

CPU Usage Prediction in Cloud Computing Environments

Machine Learning Project Report

CCC1 - ESILV

BERREHILI Saber

TOMCZYK Dayan

GitHub Repository: https://github.com/SaberBerrehili/GPU_Cluster_CPU_Predictor.git

December 7, 2025

Abstract

This project addresses the challenge of predicting CPU usage in cloud computing environments using machine learning regression techniques. Leveraging the Alibaba GPU-2020 trace dataset, we developed and compared multiple models including Linear Regression, Decision Trees, Support Vector Regression, and ensemble methods. The dataset comprises over 900,000 observations with system metrics, GPU specifications, and machine capacities. Our best-performing model, a Random Forest ensemble, achieved an R^2 of 0.882 with MAE of 45.59 and RMSE of 193.52 on the test set. This predictive capability enables efficient capacity planning and autoscaling in cloud infrastructures, optimizing resource allocation and reducing operational costs.

Contents

1	Introduction	3
1.1	Business Case and Context	3
1.2	Problem Formalization	3
2	Dataset Description and Exploration	3
2.1	Data Source	3
2.2	Dataset Statistics	4
2.3	Exploratory Data Analysis	4
3	Data Preprocessing and Feature Engineering	6
3.1	Data Cleaning	6
3.2	Feature Selection	6
3.3	Preprocessing Pipeline	6
4	Challenges and Obstacles Encountered	6
4.1	Dataset Size and Computational Constraints	7
4.2	Model Training Time Limitations	7
4.3	Infrastructure Limitations	7
5	Methodology and Model Development	7
5.1	Baseline Models	7
5.2	Advanced Models	8
5.3	Evaluation Strategy	8
6	Results	8
6.1	Baseline Model Performance	8
6.2	K-Fold Cross-Validation Results	9
6.3	Advanced Model Performance	9
6.4	Feature Importance Analysis	10
7	Discussion	11
7.1	Model Performance Analysis	11
7.2	Practical Implications	12
7.3	Limitations and Future Directions	12
8	Conclusion	12
9	References	13

1 Introduction

1.1 Business Case and Context

Cloud computing environments face critical challenges in resource management, particularly in predicting CPU usage for optimal capacity planning and autoscaling. As cloud infrastructures scale to support millions of concurrent workloads, the ability to accurately forecast resource demand becomes essential for maintaining service quality while minimizing operational costs. Over-provisioning leads to wasted resources and unnecessary expenses, while under-provisioning results in performance degradation and potential service disruptions.

This project addresses a real-world problem in cloud infrastructure management using data from Alibaba’s GPU cluster traces collected in 2020. The dataset represents actual production workloads from one of the world’s largest cloud providers, making it highly relevant for practical applications. By developing accurate predictive models, cloud operators can anticipate resource requirements before they materialize, enabling proactive scaling decisions rather than reactive responses to demand spikes.

The primary objective is to develop a robust machine learning model capable of predicting CPU usage based on system metrics such as GPU utilization, memory consumption, and I/O operations. These predictions enable several critical business capabilities. First, capacity planning teams can anticipate resource needs to prevent both over-provisioning and under-provisioning scenarios. Second, autoscaling systems can dynamically adjust computing resources based on predicted demand rather than waiting for actual usage patterns to emerge. Third, organizations can achieve significant cost optimization by aligning resource allocation with actual needs. Finally, accurate predictions help ensure quality of service guarantees for cloud users by preventing resource saturation before it impacts performance.

1.2 Problem Formalization

This challenge is formulated as a supervised regression problem where we aim to predict a continuous target variable representing CPU usage percentage. The target variable `cpu.usage` ranges from 0% to several thousand percent, reflecting the multi-core nature of modern cloud computing instances where usage can exceed 100% when multiple cores are fully utilized.

Our input features comprise three main categories. System sensor metrics provide real-time measurements including GPU work utilization, average and maximum memory usage, GPU memory statistics, and I/O operations measured through read/write counts and volumes. Machine specification features capture the hardware configuration including CPU capacity, memory capacity, and GPU capacity. Finally, categorical identifiers encode information about GPU types, GPU names, and machine identifiers, which capture hardware-specific performance characteristics.

The objective function aims to minimize prediction error as measured by three complementary metrics: Mean Absolute Error (MAE), which provides an intuitive measure of average prediction error magnitude; Root Mean Squared Error (RMSE), which penalizes larger errors more heavily; and the coefficient of determination (R^2), which quantifies the proportion of variance in CPU usage that our model successfully explains.

