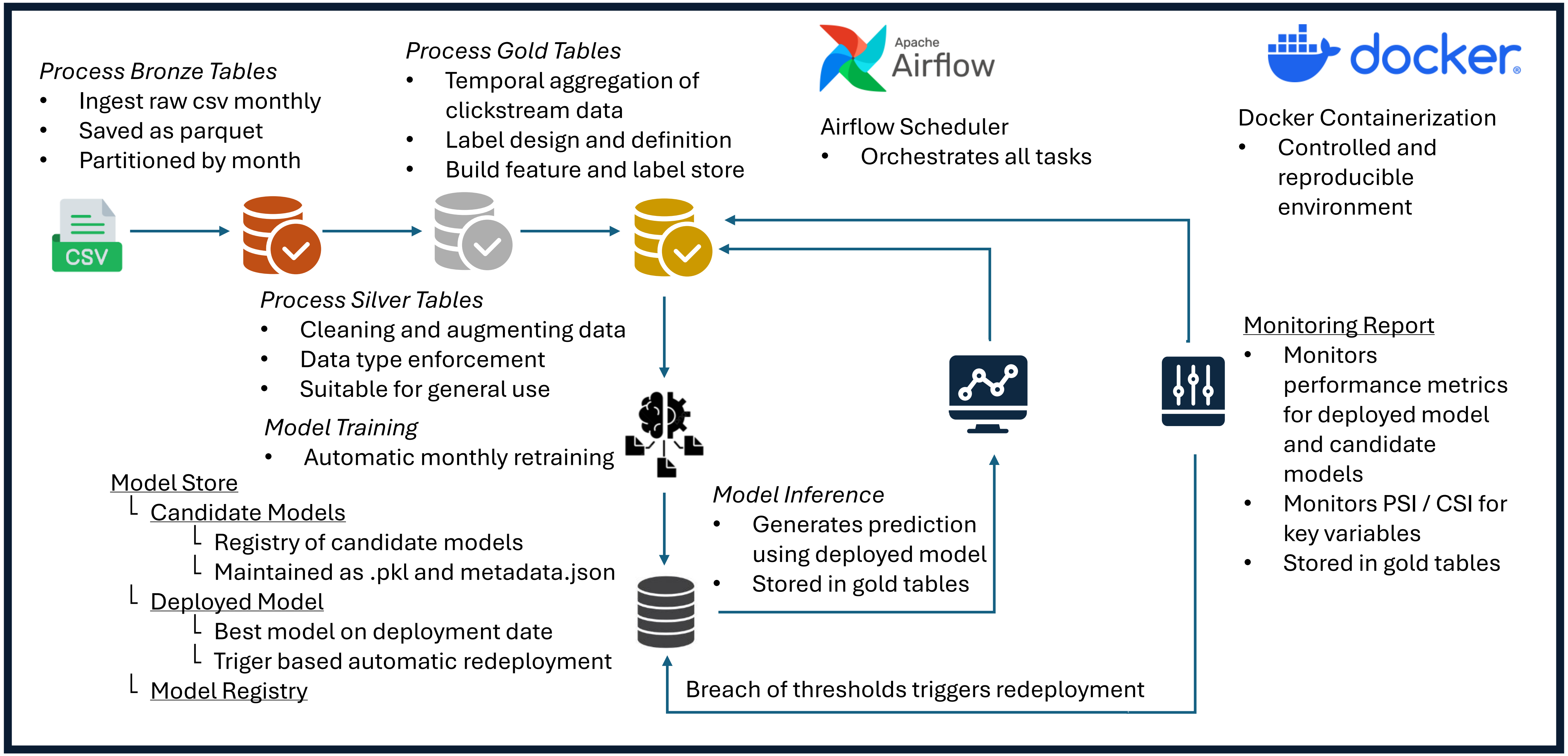
Building an end-to-end ML pipeline

Yue Jun Yuan

End-to-end Model Lifecycle



Process Bronze Tables
- Ingest raw csv monthly
- Saved as parquet
- Partitioned by month

