DATA+AI SUMMIT 2022

Designing Better MLOps Systems

Considerations for their nuts and bolts

Chengyin Eng
Senior Data Science Consultant, Databricks

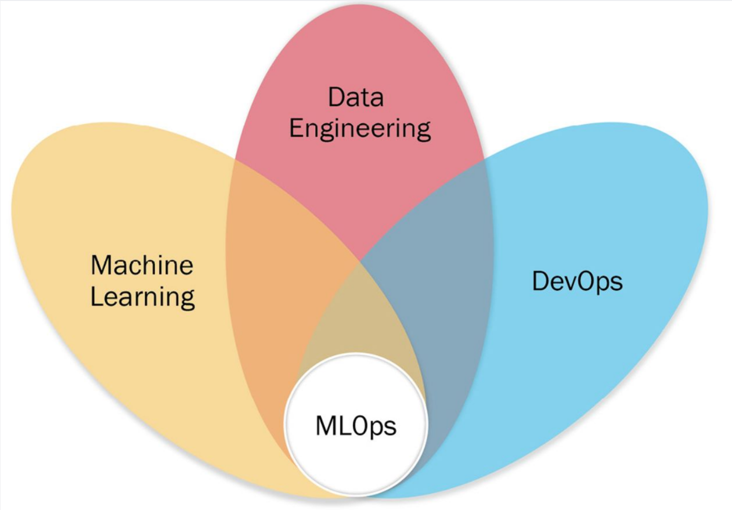
ORGANIZED BY databricks

Chengyin Eng

- Senior Data Science Consultant at Databricks
- Focus: Scalable ML, NLP, end-to-end ML pipeline and architecture design
- Master's in Computer Science
- Bachelors in Environmental Science and Statistics

What's an MLOps system?

Data + features + model + deployment + workflow and resource management



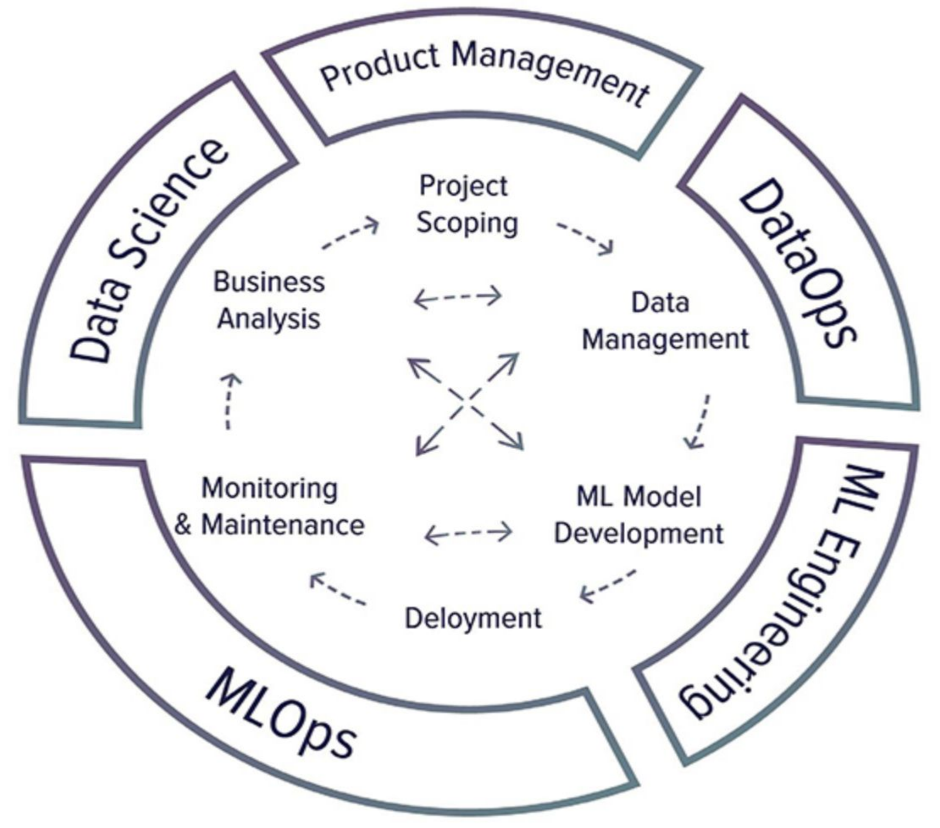
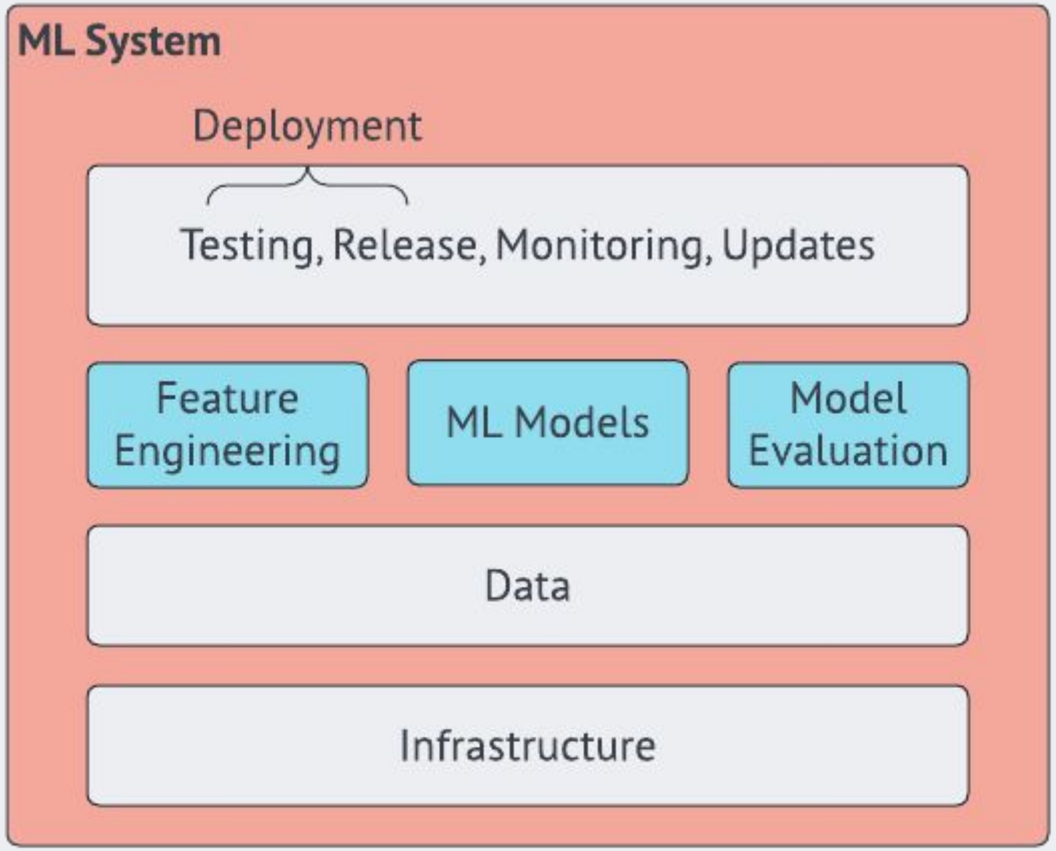


