Université de Lille
Faculté des Sciences et Technologies

Leveraging Machine Learning for Dynamic and Intelligent Cloud Commitment Recommendations

Pierre LAGUE

sudo_

Sudo Group

Département d'Informatique
Faculté des Sciences et Technologies

June, 2025

.

Candidate: Pierre LAGUE, No. 42315164,
pierre.lague.etu@univ-lille.fr

Apprenticeship Supervisor: Mike DOUIEB, m.douieb@sudogroup.fr

Company: Sudo Group

Academic Supervisor: Marc TOMMASI, marc.tommasi@univ-lille.fr

Département d'Informatique
Faculté des Sciences et Technologies
Campus Cité Scientifique, Bât. M3 extension, 59655 Villeneuve-d'Ascq

June, 2025

Acknowledgements

I would like to express my gratitude to the University of Lille and Sudo Group for offering me the opportunity to complete this apprenticeship, which has been both enriching and instructive in many different ways. I am grateful to Mr. Tommasi, my academic supervisor, for his continuous guidance and support throughout this experience. I also extend my thanks to Mr. Mike Douieb, my apprenticeship supervisor, for his involvement, expertise and pieces of advice during this apprenticeship. A warm thank you goes to my colleagues Allan, Wajih, Yuzhe, Yasmine, and Nassim, whose support, collaboration, and kindness made a real difference in my daily work and learning.

Abstract

This report outlines the work completed during my apprenticeship, which was part of my Master's program in Computer Science and Machine Learning at the University of Lille 1 (FST). In my role as a Machine Learning Engineer at a FinOps consulting firm, I was tasked with designing and implementing automated solutions to dynamically manage and optimize our clients' cloud infrastructures.

My work revolved around two key, complementary projects. The "VM Reservation" project focused on building a dynamic recommendation engine for virtual machine reservations based on predictive forecasting. The "Alpha Stability" project involved developing a model to estimate and quantify the workload stability of a virtual machine, providing a critical metric for strategic decisions.

Through these projects, I honed advanced technical skills in data analysis, predictive modeling, hyperparameter tuning, and MLOps fundamentals. This experience also enhanced my problem-solving abilities, autonomy, and deep understanding of the business imperatives in the FinOps domain. This document provides a detailed account of the methodologies, results, and key takeaways from my work.

Key Words:

- GreenIT
- FinOps
- Forecasting
- Supervised Learning
- Data Science
- Machine Learning

Contents

0.1 Legal Notice

This thesis was prepared in the course of an apprenticeship with Sudo Group, and all work, research, data, and findings contained herein constitute a work-for-hire. As such, this document and its entire contents are the sole and exclusive intellectual property of Sudo Group. This work is confidential and may not be reproduced, distributed, or disclosed in any form, in whole or in part, without the express prior written authorization of Sudo Group.

Chapter 1

Introduction

This document is submitted as part of my final year in the Master of Computer Science program, specializing in Machine Learning, which I am completing through an apprenticeship at the University of Lille 1. The document aims to provide a reflective and objective demonstration of the work and methodology I undertook during this period. It is not simply a report on my work, but rather an analysis of what has been achieved—and what has succeeded or failed—in the process of working on the following subject: **Leveraging Machine Learning for Dynamic and Intelligent Cloud Commitment Recommendations**

It was within this academic framework that I joined Sudo, a consulting firm founded in 2020 specializing in digital transformation and data. Sudo distinguishes itself through its advanced expertise in FinOps and Green IT, supporting Information Systems Departments (ISDs) and other companies that wish to innovate while optimizing their costs and reducing the environmental and social impact associated with their use of Cloud services. Beyond its consulting activities, Sudo is also developing a proprietary SaaS FinOps platform designed to automate these optimizations.

As an Apprentice ML Engineer within Sudo's Data team, my objectives for this first year were multifaceted. On a professional level, the primary goals were to fully integrate into the company's projects and culture while building practical skills in data collection, analysis, and modeling. The overarching ambition was to actively

contribute to Sudo's core mission: developing digital solutions that empower clients to control their Cloud expenditures and diminish their carbon footprint.

1.1 Context of the Apprenticeship

There was two aim for this internship. The first and main one, was to finance my studies. The second one was to figure out which type of PhD I was to follow next year. I was hesitant between a fully academic PhD in a research unit, or a CIFRE PhD where I would be working in pair with a separate company.

1.1.1 Sudo Group

Founded in 2020, Sudo is a young and dynamic consulting firm that has quickly established itself as an innovative player in the Cloud Computing ecosystem. Despite its current size of seven employees, the company has a clear ambition: to accelerate the technological transformation of its clients. Sudo's primary mission is twofold: on one hand, to optimize its clients' Cloud architectures to reduce their costs (FinOps), and on the other, to decrease their environmental and social impact (Green IT).

Sudo's culture is centered on innovation, advanced technical expertise, continuous optimization, and a strong commitment to environmental responsibility. The company's values are reflected in its pursuit of tailor-made, results-oriented solutions for its clients.

Industry Sector and Positioning

The Cloud Computing market is experiencing hypergrowth, with a projected value exceeding $700 billion in 2024 and sustained growth momentum. However, this rapid expansion comes with significant structural challenges for businesses. It is estimated that $300 billion is wasted annually worldwide on Cloud commitments, often due to erroneous forecasts or a slower-than-anticipated migration to the Cloud. Simultaneously, approximately 50% of companies struggle to understand and predict their Cloud usage as they expand their infrastructure or adopt particularly resource-intensive technologies, such as Artificial Intelligence. Finally, a large majority—nearly 80% of companies—admit to having difficulties in comprehensively and accurately measuring the carbon footprint of their Cloud applications.

It is in this dynamic environment and in the face of these critical issues that Sudo positions itself. Sudo operates at the crossroads of digital transformation, Data, and more specifically, FinOps (Financial Optimization of Cloud Operations) and Green IT (Sustainable IT). This positioning is particularly relevant in a context

where companies seek both to control their Cloud spending and to meet sustainability imperatives. The company does not merely provide consulting services but is also developing a SaaS FinOps platform, which aims to automate the optimization of costs and the carbon footprint of applications running on the Cloud, without requiring deep technical expertise from the user.

What is FinOps?
FinOps is a discipline and a cultural practice that aims to maximize the business value of the Cloud. It involves helping organizations understand their Cloud costs, make informed spending decisions, and optimize their resource utilization. This requires collaboration between financial, technical, and business teams, and is often structured around three phases: Inform, Optimize, and Operate.

Faced with the rapid evolution of Cloud technologies and the growing demand for sustainable solutions, Sudo encounters stimulating challenges, such as the need to maintain constant technological surveillance. However, it also seizes significant growth opportunities by providing concrete answers to these problems of waste, complexity, and environmental impact.

Service Offerings and Target Clientele

Sudo's offerings are structured around audit, support, and training missions. These services are designed to provide customized, results-oriented solutions, enabling clients to realize their full technological, economic, social, and environmental potential. Sudo's target clientele primarily consists of Information Systems Departments (ISDs) and companies of all sizes that are engaged in innovation and wish to optimize their use of Cloud services.

Internal Organization

With a small team, Sudo's organization is agile and promotes direct collaboration. The typical project process at Sudo begins with a phase of identifying and defining client needs, often led by Mr. Mike DOUIEB, a FinOps expert and the founder of Sudo. These needs are then translated into project objectives for the Data team, which takes charge of designing and developing the technical solutions.

Although there is no complex formal organizational chart due to the structure's size, roles are clearly defined, with leadership provided by Mike DOUIEB and specialized operational teams, including the Data team. Communication is fluid and decision-making channels are short, allowing for great reactivity.

Presentation of Roles within Sudo Group

The lifecycle of a project at Sudo involves the collaboration of several areas of expertise:

- **Product & Strategy Expertise (FinOps/Green IT):** This role is central to defining the vision for solutions, technical expertise, and product management. Mr. Mike DOUIEB embodies this expertise, particularly in the areas of FinOps and Green IT.
- **Cloud Development and DevOps Consultant:** Focused on providing technical support to clients in the implementation and optimization of Cloud solutions.
- **Data Science & Digital Transformation Consultant:** Assists clients in leveraging their data, defining data strategies, and implementing transformation projects.
- **Data Scientist / Machine Learning Engineer (my role):** Responsible for the collection, processing, analysis, and modeling of data to develop solutions and insights.

Apprenticeship Advisor

My apprenticeship supervisor is Mike DOUIEB, the founder of Sudo. In his capacity as supervisor, he is responsible for setting the objectives for my projects and ensuring their relevance to both client requirements and the company's strategic goals. My advancements in the various subjects I worked on are supported through his guidance on technical and business matters, the delegation of increasing responsibilities, and feedback.

1.1.2 Apprenticeship Mission

As an apprentice Data Scientist / ML Engineer, my role is at the heart of Sudo's strategy. I am primarily involved in the data value chain: from data collection and preparation to its modeling (statistically or via ML algorithms) and exploitation. My objective is to transform raw data into actionable information and digital solutions that directly contribute to the company's mission: reducing the costs and carbon footprint of its clients' Cloud infrastructures, specifically virtual machine usage on the cloud. I am therefore required to interact with data from various Cloud providers (Azure, GCP, AWS) and to develop tools and models to analyze and optimize them.

1.1.3 Introduction to My Host Department: The Data Team

Main Missions of the Department

The Data Team, within which I work, is the technical engine of the solutions developed by Sudo. Its main missions are to:

- Develop and maintain data analysis scripts to evaluate Cloud consumption and the financial gains from our recommendations.
- Design, train, and deploy Machine Learning models to predict trends (such as virtual machine usage on the cloud) or identify optimization opportunities.
- Contribute to the development of Sudo's SaaS FinOps platform by integrating features resulting from the team's R&D work.
- Conduct a rigorous scientific analysis of the results obtained, documenting the methodologies employed and the conclusions, in accordance with an engineering approach.

1.2 Thesis Overview

During this work we will go over 2 major parts of my work at Sudo. The first one is the essence of my work: *VMReservation*. This chapter will present the complete methodology and thought process that we went through to come up with the first stable version of the core feature of our product that allows a dynamic and intelligent recommendation system for Reserved Instances (RI) in the Azure cloud environment. I will then present *Alpha Stability* , which works hand in hand with *VMReservation* and will present the interesting contribution it represents. *AlphaStability* aims at estimating the workload of a virtual machine running a certain application in a certain working environment.

