ML Model Runtime Prediction

Methodology, Design Decisions, and Justification

1. Overview of the Approach

The goal of this phase of the project is to train a machine learning model that can **predict execution time of a workload on a target production environment**, given:

1. Runtime behavior observed on a developer machine
2. Resource configuration of the target production machine

This is framed as a **supervised regression problem**, where the target variable is the execution time (`runtime_sec`), and the input features describe workload characteristics, system utilization, and hardware capacity.

---

2. Data Preprocessing and Feature Preparation

2.1 Removal of Non-Predictive Identifiers

The dataset contains identifier columns such as:

- `run_id`
- `batch_id`

These columns were removed from the feature set because:

- They do not have a causal relationship with execution time
- Including them would introduce noise or bias
- They could lead to data leakage if used directly by the model

---

2.2 Handling of Categorical Features

- `workload_type` was encoded using one-hot encoding to allow the model to learn different performance behavior patterns across workload categories (ML, DB, WEB).
- `workload_name` was intentionally removed and replaced by `workload_complexity`, a numeric abstraction that captures relative computational cost while avoiding overfitting to specific workload labels.

- `disk_type` was removed in favor of `disk_speed_class`, which preserves the ordinal performance relationship between HDD, SSD, and NVMe storage.

2.3 Use of Engineered Features

Several derived features were included to better represent system behavior:

- **Effective CPU usage** captures absolute compute consumption instead of relative percentages.
- **Memory pressure** reflects how close a workload is to exhausting available memory.
- **I/O intensity** distinguishes CPU-bound workloads from disk-bound workloads.

These engineered features help tree-based models capture non-linear system behavior more accurately.

---

3. Train / Validation / Test Split (Correct Method)

3.1 Why Batch-Based Splitting Is Necessary

The dataset was collected in **batches**, where each batch represents:

- A specific experimental run
- A fixed environment configuration
- A group of correlated executions

If rows were split randomly, samples from the same batch could appear in both training and test sets, leading to:

- Leakage of environment-specific noise
- Overly optimistic evaluation results
- Poor real-world generalization

3.2 Correct Splitting Strategy

To prevent this, the dataset was split **by batch**, not by individual rows.

- Entire batches were assigned to exactly one split
- This ensures no environment overlap between splits
- The test set simulates completely unseen production environments

3.3 Split Ratio Used

- **Training set:** 70% of batches
- **Validation set:** 15% of batches
- **Test set:** 15% of batches

This mirrors real deployment scenarios where predictions are made for environments not seen during training.

---

4. Baseline Models (Mandatory for Research)

Baseline models were implemented to demonstrate that advanced models provide meaningful improvement.

4.1 Naive Baseline

Approach:
Always predict the mean runtime from the training set.

Purpose:

- Establishes a minimum performance benchmark
- Any useful model must outperform this baseline

4.2 Linear Regression

Why Linear Regression was used:

- Highly interpretable
- Reveals basic relationships between features and runtime
- Serves as a lower-bound machine learning baseline

Limitations:

- Assumes linear relationships
- Cannot model threshold effects or complex interactions
- Underperforms for resource contention and I/O-heavy workloads

---

5. Advanced Models (Main Results)

5.1 Decision Tree Regressor

Strengths:

- Captures non-linear thresholds (e.g., memory pressure tipping points)
- Easy to visualize and explain
- Useful as an intermediate step between linear and ensemble models

Limitations:

- Prone to overfitting
- Lower generalization compared to ensembles

5.2 Random Forest Regressor (Primary Model)

The Random Forest Regressor was selected as the **main model** due to its strong balance between accuracy, robustness, and interpretability.

Key advantages:

- Handles non-linear relationships naturally
- Learns interactions between CPU, memory, and I/O features
- Robust to noise and outliers
- Generalizes well across unseen environments
- Provides feature importance for explainability

Random Forest averages predictions across multiple decision trees, reducing variance and improving stability compared to a single tree.

5.3 Gradient Boosting / XGBoost (Optional Best Model)

Why considered:

- Often achieves the highest prediction accuracy
- Learns subtle cross-feature interactions
- Corrects errors iteratively

Trade-off:

- More complex to tune
- Slightly less interpretable than Random Forest

This model is suitable when maximum accuracy is prioritized over interpretability.

---

6. Model Evaluation Metrics

All models were evaluated using standard **regression metrics**:

6.1 Mean Absolute Error (MAE)

- Measures average absolute difference between predicted and actual runtime
- Easy to interpret in real-world units (seconds)

Interpretation:
"On average, the prediction is off by X seconds."

6.2 Root Mean Squared Error (RMSE)

- Penalizes large errors more than MAE
- Highlights worst-case prediction failures

Interpretation:
Higher RMSE indicates presence of large prediction errors.

6.3 R^2 Score

- Measures goodness of fit
- Indicates how much variance in runtime is explained by the model

Interpretation:
Values closer to 1 indicate stronger explanatory power.

---

7. Workload Types the Model Predicts Well

The model performs well for workloads that exhibit **repeatable and measurable resource consumption patterns**, including:

- SQL joins
- Database aggregations
- Machine learning training and inference
- Disk-heavy workloads
- CPU-bound workloads

Reason:
These workloads scale predictably with CPU cores, memory capacity, and disk speed.

---

8. Known Limitations

The model is not expected to perform perfectly in the following cases:

- Very short-lived scripts (< 1 second)
- Network-dominated systems
- Highly branch-dependent or data-dependent logic

These workloads are influenced by factors not fully captured by system-level metrics.

---

9. Summary

The Random Forest–based approach successfully models execution time by learning how workload behavior observed on a development machine scales under different hardware configurations. Batch-based splitting ensures realistic evaluation, baseline models establish credibility, and ensemble methods provide strong generalization across unseen environments. This makes the approach suitable for practical runtime prediction in real-world systems.