Image sources: Emmanuel Raj, Chip Huyen

What's an MLOps system?

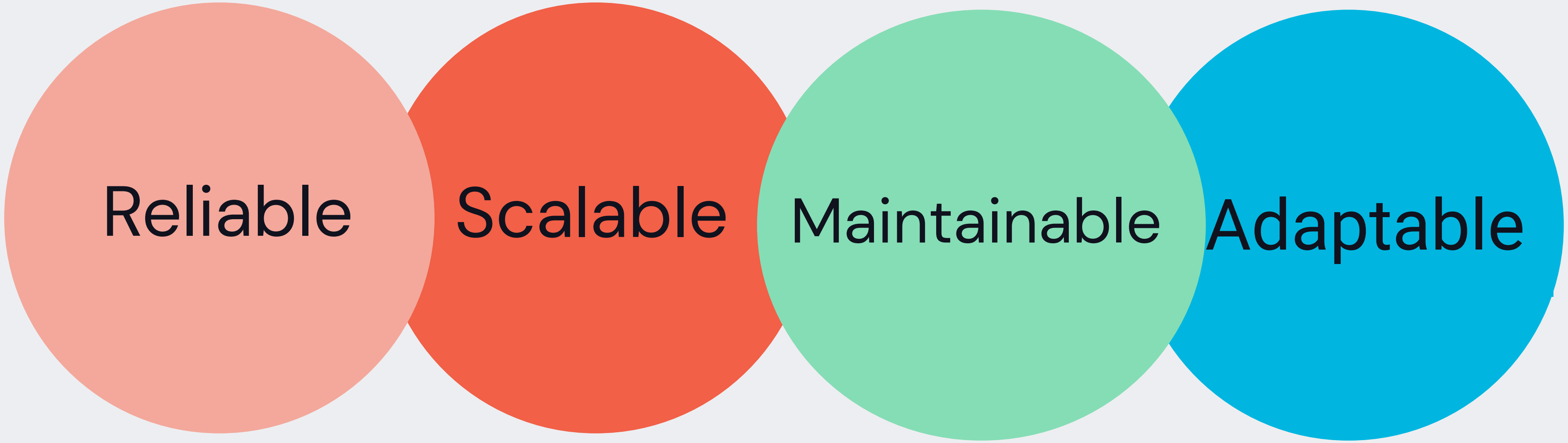
Data + features + model + deployment + workflow and resource management



Adapted from Chip Huyen

MLOps system requirements

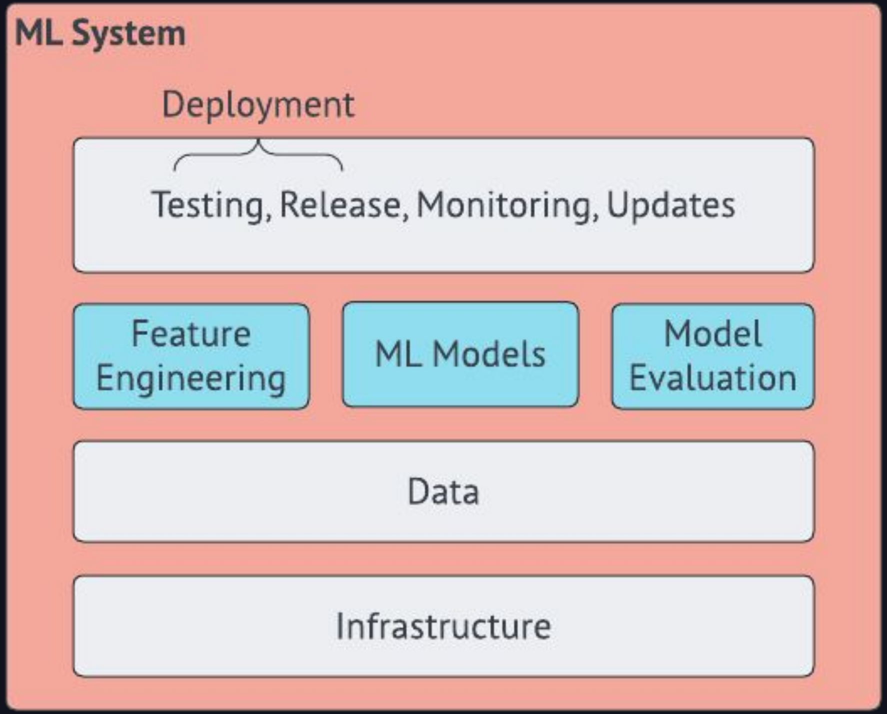
It enables and supports an iterative process



A model is not the product

A model is not the product;

the **system** is the product.