The VMReservation chapter centers on a core machine learning problem: the time series forecasting of cloud resource consumption. Specifically, the objective is to predict the future trajectory of a single, scalar metric: the Daily vCPU Usage Intensity (I) with supervised machine learning. As we will see later in this work, this metric represents the aggregated average vCPU usage for a particular combination of resource attributes (e.g., VM series, location, environment) on a given day. The task is therefore to forecast the time series constituted by the sequence of these I values over a defined future horizon. This predicted time series of Daily vCPU Usage Intensity (often referred to conceptually as 'VM Usage' in subsequent figures and discussions) forms the essential foundation upon which the data-driven Reserved Instance recommendation logic is built. This project comes hand in hand with

feature engineering tasks and model optimisation in order to have a 6 month forecast of the intensity I that offers actionable reservation recommendations.

The Alpha Stability analysis, detailed in the second chapter, addresses a related but distinct problem of quantifying historical usage predictability to provide supplementary context for these primary consumption forecasts. The aim is normalize application and environment names to build a reference catalog working as a key value system. The keys, a pair of application name and environment name, will refer a value: the stability of the workload of that application running on the given environment.

Chapter 2

Background

Cloud computing has fundamentally reshaped the IT landscape, offering unprecedented scalability, flexibility, and access to advanced technologies. Businesses increasingly migrate workloads to major cloud providers like Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP) to leverage these advantages and shift from capital expenditure (CapEx) on physical hardware to operational expenditure (OpEx) based on consumption. However, this consumption-based model, while flexible, introduces significant challenges in cost management.

2.1 Cloud Cost: Flexibility vs. Financial Waste

The pay-as-you-go pricing model (paying for a resource as you use it) offers maximum flexibility but can lead to unpredictable and escalating costs, especially for stable, long-running workloads. To address this, cloud providers offer commitment-based purchasing options, such as Azure Reserved Instances (RIs) and Savings Plans. These mechanisms provide substantial discounts (often up to 72% compared to on-demand rates) in exchange for a commitment to use specific resources (like Virtual Machines - VMs) or a certain amount of compute power for a defined term, typically one or three years.

While these commitments are powerful levers for cost optimization, they introduce a new risk: waste. Organizations often struggle with accurately forecasting their future cloud infrastructure needs. Factors like evolving application demands,

unforeseen project delays, shifting business priorities, and the inherent complexity of cloud billing data make precise prediction difficult. This imprecise understanding leads to significant financial inefficiency. In a 2023 report by Infosys, potentially over $300 billion in cloud commitments are wasted annually due to mismatches between committed resources and actual usage. This wasted spend directly impacts profitability and represents a major hurdle in maximizing the financial benefits of the cloud.

2.1.1 Quantifying Cloud Value

Effective cloud financial operations/managment (often termed FinOps) hinges on optimizing cloud spend and demonstrating clear value. Key financial metrics include:

- Total Cost of Ownership (TCO): Comparing the cost of running workloads in the cloud versus on-premises requires careful TCO analysis. Commitment discounts are crucial for making cloud TCO favorable for many workloads.
- Return on Investment (ROI): Businesses need to justify cloud investments. Optimized spending through intelligent commitments directly improves the ROI of cloud initiatives.
- Budget Predictability: Fluctuating on-demand costs complicate financial planning. Converting baseline usage to fixed-cost RIs or Savings Plans enhances budget predictability, although inaccurate commitments can negate this benefit.
- Cost Savings: Directly measurable savings achieved by leveraging discounts compared to pay-as-you-go rates are a primary driver for commitment strategies.

Mastering commitment strategies is therefore essential not just for saving money, but for achieving broader financial control and demonstrating the strategic value of cloud adoption.

2.2 Sustainability Dimension: Carbon Footprint

Beyond financial considerations, the environmental impact of cloud computing is gaining increasing attention. Data centers are significant consumers of electricity, contributing to global carbon emissions. While cloud providers invest heavily in renewable energy and efficient operations, the sheer scale of cloud infrastructure necessitates optimization at the user level as well.

Wasted cloud commitments often translate to underutilized or idle reserved resources. Even if paid for, these resources consume energy within the data center. By accurately predicting usage and right-sizing commitments, organizations can:

- Reduce Energy Consumption: Ensure that provisioned resources closely match actual demand, minimizing energy wasted on idle capacity.
- Lower Carbon Footprint: Directly decrease the carbon emissions associated with powering and cooling unnecessary infrastructure.
- Align with ESG Goals: Contribute to corporate Environmental, Social, and Governance (ESG) objectives by demonstrating responsible resource management.

Optimizing cloud commitments is thus intrinsically linked to sustainability. Efficient resource utilization is not only financially prudent but also environmentally responsible.

2.3 Need for Dynamic Intelligent Solutions

The core challenge lies in accurately "knowing" future usage patterns. Traditional methods for managing commitments often rely on manual analysis, static rules, or basic heuristics provided by cloud vendor tools. These approaches frequently lack the sophistication to handle the inherent volatility and complexity of cloud usage data, making commitment decisions risky and prone to failure. This context highlights a critical need for more dynamic, adaptive, and intelligent approaches. Machine Learning (ML) offers a promising path forward. By leveraging ML algorithms, particularly time-series forecasting techniques, organizations can analyze vast amounts of historical consumption data, identify complex patterns, predict future needs with greater accuracy, and factor in various influencing variables.

This thesis explores precisely this opportunity: leveraging Machine Learning to develop a system that provides dynamic and intelligent recommendations for Azure Virtual Machine commitments. The goal is to move beyond static, reactive cost management towards a proactive, data-driven strategy that maximizes financial savings through optimized RIs, enhances budget predictability, and implicitly supports environmental sustainability by promoting efficient resource utilization. The development of such a system addresses the technological gap in mastering cost optimization via commitments in complex cloud environments.

2.4 Related Works

Research on leveraging machine learning (ML) for dynamic and intelligent cloud commitment recommendations intersects several domains: ML-driven decision support, cloud resource optimization, and recommendation systems. While direct studies on "cloud commitment recommendations" are limited, related works address ML applications in cloud environments for decision-making, resource allocation, and policy recommendations.

ML and Cloud Computing for Decision Support

A recent study explores how ML, artificial intelligence (AI), and cloud computing can be combined to build scalable, flexible, and effective decision support systems for fraud detection in the insurance sector. The paper proposes a framework that leverages ML for accurate predictions, AI for decision automation, and cloud computing for scalability, ultimately providing actionable recommendations to insurers [1].

Another work presents a decision support system for crop recommendations using ML classification algorithms and cloud data. The system analyzes large-scale agricultural datasets in the cloud to provide personalized, context-aware recommendations to farmers, demonstrating the potential of cloud-based ML for dynamic recommendation tasks [2].

Cloud-Based Policy and Resource Recommendation

Research on cloud-based ML for flood policy in Makassar, Indonesia, utilizes multiple data sources and ML models on cloud platforms to generate policy recommendations for urban planning and disaster mitigation. This work highlights the value of integrating ML and cloud resources for dynamic, data-driven policy advice [3].

ML in Cloud Environments

A study demonstrates the deployment of ML models in the cloud for medical image analysis, emphasizing the efficiency and workflow improvements achieved through cloud-based ML. The research suggests that cloud integration enables scalable and dynamic decision support, which is relevant for commitment recommendations in other domains [4].

Similarly, ML models, such as XGBoost, have been used in cloud environments to predict air quality, integrating real-time data and providing actionable recommendations for users. This approach showcases how ML and cloud computing together facilitate timely, adaptive recommendations based on dynamic data [5].

Cloud Provider Integrated Recommendation Features

Major cloud providers, such as AWS, Azure, and Google Cloud, offer built-in tools and recommendation engines to help customers choose commitment plans and optimize resource usage. These solutions typically leverage proprietary algorithms and historical usage data to suggest reserved instances, savings plans, or other long-term commitments. While these tools provide convenience and some degree of automation, they are fundamentally designed within the provider's ecosystem and often prioritize the provider's financial interests. As a result, recommendations may be biased toward options that maximize provider revenue, such as encouraging overcommitment or steering users toward products with higher margins. This inherent bias means that while provider-driven recommendation systems can be useful, they may not always align perfectly with the customer's optimal cost-saving strategy or business objectives [6, 7]. Independent or hybrid approaches, which incorporate user-centric data and third-party analytics, are increasingly being explored to address these limitations and provide more objective, tailored recommendations for cloud commitments.

That being said, while direct literature on "dynamic and intelligent cloud commitment recommendations" is scarce, the reviewed works collectively demonstrate the feasibility and effectiveness of leveraging ML in cloud environments for dynamic, data-driven recommendation and decision support systems. These studies provide a strong foundation for further research in applying ML for cloud commitment optimization and recommendations. This brief overview also emphasizes the novelty of our work, and how we place ourselves in an effort to produce non-biased solutions that is aimed at the client's needs first.

Chapter 3

VMReservation

In this chapter, we provide an overview of the process that led to the first stable version of our Virtual Machine Recommendation system. We will focus on the key development phases, the rationale behind our design choices, and the main challenges encountered along the way.

3.1 Motivations

As mentioned in Chapter 2, commitment is one of the most powerful levers for cost optimization in the Cloud. However, due to an imprecise understanding of usage as well as delays in Cloud migration projects, more than 300 billion dollars in Cloud commitments are wasted each year (source: Cloud Radar 2023 - Infosys). The main technological padlock to mastering cost optimization via commitments is "knowing" the future usage of one's cloud infrastructure. In fact, to this day, it is hard to have a very precise understanding of cloud usage and the various external or internal factors that could make it change from one day to the next. Some technologies such as time series forecasting exist to iteratively predict the future based on past data, and in some cases the models yield significant results, making the future not so uncertain. But in the case of cloud usage, the difficulty lies within the data. In fact, cloud billing data is extremely complex, to understand, and to manipulate. Building a model in itself isn't hard, but feeding it with the appropriate data, derived features and extracting practical results out of the model's output is the real challenge that we aim to illustrate in this work.

Our strategy is to try and incorporate a forecasting and recommendation engine designed to optimize Microsoft Azure Virtual Machine (VM) utilization. By analyzing historical cloud usage patterns and predicting future needs for specific VM, this system would identify prime candidates for Azure Reserved Instances (RIs). Adopting these recommendations would present significant, quantifiable advantages in one critical area: financial expenditure.

As specified in Section 2, the total expenditure of wasted commitments goes up to 300 billion dollars. The primary financial driver for utilizing Azure Reserved Instances is substantial cost reduction illustrated by the following points:

1. Discounted Rates: Azure offers significant discounts (often up to 72% compared to pay-as-you-go rates, varying by term, region, and VM type) for committing to use specific VM resources for a 1-year or 3-year term. Our recommendation feature identifies workloads with consistent usage patterns that are ideal for maximizing these savings. By reserving capacity for these predictable workloads, the client transitions from premium, flexible pricing to heavily discounted, committed pricing.

2. Enhanced Budget Predictability: Pay-as-you-go VM costs can fluctuate based on usage variations. Committing to RIs for baseline workloads transforms a variable operational expense into a more predictable, fixed cost for the duration of the term. This greatly aids financial planning, forecasting, and budget management for cloud infrastructure. However, the lack of dynamic and "intelligent", data-driven methods make this process risky and prone to failure.