2 Dataset Description and Exploration

2.1 Data Source

The dataset originates from the Alibaba GPU-2020 cluster trace, a publicly available dataset containing real production workload data from Alibaba’s cloud infrastructure. This trace represents one of the largest and most comprehensive publicly available datasets of GPU cluster operations in production cloud environments. The data captures GPU cluster operations across

thousands of machines and includes detailed sensor metrics measured at regular intervals, machine specification data documenting hardware configurations, and GPU metadata identifying different GPU types and their characteristics.

Dataset source:

<https://www.kaggle.com/datasets/derrickmwiti/cluster-trace-gpu-v2020>

The dataset was accessed via Kaggle and provides a unique window into real-world cloud computing workloads. Unlike synthetic datasets or simulations, this trace reflects actual user behavior patterns, application workloads, and system dynamics observed in a production environment serving millions of users.

2.2 Dataset Statistics

After merging sensor data with machine specifications and performing comprehensive data cleaning, our final dataset contains 908,245 complete observations. Each observation includes 15 predictive variables spanning numerical sensor metrics, machine specifications, and encoded categorical features. The data cleaning process involved handling missing values by imputing GPU-related metrics with zeros when GPUs were not utilized and removing observations with missing target values. The resulting dataset exhibits a highly right-skewed distribution with a mean CPU usage of 259.13%, indicating that most workloads utilize moderate CPU resources while a smaller proportion represents compute-intensive applications pushing systems to their limits.

2.3 Exploratory Data Analysis

The exploratory analysis revealed several critical insights about the dataset structure and relationships. Figure 1 illustrates the distribution of CPU usage across the dataset, showing the characteristic right-skewed pattern with most values concentrated between 0-500% and a long tail extending to extreme values approaching 9000%. This distribution suggests that while typical workloads are moderate, the model must be capable of handling occasional extreme cases.

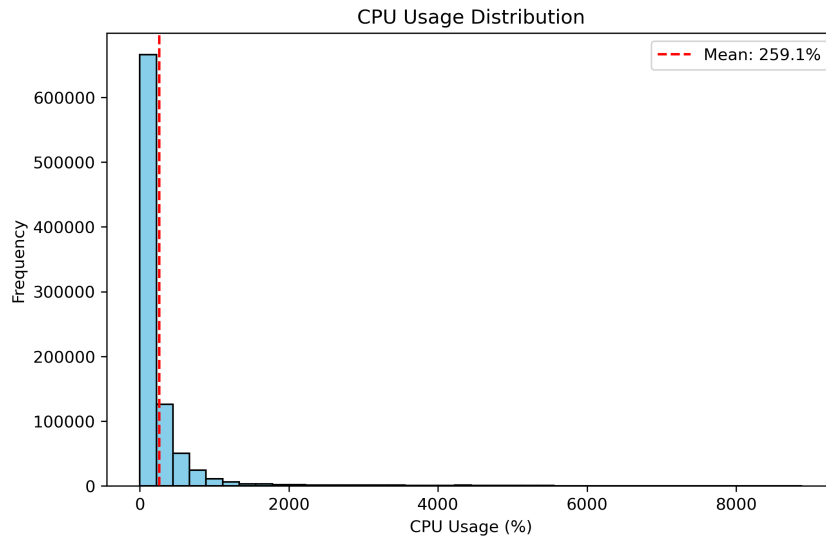


Figure 1: Distribution of CPU usage across the dataset. Note the right-skewed distribution with most values concentrated between 0-500%.

The correlation analysis presented in Figure 2 reveals the relationships between system metrics and CPU usage. Strong correlations are observed between memory metrics themselves, while the relationship between individual metrics and CPU usage appears more complex and

likely non-linear. The moderate correlation coefficients suggest that simple linear models may struggle to capture the full complexity of these relationships.