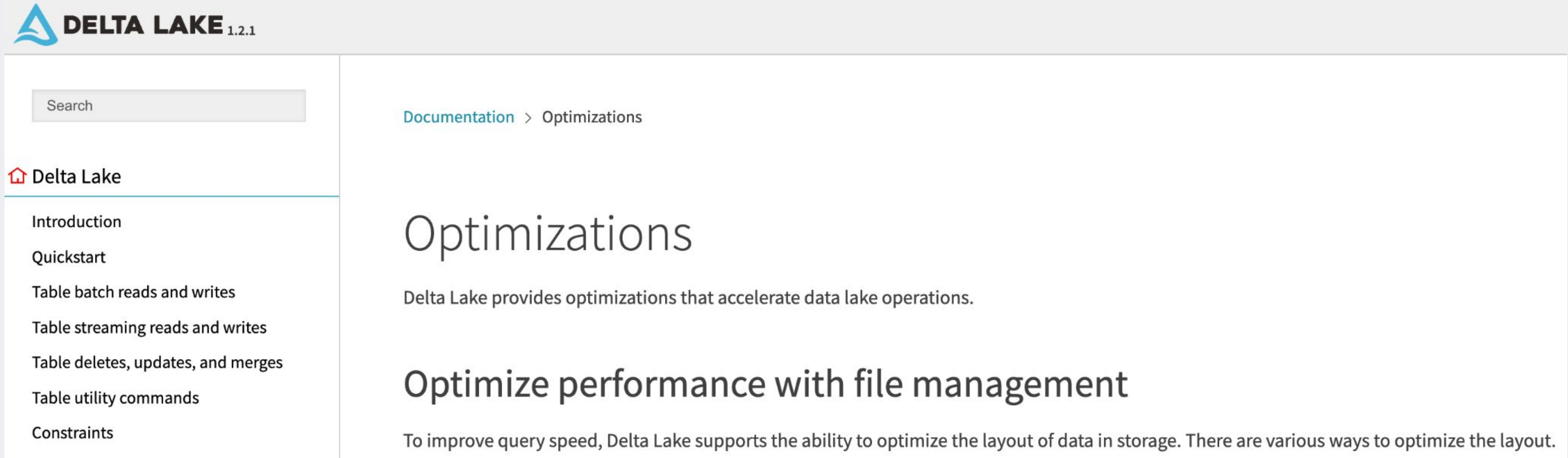


Data

Choose efficient file formats

Consider downstream usage

- File formats
  - Use column-oriented .delta or .parquet, rather than .csv
    - Especially if you access feature subsets
    - Store in partitions



DELTA LAKE

Image source

Split into train/test to avoid data leakage?

Ensure data quality = important first step to a trustworthy model

- If time-dependent, split by time
- Split before scaling features
- Split before imputation
  - Especially if you impute with data statistics
- Check for duplicates before splitting
  - CIFAR 10 and CIFAR 100 have lots of duplicates
    - 3.25 and 10% respectively

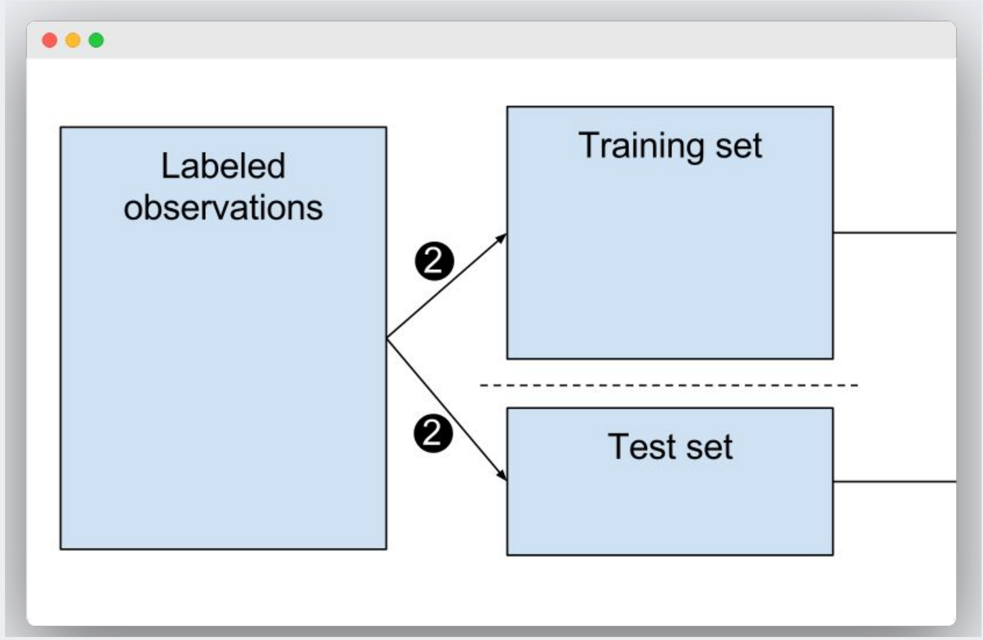


Image source

Data versioning and testing

Allows early detection of inconsistencies

- Keep track of data versions
  - Use file names?
    - Cumbersome and not robust
  - Use open source Delta storage format
  - Helps with debugging production errors due to errors (Delta time travel)

DataFrameReader options

DataFrameReader options allow you to create a DataFrame from a Delta table that is fixed to a specific version of the table.

Python

```python
df1 = spark.read.format("delta").option("timestampAsOf", timestamp_string).load("/tmp/delta/people10m")
df2 = spark.read.format("delta").option("versionAsOf", version).load("/tmp/delta/people10m")
```

For `timestamp_string`, only date or timestamp strings are accepted. For example, `"2019-01-01"` and `"2019-01-01T00:00:00.000Z"`.

Data versioning and testing

Allows early detection of inconsistencies

- Do the schemas match?
  - Including targets
- Does the data have all the expected features?

Change a column type

Python

```
spark.read.table(...) \
  .withColumn("birthDate", col("birthDate").cast("date")) \
  .write \
  .format("delta") \
  .mode("overwrite")
  .option("overwriteSchema", "true") \
  .saveAsTable(...)
```

Features

Myth: More features = better model

Prioritize relevance and maintainability

- Higher number of features can lead to:
    - Overfitting
    - Larger model size = higher memory requirements to serve the model
    - Higher model latency
    - Higher technical debt
        - Useless features

Understand feature importance

Relevance, fairness, and accountability

- Involve SMEs
- Relationship between the features and the target
- Correlation between features
- Tools:
  - built-in feature importance
  - SHAP, LIME, InterpretML
  - Fairlearn

Reduce feature engineering effort

Enable consistency and re-use

- Feature registry / store
  - Informally, feature tables in a data path
  - Formally, a specialized feature store for feature tables
    - Avoid training/serving skew

DELTA LAKE

Features (21)