3. Optimized Cloud Spend: In theory, the system pinpoints opportunities where sustained usage makes pay-as-you-go economically inefficient. By converting these specific, identified VMs to RIs based on data-driven forecasts, we ensure that budget is allocated optimally, paying the lowest possible price for required, long-term compute capacity.

The VM reservation recommendation system would serve as a critical tool for strategic cloud management. It would directly translate forecasted usage into actionable financial savings by leveraging Azure's Reserved Instance pricing model. This principle is destined to be combined with other reservable computing units (databases, disks etc.). It could also handle saving plans on top of the RI reservation to handle excess costs. It is important to understand that to this day, NO previous work has been done to implement and propose a dynamic and intelligent (leveraging ML) way of harnessing commitment methods in the Azure environment. Our work is from the ground up and we have faced countless challenges in terms of ideas, implementation and logic to achieve our first version.

3.2 Preliminary Considerations

In this section we will detail the preliminary work that had to be done in order to move beyond a simple conceptual idea of our system. We will go through a brief summary of the data we have used, how we engineered it, the choice of the model, the feature engineering and how we came up with the type of results we present to the client.

3.2.1 Details on the Data

This study is empirically grounded in data derived from the Microsoft Azure cloud environment of 70 different companies. For the purposes of this research, the initial, unprocessed collection of this information will be referred to as the "dataset". Subsequent transformations and subsets of this data will be defined as they are introduced throughout this work. A foundational aspect of our data processing methodology involved determining the optimal sequence for two principal analytical stages: (1) the comprehension and processing of cloud billing data, (2) the choice of model and overall system architecture for a Machine Learning approach, and (3) the underlying logic for proposing reserved instances with newly introduced metrics such as application-environment stability.

3.2.2 Understanding and Processing Cloud Billing Data

Billing data is an extremely complex type of data, it has numerous columns with various meanings and normalizing the data model was one of the most important steps. The final data model we work with comprises client identification information, cloud usage information, virtual machine information and datetime information. The columns used in our analysis are the following:

- Client and Billing Identifiers:
  - `invoice_section`: A categorical attribute representing a defined section within the billing invoice, utilized for client or departmental segregation.
  - `billing_account`: The primary identifier for the account responsible for financial settlement.
- Resource Specification and Usage Metrics:
  - `meter_name`: A descriptive name for the billed resource, typically identifying the virtual machine type or service.
  - `meter_id`: A unique code corresponding to the specific type of resource being measured.

  - `meter_sub_category`: A granular classification within a broader meter category, providing further specificity (e.g., "Dsv5 Series" for virtual machines).
  - `location`: The geographical Azure region of resource deployment (e.g.,'westeurope').
  - `commitment_pricing_model`: An indicator of the applied pricing structure, distinguishing between commitment-based models (e.g., Reserved Instance, Savings Plan, Spot) and standard Pay-As-You-Go rates.
  - `resource_vcpus`: The number of virtual Central Processing Units (vCPUs) allocated to the resource instance.
  - `quantity`: A measure of the resource consumed, typically in hours for virtual machines, between 0 and 24, since the data is day-based.

- Environmental and Contextual Attributes:
  - `environment`: A user-defined attribute, often derived from tags, intended to specify the resource's operational context (e.g., Production, Development, Staging).
  - `tag_key_unfiltered`: The key associated with a resource tag. This attribute is instrumental not only for potential environment inference but also for filtering transient or auxiliary resources (e.g., those associated with Databricks clusters).

- Temporal Information:
  - `usage_day_utc`:The Coordinated Universal Time (UTC) date on which the resource usage was recorded.

- Financial Attributes:
  - `billing_currency`: The currency denomination of the billing charges (e.g., USD, EUR).
  - `effective_price`: The actual financial cost incurred for the resource usage, incorporating any applicable discounts from commitments or other pricing agreements.
  - `payg_price`: The hypothetical cost that the same usage would have incurred if billed at standard Pay-As-You-Go rates, serving as a baseline for evaluating commitment savings.

The meticulous selection, interpretation, and normalization of these attributes formed the foundational data layer upon which all subsequent feature engineering, model development, and analytical interpretations are built.

3.2.3 Data Filtering and Scope Refinement

To ensure the analytical dataset was focused on relevant resource utilization patterns pertinent to commitment optimization, a series of filtering criteria were applied. The primary objective was to isolate Virtual Machine (VM) instances whose consumption characteristics and pricing models were suitable candidates for Reserved Instance (RI) consideration. Initially, the dataset was constrained to include VMs operating under "OnDemand" (Pay-As-You-Go) or "Reservation" pricing models, as these represent the primary states between which RI optimization occurs. However, further refinement was necessary to exclude specific usage types or pricing constructs that would introduce noise or were outside the immediate scope of this phase of RI analysis. Consequently, records associated with the following were excluded:

- "Spot" Pricing Model: VMs under this model are subject to preemption and represent transient, interruptible workloads, making them unsuitable for long-term commitment strategies like RIs.
- "Databricks" Workloads: Databricks pricing presents a bifurcated cost structure, comprising both the underlying cloud provider's virtual machine costs and the Databricks Unit (DBU) consumption fees. While RIs can and should be applied to the underlying VM infrastructure, the DBU component represents a separate, platform-level cost. This analysis is scoped to infrastructure-level commitments (RIs) only. Optimizing DBU costs through Databricks-specific commitments (i.e., pre-purchasing DBUs) constitutes a distinct optimization vector that introduces separate analytical requirements and is therefore considered out of scope for our work, for the moment.
- "AKS Node Pools": Azure Kubernetes Service (AKS) node pools are fundamentally comprised of Virtual Machine Scale Sets, making them technically eligible for RI coverage. However, their lifecycle is governed by the Kubernetes control plane and automation, such as the Cluster Autoscaler, which dynamically adjusts the node count based on workload demand. This introduces significant volatility in the number and even type of active instances. Committing to a static number of RIs for such a dynamic fleet carries a high risk of either underutilization (if the cluster scales in) or missed savings (if it scales out). Consequently, these workloads were excluded from this initial RI analysis to await a more specialized strategy that can accommodate their elastic nature.

These filtering steps were crucial for creating a dataset representative of stable, potentially reservable VM workloads, enhancing the signal-to-noise ratio for subsequent forecasting and recommendation tasks.

3.2.4 Environment Normalization

The operational environment of a VM significantly influences its usage patterns; for instance, production environments typically exhibit more stable and sustained utilization compared to development or testing environments, which may display more erratic or fluctuating consumption. Acknowledging this, a critical step involved the normalization and stratification of environment indicators. For the initial phase of this research, a binary classification of environments was adopted to simplify the analytical complexity while still capturing a primary axis of usage variability. Identified environment indicators were categorized into two distinct classes:

- "prd" (Production): Representing resources dedicated to live, operational workloads.
- "non_prd" (Non-Production): Encompassing all other environments, including but not limited to development, testing, staging, and quality assurance.

This categorization allows for a foundational understanding of how environment type correlates with usage stability and RI suitability. While this initial binary approach provides a tractable starting point, a more granular clustering and classification of distinct non-production environments has been subsequently developed to enable a more nuanced analysis that reflects the diverse usage profiles within the "non_prd" category. This refined multi-class environment model enhances the precision of usage forecasting and commitment recommendations by proposing a wider spectrum of operational contexts.

3.2.5 Generating vCPU Information

A key parameter for quantifying resource consumption is the number of virtual Central Processing Units (vCPUs) allocated to each Virtual Machine (VM) instance. The methodology for determining this vCPU count varied based on the pricing model attribute associated with each resource record.

- Direct Availability for OnDemand Instances: For VMs operating under an "OnDemand" (Pay-As-You-Go) pricing model, vCPU count information is typically explicitly available as a distinct attribute within the billing data, as pricing for such instances often correlates with vCPU allocation.
- Inferential Extraction for Reservation Instances: Conversely, for VMs under a "Reservation" pricing model, the direct vCPU count attribute is often unpopulated or not consistently provided. However, this information is crucial for accurate usage intensity calculations. To address this, an inferential approach was adopted. The meter_name attribute, which describes the VM type (e.g.,

"B1s"), typically embeds the vCPU count within its standardized nomenclature (e.g., "B1s" indicates 1 vCPU in the B-series). A regular expression was developed and applied to the meter_name for these "Reservation" instances to parse and extract the implicit vCPU count.

This dual approach ensured that a vCPU count could be reliably associated with each relevant VM record, regardless of its pricing model. The extracted or inferred vCPU count was then populated into a standardized resource_vcpus field, forming a critical input for the subsequent calculation of the Daily vCPU Usage Intensity metric.

3.2.6 Core Metric Calculation

Then, we were able to define the core metric of this study: the Daily vCPU Usage Intensity (I). That is, a resources' total vCPU usage distributed over a 24-hour period. This metric is computed as follows:

$$I = \frac{q * n}{24}$$

Where q is the quantity (hours a resource has been used) and n is the number of vCPU used by that resource. To normalize the data, the result is divided by 24, converting this value into average vCPU usage per day. This transformation allows us to understand the daily usage intensity for each virtual machine, ensuring that the subsequent analysis is based on consistent time-based units. I is the scalar value we will train our model to predict in the future in our forecasting-regression problem.

3.2.7 Aggregating Candidates for Reserved Instances

Following the initial data preprocessing and enrichment stages, a critical step involved defining the fundamental analytical units for assessing Reserved Instance (RI) candidacy. This was achieved by aggregating resource consumption data based on specific multi-attribute 'combinations'. Two hierarchical levels of combinations were defined:

- Precise Combination (PC): This represents the most granular level of analysis and is defined by a unique tuple of the following attributes:
  - `invoice_section` (client/departmental identifier)
  - `meter_sub_category` (specific VM type/series)
  - `location` (Azure deployment region)
  - `environment` (normalized environment category)

This granularity is hypothesized to be crucial for accurately modeling usage patterns and identifying suitable RI candidates, as environments are significant implicit drivers of consumption behavior.

- Broad Combination (BC): This represents a higher-level aggregation, defined by a unique tuple of:
  - `invoice_section`
  - `meter_sub_category`
  - `location`

The BC effectively subsumes multiple PCs that share the same client, VM type, and region, irrespective of their specific environment.

A significant methodological challenge lies in finding the optimal analytical scope for these combinations, particularly when considering the distinct requirements of model training versus practical inference and recommendation generation. For model training (e.g., forecasting usage stability or future consumption, later described in this work), the Precise Combination (PC) is generally preferred. Its granularity allows the model to learn nuanced patterns influenced by specific virtual machines, locations, invoice sections, and environments, which might be obscured at higher aggregation levels.