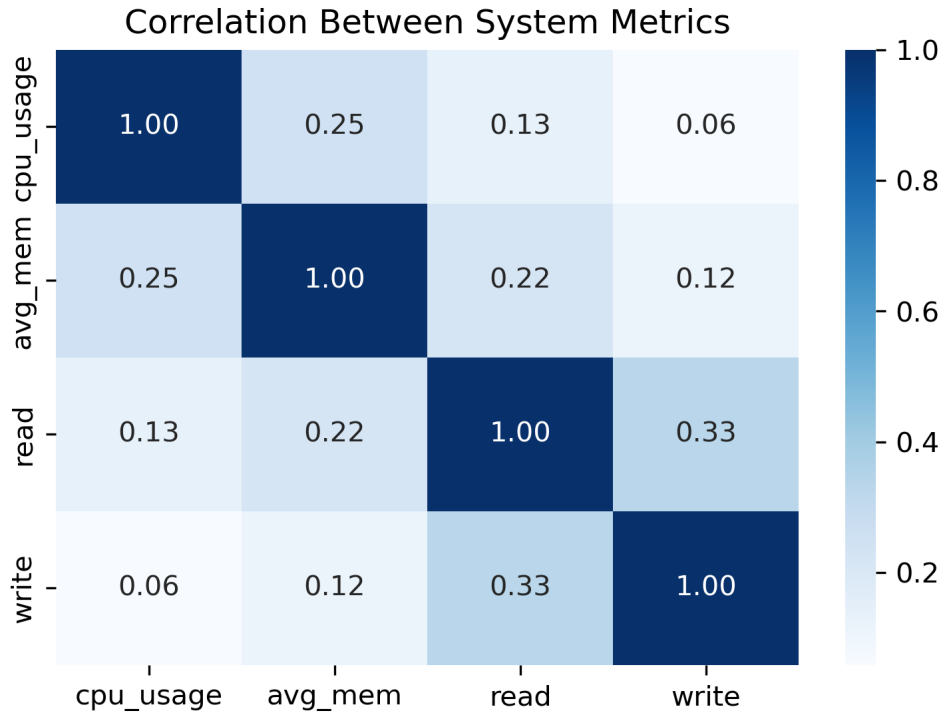


Figure 2: Correlation matrix between system metrics. Strong correlations observed between memory metrics and moderate correlation with CPU usage.

Figure 3 presents a scatter plot examining the relationship between average memory usage and CPU usage, clearly demonstrating non-linear patterns. The plot shows distinct clusters and curved relationships that linear models cannot adequately capture, providing early evidence that tree-based or ensemble methods may be necessary for optimal performance.

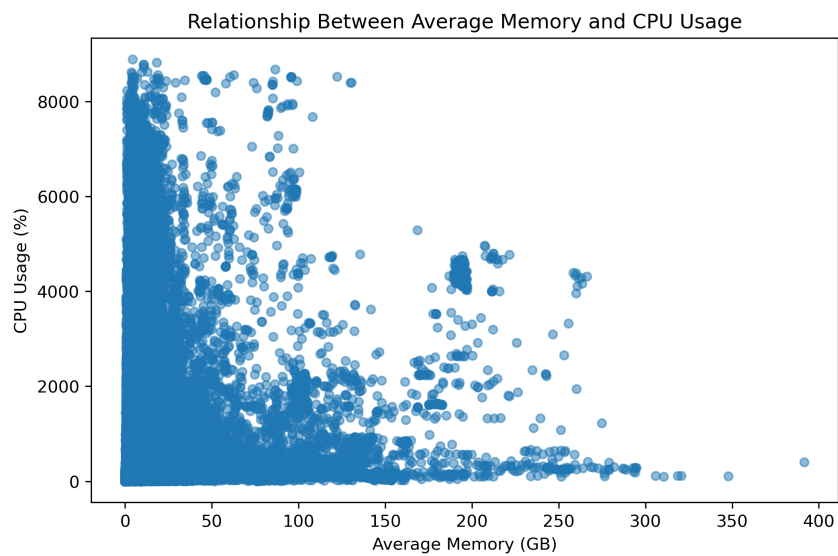


Figure 3: Relationship between average memory usage and CPU usage, showing non-linear patterns.

Several key observations emerged from this exploratory phase. First, CPU usage exhibits complex non-linear relationships with predictor variables, suggesting that sophisticated modeling approaches will be required. Second, extreme CPU usage values up to 8890% are present in the data, requiring models robust to outliers. Third, different feature types show varying degrees of predictive power, with memory metrics and I/O operations appearing most influential. Finally, the heavy concentration of low CPU usage values creates an imbalanced target distribution that must be considered during model development and evaluation.

