \times k = \textbf{1,000,000}$$
trainable parameters

Xebia | Data

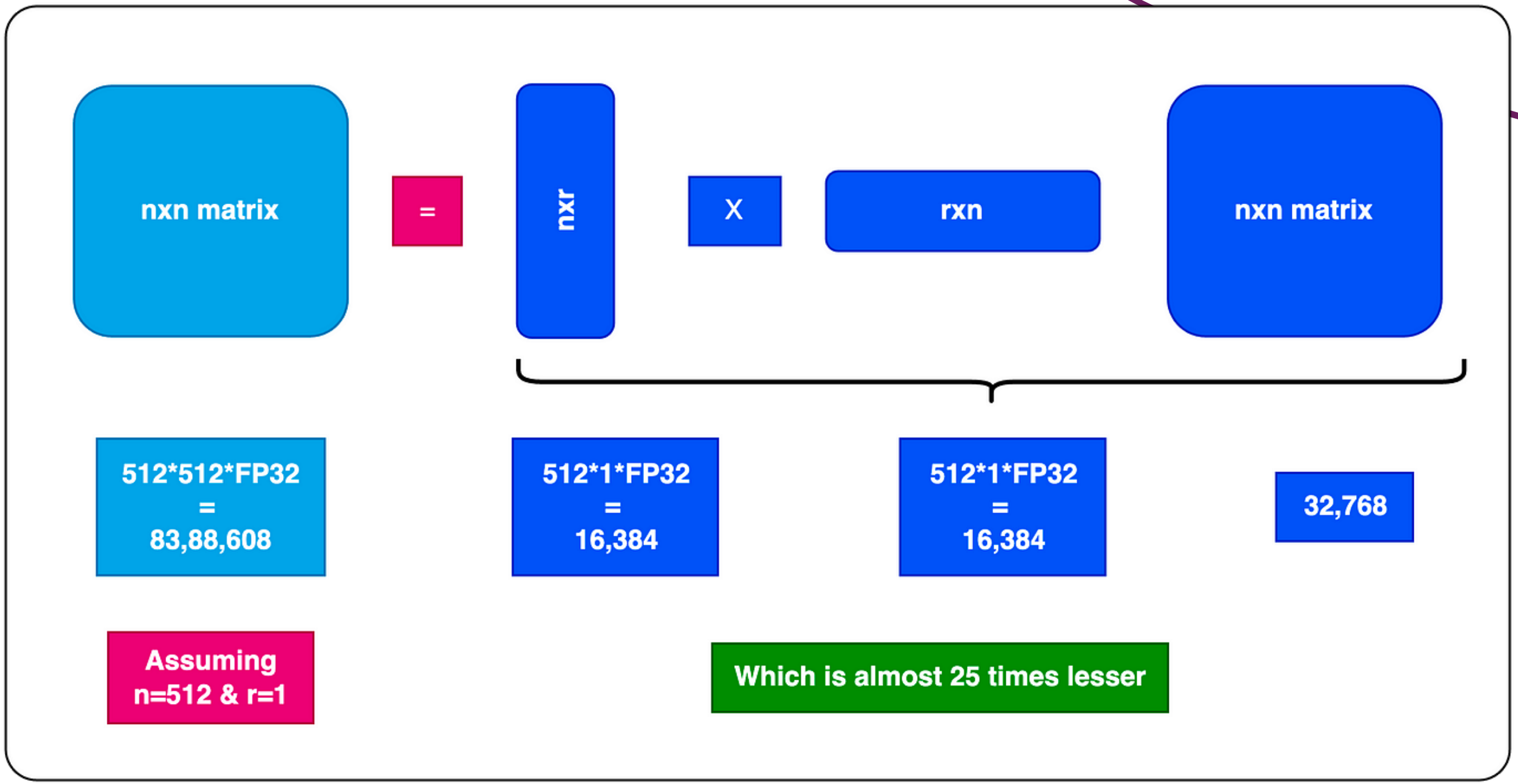
Solution: Low Rank Adaptation (LoRA)