Process Silver Tables
- Cleaning and augmenting data
- Data type enforcement
- Suitable for general use

Process Gold Tables
- Temporal aggregation of clickstream data
- Label design and definition
- Build feature and label store

Airflow Scheduler
- Orchestrates all tasks

Docker Containerization
- Controlled and reproducible environment

Model Training
- Automatic monthly retraining

Model Store
└ Candidate Models
  └ Registry of candidate models
  └ Maintained as .pkl and metadata.json
└ Deployed Model
  └ Best model on deployment date
  └ Triger based automatic redeployment
└ Model Registry

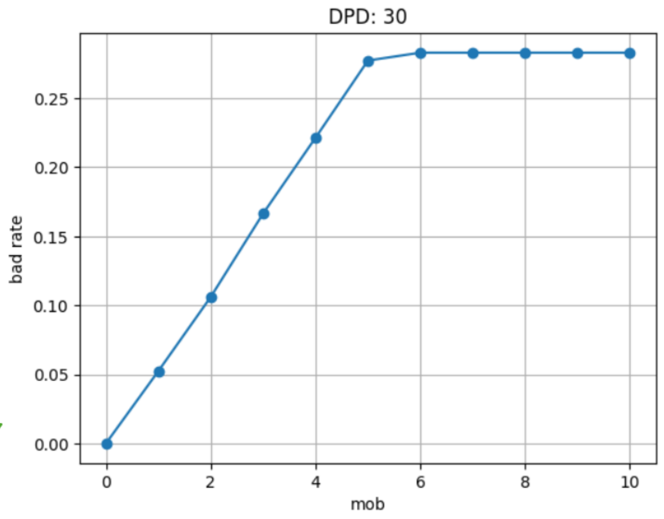
Model Inference
- Generates prediction using deployed model
- Stored in gold tables

Monitoring Report
- Monitors performance metrics for deployed model and candidate models
- Monitors PSI / CSI for key variables
- Stored in gold tables

Breach of thresholds triggers redeployment

Main ETL Considerations

1. Data Quality handling and data type enforcement handled on Silver layer → Fit for general analytics use
2. Temporal aggregation of clickstream data (AVG / SUM L3M, LM)
   - ➢ Implication: No gold layer (feature / label stores) generated for first two months
3. Multi-hot encode "Type of Loan" variable – gold_config.yaml defines list of expected values
   - ➢ Implication: **Ensures consistent schema** enforced across splits
4. Label Definition Considerations
   - ❖ Observation: Bad rate tapers off on the 5th – 6th month → 30DPD_6MOB ✗
   - ➢ Implication: More months required for labels to materialize – reduced amount of data available for training, creates long gap between inference and monitoring
   - ❖ Observation: **97.03%** of customers having any overdue amount over the loan tenor has an overdue amount within first 4 months → **30DPD_4MOB** ✓
   - ➢ Implication: Increasing from 4 MOB to 5 MOB yields very marginal improvement in capture rate. 4 MOB balances trade-off between operational requirements (**timely monitoring**) and label integrity



MOB Period	Capture Rate
5	100%
4	97.03%
3	82.18%

5. Feature and Label store nomenclature captures "snapshot_date" based on when airflow executes the ETL
   - ➢ **Lag month = 4** logic applied when aligning for training / monitoring

feature_store date	label_store date
2023-04-01	2023-08-01
2023-05-01	2023-09-01
…	…
2023-12-01	2024-04-01

