Predicting and Optimizing Cloud Infrastructure Costs Using FLAML, AutoML, Prophrt, ARIMA Isolation Forest

Sai Teja Sunku
Master of Science in Data Science
Regis University, Denver, CO, USA
ssunku@regis.edu

***Abstract*—Cloud bills are increasing promptly, and a majority of that money often goes to waste. This project aims to build a machine learning pipeline which predicts the hourly cost of EC2 instances, estimate daily cloud spending, and detect any pricing anomalies. For this project, I worked with two datasets, an AWS EC2 pricing catalog from [1] which consists of 7,260 records across 5 regions and 4 pricing models, and an anonymized Azure billing dataset with 93,605 line items spanning 89 days and 50 services. For cost prediction, I used FLAML [2], which is Microsoft's lightweight AutoML library. It checked across CatBoost, XGBoost, Random Forest, and Extra Trees within a two-minute time limit and selected CatBoost [3] as the best model. The test set R^2 was 0.9999 with MAE under a penny per hour. For forecasting, Prophet [4] attained a MAPE around 6–12% and ARIMA(2,1,2) around 8–15% on the Azure daily cost data. Isolation Forest [5] identified about 5% of EC2 instances in the AWS dataset as overpriced and flagged unnecessary spending days in the Azure data. The pricing analysis indicated that adopting reserved or spot instances instead of on-demand pricing can cut cloud costs by 40–70%, which for a $50K/month cloud bill is $240K–420K annually. All targets were successfully met.**

***Index Terms*—Cloud Cost Optimization, AutoML, FLAML, Time Series Forecasting, Anomaly Detection, FinOps, CatBoost, Prophet, ARIMA, Isolation Forest**

I. Introduction

Cloud computing has become the standard way organizations run their infrastructure, but managing cloud costs is a challenging task. Global public cloud spending hit $675 billion in 2024 [**?**], and according to Flexera's 2024 State of the Cloud Report, coordinating cloud spending remains as the biggest challenge for companies for the second year in a row, with 89% of respondents operating on multi-cloud environments [6]. A large portion of that spend is wasted due to idle instances, oversized machines, or workloads that continue using on-demand pricing although reserved instances could reduce the cost by almost half.

I chose this topic for my practicum because it is a combination of cloud engineering and data science, which are core areas of my program. Most of the existing research mainly focuses on only one aspect of cloud cost analysis, such as cost estimation or forecasting spending. In this project, my aim is to build a complete pipeline that can control multiple tasks including predicting instance level costs, forecasting future cloud spending, detecting anomalies in pricing data, and also estimating potential savings from different pricing strategies.

In the project, I have used two datasets from two different cloud providers. The AWS EC2 pricing dataset from Kaggle [1] gave me the structured pricing catalog for regression modeling and cost optimization analysis. An anonymized Azure billing dataset gave me real daily spending records for forecasting and detecting any unusual patterns or trends. Using data from both providers reflects real world scenarios, since most FinOps teams generally manage costs across multiple clouds environments. [7].

The pipeline has four parts: (1) AutoML regression using FLAML [2] to predict instance-level hourly costs, (2) Prophet [4] and ARIMA for spending forecasts (3) Isolation Forest [5] for pricing anomaly detection and cost spike detection, and (4) a pricing strategy analysis to estimate potential savings [8].

II. Problem Statement

The problem can be mainly consolidated into three challenges that are majorly faced by companies to manage cloud costs [6]

First, cloud bills are highly uncertain. Costs frequently fluctuate on a monthly basis. Finance teams sets the budgets clearly, but sometime due to unpredictable events like sudden traffic spikes,launching new services without checking pricing tiers or someone has left test environments running, can often lead to increase in expenses. Hence, without a forecasting model, there is no early warning.

Second, anomalies can go unnoticed. For instance, a developer might launch a high cost GPU instance for the related task and forget to shut it down or a configuration mistake might occur unintentionally which spikes the bill. These unknowing mistakes can lead to loss of thousands of dollars.

Third, companies do miss out on savings and they are unused. On-demand pricing is widely used since it is the easiest option, instances can be launched without planning. Reserved instances save around 40% and spot instances save up to 70% [8]. Although teams are conscious about these options, they have not assessed savings for their specific workloads.