One Parameter Efficient approach to fine-tuning is LoRA

The key to LoRA matrix decomposition

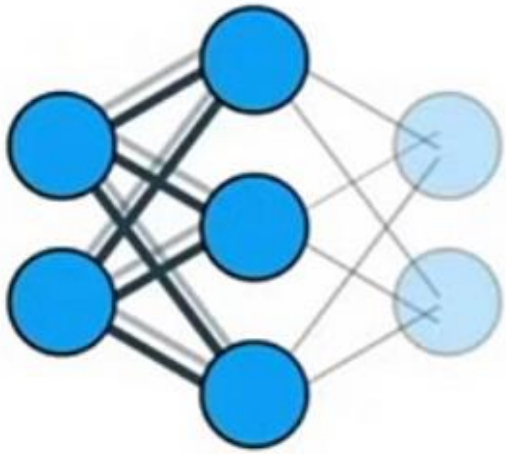


https://arxiv.org/pdf/2106.09685



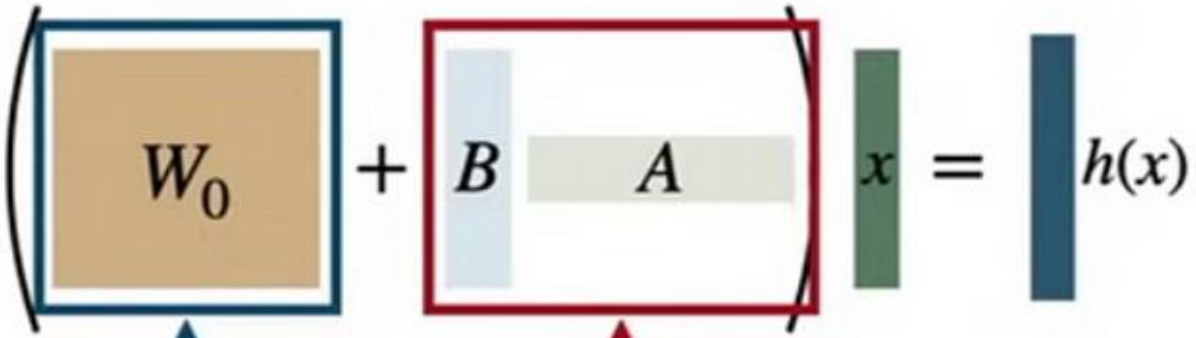
https://abvijaykumar.medium.com/fine-tuning-llm-parameter-efficient-fine-tuning-peft-lora-qlora-part-1-571a472612c4

Low Rank Adaptation (LoRA)



$x \rightarrow h(x)$

$$h(x) = W_0 x + \Delta W x \qquad \Delta W = BA$$
$$= W_0 x + BAx$$



Frozen

Trainable

$d = 1{,}000$
$k = 1{,}000$
$r = 2$

$\Longrightarrow$ $(d \times r) + (r \times k) = 4{,}000$ **trainable parameters**