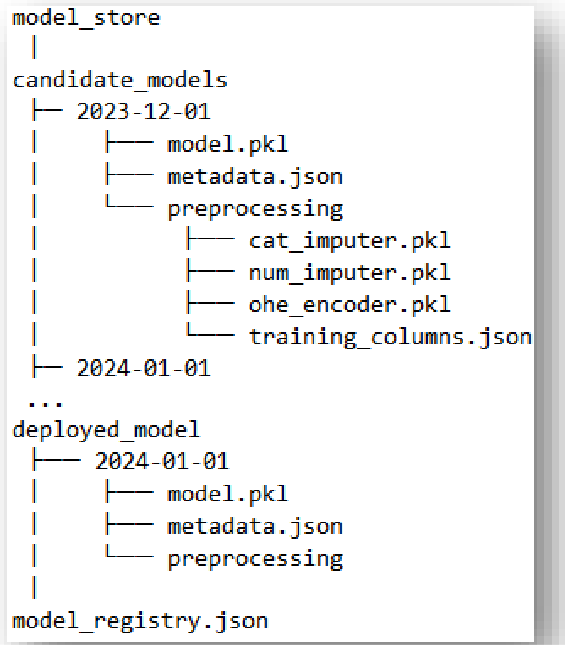
Model Training Considerations

1. Trigger Condition: **Check existence** of feature and label stores → check **sufficiency of data** (including whether all required labels have materialized – 4 months lag due to 30DPD_4MOB)
2. **ML_config.yaml** governs the following: split ranges, variable level preprocessing strategy, model configuration and hyperparameter search space, evaluation metrics
3. Build **preprocessor** (ColumnTransformer object) based on imputation strategy defined in ML_config.yaml
   - ➢ Preprocessor **fits on training set** and transforms all other splits
   - ➢ Underlying pipeline object fit based on train dataset is saved along with model in model_store to be used during inference
4. Train and output extensive **metadata** including: model version, training window, best hyperparameters, evaluation metrics, feature importance
5. Once there is sufficient data, airflow scheduler continues to **execute model training every month** → shifting training window to capture more recent data
   - ➢ Best model from each month is saved as candidate models
6. Central Model Registry is updated every month with key metadata of each model

Model Deployment → 2024-06-01 → What actually happens?

Promote best model task is triggered

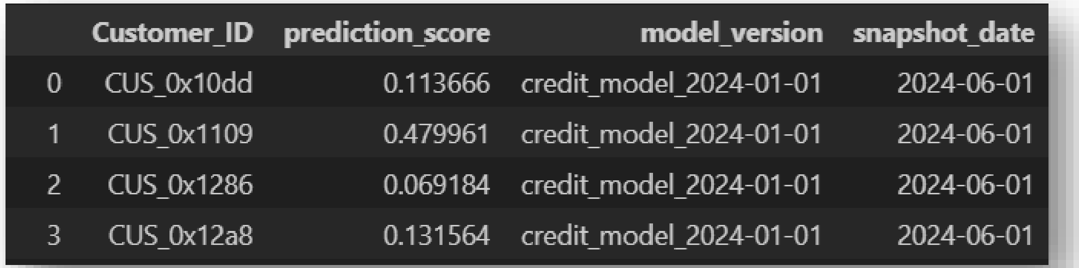
- ➢ searches metadata of all available candidate models and **“deploys” model** with best **OOT AUC**
- ➢ Deployed model will start being **used for inference**



```
model_store
 |
candidate_models
 ├─ 2023-12-01
 |     ├── model.pkl
 |     ├── metadata.json
 |     └── preprocessing
 |           ├── cat_imputer.pkl
 |           ├── num_imputer.pkl
 |           ├── ohe_encoder.pkl
 |           └── training_columns.json
 ├─ 2024-01-01
 ...
deployed_model
 ├── 2024-01-01
 |     ├── model.pkl
 |     ├── metadata.json
 |     └── preprocessing
 |
model_registry.json
```