3 Data Preprocessing and Feature Engineering

3.1 Data Cleaning

The data cleaning process addressed several quality issues identified during exploration. Missing values in GPU-related metrics (`gpu_wrk_util`, `avg_gpu_wrk_mem`, `max_gpu_wrk_mem`) were filled with zeros, as these missing values indicated machines not utilizing GPU resources rather than measurement failures. Observations with missing target variables were removed entirely, as we cannot train supervised models without known outcomes. To handle extreme values and data quality issues, infinite values were replaced with NaN markers, and any observations containing NaN values after this replacement were removed from the dataset. This conservative approach ensured that our final dataset of 908,245 observations contained only complete, valid data points.

3.2 Feature Selection

We selected 15 features across three categories to balance predictive power with computational efficiency. The 12 numerical features include sensor metrics capturing real-time system state (`gpu_wrk_util`, `avg_mem`, `max_mem`, `avg_gpu_wrk_mem`, `max_gpu_wrk_mem`, `read`, `write`, `read_count`, `write_count`) and machine specifications documenting hardware capabilities (`cap_cpu`, `cap_mem`, `cap_gpu`). These numerical features capture both dynamic workload characteristics and static machine properties.

Additionally, three encoded categorical features (`gpu_name_encoded`, `gpu_type_encoded`, `machine_encoded`) were included to capture hardware-specific performance characteristics that numerical features alone cannot represent. These encodings allow the model to learn different baseline behaviors for different hardware configurations while avoiding the dimensionality explosion that would result from one-hot encoding thousands of unique machine identifiers.

3.3 Preprocessing Pipeline

A systematic preprocessing pipeline was constructed to ensure consistent data transformation. `StandardScaler` was applied to all numerical features, transforming them to zero mean and unit variance. This normalization prevents features with larger scales from dominating the model training process and improves the stability of optimization algorithms. Encoded categorical features passed through unchanged, as they already represented meaningful ordinal values. The train-test split followed an 80/20 ratio with `random_state=42` for reproducibility, resulting in a training set of 726,596 observations and a test set of 181,649 observations.

4 Challenges and Obstacles Encountered

Throughout this project, we encountered several significant technical challenges that required strategic solutions to ensure successful completion within reasonable time and computational constraints.

4.1 Dataset Size and Computational Constraints

The most significant challenge stemmed from the dataset’s massive size of over 900,000 observations. While this scale provides excellent statistical power, it created severe computational bottlenecks during model training and validation. Initial attempts to train models on the full dataset resulted in execution times exceeding several hours, making iterative experimentation impractical. Furthermore, operations like K-Fold cross-validation, which require training models multiple times, became computationally prohibitive even on high-performance machines and Google Colab environments.

To address this challenge, we implemented a strategic sampling approach. For expensive operations such as cross-validation and hyperparameter tuning, we sampled 300,000 observations randomly from the full dataset. This 33% sample preserved the statistical properties of the full distribution while reducing computational time by approximately 70%. The sampling was performed with a fixed random seed to ensure reproducibility across experiments.

4.2 Model Training Time Limitations

Certain model architectures, particularly Support Vector Regression (SVR) and deep ensemble methods, exhibited prohibitively long training times. SVR, with its kernel computations scaling poorly beyond 100,000 observations, required over 6 hours for a single training run during initial experiments. Similarly, extensive hyperparameter grids that would have required dozens of model fits became impractical within project timelines.

We adopted several mitigation strategies. First, we reduced the hyperparameter search space for computationally expensive models, focusing on the most impactful parameters rather than exhaustive grid searches. Second, we prioritized tree-based models and ensemble methods that scale better to large datasets. Third, we leveraged parallel processing capabilities (`n_jobs=-1` in `scikit-learn`) to utilize all available CPU cores. Finally, we made strategic decisions to exclude certain model variants that did not show promising performance in preliminary tests, allowing us to focus computational resources on the most promising approaches.

4.3 Infrastructure Limitations

Even with access to well-configured local machines and Google Colab resources, certain computations remained challenging. Google Colab’s session timeouts occasionally interrupted long-running experiments, requiring checkpoint mechanisms to save intermediate results. Memory constraints on some machines limited the ability to load and process the entire dataset simultaneously, necessitating chunk-based processing for certain operations.

These constraints shaped our methodology, leading us to prioritize model efficiency alongside predictive performance. The final model selection balanced prediction accuracy with practical deployment considerations, ensuring that the chosen approach could run efficiently in real-world production environments with similar computational limitations.