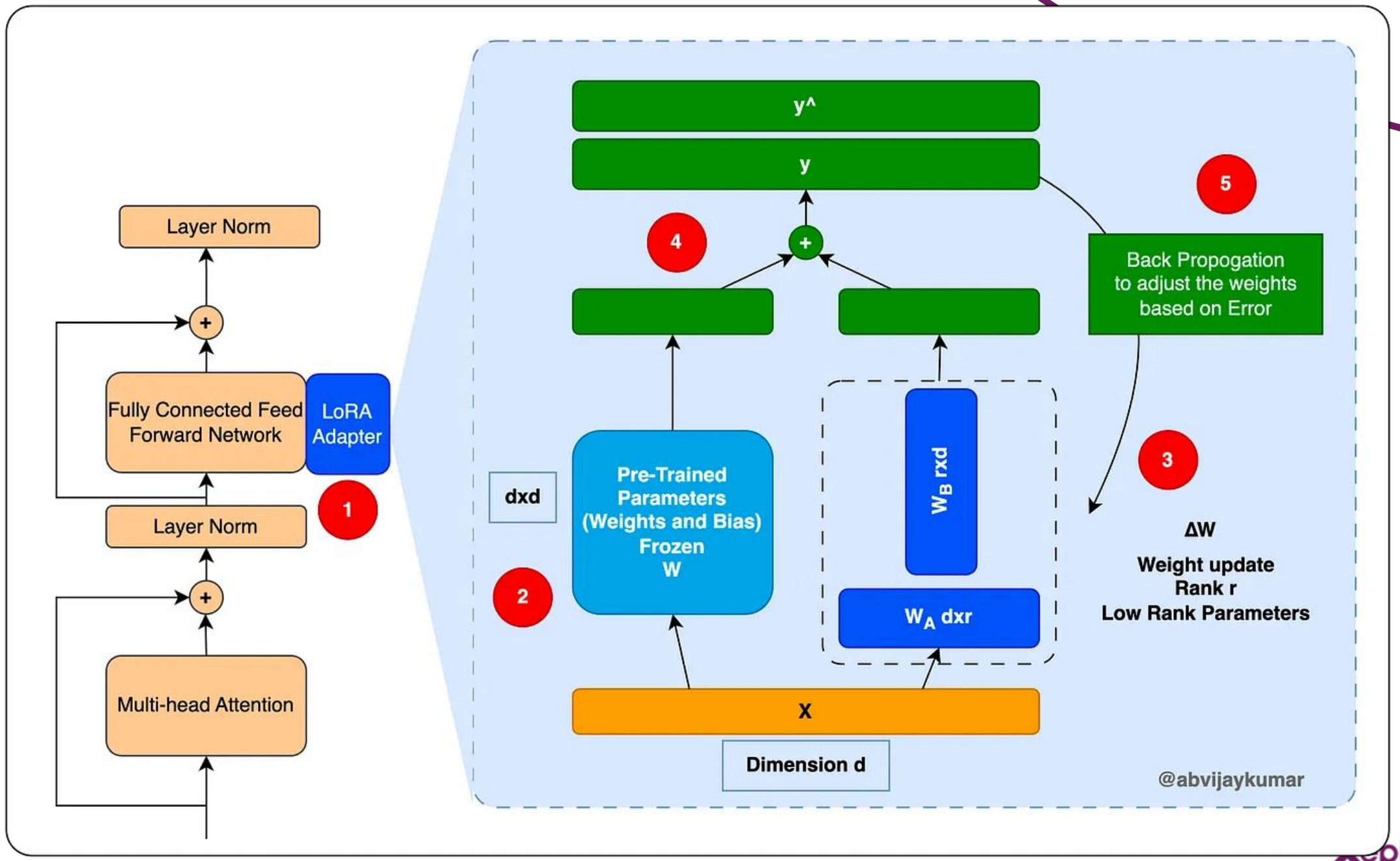
$W_0, \Delta W \in R^{d \times k}$

$B \in R^{d \times r}$

$A \in R^{r \times k}$

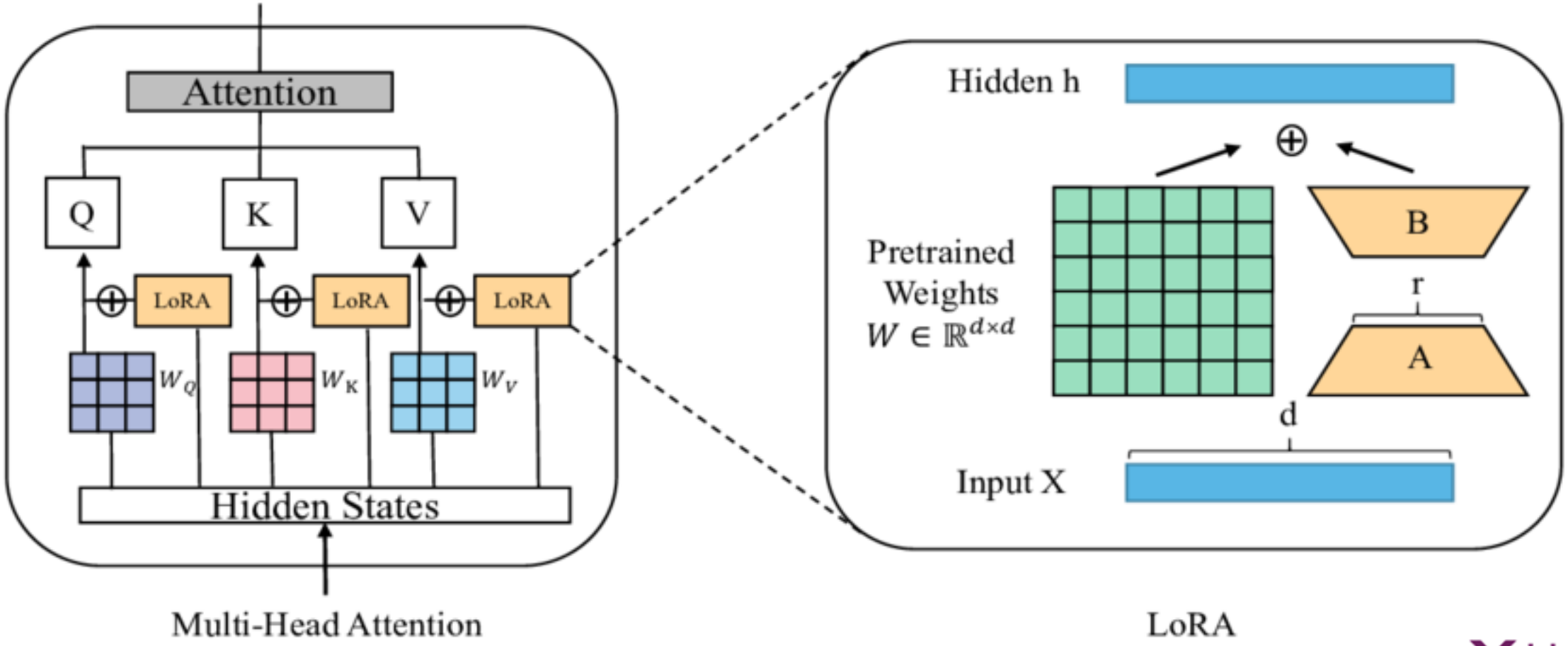
$h(x) \in R^{d \times 1}$

Xebia | Data

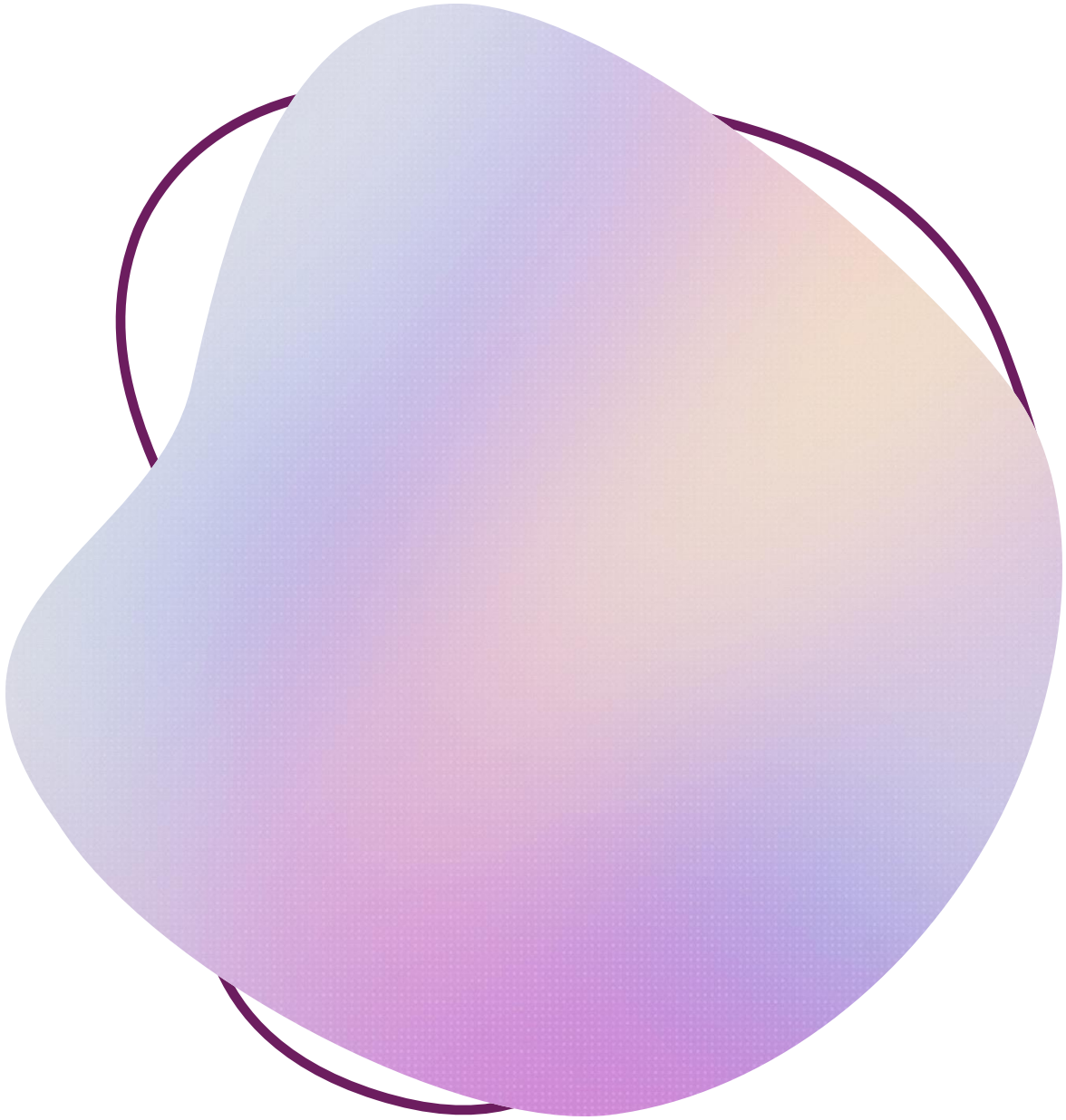


https://abvijaykumar.medium.com/fine-tuning-llm-parameter-efficient-fine-tuning-peft-lora-qlora-part-1-571a472612c4

LoRA can be applied to different attention components



Source: https://medium.com/datadreamers/exploring-lora-unveiling-parameter-efficient-tuning-and-self-attention-mechanisms-in-depth-58e4c3b5ce30