However, for generating actionable RI recommendations, which are often made at the level of a VM type within a location for a given client (approximating the Broad Combination - BC), a decision must be made. Recommendations cannot always be tied to a highly specific environment, as RIs are typically purchased for a VM series in a region. Therefore, a strategy is required to either train models at the PC level and then intelligently aggregate their predictions or stability assessments up to the BC level for recommendation, or to potentially train separate, perhaps simpler, models directly at the BC level using aggregated historical data. This study primarily leverages PCs for detailed stability and pattern analysis, with subsequent aggregation of insights for broader recommendation scopes.

3.3 Machine Learning Approach

Optimizing Reserved Instance (RI) commitments within the dynamic Azure cloud environment presents a significant analytical challenge, necessitating accurate, long-term usage forecasting. Azure resource consumption exhibits considerable heterogeneity across different virtual machine series, locations, invoice sections, and operational environments. This complexity is further compounded by inherent consumption volatility, potential seasonality, and the occurrence of data gaps or anomalies

within billing records. The primary objective of this research phase was to develop a robust forecasting methodology capable of navigating these complexities to inform strategic 1-year and 3-year RI purchase decisions, directly minimizing cloud expenditure while ensuring adequate capacity coverage.

3.3.1 Initial Combination-Specific Modeling Strategy

The initial methodological approach centered on capturing the specific consumption nuances of each resource combination by training individual time-series forecasting models for every distinct Broad Combination (BC), as previously defined (i.e., a unique invoice_section, meter_sub_category, and location). The selection of an appropriate model architecture was a key consideration.

Tests with Facebook Prophet

Initial evaluations explored the Prophet forecasting model, developed by Facebook. Prophet, known for its robust handling of seasonality, trend changes, and holidays, and its foundation in additive regression models (conceptually similar to generalized additive models, GAMs, often incorporating ARIMA-like error structures), was tested for its adaptability. However, this model proved suboptimal for the specific characteristics of many cloud VM consumption patterns. Numerous VM workloads, particularly those targeted for RI consideration, exhibit sustained, high utilization (approaching 100which inherently lacks strong seasonality or cyclical components that are a primary strength of Prophet. Consequently, the Prophet-based approach did not yield sufficiently accurate or reliable forecasts for this domain.

Transition to XGBoost: A More Robust Solution

Subsequently, attention shifted to the XGBoost (Extreme Gradient Boosting) model, a gradient-boosted decision tree ensemble known for its high performance and flexibility in capturing complex, non-linear patterns in data that may not exhibit regular cyclicality. This model was chosen for the following reasons:

The development of the XGBoost models involved extensive feature engineering, incorporating a rich set of predictors derived from the time series data. These included:

- Temporal Components: Date-part features (e.g., day of week, month, year), time indices, and polynomial trend representations.
- Cyclical Encodings: Sine and cosine transformations of time-based features to capture potential periodicities.
- Lag Features: Past consumption values at various lags.

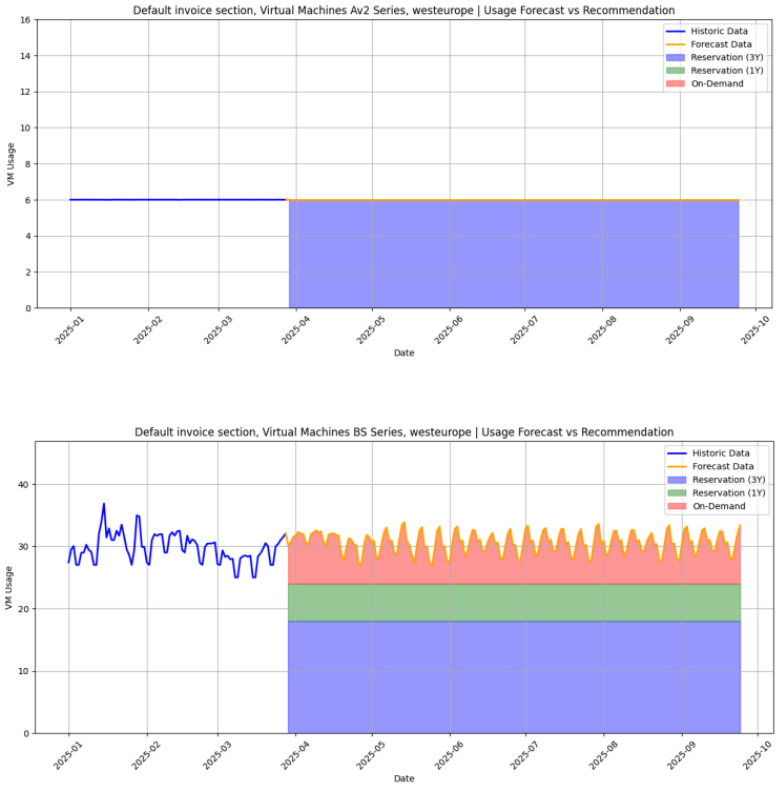

Figure 3.1: Combination-specific models provide an accurate forecast w.r.t the historical data, but offer poor deployability.

- Rolling Window Statistics: Moving averages, standard deviations, and other statistical measures calculated over defined historical windows.
- Difference-Based Metrics: First and second-order differences to capture rates of change (velocity) and acceleration in consumption.

The rationale underpinning this highly localized, combination-specific modeling strategy was the hypothesis that individual models, tailored to the unique historical behavior of each BC, would yield the most accurate representations of their consumption dynamics.

Re-Evaluation the Combination-Specific Strategy

Figure 3.1 illustrates 2 inferences from 2 individual models. The blue line is the historical usage of the virtual machines both located in "westeurope" servers. The first one (above) illustrates the model that was trained on flat historical daily vCPU usage intensity (I) and proposed a flat forecasted $\hat{I}$ over 6 months in the future. The blue zone represents an actionable reservation of 6 vCPU of the virtual machine "Av2 Series" for a 3 years commitment. More details on these will come later in this work.

The second graph of Figure 3.1 (under) illustrates a model that was trained on historically fluctuating data. It proposes a forecast $\hat{I}$ that seemingly follows such fluctuating trends. The blue zone represents the actionable reservation of vcpu for a 3 years commitment, the green zone represents the actionable reservation of vcpu for a 1 year commitment, and the red zone represents the excess on-demand usage each for a virtual machine "BS Series".

Despite the theoretical appeal and the demonstrable accuracy of individual XGBoost models on their specific training data, this initial strategy encountered substantial practical and methodological obstacles that limited its viability for strategic, long-term RI decision-making:

- Computational Overhead: The training of potentially hundreds or thousands of individual XGBoost models, one for each BC, imposed a significant computational burden. This was particularly noted by the necessity of performing hyperparameter optimization (e.g., via GridSearchCV or randomized search) for each distinct model to achieve optimal performance, leading to infeasible execution times for a production-scale system
- Forecast Reliability for Strategic Decisions: While individual models exhibited high fidelity to past data, their long-range forecasts (1-3 years) often proved unreliable or lacked the intuitive coherence required for making substantial financial commitments. Highly localized models can be prone to overfitting to specific historical idiosyncrasies, leading to divergent or implausible long-term extrapolations.
- Scalability and Maintenance: Managing and maintaining a large ensemble of independent models presents considerable operational challenges.

These limitations necessitated a re-evaluation of the forecasting approach, prompting an investigation into more scalable and robust methodologies capable of balancing localized accuracy with generalizability and computational tractability for strategic RI planning.

3.3.2 Transition to a Global Forecasting Model and Methodological Refinements

The computational intractability and deployment complexities associated with the combination-specific modeling strategy necessitated a significant methodological pivot. A global forecasting model, trained on a concatenated dataset comprising time series from all relevant Broad Combinations (BCs), was adopted. While this approach inherently addressed the scalability and computational challenges, it introduced the

new scientific problem of enabling a single model to effectively differentiate and learn the unique consumption behaviors of diverse BCs.

Initial Challenges with a Global Model

Initial iterations of the global model exhibited several common failure modes, producing forecasts unsuitable for reliable Reserved Instance (RI) planning. These included:

- Unrealistic Growth/Decline: Forecasts demonstrating explosive, unbounded growth (often attributable to the instability of polynomial time features) or precipitous declines immediately following the historical data horizon (suggesting issues with feature calculation at prediction boundaries or excessive sensitivity to end-of-series noise).
- Inertial Flat-Lining: Projections remaining inexplicably flat despite evident historical variations, indicative of potential over-regularization or the model's failure to capture underlying dynamics.
- Lack of Differentiation: A critical issue was the model's tendency to produce nearly identical forecast patterns for distinct resource types (e.g., 'Eav4' vs. 'BS' series VMs), indicating an inability to effectively leverage categorical information to discern combination-specific nuances.

Such unreliable forecasts rendered the subsequent RI optimization calculations (e.g., determining the optimal mix of 1-year RIs, 3-year RIs, and On-Demand capacity) highly uncertain and prone to suboptimal outcomes.

Methodological Adaptations for the Global Model

To address these shortcomings and enhance the global model's capacity to learn differentiated, stable forecasts, several key adaptations were implemented:

- Training Target Consistency: The model was trained directly on the individual time series of each BC, rather than on aggregations or transformations that might obscure specific patterns.
- Consistent Temporal Feature Engineering: Rigorous attention was paid to ensuring that time-based features (e.g., time_index) were calculated consistently, relative to a global reference point (e.g., a fixed start date), across both historical data and the future prediction horizon.
- Stabilizing Feature Selection: Features identified as inherently unstable or prone to noise amplification, such as high-order polynomial time trends and

highly sensitive difference-based metrics (e.g., acceleration), were systematically excluded or regularized.

- Explicit Combination Encoding and Interaction Features: Categorical identifiers (e.g., VM series, location, invoice section) were transformed using One-Hot Encoding. Crucially, interaction features were engineered by combining these one-hot encoded identifiers with key temporal and lag features. This explicitly guides the model to learn distinct trends, seasonalities, and responses to lagged consumption specific to each category or combination of categories.
- Data Integrity and Preprocessing: Time series were preprocessed to handle discontinuities. Gaps in historical data were imputed using forward-filling to provide more continuous input for lag and rolling window features. Conversely, specific periods or combinations identified as exhibiting extreme, unrepresentative noise were selectively excluded from the training set to prevent undue influence on the global model.
- Regularization and Model Complexity Control: Careful tuning of XGBoost hyperparameters (e.g., max_depth, gamma, min_child_weight, lambda, alpha) was performed, in conjunction with techniques like early stopping during training, to achieve a balance where the model could capture genuine variations without overfitting to noise present in the aggregated dataset.