Feature	Data Type	Consumers			
		Models	Endpoints	Jobs	Notebooks
accommodates	DOUBLE	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference
bathrooms	DOUBLE	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference
bed_type	INTEGER	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference
bedrooms	DOUBLE	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference
beds	DOUBLE	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference
host_total_listings_count	DOUBLE	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference
latitude	DOUBLE	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference
longitude	DOUBLE	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference
minimum_nights	DOUBLE	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference
neighbourhood_cleansed	INTEGER	chengyin_airbnb_fs_model_83966f/1	-	-	01_Distributed_Inference

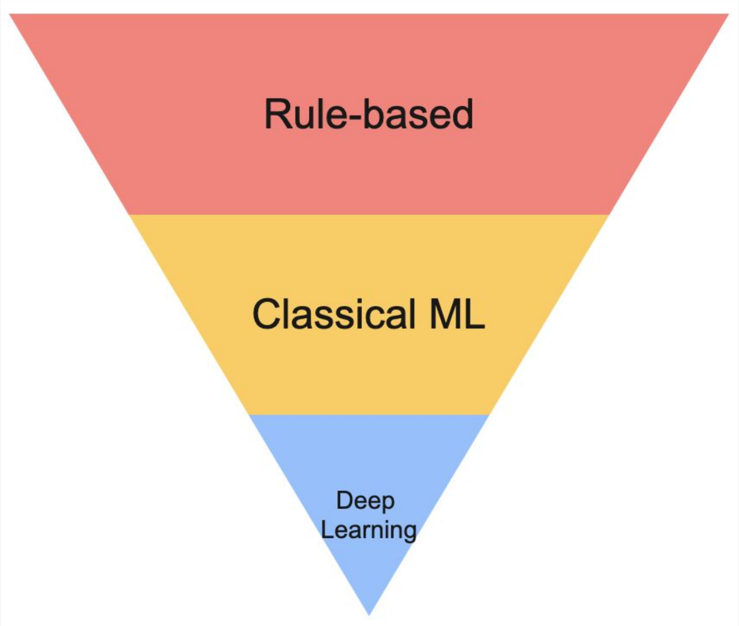
< 1 2 3 >

Models

Inverse pyramid of complexity

Iteration is your best friend

- Start with simple models
  - Evaluate model architectures under similar setups
- Check model assumptions
- Single node first, before distributed



Don’t chase after the shiny stars

Applied ML != Research ML

- SOTA on research data != SOTA on your data
- Is there community support?


François Chollet
@fchollet

The thing is, applied ML engineers have opposite needs to those of researchers. When you do applied ML, you need a framework that's feature-complete, reasonably prescriptive, high-level, that guides you towards industry best practices. And ofc you want it to be production-ready.

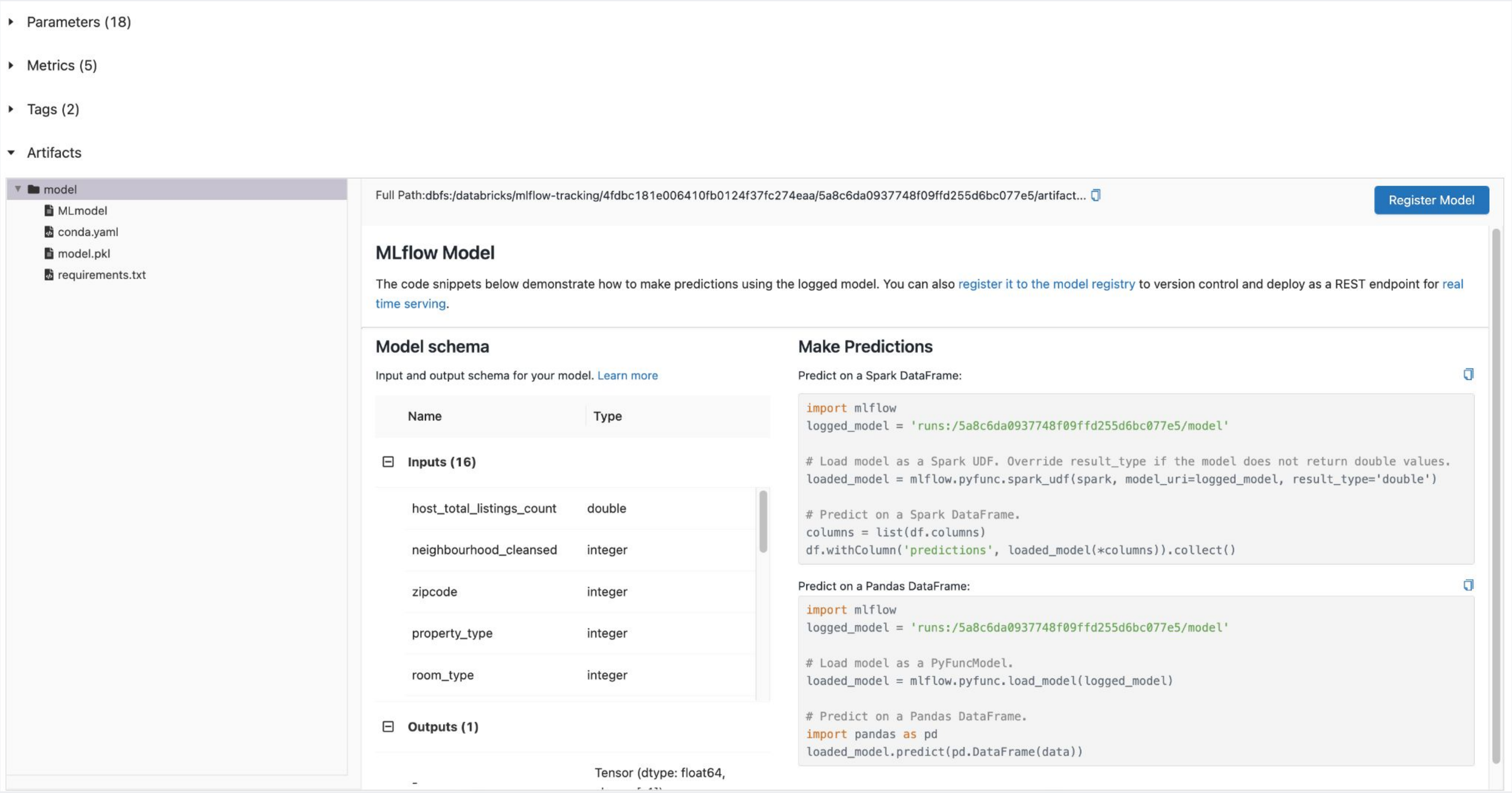
- Researchers want:
  - Minimalistic library
  - Non-prescriptive, low-level
  - Simple mental models, as few features as possible
- Applied ML engineers want:
  - Feature-complete framework
  - Prescriptive, guide towards best practices
  - Easy to use, high-level
  - Stable over time