Model Inference Considerations

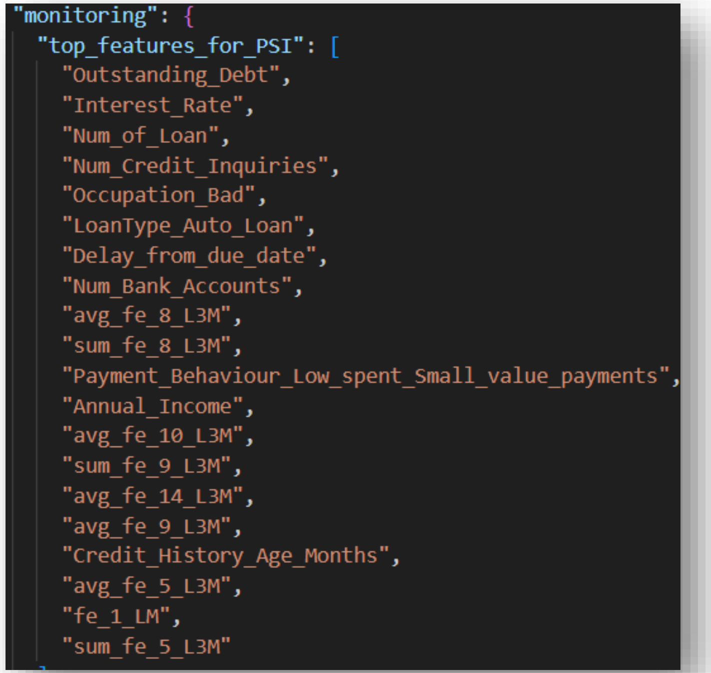
1. Trigger condition: Inference task will skip unless **current date >= deployment date**
2. Loads deployed model along with **associated preprocessors and list of columns** used during training
3. Which customers to make prediction: **“loan_start_date” == “snapshot_date”**
4. Apply preprocessing transformation to filtered feature set and enforce training schema → prevents any schema drift from causing an exception
5. Output **probability of default** for each customer and save to gold layer on datamart



	Customer_ID	prediction_score	model_version	snapshot_date
0	CUS_0x10dd	0.113666	credit_model_2024-01-01	2024-06-01
1	CUS_0x1109	0.479961	credit_model_2024-01-01	2024-06-01
2	CUS_0x1286	0.069184	credit_model_2024-01-01	2024-06-01
3	CUS_0x12a8	0.131564	credit_model_2024-01-01	2024-06-01

Model Monitoring Considerations

1. Trigger condition: First monitoring date set to be 4 months after deployment date → **Allows labels to materialize**
   - Could potentially begin monitoring earlier only for underlying data related metrics (PSI / CSI)
   - Decision aligned with business to provide first monitoring report 4 months after deployment as model performance not expected to degrade too quickly, allows for **standardized monitoring report formats**
2. How to define key variables for monitoring:
   - Alignment sought with business based on **feature importance** (by gain) generated by earlier candidate models (prior to deployment) as captured in model metadata → shortlist defined in **monitoring_config.yaml**



```
"monitoring": {
  "top_features_for_PSI": [
    "Outstanding_Debt",
    "Interest_Rate",
    "Num_of_Loan",
    "Num_Credit_Inquiries",
    "Occupation_Bad",
    "LoanType_Auto_Loan",
    "Delay_from_due_date",
    "Num_Bank_Accounts",
    "avg_fe_8_L3M",
    "sum_fe_8_L3M",
    "Payment_Behaviour_Low_spent_Small_value_payments",
    "Annual_Income",
    "avg_fe_10_L3M",
    "sum_fe_9_L3M",
    "avg_fe_14_L3M",
    "avg_fe_9_L3M",
    "Credit_History_Age_Months",
    "avg_fe_5_L3M",
    "fe_1_LM",
    "sum_fe_5_L3M"
```

Model Monitoring Considerations - Continued

1. Thresholds and action points as agreed with business:
2. **AUC** measures model's ability to discriminate between defaulters and non-defaulters → directly **aligned with business use** of output
3. **Gini** (despite being a function of AUC) is more commonly understood by business and regulators → **0 to 1 scale**
4. **KS Statistic** measure how well model separates the defaulters and non-defaulters → **discriminatory power**

Priority	Metric	Threshold	Actions if breached
P0	AUC	0.70	Best candidate model (OOT_AUC) from auto-ML triggered for redeployment; DS team alerted to review
P0	Gini	0.50	
P1	KS	0.4	Two consecutive breaches triggers redeployment of best candidate models (OOT_AUC) from auto-ML; DS team alerted to review
P2	F1	No defined thresholds for monitoring	Metrics provided as supplementary information
P3	Precision		
P3	Recall		
P4	Accuracy		