Addressing Data Scarcity through Data Augmentation

A significant challenge encountered during the training of the global forecasting model was the influence of historical data imbalances and the limited availability of recent, representative consumption patterns. Initial training attempts utilizing the full available historical data (approximately 2-3 years, 1000 daily data points per combination) resulted in models that exhibited a pessimistic bias. This was attributed to the prevalence of low-usage periods in the older historical data, often preceding significant "move-to-cloud" transitions or substantial workload expansions within organizations. For instance, if a substantial increase in usage only occurred in the most recent 5 months, approximately 70% of the training data might represent a low-utilization regime, unduly influencing the model to under-predict future consumption, particularly for combinations that had recently scaled. This was evidenced by forecasts initiating at levels below the most recent observed values, indicating an inability to adequately capture recent upward trends. To mitigate this temporal bias and improve the model's responsiveness to recent trends, two key strategies were adopted:

- Focused Training Window: The training dataset was temporally constrained to the most recent period of significant, representative usage, typically the latest

6 months (approximately 180 days). While this reduces the overall volume of historical data, it concentrates the model's learning on patterns most relevant to current and future consumption levels.

- Time Series Data Augmentation: To compensate for the reduced size of the training window and to enhance model robustness, data augmentation techniques specific to time series were employed. These techniques artificially expand the training dataset by creating plausible synthetic variations of the original sequences. The primary methods utilized were:
  - Jittering: Introducing small, random Gaussian noise to the consumption values of the training series. This helps the model become less sensitive to minor, exact fluctuations in the historical data.
  - Time Warping: Applying smooth, non-linear distortions to the time axis of segments within the training series (locally stretching or compressing time). This makes the model more resilient to slight variations in the timing of events or patterns.

Hyperparameter Optimization

Given the complexity of the cloud consumption data, the engineered feature set, and the XGBoost algorithm itself, achieving optimal predictive performance necessitates careful tuning of the model's hyperparameters. Default parameters often lead to suboptimal outcomes, such as underfitting (failing to capture complex patterns) or overfitting (memorizing training data noise and generalizing poorly to unseen data).

To systematically identify the optimal hyperparameter configuration for the global XGBoost model, an automated optimization routine utilizing the Optuna framework was implemented. Optuna was selected for its efficiency in navigating large hyperparameter search spaces compared to exhaustive methods like Grid Search. It employs intelligent sampling strategies (e.g., Tree-structured Parzen Estimator - TPE) to probabilistically explore the defined space, concentrating computational effort on more promising regions. A standard routine is the following:

- Defining a search space for key XGBoost hyperparameters (e.g., n_estimators, max_depth, learning_rate, gamma, min_child_weight, regularization parameters lambda and alpha).
- Iteratively training and evaluating the XGBoost model with different hyperparameter combinations suggested by Optuna.

- The objective function for Optuna is to minimize a predefined error metric (e.g., Mean Absolute Error - MAE) on a dedicated, temporally distinct validation dataset. This validation set preserves the temporal ordering of the data, ensuring that optimization is geared towards generalization to future, unseen time steps.

The hyperparameter optimization routine is crucial for maximizing the global model's predictive accuracy. By systematically searching for and selecting the hyperparameter set that minimizes error on the validation data, this process enhances the model's ability to generalize. The final forecasting model is then retrained on the combined training and validation data (or the full relevant historical window) using these optimized best parameters. This data-driven approach to hyperparameter selection significantly increases the likelihood of achieving robust and reliable performance on the final, unseen test set and in production forecasting scenarios. The routine is designed to be re-executable (e.g., weekly) to allow for continuous model improvement as new data becomes available and consumption patterns potentially evolve.

Performance Assessment of the Global Model

The global model, incorporating these refinements, offers significant advantages in terms of reduced computational intensity for training and streamlined deployment compared to the combination-specific approach. While the visual fidelity of its forecasts for individual BCs may not always match the close historical fit of highly localized models, the global model demonstrates improved overall stability and generalizability in its predictions, particularly for longer forecast horizons.

However, certain limitations persist. The model can exhibit challenges in accurately forecasting periods of sustained flat-line usage or precisely replicating sharp, short-term fluctuations, sometimes tending towards a pseudo-mean prediction based on the diverse patterns observed across the entire dataset. This behavior suggests that while categorical features aid differentiation, the model can still be influenced by the aggregate characteristics of the training data, particularly for combinations with less distinct historical signals or those susceptible to minor input variations.

Figure 3.2 illustrates 2 inference from 1 refined model on the same historical use of the virtual machines as displayed on Figure 3.1 (with a slight difference in time as this version had to be deployed on more recent data, therefore only the test set differ). On the graph above, we can see that the historically flat I is not well forecasted. In fact, the 6 month forecast displays noise that illustrates some limitations of our global model, leading to a reservation zone (in blue) that is underutilized at some periods (meaning the reservation exceeds the forecasted $\hat{I}$).

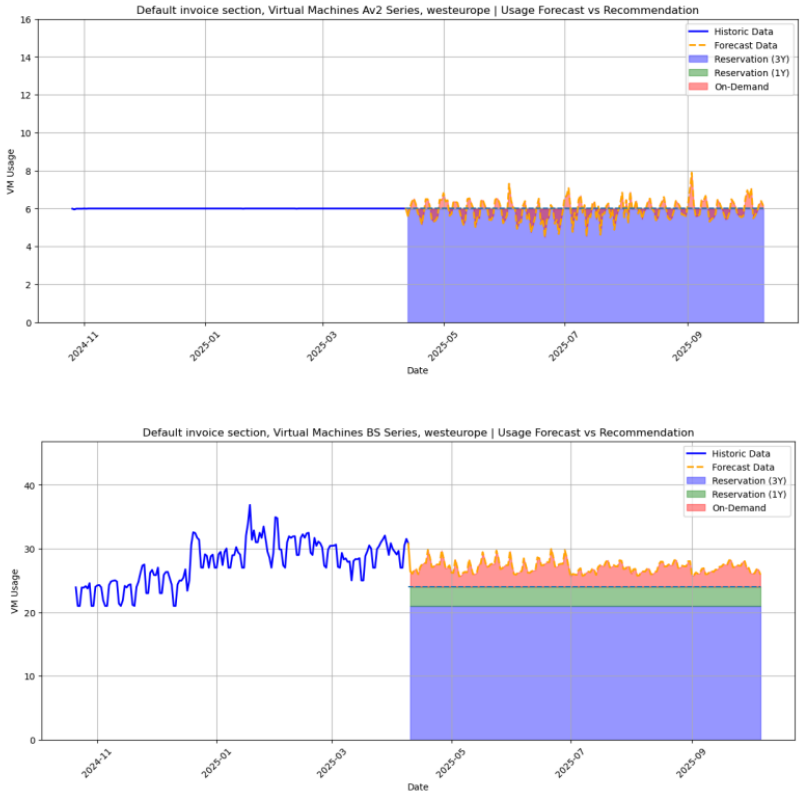

Figure 3.2: The general model still lacks forecasting quality. The flatline isn't properly forecasted on the upper image and the fluctuations are not properly illustrated in the lower image which tends to predict a pseudo-mean value out of the historical data.

The second plot illustrates the forecast of the fluctuating historical data of virtual machine "BS Series" with a forecast of poor quality. In fact, as soon as it begins, $\hat{I}$ suffers a violent drop that does not seem to follow the growing trend, then the intensity spikes do not follow the profile of the historical data. Both of these plots illustrate the presence of noisy data in the training data, making our model inaccurate and unreliable.

	Initial Global Model	Refined Global Model
Mean-Average-Error	8.729	1.135
R^2	0.119	0.975
Number of Features	22	172

Table 3.1: Model performance on the test set after training. The refined model incorporates data augmentation, 180 days of historical data, enhanced feature engineering, and hyperparameter optimization. While yielding better results, it is not reliable and does not provide the expected forecasts.

Quantitative evaluation on a held-out test set indicates that this revised methodology yields forecasts with improved error metrics compared to the less stable extrapolations from the initial global model attempts as displayed in Table 3.1. The metrics used are the following:

- **MAE:** The Mean-Average-Error measures the absolute difference between the predicted values ($\hat{I}_t$) and the actual values (y_t):
  - $MAE = \frac{1}{n}\sum_{t=1}^{n} |\hat{I}_t - y_t|$, with n being the number of samples in our test set.
  - The rationale for this metric in our forecasting setup is that it gives a clear measure of average error magnitude, as well as being robust to outliers compared to squared error metrics.
- **R-squared:** The Coefficient of Determination measure the proportion of variance in the observed data that is predictable from the input features:
  - $R^2 = 1 - \frac{\sum_{t=1}^{n}(y_t - \hat{I}_t)^2}{\sum_{t=1}^{n}(y_t - \bar{y})^2}$, with n being the number of samples in our test set.
  - * If R-squared = 1, then the model achieves perfect predictions
    * R-squared = 0, then the model performs no better than predicting the mean
    * R-squared < 0, then the model performs worse than simply predicting the mean.

  - The rationale for this metric is that it explains how well our model captures the variance of our data, which in our case is extremely important. Also, it stands as a good baseline metric for comparison with other models.

Using both of these metrics helps balance interpretability and statistical rigor, especially important in time series forecasting, where both the size of errors and the structure of the underlying signal matter.

3.3.3 Scalability Challenges for Production Deployment

While the global forecasting model offers significant advantages in terms of centralized training and deployment, its application to extensive, multi-organizational datasets (e.g., encompassing data from over 70 distinct companies) revealed substantial scalability challenges. These challenges primarily manifest in two interconnected areas: feature dimensionality and computational overhead during inference.

Proliferation of Features in Diverse Datasets

The one-hot encoding strategy for categorical identifiers (such as meter_sub_category, location, invoice_section, and environment), while crucial for enabling the global model to learn combination-specific nuances, leads to a rapid expansion of the feature space as the diversity of these categorical values increases. In larger, aggregated datasets, the number of unique values for these identifiers can grow substantially, resulting in models with a very high number of input features (e.g., potentially exceeding 900 features in some scenarios).

This high dimensionality presents several problems:

- Increased Model Complexity: Models with a vast number of features can become more difficult to train, interpret, and debug.
- Computational Cost of Feature Engineering: The necessity to generate these numerous features (including one-hot encoded categories and their interactions with temporal/lag features) for each inference step becomes computationally expensive.
- Inference Latency for Autoregressive Forecasting: Given the autoregressive nature of the forecasting model (where predictions are fed back as inputs for subsequent steps), generating a long-range forecast (e.g., 180 days) requires iterative feature creation and prediction. High feature counts significantly increase the latency of this iterative process, potentially rendering it impractical for timely decision-making (inference for up to 45 minutes with a 900 feature model, as well as intensive memory usage).

Mitigating Scalability Issues and Enhancing Efficiency

To address these scalability concerns and ensure the practical viability of the global model, several data reduction and feature management strategies have been investigated. The principle is to intelligently filter and simplify the input data and feature set before model training and inference, thereby reducing computational load without catastrophically sacrificing predictive performance on key combinations (less is more).