5:54 PM · Jun 21, 2022 · Twitter Web App

Track and version models for reproducibility

Don’t forget about the environment details

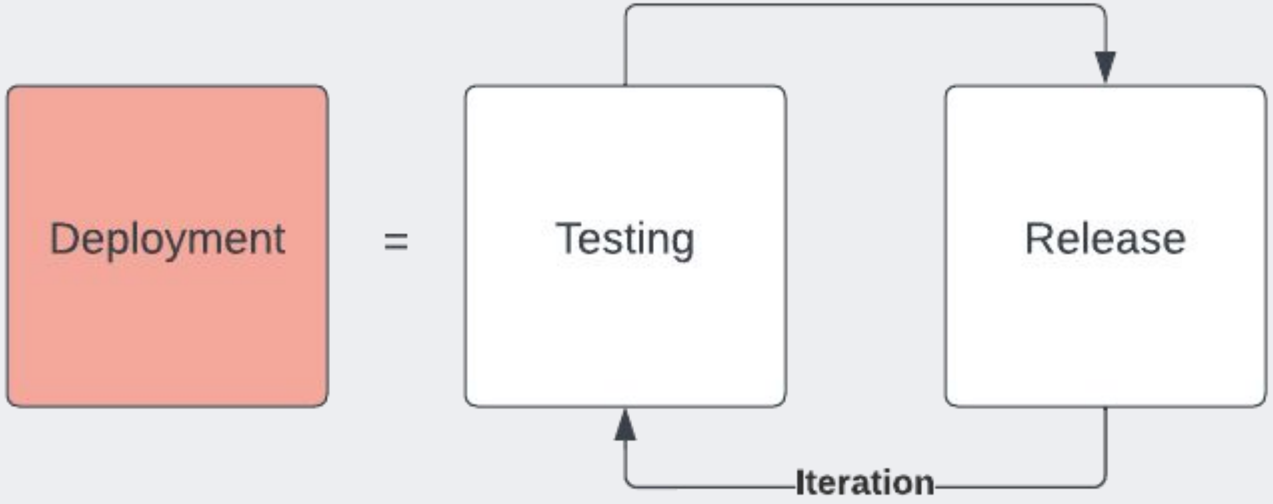
- Hyperparameters
- Evaluation metrics
  - Know your project goal: are there any business metrics?
  - What’s the class imbalance ratio?
- Artifacts
  - Feature importance file, model object, evaluation plots
- Environment details
  - Same package versions in dev and prod; cloud, containers


Parameters (18)
Metrics (5)
Tags (2)
Artifacts
model
MLmodel
conda.yaml
model.pkl
requirements.txt
Full Path:dbfs:/databricks/mlflow-tracking/4fdbc181e006410fb0124f37fc274eaa/5a8c6da0937748f09ffd255d6bc077e5/artifact...
Register Model
MLflow Model
The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry to version control and deploy as a REST endpoint for real time serving.
Model schema
Input and output schema for your model. Learn more
Name
Type
Inputs (16)
host_total_listings_count
double
neighbourhood_cleansed
integer
zipcode
integer
property_type
integer
room_type
integer
Outputs (1)
Tensor (dtype: float64,
Make Predictions
Predict on a Spark DataFrame:
import mlflow
logged_model = 'runs:/5a8c6da0937748f09ffd255d6bc077e5/model'
# Load model as a Spark UDF. Override result_type if the model does not return double values.
loaded_model = mlflow.pyfunc.spark_udf(spark, model_uri=logged_model, result_type='double')
# Predict on a Spark DataFrame.
columns = list(df.columns)
df.withColumn('predictions', loaded_model(*columns)).collect()
Predict on a Pandas DataFrame:
import mlflow
logged_model = 'runs:/5a8c6da0937748f09ffd255d6bc077e5/model'
# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(logged_model)
# Predict on a Pandas DataFrame.
import pandas as pd
loaded_model.predict(pd.DataFrame(data))

Deployment

Deployment = test + release

Test robustness and performance before a model release



Model testing

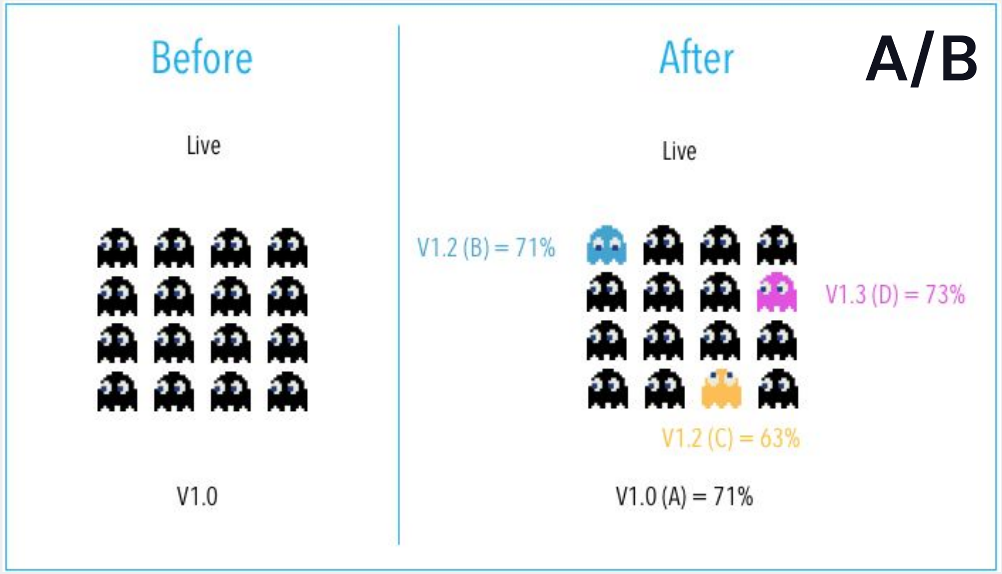
Write up a checklist of required and nice-to-have tests

- Key metrics
- Testing on random data points
- Unit tests for model robustness using real data
- Post-training tests:
  - Invariance test: data augmentation – do the augmented vs. original inputs affect model outputs?
  - Minimum functionality test – does the model perform well on "very easy" samples?
  - Directional expectation – does an increase/decrease in inputs affect model outputs?
- Example tool: deepchecks

Deployment methods

What about acceptable fallbacks? A “dumb” model or the last model?