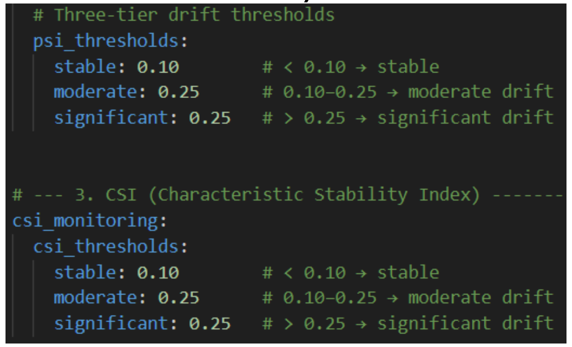
Breach of P0 / P1 metrics → What happens?

1. Monitoring Reporting maintains separate flag indicators tracking if each metric's thresholds are breached
    1. **Breach of AUC or Gini** thresholds result in "breach_flag" marked as true in the same month
    2. **Breach of KS** → monitoring pipeline checks if previous month KS was also breached to determine "breach_flag" indicator
2. At the start of each month, promote_best_model task **checks previous month's monitoring report** for breach_flag indicator → If breach detected, **current deployed model is decommissioned** and **best performing candidate model is deployed** → Deployment event **logged** with reason and **alerts** sent out to DS team
3. Newly deployed model is also reflected in Model Registry

Model Monitoring Report

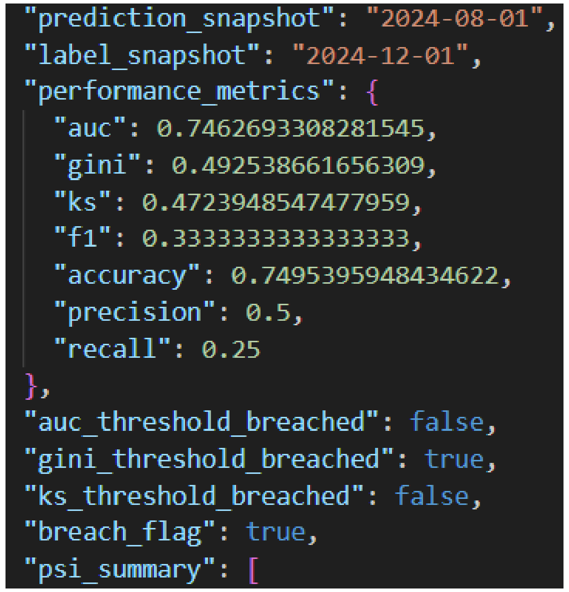
1. Population distribution of key variables on reference month (latest training window of deployed model) overlayed with monitoring snapshot (relative to prediction month)

- ➢ PSI / CSI callouts for each variable color-coded to emphasise severity
- ➢ PSI and CSI can be an **early signal** to degradations in model performance → Potentially worth considering separation of PSI / CSI metrics from performance metrics to get "earlier" signals