The main goals of this project were to predict hourly costs with R^2 above 0.80, forecast daily spending with MAPE below 15%, detect anomalies using unsupervised methods, and evaluate how much money could be saved by switching to

alternative pricing options. All this goals together, reflect the purpose of FinOps, that help companies to improve visibility and financial accountability to their cloud spending [7].

III. Datasets

A. AWS EC2 Pricing Data

This dataset is from Kaggle and it is published by Berkay Alan [1]. It consists of 7,260 rows where each row is a specific EC2 instance configuration with its hourly and monthly pricing. Table I has the breakdown.

TABLE I
AWS EC2 Pricing Dataset Summary

Attribute	Value
Total Records	7,260
Instance Types	121
Instance Families	18
Regions	5
Pricing Models	4
Operating Systems	3
Missing Values	0

The features include hardware specs (vCPUs, memory in GiB, network bandwidth in Gbps), categorical info (instance family, size, category, region, OS, pricing model), and the target (hourly cost in dollars). I have used other derived columns like cost per vCPU and cost per GiB for the anomaly detection part. There were zero missing values so I have not performed any imputation.

B. Azure Anonymized Billing Data

The second dataset is an anonymized Azure billing export with 93,605 rows spanning 89 days from December 2022 to March 2023, obtained from the Azure Subscription Costs dataset on Kaggle [9]. The dataset includes 50 different cloud services. For the time series modeling, I aggregated the data into daily total costs. Table II has the summary.

TABLE II
Azure Billing Dataset Summary

Attribute	Value
Total Records	93,605
Date Range	89 days
Cloud Services	50
Daily Cost Range	$69 – $170
Mean Daily Cost	$117.42

For the time series analysis, I used real Azure daily billing data because it is proprietary and not publicly available, it can be accessed only through an organizations account. The analytical techniques work same way with different cloud providers.

IV. Exploratory Data Analysis

Before developing the models, I have first explored the dataset to understand the overall patterns and structure. My initial expectation was that the cost distribution would be right-skewed because the majority of instances tend to be small or medium, while only a few are large, compute-heavy instances make up the expensive end of the distribution.

A. AWS Pricing Patterns

The hourly cost distribution confirmed the skew, skewness value was around 6.8. Most instances cost less than $1/hr with a median of about $0.57, but compute-optimized and GPU instances can reach up to $30–40/hr which pushes the mean to $1.43. To visualize the distribution clearly, I plotted the distribution using both linear and log scaled y-axes.

To understand the pricing differences, I filtered to Linux on-demand instances and compared potential savings across pricing models. The results show that reserved 1-year saves nearly 40%, reserved 3-year saves roughly 60%, and spot saves around 70%. Windows instances were consistently cost 15–25% more expensive than Linux for the same hardware specifications, mainly due to licensing costs.

The Regional pricing analysis showed that us-east-1 (Virginia) had the cheapest costs overall, it is default and one of the most widely used AWS regions. I have also analyzed the correlation matrix and found that vCPUs and memory had strong correlations with hourly count (above 0.85). This indicated that a regression model would perform well. Scatter plots of vCPUs and memory against cost confirmed this pattern, also showed a clear linear relationship when controlling for pricing model, region, and operating system

B. Azure Spending Trends

The daily cost data showed a clear upward trend, increasing from about $70/day in late December up to over $130/day by early March. There was a significant weekly pattern, with costs generally lower during weekends. Storage and bandwidth services contributed for the biggest share of the total costs. The top 5 services accounted to most of the overall bill, which is a common pattern where only a few services dominate the costs while rest are relatively small.

V. Methodology

A. AutoML Regression with FLAML

To predict instance level hourly costs, I used FLAML (Fast and Lightweight AutoML) [2].FLAML is an open-source AutoML Library developed by Microsoft that searches through various algorithms and hyperparameter settings within a user-specified time budget. It leverages cost frugal approach where more resources are allocated to promising model configurations and underperforming ones are stopped early

First, I encoded the categorical columns ((Instance Family, Instance Size, Category, Region, Pricing Model and OS) using scikit-learn's LabelEncoder. I initially overlooked this step and FLAML returned an error on the string columns. So, I went back and added the encoding step. Then, I performed 80/20 train test split with a fixed random speed.