Key strategies include:

- Pre-filtering of Low-Impact Combinations: A primary approach involves identifying and excluding Broad Combinations (BCs) or Precise Combinations (PCs) that contribute minimally to overall consumption or RI optimization potential. This can be achieved by using thresholds of consistency checks.
- Strategic Categorical Feature Management: Instead of one-hot encoding every unique environment string, a more structured approach involves mapping diverse raw environment inputs to a smaller, predefined set of canonical environment categories (e.g., 'production', 'development', 'testing', 'staging', 'sandbox'). Further aggregation, such as grouping 'development' and 'sandbox' if their usage patterns are statistically similar, can further reduce dimensionality
- Iterative Feature Selection and Pruning: Post-initial model training, feature importance analysis (e.g., using SHAP values or feature permutation importance from XGBoost) can identify features (including specific one-hot encoded categories or interaction terms) that have minimal predictive power. These can be pruned to create a more parsimonious model without significant loss of accuracy on critical combinations.

The objective of these strategies is to achieve a "Turing-esque" efficiency – to accomplish more (accurate and timely forecasting for key entities) with less (reduced data volume, fewer features, lower computational cost). This involves a continuous cycle of intelligent filtering, feature engineering refinement, and model optimization to ensure the forecasting system remains scalable and performant when deployed across increasingly large and diverse cloud environments.

3.3.4 Results of the Data and Feature Filtering Strategy

The implementation of a multi-faceted filtering strategy, targeting both noisy data combinations and non-contributory features, yielded significant improvements in the performance and efficiency of the global forecasting model (Table 3.2). Applying filtering criteria to consumption combinations, specifically excluding those with usage

intensity below a defined threshold set to 6 or exhibiting excessively high historical coefficient of variation set to 0.02, successfully removed data points unlikely to represent stable or reservable workloads.

Figure 3.4 illustrates the inference of our filtered model on the same historical I of the virtual machines displayed on Figure 3.2. The first graph illustrates a near perfect forecast of a historically flat I on the virtual machine "Av2 Series". Only very little noise is present on the forecast, but such variations can be ignored at this scale.

The second graph of Figure 3.4 illustrates the forecast for a historically fluctuating I. The forecast displays appropriate trends and spike profiles illustrating a cyclical tendency that is significantly better than any of the previous versions of the global model for the virtual machine "BS Series".

	Initial Global Model	Refined Global Model	Filtered Global Model
Mean-Average-Error	8.729	1.135	**0.705**
R^2	0.119	0.975	**0.983**
Number of Features	22	172	**24**

Table 3.2: Comparison of the versions of the model. The filtered model has achieved the best results so far. The filtered model is the refined model, with a filtering strategy.

This data-level filtering enhanced the signal-to-noise ratio within the training data, enabling the model to better discern and learn representative usage patterns. Complementing this, a systematic feature selection analysis was conducted. This analysis revealed that, contrary to initial hypotheses regarding complexity, the interaction features engineered between categorical identifiers and temporal/lag variables did not provide significant predictive benefit for this particular dataset and model architecture.

Consequently, the final feature set was streamlined to retain only the base temporal and engineered features alongside the one-hot encoded categorical identifiers. This combined data and feature filtering approach significantly reduced the feature dimensionality, decreasing the number of input features from approximately 100 to a more parsimonious set of around 20. The result was a substantial improvement in overall predictive performance, observed consistently both on the independent test set used for model validation and during combination-specific inference, demonstrating that a more focused and simplified input indeed allowed the model to achieve greater accuracy and robustness. Although the performance of the filtered model on the test set is similar to the refined models' performance, the results are better with

a smaller amount of features (Figure 3.3). As we can see, the filtered model has a close to perfect fit to the test data, harnessing strong fluctuations and patterns.

Interestingly, evaluation of the models (Table 3.2) revealed a significant outcome of the filtering strategy. The ability of the model to explain the variance in the data, as quantified by the R^2 metric, remained comparable between the refined model and the filtered model. Crucially, this similar level of variance explanation was achieved despite the filtered model utilizing a significantly reduced number of input features. This finding strongly suggests that a substantial portion of the features in the original, more complex model did not contribute to capturing the underlying data variance. Furthermore, the filtered model demonstrated superior predictive performance, exhibiting a lower Mean Absolute Error (MAE). This indicates that the removed features were not merely redundant but likely introduced noise or complexity that negatively impacted the model's ability to generalize, making it more 'error-prone' on unseen data. Achieving better performance with a substantially smaller feature set highlights the efficiency and robustness gains of the filtering strategy.

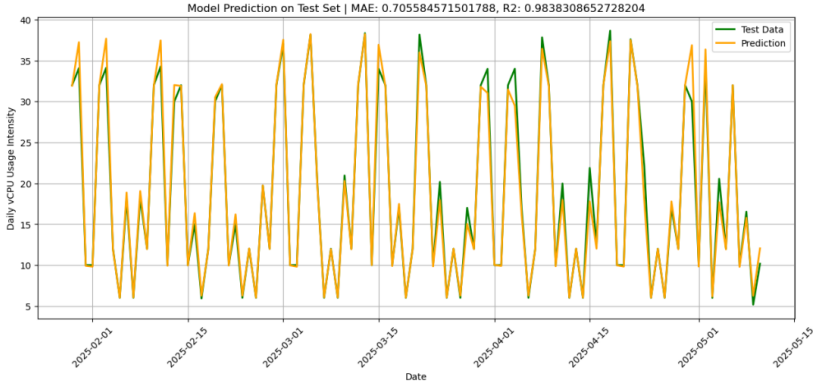

Figure 3.3: Plot of the predictions of our filtered model on the test set. The model is able to fit almost perfectly to highly fluctuating data yielding results that match our expectations in terms of reliability.

3.4 Reserved Instance Recommendation Strategy

Following the generation of usage forecasts by the global model, the subsequent critical phase involves translating these predictions into actionable Reserved Instance (RI) commitment recommendations. This section details the methodology developed for this translation, focusing on the derivation of a conservative baseline for reservations and the rationale behind its calculation.

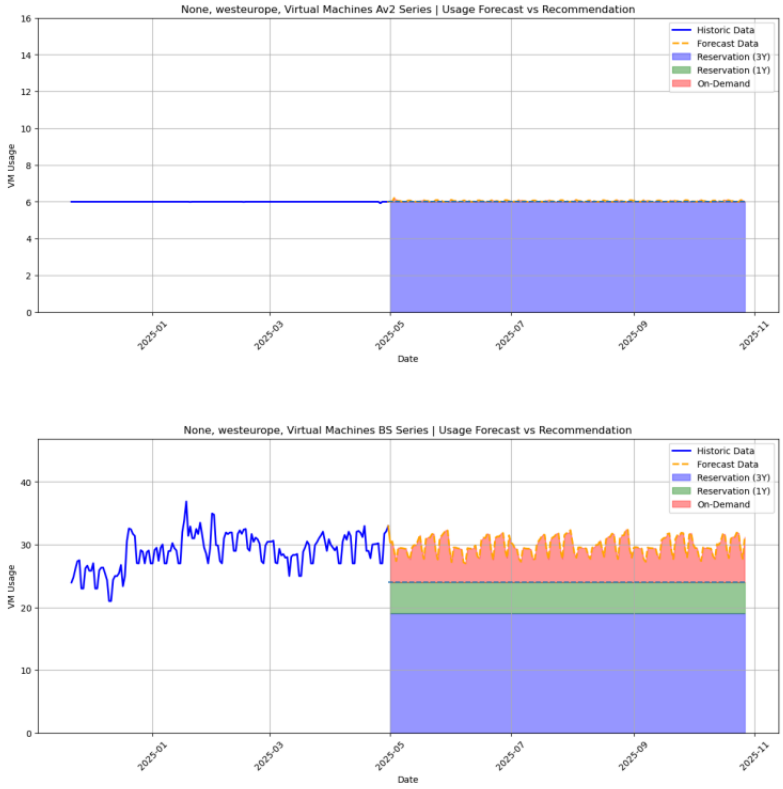

Figure 3.4: By filtering the data before training, we have removed a substantial amount of noisy data, allowing the inference to be only focused on the proper candidates for optimization. The model performs well on the test set, and displays good generalization capabilities.

3.4.1 Bridging vCPU Usage Forecast and Actionable Recommendations

While the forecasting model provides predictions of future Daily vCPU Usage Intensity (I), directly converting these point forecasts into RI purchase decisions is imprudent. Forecasts are inherently probabilistic and subject to error. Committing to RIs based solely on mean predicted usage carries a significant financial risk: if actual usage falls below the forecast, the reserved capacity becomes underutilized, negating potential savings and possibly incurring net losses. Therefore, a robust methodology is required to determine a "safe" level of commitment that balances cost-saving opportunities with the risk of over-commitment.

The Minimum Lower Bound (MLB)

The Minimum Lower Bound (MLB) is a statistically derived, conservative estimate of future usage that forms the basis of a Reserved Instance (RI) recommendation strategy. The goal is to identify a usage level that can be covered by long-term RIs (3-year and 1-year) with high confidence, minimizing the risk of underutilization.

Predicted usage above the MLB is handled by more flexible options like On-Demand capacity.

Initial and Refined Derivation The MLB is calculated using the model's point forecast ($\hat{I}$) and the standard deviation of its Mean Absolute Error on a test set, $\sigma(MAE_{test})$, which quantifies the variability of the model's prediction errors.

The initial formula was based on a 95% confidence interval:

$$MLB = \hat{I} - 1.96 * \sigma(MAE_{test})$$

However, this was found to be overly restrictive and led to conservative recommendations that missed potential savings.

To achieve a better practical balance between risk mitigation and cost optimization, the formula was refined to:

$$MLB = \hat{I} - 3 * \sigma(MAE_{test})$$

The choice of the factor 3 is justified by:

- **Increased Confidence:** It provides a more robust lower bound, referencing Chebyshev's inequality and capturing a very high percentage of deviations (99.7% for a normal distribution).
- **Error-Adaptive Conservatism:** The MLB's conservatism dynamically adjusts to model performance. A high $\sigma(MAE_{test})$ (inconsistent errors) pushes the MLB lower for a more conservative recommendation, while a low $\sigma(MAE_{test})$ (consistent errors) keeps the MLB closer to the forecast $\hat{I}$.

Finally, a **historical floor sanity check** is applied. The methodology takes the maximum of the forecast-based MLB and a conservative percentile of recent historical usage. This acts as a failsafe, ensuring the recommendation remains robust even if the forecast model is temporarily inaccurate.