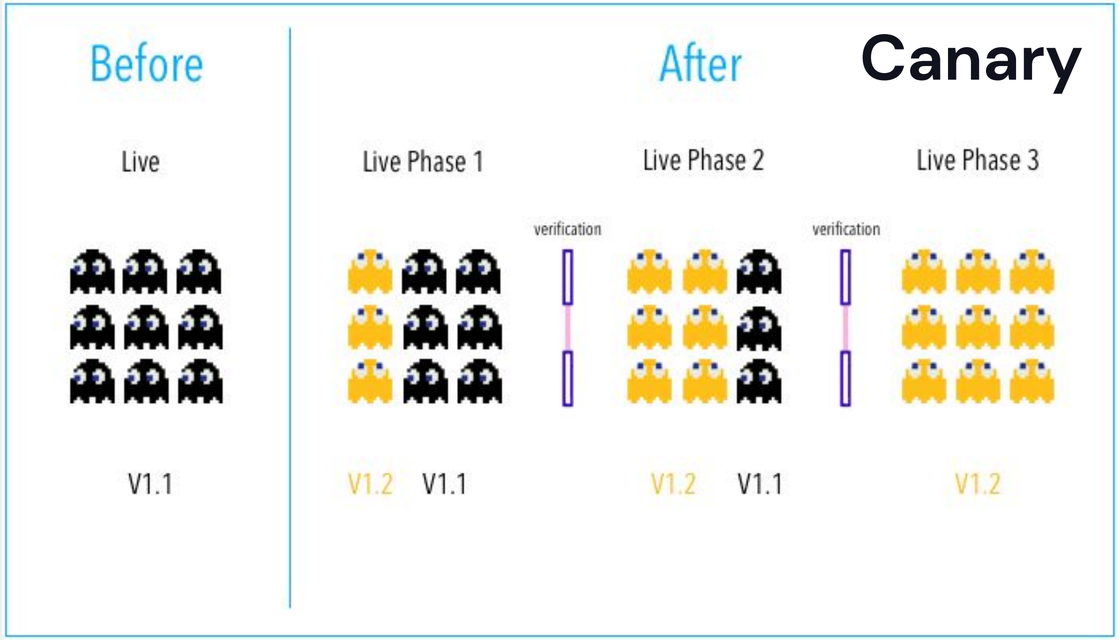
- **Shadow**: Deploy in parallel with the existing model
  - Only serve the existing model’s prediction
- **A/B:** Deploy in parallel.
  - The new model serves a percentage of traffic.



Deployment methods

What about acceptable fallbacks? A “dumb” model or the last model?

- **Canary:** Roll out incrementally to a subset of users
- **Interleave**: Show both new and existing models’ predictions

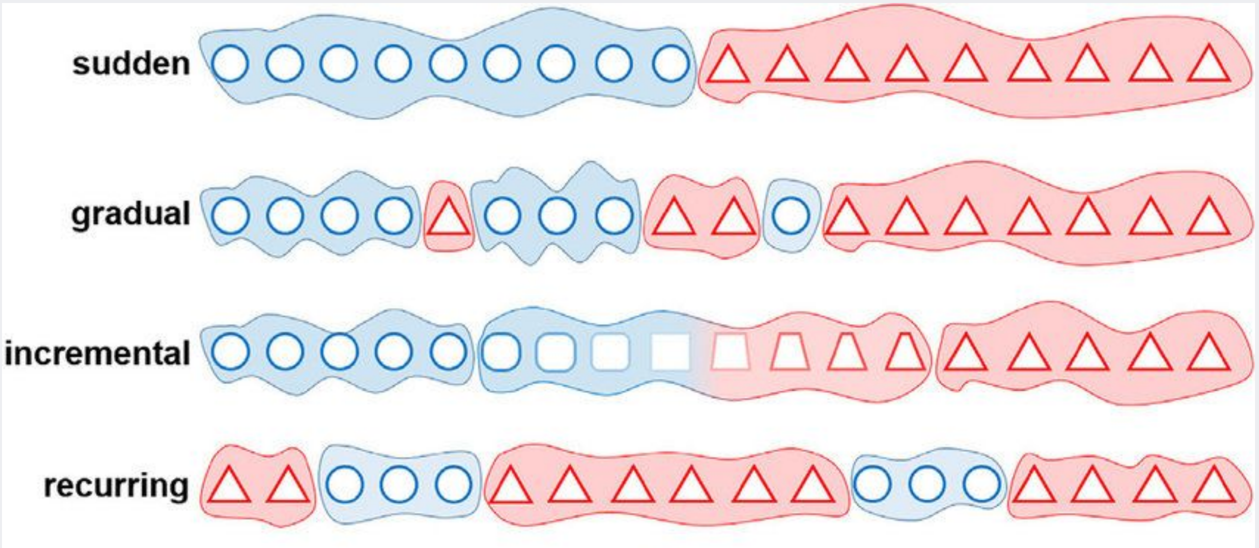


Monitoring and Observability, Retraining

Monitoring

Tracks pre-declared metrics, aka the known-unknowns

- Data drifts
  - Happens more often than assumed (not just black swan events)
  - Types of drift
    - Feature drift
    - Label drift
    - Prediction drift
    - Concept drift



- Establish a trusted baseline data
  - Analyze on subsets/slices – aggregate metrics may not reveal non-benign shifts

Image source

Monitoring

Tracks pre-declared metrics, aka the known-unknowns

- Model evaluation metrics
  - Business KPIs

- Infrastructure
  - # of predictions served
  - CPU/GPU utilization
  - System uptime / downtime

Observability

Understand the **unexpected**, aka the unknown-unknowns

- Observe all the possible behaviors that a system might exhibit
- More fine-grained: you can reconstruct the circumstances
  - Logs with high cardinality: e.g. container ID, which event, who, what function, metadata, etc.
- Sociotechnical: enabled by team

Retraining

Consider the nitty-gritty before a blind retrain

- How?
- What data?
- When?