Quantization

QLoRA — Quantized Low Rank Adaption Method

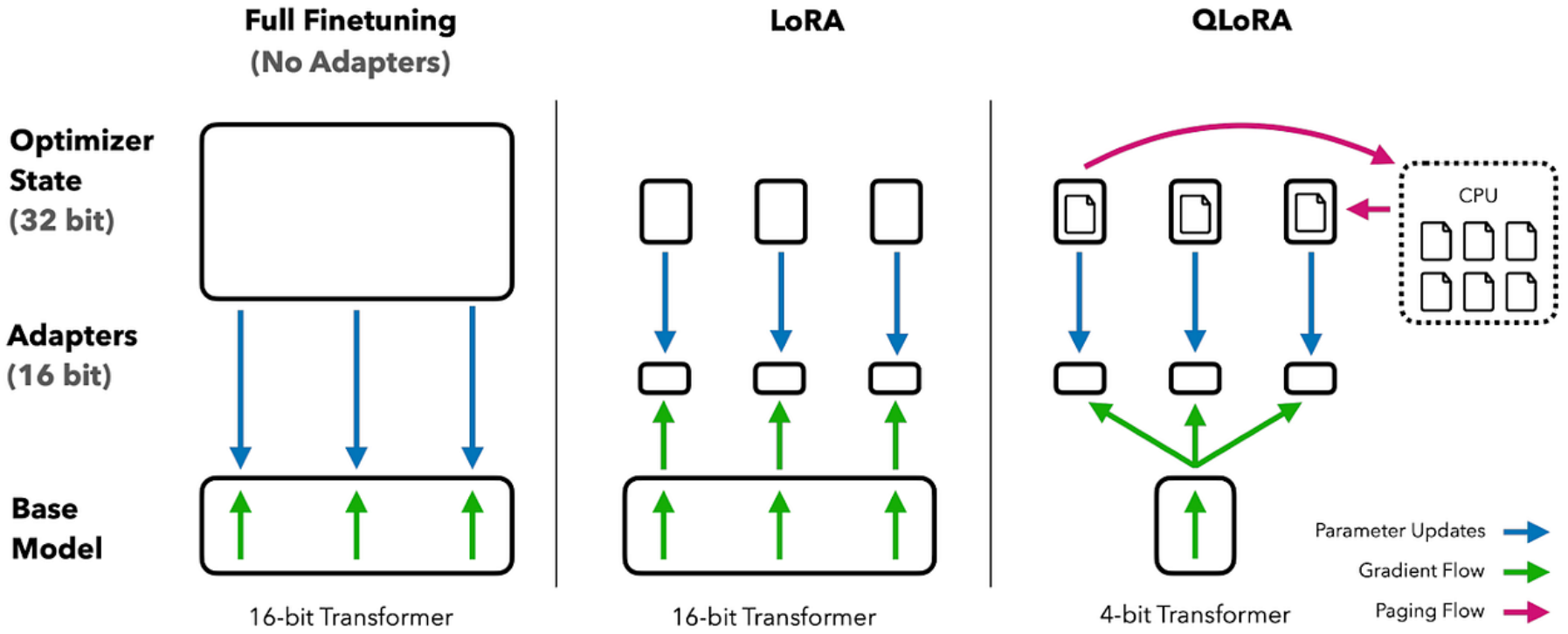


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

https://arxiv.org/pdf/2305.14314

There are 3 Key optimizations that QLoRA brings on top of LoRA

1. 4-bit NF4 Quantization

4-bit NormalFloat4 is an optimized data type that can be used to store the weights of the original model, which brings down the memory footprint considerably.
It is a 3-step process:

2. Normalization: As part of the normalization step, the weights are adjusted to a zero mean, and a constant unit variance.

There are 3 Key optimizations that QLoRA brings on top of LoRA

1. 4-bit NF4 Quantization

4-bit NormalFloat4 is an optimized data type that can be used to store the weights of the original model, which brings down the memory footprint considerably.
It is a 3-step process:

2. **Quantization**: A 4-bit data type can only store 16 numbers. So, after normalization, instead of storing the weights, the nearest position is stored.

For example, let's say we have a FP32 weight, with a value of 0.2121. a 4-bit split between -1 to 1 will be the following number positions. 0.2121 is closest to 0.1997, which is the 10th position. Instead of saving the FP32 of 0.2121, we store 10.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
-1	-0.8667	-0.7334	-0.6001	-0.4668	-0.3335	-0.2002	-0.0669	0.0664	0.1997	0.333	0.4663	0.5996	0.7329	0.8662	1

Xebia | Data

https://abvijaykumar.medium.com/fine-tuning-llm-parameter-efficient-fine-tuning-peft-lora-qlora-part-1-571a472612c4

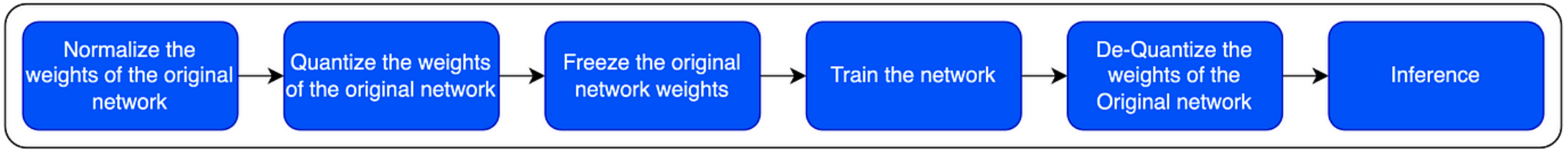
There are 3 Key optimizations that QLoRA brings on top of LoRA

1. 4-bit NF4 Quantization

4-bit NormalFloat4 is an optimized data type that can be used to store the weights of the original model, which brings down the memory footprint considerably.
It is a 3-step process:

3. **Dequantization**: Once all the training is done, the original weights will be de-quantized, i.e. reverse the process.

Note, the 4-bit NormalFloat quantization is applied only to the original weights of the model. The LoRA adapter weights will be left as FP32, as the training will happen on these weights.



Xebia | Data

https://abvijaykumar.medium.com/fine-tuning-llm-parameter-efficient-fine-tuning-peft-lora-qlora-part-1-571a472612c4

There are 3 Key optimizations that QLoRA brings on top of LoRA

2. Double Quantization

In 4-bit NF4 Quantization, a "quantization constant" is used to scale the weights

$$Q(w) = round(\frac{w - z}{s})$$

w = Original high-precision weight.
z = An offset used to align the quantized range with the actual weight distribution.
s = Quantization constant (scaling factor), dependent on the max weight value.
$Q(w)$ = Quantized weight, in a lower precision format (i.e.., 4-bit integer)

There are 3 Key optimizations that QLoRA brings on top of LoRA

2. Double Quantization

Double quantization, further reduces the memory footprint, by quantizing the quantization constants.

Let's imagine we have grouped 64 parameters/weights into a block to quantize, and each quantization constant takes 32 bits, as it is FP32. It adds a 0.5 bit per parameter on average, which means we are talking of at least 500,000 bits for a typical 1Mil parameter model.

With Double quantization, we apply quantization on these quantization constants, which will further optimize our memory usage.

There are 3 Key optimizations that QLoRA brings on top of LoRA

3. Unified Memory Paging

Coupled with the above techniques, QLoRA also utilizes the nVidia unified memory feature, which allows GPU->CPU seamless page transfers, when GPU runs out of memory, thus managing the sudden memory spikes in GPU, and helping memory overflow/overrun issues.