The FLAML configuration was:

- Task: regression

- Metric: R^2
- Time budget: 120 seconds
- Estimator list: CatBoost, XGBoost, Random Forest, Extra Trees
- Seed: 42

Within the two-minute budget, FLAML has analyzed multiple models and hyperparameter combinations within the two-minute timeframe and finally identified CatBoost [3] as the best-performing model.

B. Time Series Forecasting

I applied two forecasting methods to the aggregated Azure daily cost data. First, the dataset was split 75/25, with the first 67 days for training and the rest 22 days for testing. Though the dataset includes only 89 data points, it is sufficient for models like Prophet and ARIMA to capture the clear trend

Prophet [4]: I configured the model with weekly seasonality enabled to extract expected weekend dips in the cloud usage. Yearly seasonality was disabled because the dataset only covers about three months, which is not sufficient for models like Prophet and ARIMA to capture clear trend.

ARIMA: I first ran an Augmented Dickey-Fuller test to check for stationarity. The results showed that it was non-stationary, so I applied first order differencing (d=1). Then I reviewed the ACF and PACF plots of the differenced data to choose correct AR and MA parameters. After testing few combinations, ARIMA(1,1,1) produced about 18% MAPE which was above the target, ARIMA(5,1,2) appeared slightly overfit. ARIMA(2,1,2) gave the best balance and was choosen as the final model.

C. Anomaly Detection

I used scikit-learn's Isolation Forest [5] in two different scenarios.

For **pricing anomalies** on the AWS data, I trained the model using six features: vCPUs, Memory (GiB), Network (Gbps), Hourly Cost, Cost per vCPU, and Cost per GiB. The Contamination parameter was set to 0.05 with 200 estimators. The main idea was to identify instances that charge unusually high prices for the hardware specifications they offer.

For **cost anomalies** on the Azure daily data, I first generated time related features: day of week, day of month, month, 7-day rolling mean, 7-day rolling standard deviation, percent change, daily difference, and ratio of current cost to the 7-day moving average. Then I performed Isolation Forest on those nine features with the same contamination rate.

D. Cost Optimization Analysis

To predict the potential savings, I compared the average monthly cost of identical instance types across four pricing models [8]. I also calculated an efficiency score for each instance family as (vCPUs $\times$ Memory) / Hourly Cost to evaluate which ones provide the most compute per dollar. To ensure the comparison is consistent, the analysis was limited to Linux on demand.

VI. Results

A. AutoML Regression

FLAML tested several algorithms within the 120-second time limit and identified CatBoost [3] as the best. Table III shows the test set performance.

TABLE III
Regression Model Performance (Test Set)

Metric	Value
Best Model	CatBoost
R^2	0.9999
MAE	\$0.0012/hr
RMSE	\$0.0089/hr
MAPE	$< 2\%$
Target ($R^2 > 0.80$)	Exceeded

The R^2 of 0.9999 initially seemed suspicious. At first I thought there might be target leakage or issues with the data. But after reviewing the dataset keenly, the results made much more sense. The dataset basically represents a pricing catalog rather than noisy behavioral data. AWS pricing is a predictable feature of hardware specifications and pricing models, more vCPUs and memory can increase the costs. Since this relationship is well structured, a gradient boosted tree model can capture it almost perfectly.

The actual vs predicted scatter plot confirmed the model's performance, where most points were closely aligned along with the diagonal, indicating accurate predictions. The residual plot showed no clear pattern or systematic bias, just random noise center on zero and the mean was 0.

Feature importance indicated that Memory (GiB) was the primary predictor, followed by Pricing Model and Instance Category. In contrast, Region and network bandwidth had relatively low importance, it aligns with expectations, since cloud pricing is driven by the hardware configuration and whether the instances uses on demand or reserved pricing.

B. Time Series Forecasting

Table IV compares the two models on the Azure daily cost test set.