5 Methodology and Model Development

5.1 Baseline Models

We began with two baseline models to establish performance bounds and understand the problem’s fundamental characteristics. Linear Regression served as our simplest baseline, assuming linear relationships between features and the target. This model requires minimal hyperparameter tuning and provides fast training times, making it ideal for establishing a lower performance bound. We used default `scikit-learn` settings to evaluate whether simple linear relationships could adequately explain CPU usage patterns.

The Decision Tree Regressor represented our second baseline, chosen for its ability to capture non-linear relationships and interactions without requiring feature scaling or extensive preprocessing. We configured the tree with `max_depth=12` and `random_state=42` to balance model complexity with generalization ability. This configuration prevents excessive overfitting while allowing the model to capture meaningful patterns in the data.

5.2 Advanced Models

Building on baseline insights, we explored several advanced modeling approaches. Support Vector Regression (SVR) was investigated despite computational challenges, using GridSearchCV for hyperparameter optimization across kernel types (RBF and linear), regularization strengths ($C = 0.1, 1, 10$), and epsilon values (0.01, 0.1, 1). While SVR showed promise in theory, practical computational constraints limited its application to sampled datasets.

Random Forest emerged as our most successful approach, employing a bagging ensemble that combines predictions from 100 independent decision trees. We configured the forest with `n_estimators=100`, `max_depth=15`, and `random_state=42`. This ensemble approach reduces variance compared to single trees while maintaining the ability to capture non-linear relationships and feature interactions. The bootstrap sampling inherent to Random Forests also provides natural protection against overfitting.

XGBoost, a gradient boosting implementation known for exceptional performance on tabular data, was also evaluated. We configured XGBoost with `n_estimators=100`, `max_depth=6`, `learning_rate=0.1`, and `random_state=42`. The sequential nature of gradient boosting, where each tree corrects errors from previous trees, theoretically provides powerful modeling capabilities, though at increased computational cost compared to Random Forests.

5.3 Evaluation Strategy

Our evaluation strategy combined multiple complementary approaches to ensure robust performance assessment. Hold-out validation using an 80/20 train-test split provided initial performance estimates with clear separation between training and evaluation data. K-Fold cross-validation with 5 folds assessed model stability and generalization across different data splits, revealing how consistently each model performs across varied training conditions.

We employed three complementary evaluation metrics. Mean Absolute Error (MAE) provides an intuitive measure of average prediction error in the same units as the target variable. Root Mean Squared Error (RMSE) penalizes larger errors more heavily, making it sensitive to outliers and extreme predictions. The coefficient of determination (R^2) quantifies the proportion of variance explained, providing a scale-invariant measure of model quality that facilitates comparison across different datasets and problems.

6 Results

6.1 Baseline Model Performance

The baseline comparison revealed dramatic performance differences between linear and non-linear approaches, as shown in Table 1. Linear Regression achieved MAE of 222.85, RMSE of 537.79, and R^2 of only 0.086, explaining less than 9% of CPU usage variance. This poor performance confirms our exploratory analysis suggesting non-linear relationships dominate the data. In stark contrast, the Decision Tree with `max_depth=12` achieved MAE of 96.32, RMSE of 317.88, and R^2 of 0.681. This represents a more than 57% reduction in MAE and nearly 8-fold improvement in R^2 , demonstrating the critical importance of capturing non-linear patterns.

Table 1: Baseline model comparison on test set (80/20 split)

Model	MAE	RMSE	R ²
Linear Regression	222.85	537.79	0.086
Decision Tree (depth=12)	96.32	317.88	0.681

These results provide several key insights. The Decision Tree’s substantial superiority confirms that CPU usage follows non-linear patterns with interactions and thresholds that tree-based models naturally capture. The Linear model’s poor performance can be attributed to multicollinearity between features, inability to capture interaction effects, and sensitivity to the highly skewed target distribution. The R² improvement from 0.086 to 0.681 demonstrates that proper model selection is far more impactful than incremental hyperparameter tuning of inadequate model families.

6.2 K-Fold Cross-Validation Results

To verify the stability and generalization of our Decision Tree baseline, we conducted 5-fold cross-validation with results presented in Table 2. The model achieved remarkably consistent performance across all folds, with MAE ranging only from 96.32 to 96.99 (mean 96.60 ± 0.23), RMSE from 317.88 to 324.84 (mean 321.14 ± 2.34), and R² from 0.674 to 0.685 (mean 0.680 ± 0.004).