Tim Dettmers provides a nice explanation: https://github.com/bitsandbytes-foundation/bitsandbytes/issues/962

Important Concepts: QLoRA

- **LoRA: Low rank adapters**
  - Reduce the trainable parameters
  - Llama2 13b
    - trainable params: 36,372,480 || all params: 13,052,236,800 trainable: 0.28%
  - Mistral 7b
    - trainable params: 28,311,552 || all params: 7,270,043,648 trainable: 0.39%
- **Quantization**
  - Bfloat-16 / NF4
  - Quantization error
    - Recovers with fine-tuning
    - Re-occurs with merging weights
  - Rank
    - Alpha = 2*rank

```yaml
lora_config:
  # See https://arxiv.org/pdf/2110.04366.pdf, p.8
  target_modules:
    - down_proj
    - up_proj
    - gate_proj
    # - k_proj
    # - v_proj
    # - q_proj
    # - o_proj
    # - lm_head
```

```yaml
- lora_target_modules:
    - q_proj
    - v_proj
    - k_proj
    - o_proj
    - gate_proj
    - down_proj
    - up_proj

    -  c_attn
    -  c_proj
```

Xebia | Data

- **Best practices**

- Efficiency Methods

QLoRA, FlashAttention and paged_adamw_32bit for efficient training

Hardware specific optimization (TensorRT, TensorRT-LLM)

Performance

- Ensemble
- Multi LoRA

https://xebia.com/blog/llm-fine-tuning-challenge/
https://xebia.com/blog/winning-recipe-for-finetuning-llm/

Xebia | Data

Make use of all available optimizations

Method/tool	Improves training speed	Optimizes memory utilization
Batch size choice	Yes	Yes
Gradient accumulation	No	Yes
Gradient checkpointing	No	Yes
Mixed precision training	Yes	(No)
Optimizer choice	Yes	Yes
Data preloading	Yes	No
DeepSpeed Zero	No	Yes
torch.compile	Yes	No

https://huggingface.co/docs/transformers/perf_train_gpu_one#batch-size-choice

Inference and efficiency

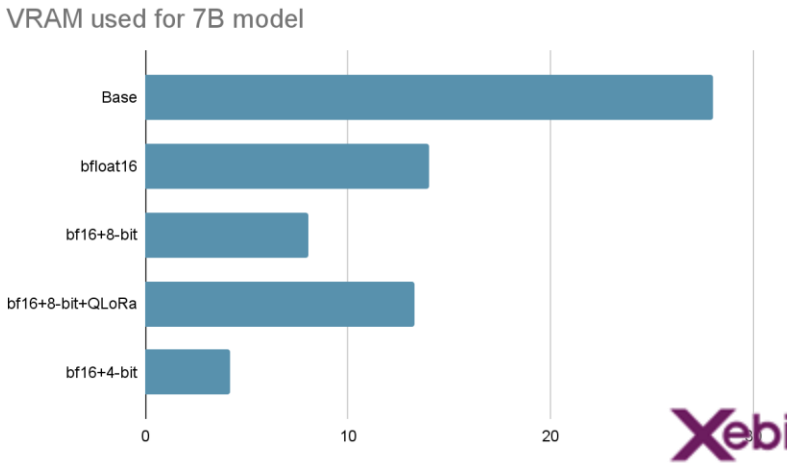
- **Experiment with efficient use of GPU VRAM**
  - Rule of thumb: model with Xbn params will take up ~4X VRAM in float32 precision.
  - Using bfloat16, it's ~2X
  - Using bfloat16 + 8-bit quantization, its ~1.1X
  - But merging the QLoRa model weights in at inference time blows things up to ~1.8X compared to normal 8-bit + bfloat16
  - Best thing: merge and dump to disk, then load at inference time. This, too, requires a lot of VRAM.
- **Be mindful of too much quantization**
  - In practice, 4-bit inference saves you ~40-50% VRAM compared to 8-bit
  - But performance is also worse . . .

4-bit

Name	Value
accuracy_MMLU__EM	0.497
accuracy_TruthfulQA__EM	0.333
calibration_MMLU__ECE_10bin	0.688
fairness_MMLU__EM_Fairness	0.497
fairness_TruthfulQA__EM_Fairness	0.333