Retraining

Consider the nitty-gritty before a blind retrain

- How?
  - Train from scratch
  - Train the existing model on new data (stateful training), e.g. DL
- What data?
  - When the data started to shift? More/less recent than that?
- When?
  - Manual or scheduled trigger
  - Based on metrics (automated)
    - Performance-based
    - Volume-based
    - Drift-based

Human oversight is crucial.

Not everything can/should be automated.

Case in point: Avoid degenerate retraining feedback loops

Ingest Bad Data → Retrain →

Image source

Workflow and Resource Management

Infrastructure

Your workflow should guide the infrastructure and tooling decisions

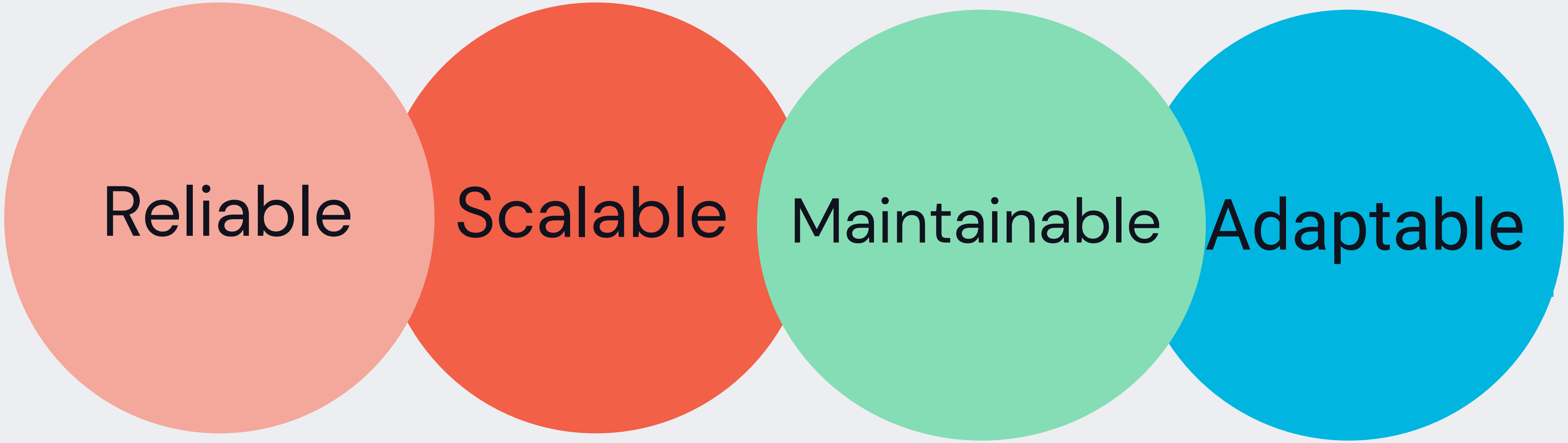
- What's the overall workflow?
  - Do I want to use a model store, feature store, etc.?
  - Do I have a dev/staging/prod environment?

- Resource management
  - Can I afford to deploy a compute-intensive model over the long term?
  - Does it enable reliability and innovation?
  - Do I use containers?
  - Do I buy or build?

- Job orchestration
  - Trigger jobs based on artifacts, schedules, Git, API?

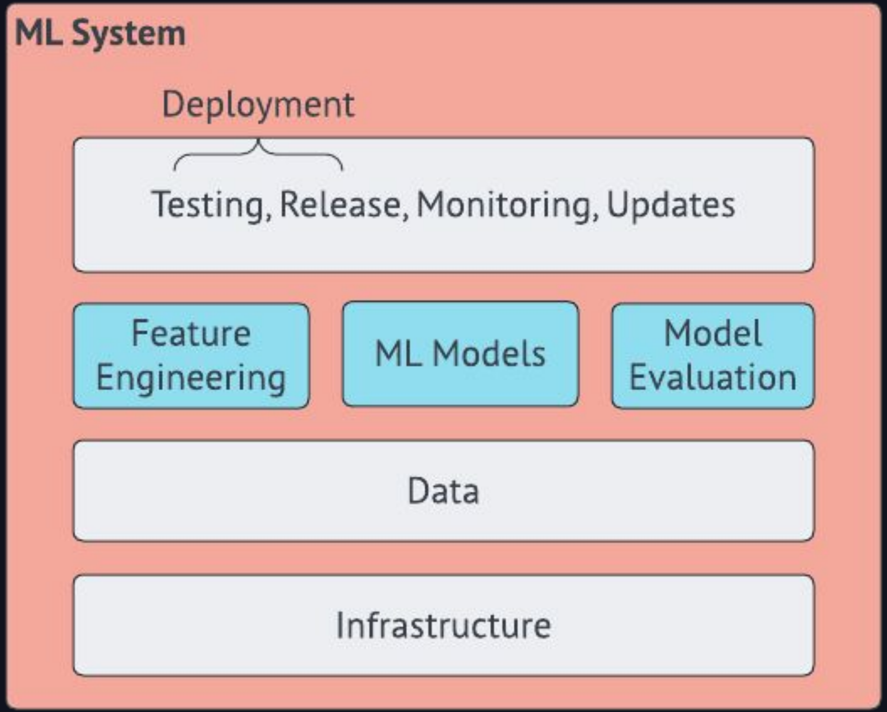
Wrap Up

Each nut and bolt makes up the entire robust MLOps system



A model is not the product;

the **system** is the product.



Best Practices

- Documentation is important!
- Code modularization
  - No commenting out code
  - Maintainability over complexity
  - Even SQL formatting matters!
- Don’t be tool zealots – be agnostic!
- Think about trade-offs
- Conduct regular architectural reviews to address weak spots
- Tracking and versioning helps with debugging outages
- Conduct post-mortems - be truth-seeking and blameless!