Incorporating Application-Environment Stability Indicators

The intrinsic value of a Reserved Instance is maximized when aplied to predictably utilized workloads. A critical factor influencing this predictability is the combined stability of the operational environment and the specific application running on the VM. While the MLB provides a conservative usage floor, allocating this reservable capacity across different RI terms (e.g., 1-year vs. 3-year) or adjusting the proportion of the MLB to reserve can be further refined by assessing the historical stability of each application-environment pairing. Conventional heuristics, such as fixed percentage allocations (e.g., an 80/20 split of the MLB between production

and non-production reservations), lack the granularity to adapt to diverse stability profiles and may lead to suboptimal commitment strategies.

To address this, a dynamic stability factor (α) is introduced. This factor, derived from a separate stability analysis for each unique application-environment pair, quantifies the historical consistency and predictability of their usage patterns. α typically ranges from 0 to 1, where higher values indicate greater stability.

Instead of a fixed percentage, α is used to modulate the portion of the MLB considered for longer-term, higher-discount RIs (e.g., 3-year terms). For example, the capacity recommended for 3-year RIs for a specific application-environment combination might be calculated as α * MLB. More stable combinations (higher α) would thus be recommended for a larger proportion of their MLB to be covered by longer, more cost-effective commitments, while less stable applications or environments would see a more conservative allocation to such terms. This approach allows for a data-driven, adaptive allocation of the reservable capacity, moving beyond static rules to tailor recommendations to the observed behavior of each specific workload. This approach will be detailed in a later chapter as it represents a study of its own.

3.4.2 Reflection on the Stability Indicator

Assessment of the dynamic stability indicator against a simpler, static heuristic allocation method (specifically, a fixed 80/20 split between broad production and non-production categories) revealed outcomes that must be contextualized by the scope of the initial analysis. This study was conducted within the operational framework of a single small-to-medium-sized enterprise, whose cloud footprint is characterized by a relatively simple structure: costs are concentrated in a few VM families, and workloads are predominantly production-oriented. Within this specific, homogenous environment, the granular, data-driven approach of α did not yield a significant improvement in the financial effectiveness of the Reserved Instance recommendations over the simpler heuristic.

Furthermore, the implementation of this more granular approach introduced additional complexity into the data processing pipeline, notably requiring the consistent inclusion of the application identifier as a key grouping dimension, thereby increasing computational overhead. Given that the recommendations derived using α were comparable to those from the straightforward 80/20 rule in this limited test case, we have retained the simpler heuristic for the current model. We posit that the full value of the dynamic stability factor will become apparent when applied to more complex and heterogeneous environments. To this end, we plan to evaluate

the α indicator on a much broader dataset, such as the one encompassing the 70 distinct corporate entities, where we anticipate its ability to model nuanced stability differences will prove to be of greater impact and interest.

	Stability-enhanced Model	**No Stability Model**
Mean-Average-Error	2.68	0.705
R^2	0.823	0.983
Number of Features	50	24

Table 3.3: Model performance comparison. The no stability model outperforms the stability-enhanced model, which has more granular information (detailed environments and application). This increases the number of features and complexifies the model, yielding less accuracy.

The results in Table 3.3, which show the 80/20 model outperforming the more complex α-based model, should be interpreted with this context in mind. For this specific test set, the simpler model not only performed better but also generated more financially optimistic recommendations while remaining within a safe risk profile. This reinforces the conclusion that for a relatively stable and simple cloud estate, a broad heuristic is highly effective. However, this outperformance is considered a function of the dataset's simplicity rather than a definitive weakness of the α indicator itself.

Other Use for the Stability Indicator

While the direct integration of the stability indicator into the primary allocation formula did not yield the anticipated performance gains (Table 3.3), the underlying "Alpha Stability Catalog" resulting from this analysis retains significant value as a data product. This catalog, which provides a quantified metric of empirical stability for each application-environment pair, offers several alternative research and practical perspectives. Notably, it can serve as a valuable confidence indicator for the recommendations generated by the system; a higher α for a given combination suggests that its historical usage has been more predictable, thereby providing a basis for higher confidence in the associated forecast and RI recommendations.

Furthermore, the core methodology developed for calculating α – assessing workload predictability based on historical usage patterns – is conceptually transferable and can be applied to evaluate the stability and suitability for commitment of workloads running on other Azure services beyond Virtual Machines (e.g., databases, container instances), offering a broader toolset for cloud resource optimization and workload management.

3.4.3 Incorporating Existing Reservations

Any new RI recommendations must consider the organization's current RI portfolio to avoid over-reservation and to accurately determine the net new commitment required. Simply recommending the full MLB without accounting for existing reservations would lead to redundant and financially inefficient commitments. The forecasting and MLB calculation are performed on the total observed usage, inclusive of workloads already covered by existing RIs. This is because the underlying usage pattern, regardless of current RI coverage, is what needs to be understood for future planning. Existing reservations, which typically manifest as sustained, flat-line usage at or near 100% utilization for the reserved capacity, do not require complex forecasting for their own usage (as their usage is, by definition, committed).

Therefore, to determine the additional capacity to reserve, the sum of vCPUs already covered by active, relevant existing RIs for a given combination (RI_{exist}) is substracted from the calculated MLB. The net new reservable vCPU quantity (RI_{new}) is then:

$$RI_{new} = MLB - RI_{exist}$$

This RI_{new} value, if positive, then becomes the target for new RI purchaase recommendations, to be allocated across different RI terms based on factors like the stability indicator α and organizational commitment preferences. This standard and logical approach ensures that recommendations are based on the incremental capacity required to meet the conservative usage floor (MLB) once existing commitments are factored in. It directly quantifies the additional vCPU units that should be reserved to achieve the target conservative coverage.

3.4.4 Deriving and Allocating the Global RI Recommendation

With the Minimum Lower Bound (MLB) established, application-environment stability factor α quantified, and existing RIs accounted for, the next step is to formulate a global RI recommendation (GR) for the net new capacity required. This recommendation must then be optimally allocated between 1-year and 3-year terms.

Formulation of the Initial Global RI Recommendation

Initial conceptualizations considered a heuristic allocation of the net reservable capacity based on broad environment categories. For instance, a naive approach might allocate 80% of this net capacity for production environments and 20% for non-production environments, assuming a binary environment classification:

$$GR = ((0.8 * MLB) + (0.2 * MLB)) - RI_{exist}$$

with RI_{exist} being the existing RI. However, this static percentage-based approach lacks the granularity to adapt to the diverse stability profiles of specific application-environment pairings.

Formulation of the Refined Global RI Recommendation

Leveraging the α factor, which reflects the historical stability of a specific application-environment combination (PC), allows for a more nuanced determination of the portion of the MLB that should be confidently targeted for reservation. The global net new RI recommendation (GR) for a given PC is therefore calculated as:

$$GR = max(0, round(\alpha * MLB - RI_{exist}))$$

Where:

- α is the stability factor for the specific application and environment the recommendation is being built upon.
- MLB is the Minimum Lower Bound calculated for that combinations' forecast.
- RI_{exist} are the vCPUs already covered by existing RI for that combination.
- round() is applied for practical whole vCPU or instance recommendations.
- max(0, ...) ensures that the recommendation is non negative.

This formulation directly links the recommended reservable capacity to the demonstrated stability of the workload; more stable workloads (higher α) will have a larger proportion of their MLB (after accounting for existing RIs) recommended for new commitments. The necessary application and environment identifiers are available during inference as they were integral to the PC definition used for forecasting and stability analysis, allowing for a direct lookup of the relevant α from a pre-computed "Alpha Catalog".

3.4.5 Allocation of 1-Year and 3-Year RI Terms

Once the global recommendation (GR) for a given broader scope (BC) is determined, it must be strategically allocated between 1-year and 3-year RI terms. While 3-year RIs offer the highest discounts, they also entail a longer commitment period, increasing the risk associated with potential future decreases in usage that might fall below the reserved level. A dynamic allocation method is required, moving beyond fixed percentage splits to one that considers the confidence and stability of the forecast itself. To address this, the allocation strategy incorporates a measure of the forecast's own volatility. The principle is that more stable, less fluctuating forecasts warrant a greater proportion of the GR to be allocated to 3-year RIs, while

more volatile forecasts suggest a more cautious approach with a larger allocation to 1-year RIs.

The capacity allocated to 3-year RIs (RI_{3Y}) is determined by further adjusting the GR based on the standard deviation of the forecast itself $\sigma(\hat{I})$ over the relevant commitment horizon:

$$RI_{3Y} = max(0, round(GR - k_{split} * \sigma(\hat{I}))$$

Where:

- $\sigma(\hat{I})$ is the standard deviation of the forecasted daily usage intensity over a period of 180 days. A higher $\sigma(\hat{I})$ indicates a greater predicted fluctuation or uncertainty and vice-versa.
- k_{split} is a risk aversion coefficient. We have decided to set k_{split} to 3, in order to have the same coefficient as when building the minimum lower bound (MLB).

The remaining portion of the GR is allocated to 1-year RIs (RI_{1Y}:

$$RI_{1Y} = GR - RI_{3Y}$$

This approach ensures that if the forecast is highly stable (low $\sigma(\hat{I})$), RI_{3Y} will be close to GR as it is the most profitable term option. Conversely, if the forecast exhibits significant fluctuations (high $\sigma(\hat{I})$), RI_{3Y} will be reduced, shifting more capacity to the more flexible 1-year term, mitigating the risk of under-utilization for long-term commitments. An illustrative example of a stable forecast lending itself to a higher 3-year RI proportion is shown in Figure 3.4.

3.4.6 Estimating Financial Gain

Quantifying the potential financial gain represents a critical component of the Reserved Instance (RI) recommendation process, serving as the primary metric for demonstrating value to stakeholders. This estimation process translates the recommended commitment quantities into projected cost savings relative to standard Pay-As-You-Go (PAYG) rates.

To ensure stability and consistency in financial projections, the methodology shifted to basing cost and savings estimations solely on the empirical data derived from the Azure billing records themselves. This approach leverages the effective price (actual cost after discounts) and pay-as-you-go price (hypothetical cost at standard rates) attributes present in the processed historical data. In order to have a reliable estimation of the discounts and retail prices of each virtual machine in a given region, we have built a script which creates a CSV file updating each month

if pricing information changes. The CSV file is then accessed and queried with the virtual machine category and the location in which it is used by our clients. This allows us to extract information such as 1 year and 3 years discounts, as well as the retail price (D_{1Y}, D_{3Y}, and P_{hour} respectively). This shift also allows for better alignment between the usage data being forecasted and the pricing data used for estimation.