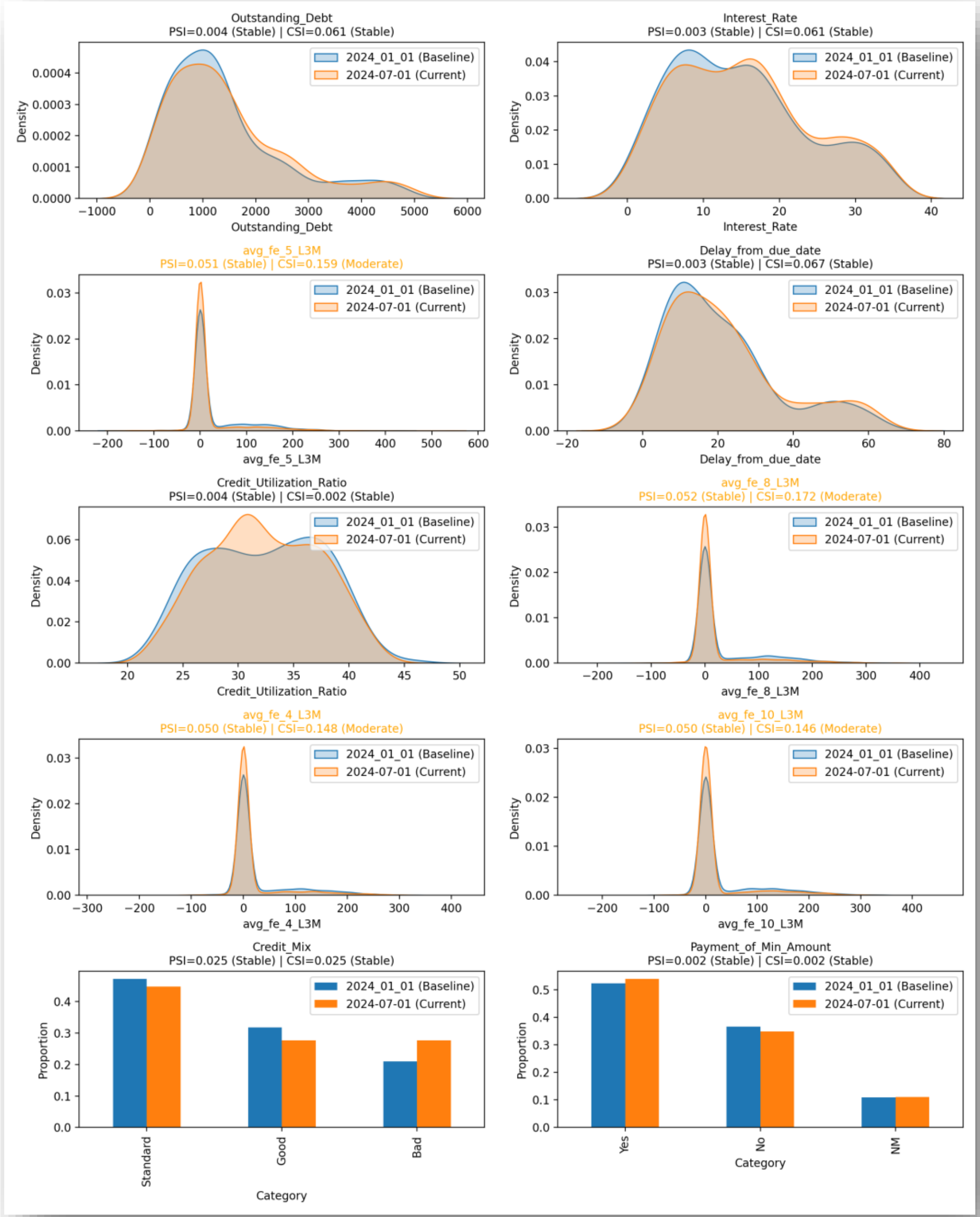
```
  # Three-tier drift thresholds
  psi_thresholds:
    stable: 0.10         # < 0.10 → stable
    moderate: 0.25       # 0.10–0.25 → moderate drift
    significant: 0.25    # > 0.25 → significant drift

# --- 3. CSI (Characteristic Stability Index) -------
csi_monitoring:
  csi_thresholds:
    stable: 0.10         # < 0.10 → stable
    moderate: 0.25       # 0.10–0.25 → moderate drift
    significant: 0.25    # > 0.25 → significant drift
```

❑ monitoring_config.yaml snippet



```
"prediction_snapshot": "2024-08-01",
"label_snapshot": "2024-12-01",
"performance_metrics": {
  "auc": 0.7462693308281545,
  "gini": 0.492538661656309,
  "ks": 0.4723948547477959,
  "f1": 0.3333333333333333,
  "accuracy": 0.7495395948434622,
  "precision": 0.5,
  "recall": 0.25
},
"auc_threshold_breached": false,
"gini_threshold_breached": true,
"ks_threshold_breached": false,
"breach_flag": true,
"psi_summary": [
```

❖ Visualizations from **2024-07-01** monitoring report highlighted **moderate drifts** in a few key variables, and **2024-08-01** report showed a **breach in KS threshold**