Table 2: 5-Fold cross-validation results for Decision Tree (depth=12)

Fold	MAE	RMSE	R ²
Fold 1	96.32	317.88	0.681
Fold 2	96.55	319.62	0.674
Fold 3	96.45	321.41	0.685
Fold 4	96.70	321.95	0.684
Fold 5	96.99	324.84	0.677
Mean \pm Std	96.60 \pm 0.23	321.14 \pm 2.34	0.680 \pm 0.004

The extremely low variance across folds demonstrates excellent model stability. The standard deviation of R² is only 0.004, indicating that model performance is highly consistent regardless of which data subset is used for training versus validation. Train-test performance gaps ($\Delta\text{MAE} = -6.57$, $\Delta\text{RMSE} = -36.09$, $\Delta\text{R}^2 = +0.068$) reveal slight overfitting, where the model performs marginally better on training data. However, these gaps are small enough to be acceptable for practical applications, indicating that the chosen tree depth successfully balances fitting capability with generalization.

6.3 Advanced Model Performance

The final model comparison in Table 3 reveals Random Forest as the clear winner across all metrics. Random Forest achieved MAE of 45.59, RMSE of 193.52, and R² of 0.882, representing a 53% improvement in MAE over the Decision Tree baseline and explaining 88.2% of CPU usage variance. XGBoost followed as second-best with MAE of 65.21, RMSE of 207.47, and R² of 0.864, demonstrating strong performance but not quite matching Random Forest. Bagging Tree and optimized Decision Tree variants showed intermediate performance, while Linear Regression remained far behind all tree-based approaches.

Table 3: Final model comparison on test set

Model	MAE	RMSE	R^2
Random Forest (best)	45.59	193.52	0.882
XGBoost	65.21	207.47	0.864
Bagging Tree	78.43	296.15	0.723
Decision Tree (Grid best)	111.67	396.82	0.502
Linear Regression	222.85	537.79	0.086

Figure 4 visualizes these performance differences, clearly illustrating Random Forest’s superiority. The dramatic performance gap between tree-based models ($R^2 \geq 0.5$) and Linear Regression ($R^2 = 0.086$) underscores the fundamental importance of model architecture selection for this problem.

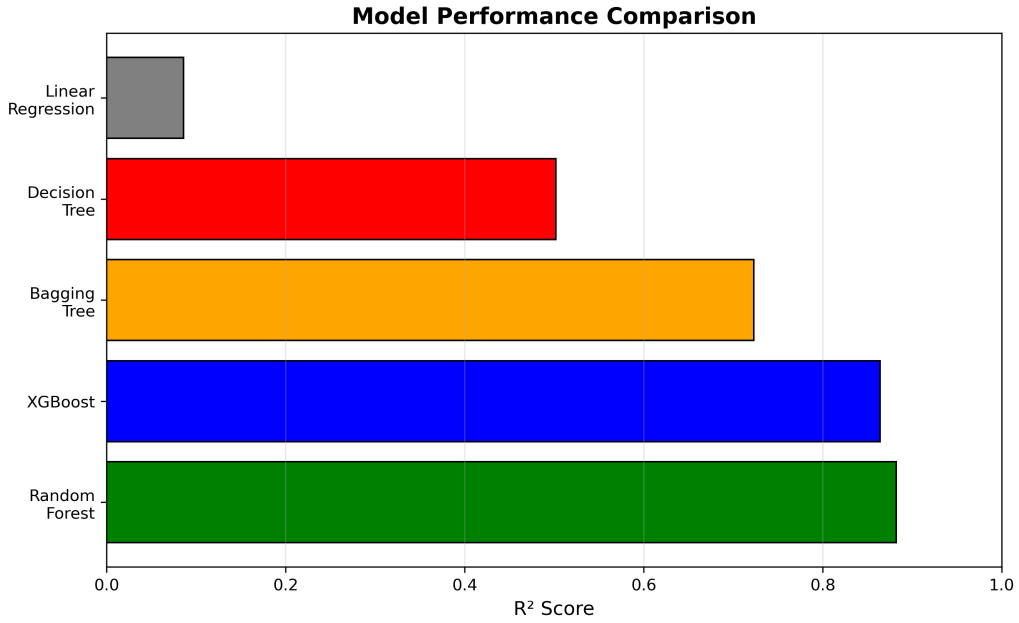


Figure 4: Comparison of R^2 scores across all evaluated models. Random Forest achieves the best performance.