8-bit

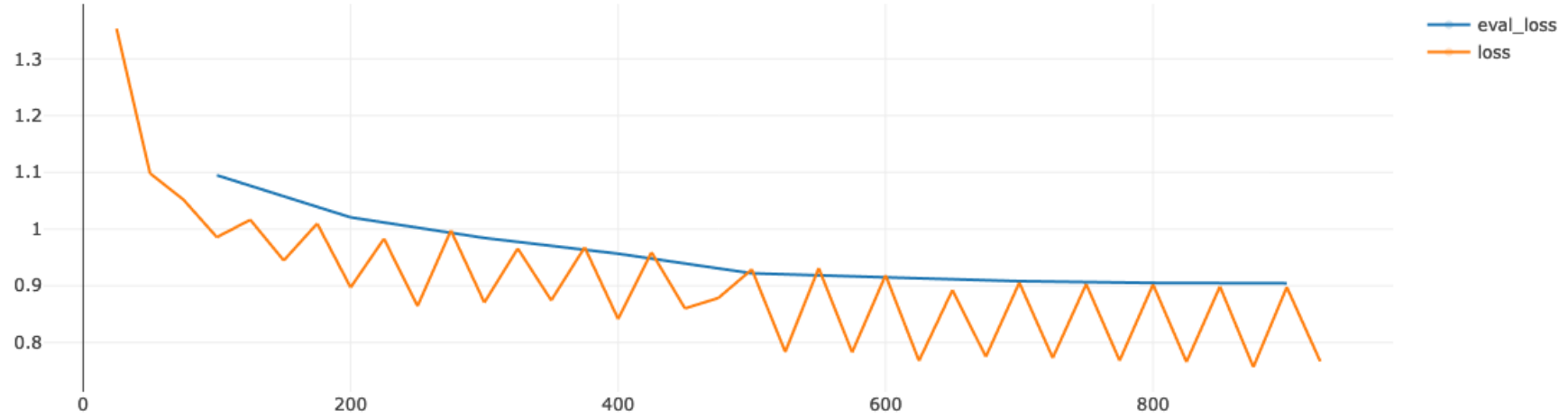
Name	Value
accuracy_MMLU__EM	0.655
accuracy_TruthfulQA__EM	0.889
calibration_MMLU__ECE_10bin	0.278
fairness_MMLU__EM_Fairness	0.623
fairness_TruthfulQA__EM_Fairness	0.889