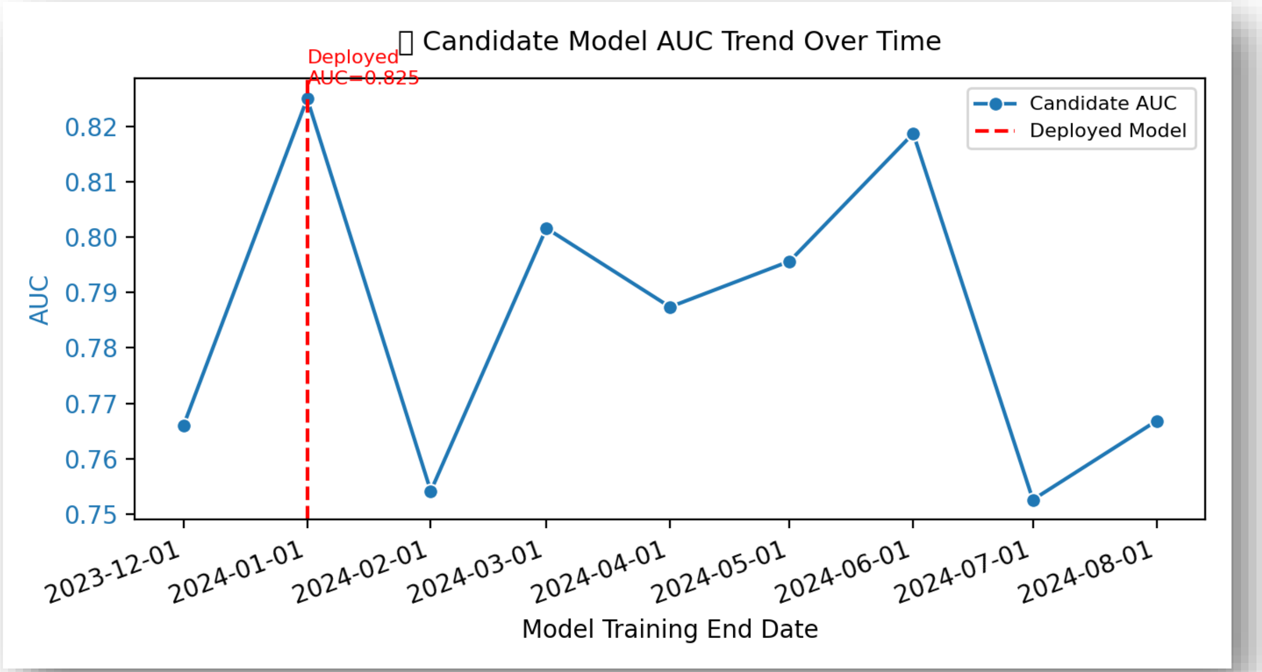


❑ Monitoring Report JSON (2024-08-01) ❑ Monitoring Report Dashboard (2024-07-01)

Model Monitoring Report - Continued

1. Performance of all candidate models (including deployed model) also tracked over time as part of model (pipeline) monitoring
   - ➢ Provides a **pre-emptive signal** in explaining any degradation in performance of the deployed model picked up during monitoring → Similar drop in performance detected in newly trained candidate model could point to potential **concept drift**

2. Conversely, an uptrend in candidate model performance could also indicate a raise in the **performance ceiling** → DS team may suggest **manual redeployment** of model despite no breaches in threshold



Overall Pipelining Considerations

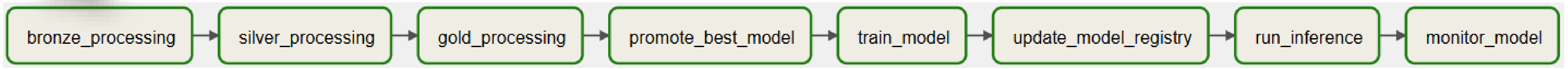
Storing Configuration / Parameters in YAML

- Separation of concerns – separation of *how things are done* from *what to do*
  - Modularity and scaling → Bank introduces new loan product → Simple YAML file update
- Governance and Change Management – "code freeze, config agile"
  - Reduced bureaucracy → code changes (script logic) post deployment requires a much more stringent set of change controls relative to parameter change (YAML update) → Improved auditability and version control