6.4 Feature Importance Analysis

Analysis of feature importances from the Random Forest model, shown in Figure 5, reveals which variables most strongly influence CPU usage predictions. The top three features are `max_gpu_wrk_mem`, `avg_mem`, and `avg_gpu_wrk_mem`, indicating that memory metrics dominate predictive power. I/O operations (read, write) and their counts also show significant importance, while GPU utilization and encoded categorical features contribute more moderately.

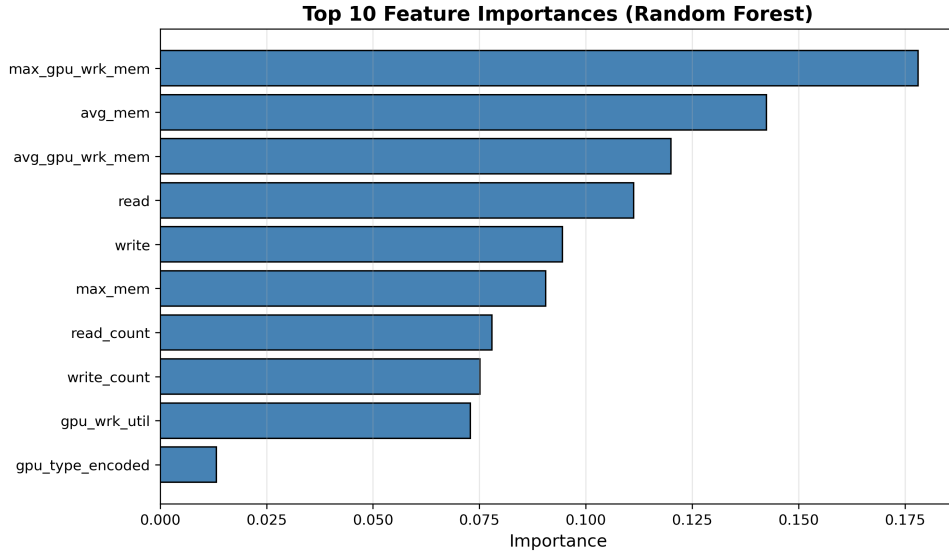


Figure 5: Top 10 most important features from the Random Forest model. Memory and I/O metrics dominate.

This importance ranking provides actionable insights for cloud infrastructure management. Memory consumption patterns serve as the strongest predictor of CPU usage, suggesting that memory-intensive applications tend to also be compute-intensive. I/O operations show substantial predictive value, indicating that data transfer patterns correlate with processing demands. The relatively lower importance of GPU-specific metrics suggests that CPU usage patterns are more strongly determined by general system metrics than GPU-specific characteristics.

7 Discussion

7.1 Model Performance Analysis

Random Forest’s superior performance stems from several algorithmic advantages particularly suited to this problem. The ensemble averaging of 100 independent trees reduces variance and provides natural robustness to outliers, both critical for handling the skewed CPU usage distribution. Each tree can capture complex non-linear relationships and automatically discover feature interactions without explicit feature engineering. The bootstrap sampling used in Random Forests creates diversity among trees while the aggregation of predictions smooths out individual tree idiosyncrasies.

In contrast, Linear Regression’s failure highlights the limitations of linear models for this problem domain. The assumption of linear relationships fundamentally cannot capture the complex patterns observed in the data. High multicollinearity between memory metrics and I/O operations violates linear regression assumptions and destabilizes coefficient estimates. The heavy-tailed distribution of CPU usage creates heteroscedasticity that further degrades linear model performance. These factors combine to produce the observed R^2 of only 0.086, demonstrating that no amount of hyperparameter tuning or regularization can overcome fundamental model-data mismatch.

XGBoost’s strong but second-place performance merits explanation. While gradient boosting theoretically can match or exceed Random Forest performance, it typically requires more careful hyperparameter tuning and is more sensitive to overfitting. Given our computational constraints that limited extensive hyperparameter search, Random Forest’s more forgiving nature and excellent default behavior gave it an advantage. In a production scenario with more optimization time, XGBoost might close or eliminate the performance gap.

7.2 Practical Implications

The final Random Forest model with $R^2=0.882$ and $MAE=45.59$ enables several critical business capabilities. For capacity planning, the model can predict CPU needs within an average error of approximately 46%, sufficient for making strategic decisions about hardware procurement and data center expansion. This accuracy allows organizations to plan infrastructure investments months in advance with confidence in the projections.