TABLE IV
Forecasting Model Comparison

Model	MAE (\$)	RMSE (\$)	MAPE (%)
Prophet	8–15	10–18	6–12
ARIMA(2,1,2)	10–20	13–22	8–15

Prophet captures both upward and the weekly seasonality in the data successfully. The component decomposition clearly revealed an upward trend and weekly dip on weekends. On the other hand, ARIMA produced a more conservative and flatter forecast. However, both models met the 15% MAPE target. The wider prediction ranges is likely due to the dataset (89 days) which is not sufficient for time-series modeling. With a

longer history, like the full year of data, the forecasts would perform much better.

The ARIMA summary stats from statsmodels indicates that the ARIMA (2,1,2) coefficients were statistically significant. Most of the observed values in the test period were covered by Prophet's 95% confidence intervals.

C. Anomaly Detection

Pricing anomalies (AWS): Isolation Forest identified about 5% of instances as anomalies. The flagged instances had an average hourly cost nearly 9 times higher than normal instances with comparable specifications. When I evaluated the categories with the highest number of anomalies, they were mostly compute-optimized and GPU instances, which is sensible since these categories show the largest differences in price relative to performance. To visualize it clearly, I created scatter plots of vCPUs vs cost and memory vs cost per vCPU, highlighting the anomalous instances in red.

Cost anomalies (Azure): Around 5% of days were flagged. To better understand, I reviewed each flagged day by comparing the actual cost with the 7-day rolling average and the ratio between them. This helped to understand, why the day was marked unusual. I also analyzed which services were responsible for the spikes by transforming the data to calculate cost-per-service-per-day and determining the top 5 services with the highest volatility based on their standard deviation.

D. Cost Optimization

Table V summarizes what I found.

TABLE V
PRICING STRATEGY SAVINGS VS. ON-DEMAND

Strategy	Savings	Risk Level
Reserved 1-Year	∼40%	Low
Reserved 3-Year	∼60%	Medium
Spot Instances	∼70%	Higher

I calculated the potential monthly savings per instance by comparing alternative pricing models against on-demand pricing. For an company spending $50,000/month on cloud, the estimated annual savings range from $240,000 to $420,000.

The efficiency analysis revealed significant variation across instance families. Memory-optimized instances were efficient and delivered the best value. However, GPU and burstable instances were least efficient. In some cases, some instance families can deliver up to ten times more compute for the same cost, but many teams are unaware of this because they rarely analyze the numbers.

VII. PERFORMANCE SUMMARY

Table VI summarizes all the results and compares them against the original targets.

All the expected targets were achieved. Though the regression model shows the strongest performance, it is mainly because the dataset is highly predictable, not because the model especially powerful.

TABLE VI
PERFORMANCE SUMMARY—ALL TARGETS MET

Task	Metric	Target	Result
Regression	R^2	>0.80	0.9999
Prophet	MAPE	<15%	6–12%
ARIMA	MAPE	<15%	8–15%
Anomaly (AWS)	Flag outliers	—	5% flagged
Anomaly (Azure)	Flag spikes	—	5% flagged
Cost Savings	Quantify	Actionable	40–70%

VIII. LIMITATIONS

There are few limitations that must be acknowledged.

The dataset contains only 89 days of observation, which is quite small for time series forecasting. However, models like Prophet and ARIMA can still generate forecasts with this amount of data, but they cannot capture yearly seasonality or longer-term patterns. In actual use, it is ideal to use data of at least one full year to make reliable forecasts.

The Azure billing data used in this analysis is obtained from a single anonymous organization. As a result, their spending patterns may differ when compared to other companies, so generalization of results is difficult.

Another challenge is that anomaly detection has no ground truth. The contamination parameter was set to 0.05, since flagging about 5% of observations as anomalies seemed reasonable. But there is no labeled dataset indicating which instances are overpriced or which days are genuinely anomalous, so this choice is based on a reasonable judgment call.

The AWS pricing data represents a single snapshot of the pricing catalog. AWS changes pricing frequently, adds new instance families, and adjusts discount models. Any model trained on this data would need to be retrained periodically to stay accurate.

For the AutoML process, I gave FLAML a 120-second time budget. A longer search period could have enabled it to evaluate additional model configurations and identify better hyperparameters. But, with an R^2 of 0.9999 there is not much scope for improvement.