Xebia | Data

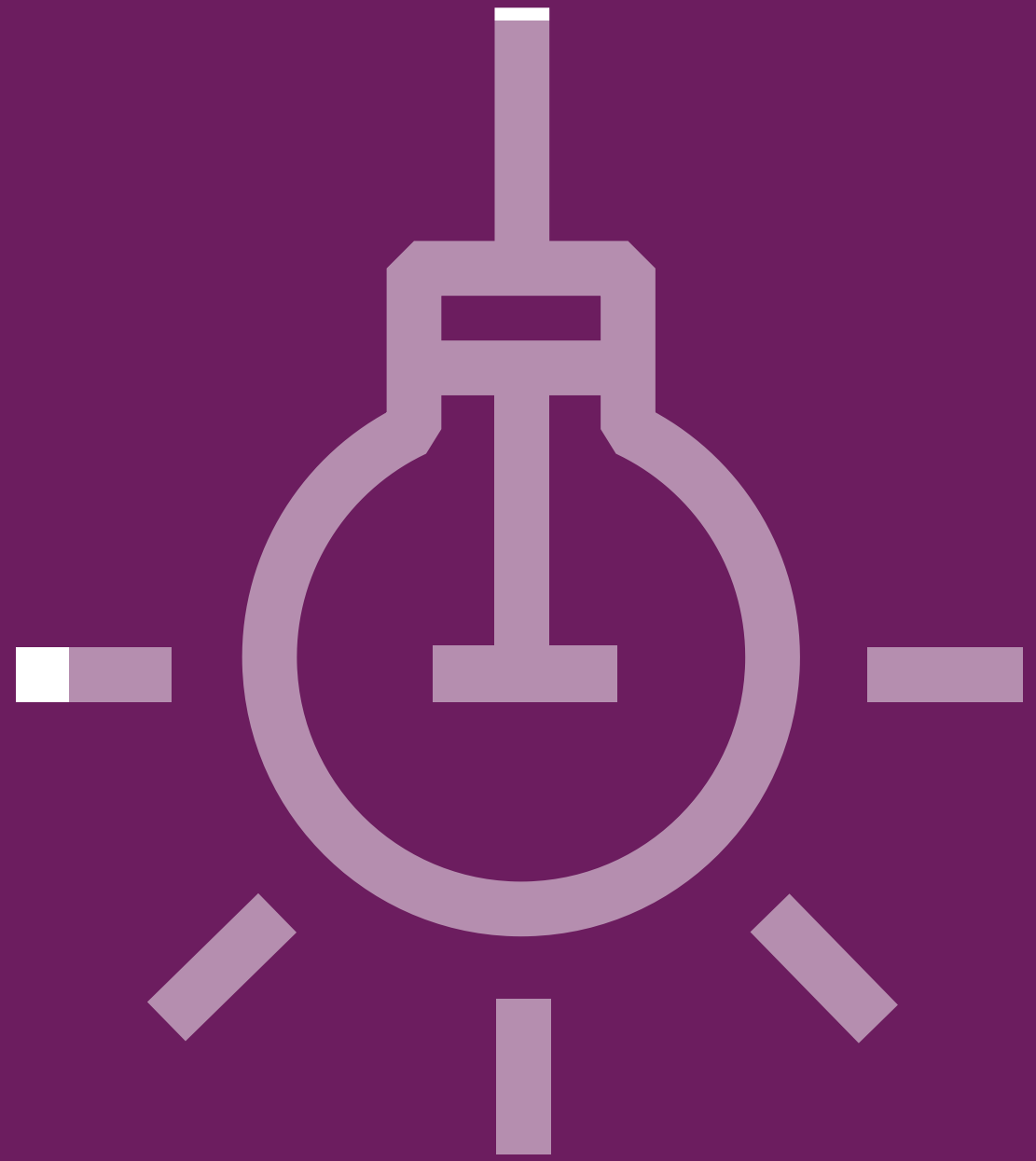
Don’ts

- **Grouping by length**
  - GPU FLOPS utilization is not optimal when training on sequences of different lengths. Text sequences can be padded, but that leads to a lot of wasted compute. Instead, you can group sequences by length and pad each group separately. This way, you can reduce the amount of padding and increase the utilization of the GPU. However, this results in unstable training.
  - The figure below shows one of the earlier training runs. The training loss is very unstable. We expect this to come from the fact that we finetune on a wide range of tasks. Grouping by length in our case also means roughly grouping by task. At the end of a batch, the loss is evaluated for the examples in the batch. This means that if the model performance varies per task, the training loss is going to fluctuate per task.
  - This in itself does not have to be harmful. However, it shows that this process potentially updates the model for a specific task per batch, which can make for unstable training. It might be mitigated by using larger batch accumulation (more observations per forward pass before an update step).

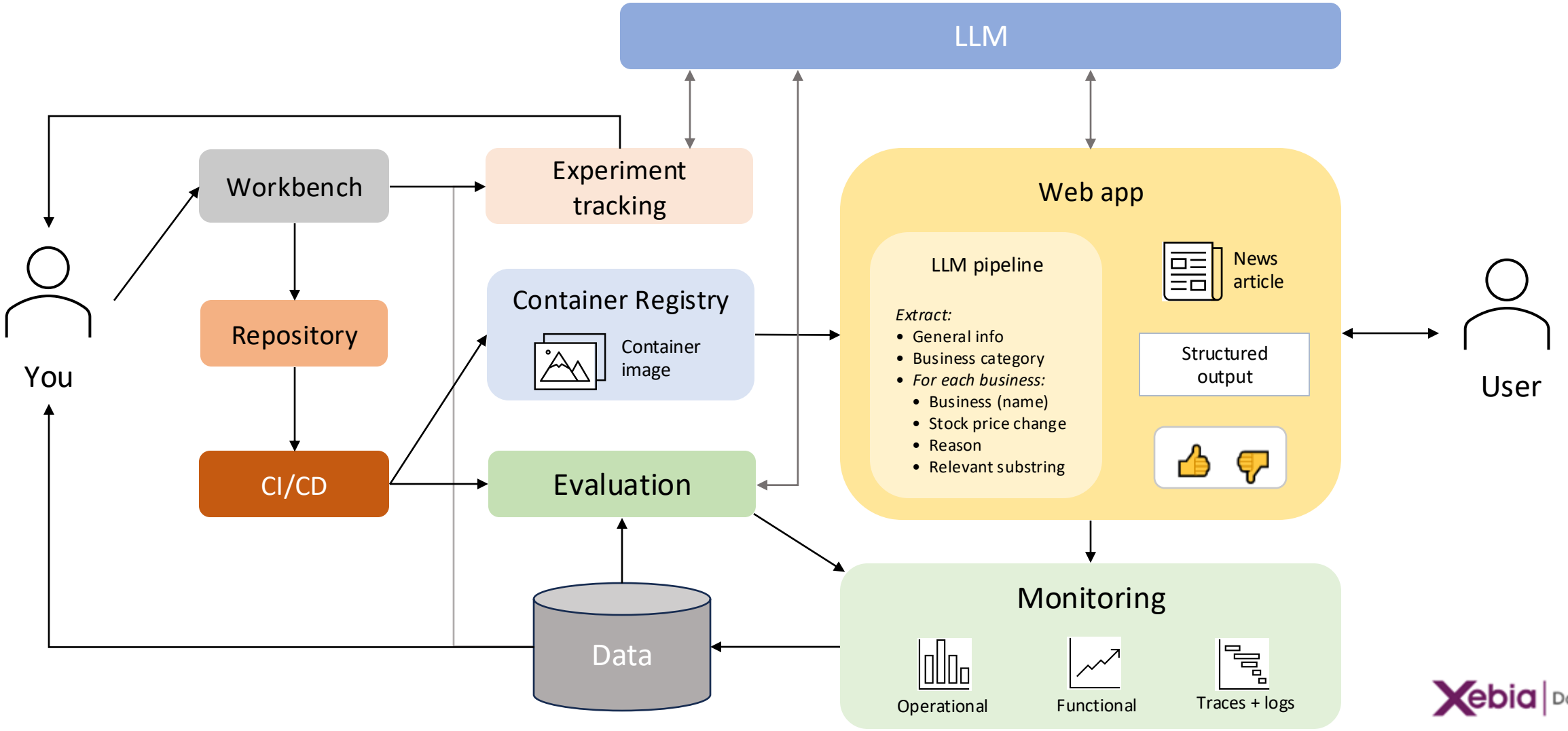


Xebia | Data

Wrap-up



Started from a notebook now we're here!



Xebia | Data

Please submit feedback and help us improve 🙏

https://xebia.ai/llmops-survey

Xebia | Data

Final notes

- Don’t forget to push everything to your branch
- Clone/Fork the Repository
- If you need your code space contingent for sth else, don’t forget to delete your codespace (but all un-pushed / un-committed changes will be lost!)
- I will leave the Azure Resource Group running for another week for you to play around in (pls don’t abuse our compute)
- You can replicate the Azure Resource Group by creating a new one on Rabo compute afterwards (follow the `trainer_docs` in the repository to set this up)

Thanks!