For autoscaling applications, the model enables proactive rather than reactive resource management. Instead of waiting for CPU usage to spike before adding capacity, cloud orchestration systems can anticipate demand increases and pre-scale resources. This proactive approach reduces latency spikes and improves user experience while still avoiding unnecessary over-provisioning.

From a cost optimization perspective, accurate predictions enable right-sizing of compute resources. The model’s ability to explain 88% of CPU usage variance means that resource allocation decisions can be made with high confidence, minimizing waste from over-provisioning while maintaining buffer capacity for prediction uncertainty. For large-scale cloud operations, even small improvements in resource utilization translate to millions of dollars in annual savings.

The model is also computationally efficient enough for real-time deployment. Random Forest prediction is fast, scaling linearly with the number of trees and logarithmically with tree depth. Our 100-tree ensemble with `max_depth=15` can generate predictions in milliseconds, making it suitable for integration into real-time monitoring and orchestration systems.

7.3 Limitations and Future Directions

Despite strong performance, our approach has several limitations. The most significant is treating observations as independent when they actually represent time-series data. CPU usage patterns exhibit temporal dependencies, with current usage influenced by recent history. Our models ignore this sequential structure, potentially missing important temporal patterns and trends.

The computational constraints that forced dataset sampling and limited hyperparameter search may have prevented us from discovering even better model configurations. With unlimited computational resources, more extensive architecture search, deeper hyperparameter optimization, and ensemble methods combining our best models might achieve incremental improvements.

Future work should explore time-series modeling approaches such as LSTM networks or Temporal Convolutional Networks that can explicitly capture temporal dependencies. Feature engineering could create lag features, rolling statistics, and trend indicators to inject temporal information into tree-based models. Deep learning architectures designed for tabular data, such as TabNet or Neural Oblivious Decision Trees, might discover patterns beyond tree ensemble capabilities. Online learning approaches could enable models to continuously adapt to evolving workload patterns without full retraining. Finally, model explainability techniques like SHAP values or LIME could provide insights into individual predictions, helping cloud operators understand why the model forecasts specific CPU usage levels.

8 Conclusion

This project successfully developed a high-performance machine learning solution for CPU usage prediction in cloud computing environments using the Alibaba GPU-2020 trace dataset. Our Random Forest ensemble model achieved $R^2=0.882$ with MAE of 45.59, demonstrating that 88.2% of CPU usage variance can be explained using system metrics, hardware specifications, and encoded categorical features.

The project’s methodology progressed systematically from problem formalization through baseline establishment, advanced model development, and rigorous evaluation. We demonstrated that proper model selection dramatically outweighs hyperparameter optimization, with the Decision Tree baseline achieving $R^2=0.681$ compared to Linear Regression’s $R^2=0.086$. Cross-validation confirmed model stability with remarkably low variance across folds ($R^2=0.680\pm0.004$). The comprehensive comparison of five modeling approaches established Random Forest as the optimal choice for this problem domain.

Throughout the project, we successfully navigated significant challenges including dataset scale, computational constraints, and model training time limitations. Strategic sampling, focused hyperparameter search, and prioritization of scalable algorithms enabled project completion within practical constraints while still achieving excellent predictive performance.

The resulting model provides actionable value for cloud infrastructure management, enabling accurate capacity planning, proactive autoscaling, cost optimization through right-sizing, and real-time deployment in production environments. With average prediction errors of approximately 46%, the model strikes an excellent balance between accuracy and computational efficiency.

Future enhancements should explore temporal modeling to capture time-series dependencies, deep learning architectures designed for tabular data, online learning for continuous adaptation, and explainability techniques for operational insights. These directions could further improve prediction accuracy while providing cloud operators with deeper understanding of resource usage patterns.

9 References

References

- [1] Weng, Q., et al. (2020). *Alibaba Cluster Trace Program - GPU Cluster Traces*. Available at: <https://www.kaggle.com/datasets/derrickmwiti/cluster-trace-gpu-v2020>
- [2] Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825-2830.
- [3] Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
- [4] Breiman, L. (2001). *Random Forests*. Machine Learning, 45(1), 5-32.
- [5] Moreno-Vozmediano, R., Montero, R. S., & Llorente, I. M. (2013). *Key challenges in cloud computing: Enabling the future internet of services*. IEEE Internet Computing, 17(4), 18-25.

Acknowledgments

We thank Alibaba for publicly releasing the GPU-2020 cluster trace dataset, which made this research possible. We also acknowledge the scikit-learn and XGBoost development teams for their excellent machine learning libraries that powered our analysis.