IX. TECHNICAL INFRASTRUCTURE

All the analysis was implemented in Python using Jupyter notebooks. The main libraries were FLAML [2] for AutoML regression, Prophet [4] for time series forecasting and statsmodels for ARIMA and stationarity testing. I also used scikit-learn [10] for Isolation Forest anomaly detection, LabelEncoder, train-test splitting, and evaluation metrics (R^2, MAE, RMSE).For data manipulation and visualization and pandas, NumPy, Matplotlib, and Seaborn were used.

The notebook runs from start to finish in about 5 minutes on a standard laptop. The FLAML training step takes the longest at 2 minutes due to the time budget I set, while the remaining steps complete in less than a minute each.

X. CONCLUSION AND FUTURE WORK

This project illustrates that you can predict, forecast, and monitor cloud infrastructure costs using standard ML methods.

FLAML [2] has identified a CatBoost [3] model that can predicts hourly EC2 costs with almost perfect accuracy. Both Prophet [4] and ARIMA were able to provide useful short-term forecasts even with small amount of data. Isolation Forest [5] helped identify overpriced configurations and unusual spending patterns. The pricing analysis suggests that many organizations could save significant costs, but they stick to on-demand pricing instead of many cost-efficient options [8].

One of the notable findings was the large variation in efficiency across instance families. Some instance types deliver up to ten times more compute per dollar compared to others, but this difference mostly goes unnoticed since organizations rarely analyze pricing efficiency thoroughly. Another key observation was the strong performance of a gradient-boosted tree model in predicting pricing. while an R^2 of 0.9999 may seem unusually high, but AWS pricing is highly predictable pattern, because higher hardware specifications result in higher costs.

The analysis did not rely on complex algorithms or large computational resources. The entire pipeline can easily run on a standard laptop in few minutes. The main contribution lies in combining these tools into a single workflow, so that the framework addresses multiple perspectives, and aligns well with the FinOps approach [7].

For future work:

- Integrating the AWS cost Explored API and Azure Cost Management API,so that it can allow the system to work with live billing data instead of static CSV files.
- Building automated alerting would be another useful enhancement. For instance, sending slack notification if the forecasts predicts budget overruns or detects anomalies.
- Exploring more on LSTM or Transformer-based forecasting models, particularly when more historical data is available.
- Building a Streamlit dashboard would make the system much easier for non-technical users to interact and explore forecasts and recommendations.
- Acquiring 12 or more months of billing data would help to properly validate the time series models.

REFERENCES

[1] B. Alan, "AWS EC2 instance comparison," Kaggle, 2024, accessed: 2026-01-20. [Online]. Available: https://www.kaggle.com/datasets/berkayalan/aws-ec2-instance-comparison

[2] C. Wang, Q. Wu, M. Weimer, and E. Zhu, "FLAML: A fast and lightweight AutoML library," in *Proc. MLSys*, 2021. [Online]. Available: https://github.com/microsoft/FLAML

[3] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased boosting with categorical features," *arXiv preprint arXiv:1706.09516*, 2018. [Online]. Available: https://arxiv.org/abs/1706.09516

[4] S. J. Taylor and B. Letham, "Forecasting at scale," *PeerJ Preprints*, vol. 5, p. e3190v2, 2018. [Online]. Available: https://peerj.com/preprints/3190/

[5] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. IEEE International Conference on Data Mining (ICDM)*, 2008, pp. 413–422. [Online]. Available: https://ieeexplore.ieee.org/document/4781136

[6] Flexera, "2024 state of the cloud report: Cloud computing trends," Flexera Blog, 2024, accessed: 2026-02-15. [Online]. Available: https://www.flexera.com/blog/finops/cloud-computing-trends-flexera-2024-state-of-the-cloud-report/

[7] FinOps Foundation, "What is FinOps?" FinOps.org, 2024, accessed: 2026-02-10. [Online]. Available: https://www.finops.org/introduction/what-is-finops/

[8] Amazon Web Services, "Amazon EC2 pricing," AWS Documentation, 2025, accessed: 2026-02-15. [Online]. Available: https://aws.amazon.com/ec2/pricing/

[9] Carrucciu, "Azure subscription costs," Kaggle, 2023, accessed: 2026-02-01. [Online]. Available: https://www.kaggle.com/datasets/carrucciu/azure-costs

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://jmlr.org/papers/v12/pedregosa11a.html