Pipeline logging system

- Pipeline logging captures all script execution, completions, and detailed actions taken within each task
  - Timestamped log files created on each run enables traceability and easier debugging

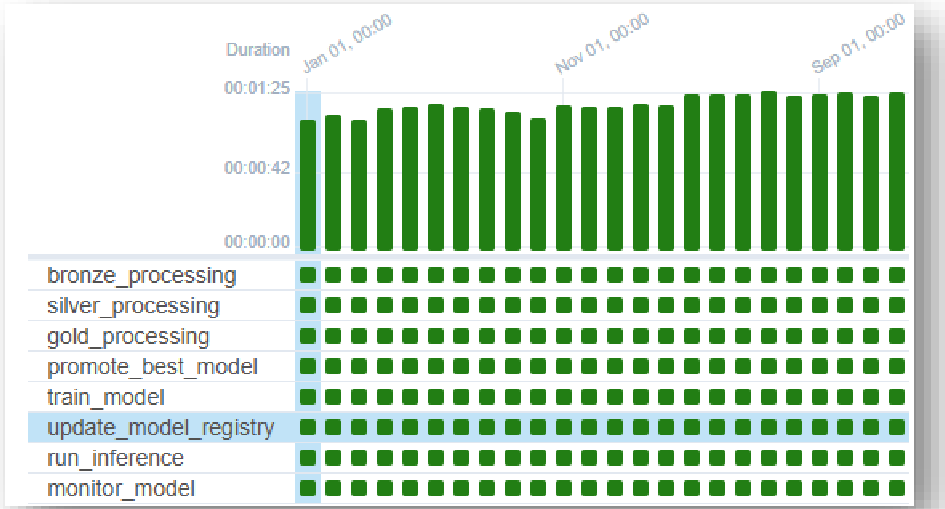
Single-sequence workflow

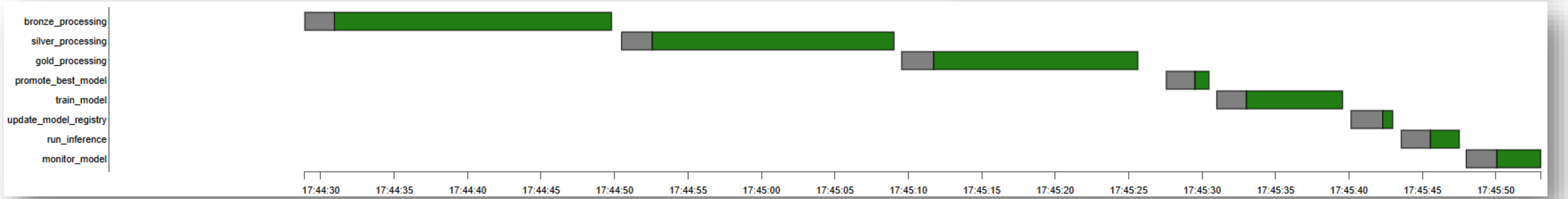


- Tasks executed in strict sequential order from data extraction to model monitoring
- Simplicity in design and predictable execution → dependencies handled within each script
- promote_best_model picks from pool of existing candidate models (excluding the one trained on the same month) for deployment → Model deployment require governance procedures that requires time

Pipeline Evaluation

1. Each month's run takes approximately 1 minute 25 seconds to complete entire process from ETL, training, inference, to monitoring
   - ➢ Earlier months takes a slightly shorter time (1 minute) as tasks such as model training / inference / monitoring have not kicked in





2. Gnatt chart analysis shows that bulk of time taken relates to the ETL process
   1. ETL process requires SparkContext initialization
   2. Processing Bronze tables takes the longest time despite having the least processing (only filtering), this is due to the bronze processing task interacting directly with raw CSV files which involves inferring schema, parsing text into numeric and datetime formats, etc → Silver and Gold processing despite much more work being done works directly with parquet files