Resources

Big-picture questions

Consider these first before writing any code

- Where is the data?
  - What's the data quality?
- Where is the code? Code versioning?
- What's the model impact?
  - Model interpretability?
  - Why are we building this?
- Who consumes the model outputs?
  - How to pass the results to the end users?
- What are the existing pain points?
- How often do you need to run the model?
- What's the desired model latency?

Typical software engineering doesn't 100% apply

ML system = (changing) data + code

- Software engineering: Waterfall -> Agile -> DevOps
- Agile / scrum is not always suitable for ML projects
  - One day sprint could be better than planning out 2-week sprint
  - No rewriting JIRA tickets!

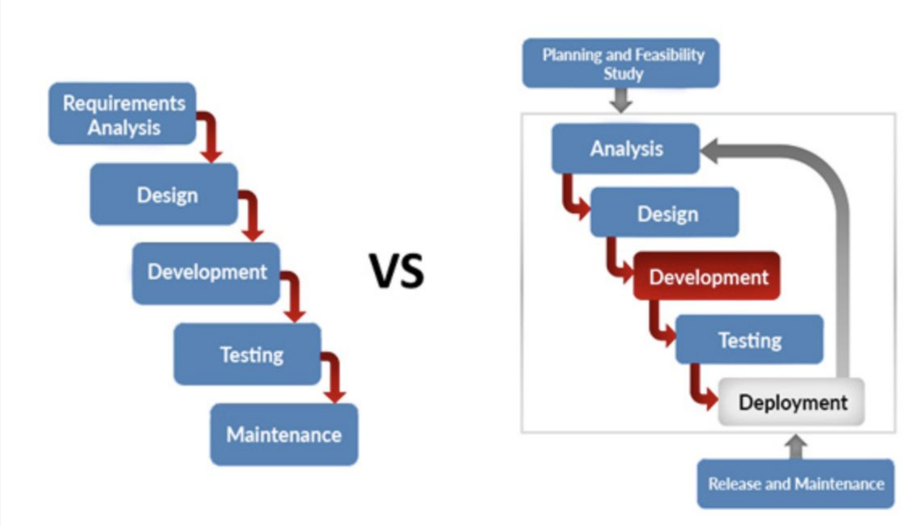


Image source: Emmanuel Raj

References

- Designing ML Systems by Chip Huyen
- Reliable Machine Learning by Chen et al
- Engineering MLOps by Emmanuel Raj
- Eugene Yan: Blog Post on DS and Agile

In-progress:

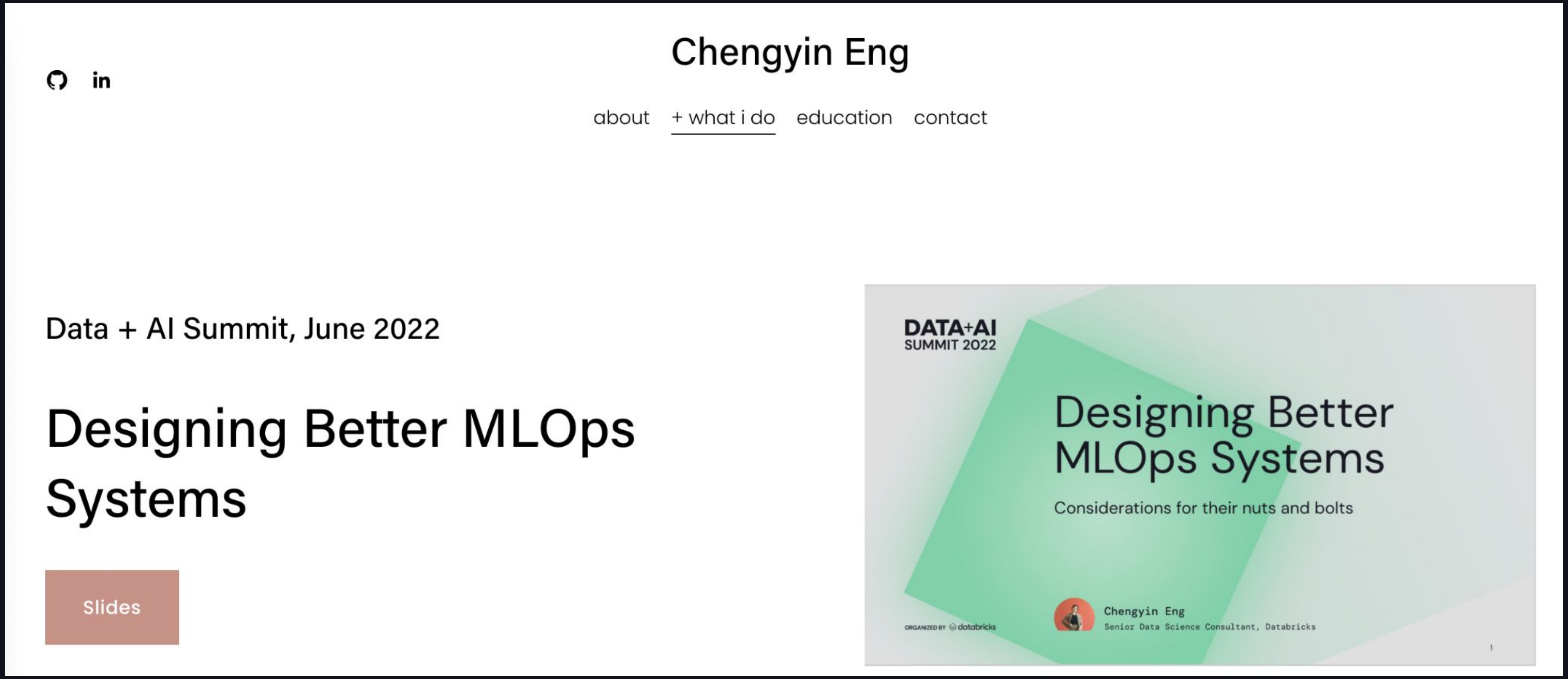
- Effective Python: 90 Specific Ways by Brett Slakin
- Observability Engineering by Majors et al

To-read:

- Data Quality Fundamentals by Moses et al

Slides are on

bit.ly/cy_talks



DATA+AI SUMMIT 2022

Thank You! Questions?

Chengyin Eng

Senior Data Science Consultant