In previous versions, the script relied on hard-coded values for the discounts and retail price, which made the financial estimation of the recommendation error-prone. The estimated total global savings (GS) resulting from adopting the recommended new RI commitments (RI_{1Y} and RI_{1Y}) is then calculated using these internally derived pricing components. The calculation considers the volume allocated to each commitment term and their corresponding estimated discounts. The formula for estimating the total monthly savings is:

1. $$S_{1Y} = -H_{month} * P_{hour} * D_{1Y} * RI_{1Y}$$

2. $$S_{3Y} = -H_{month} * P_{hour} * D_{3Y} * RI_{3Y}$$

3. $$GS = S_{1Y} + S_{3Y}$$

Where:

- RI_{1Y} and RI_{3Y} represent the recommended net new capacity (e.g. in virtual machine instance count) for 1 year and 3 years RI terms.
- D_{1Y} and D_{1Y} are the estimated percentage discounts for 1 year and 3 year RIs for the specific VM type extracted from the current broad combination (BC).
- P_{hour} is the standard pay-as-you-go hourly price of the relevant VM, derived from the pricing CSV.
- H_{month}, represents the number of hours in a month. It is set to 730, as we assume a continuous use of the VM.

The resulting GS value represents the projected monthly financial negative saving anticipated from implementing the system's RI recommendations for a given broad combination. This quantitative impact assessment is a pivotal output, providing the necessary justification for commitment decisions and forming a key component of the final recommendation report presented to clients.

Illustrative Application and Financial Impact

In practice, the system analyzes historical consumption to generate forecasts (e.g., six-month or longer) for the daily vCPU Usage Intensity (I) at aggregated BC level (e.g., BS Series VMs in West Europe for a specific client). For workloads demonstrating stable forecasted demand (e.g., consistent 8 vCPU daily intensity from four E2a v4 instances), the methodology recommends reserving the appropriate number of VMs (e.g., four E2a v4 instances, or an equivalent vCPU capacity using the most cost-effective instance sizes within that series) for the calculated optimal terms.

Recommendations prioritize covering the stable baseline (GR) with the 1-year/3-year split informed by forecast volatility. The system quantifies the financial impact by calculating projected monthly savings against estimated Pay-As-You-Go costs (e.g., a hypothetical 61% discount yielding €1524.70 monthly savings for an illustrative Eav4 scenario if fully committed under optimal terms).

Chapter 4

Alpha Stability

4.1 Context

FinOps focuses on rightsizing cloud usage and optimizing cloud costs on platforms like Microso Azure. A widely adopted cost optimization strategy involves resource-based commitments—such as Reserved Instances (RIs)—to reserve specific types of Virtual Machines (VMs) in designated regions. This approach is oen considered one of the most effective ways to reduce cloud spend. However, in environments with large VM fleets, managing reservations at the individual resource level becomes increasingly tedious and inefficient. To address this, many organizations choose to reserve VMs in bulk at higher levels of the organizational hierarchy, targeting groups of VMs with similar characteristics. While this top-down, holistic strategy improves manageability and reservation coverage, it introduces trade-offs: reduced visibility into individual workloads and uncertainty about whether the applications running on these VMs will continue to be used over time. This makes it more difficult to determine the optimal reservation scope and term. A promising way to address this challenge is by evaluating the stability of the applications running on the VMs. Application stability provides a more business-aware perspective, which is often missing from standard usage forecasts. By factoring in stability, reservation recommendations become more reliable—for example, avoiding commitments on VMs running short-lived or unstable workloads. In addition, application stability can serve as a valuable feature for forecasting usage patterns, helping to model demand volatility more accurately. Our methodology analyzes the cloud billing data of over 70

companies (we will refer to it as the dataset) to derive stability insights based on historical VM usage. These insights can then be integrated into predictive models through supervised learning and unsupervised hierarchical clustering, using techniques such as Levenshtein distance and linkage analysis. A key part of this process is the estimation of an application stability metric (alpha), computed by analyzing normalized application usage over time. By systematically identifying relevant tags and standardizing application names, we create a robust framework for associating usage patterns with specific applications. This enables a scalable, data-driven approach to assessing workload stability and improving cost optimization decisions within the Azure environment.

4.2 Details on the Data

This study is empirically grounded in data derived from the Microsoft Azure cloud environment, specifically utilizing consolidated cloud billing records from a cohort of over 70 organizations. For the purposes of this research, the initial, unprocessed collection of this information will be referred to as the "dataset". Subsequent transformations and subsets of this data will be defined as they are introduced throughout this work.

A foundational aspect of our data processing methodology involved determining the optimal sequence for two principal analytical stages: (1) the identification and extraction of execution environment information, and (2) the classification of application-related tag keys followed by the clustering of their corresponding tag values. After careful consideration of data characteristics and analytical objectives, the extraction of environment information was strategically prioritized to precede the application-specific data refinement processes. This decision was deduced on a key observation regarding data scope: attributes pertinent to application identification and characterization represented a relatively small fraction of the overall dataset. Conversely, indicators of execution environments were found to be more pervasively distributed across the data. Performing application-centric filtering at an earlier stage would have resulted in the premature exclusion of a significant volume of data. While this excluded data might not be directly related to application specifics, its retention during the initial environment extraction phase was deemed crucial for preserving broader contextual integrity, which could be vital for comprehensive analysis or for accurately mapping refined application data back to the original, more expansive dataset. Therefore, by addressing environment extraction first, we aimed to maximize the information available for, and the contextual richness of, the subsequent, more granular application-level analyses. Finally, we consider

the dataset as a whole and do not proceed to the filtering of data that would be considered as noisy in other setups.

4.3 Extracting Relevant Tag Keys

Tags are a critical component of billing and metadata management. By allowing users to assign key–value pairs to resources, tags enable categorization by usage, ownership, application, and other business dimensions. However, tagging conventions are inconsistently applied in practice, with widespread deviation from expected structures. In particular, the challenge of identifying VMs associated with applications emerges due to the lack of standardized naming practices. Although some tag keys clearly reference applications ("application", "app", etc) there is no universal or enforced convention. This results in high variability and semantic ambiguity in the tags used. An early attempt to solve this through a rule-based system (nested if statements) quickly revealed its limitations in terms of scalability and maintainability. It became evident that a more intelligent, adaptive approach was required.

4.3.1 Baseline Model: Naive Bayes Classifier

The first machine learning-based solution implemented was a Naive Bayes classifier. Naive Bayes is a probabilistic model based on Bayes' theorem, which calculates the probability of a class C given a set of features F:

$$P(C|F) = \frac{P(F|C) * P(C)}{P(F)}$$

The "naïve" part is its core simplifying assumption: it treats all features as independent of each other given the class. This allows the likelihood $P(F|C)$ to be calculated simply as the product of individual probabilities:

$$P(F|C) = \prod_i P(f_i|C)$$

Thus the model predicts the class C that maximizes $P(C) * \prod_i P(f_i|C)$. It was initially chosen due to this inherent simplicity, computational efficiency (especially with high-dimensional data), and ease of interpretability, making it a suitable baseline model, particularly when working with limited initial datasets or when a fast, understandable first pass is needed. It was trained on a small set of manually labeled tag keys using a basic feature set. The tags were extracted from our initial data from 70 companies. They were labelled thanks to the expertise of domain experts. This approach yielded an initial accuracy of 0.8558.

However, upon inspection, the model demonstrated systematic misclassifications, notably false positives and false negatives, that domain experts could intuitively refute. In order for the classification to be deemed reliable our model needed to achieve at the very least 0.95 accuracy, to ensure that application-related tags were in fact classified as such. This highlighted the limitations of Naive Bayes in capturing the complexity of the tagging noise and the need for a more expressive model.

4.3.2 XGBoost: A More Robust Solution

To address these shortcomings and capture the intricate patterns missed by Naive Bayes, the next iteration involved implementing an XGBoost (Extreme Gradient Boosting) classifier. XGBoost is a powerful and widely-used gradient boosting framework based on decision tree ensembles. In a classification setup, it was selected for the following reasons:

- Ensemble Method (Boosting): Instead of relying on a single model like Naïve Bayes, XGBoost builds an ensemble of decision trees. It does this sequentially: each new tree is trained to correct the errors (specifically, the residuals or gradients of the loss function with respect to the predictions) made by the ensemble of previously trained trees. This iterative refinement process allows the model to gradually improve its predictions

- Regularization: A key strength of XGBoost is its built-in regularization techniques (L1 - Lasso, and L2 - Ridge regularization on tree complexity and leaf weights). This helps prevent overfitting by penalizing overly complex models, leading to better generalization on unseen data. This is crucial when moving from a simpler model like Naive Bayes, as more complex models have a higher risk of overfitting.

- Handling Complex Interactions: Unlike Naive Bayes's feature independence assumption, the tree-based structure of XGBoost naturally models non-linear relationships and interactions between features. The model can learn that the effect of one feature depends on the value of another, which was likely a source of Naive Bayes's misclassifications

- Efficiency and Scalability: XGBoost is designed for computational speed and efficiency. It incorporates techniques like parallel processing for tree construction, cache-aware access, and out-of-core computation, making it suitable for larger datasets and faster training cycles compared to some other gradient boosting implementations.

By leveraging these characteristics, XGBoost was expected to provide a significant uplift in performance, better capture the nuances of the tagging data, and

overcome the systematic errors observed with the Naive Bayes model. The setup of the XGBoost model and appropriate data engineering techniques involved the following steps:

- Manual Labelling: An initial set of 118 manually labeled tag keys extracted from our dataset was established. Covering both application-related and unrelated examples. Given the limites size and class imbalance, the dataset alone was insufficient without augmentation.
- Data Normalization: Before training all tag keys were normalized by transforming the characters in lowercase, removing whitespace and applying unicode normalization.
- Data Augmentation: We developed functions that could augment the data by creating variants of each tag keys. Such functions would swap characters, randomly insert or delete characters or reorder the tag key by chunks given a delimiter. This would mimic typos and common naming habits.
- Feature Engineering: each tag key (original and augmented) was vectorized using a rich feature set encompassing lexical features, semantic proximity with words such as "application" or "app", syntactic features and diversity metrics (entropy and unique character ratio).
- Class Imbalance: Naturally the dataset exhibited a strong class imbalance, with a majority of tag keys unrelated to applications. We addressed this using oversampling of the minority (positive) class within the training split to improve learning sensitivity.
- Hyperparameter Optimization: To optimize the XGBoost classification task beyond the Naïve Bayes baseline and various augmentation we set up to enhance our models' robustness, a hyperparameter search was conducted using k-fold cross validation GridSearchCV. This process allows to iteratively train, validate and test a set of hyperparameters. The best set of hyperparameters is then saved and the final model uses them.

4.3.3 Results and Evaluation

To train and later test our model we followed the standard procedure using 70% of our labeled dataset for the training process and the 30% remaining for the testing process, in order to evaluate our models' performance on unseen data. The training data was shuffled to ensure no bias was made with our labeling order. The optimized XGBoost model, trained on our augmented and feature-engineered dataset, achieved a significantly higher accuracy of 0.9576.

Compared to the Naive Bayes baseline, this model demonstrated:

- Superior precision and recall.
- Reduced false positives/negatives.
- Better alignment with expert intuition.

This confirmed that XGBoost could capture the nuanced patterns necessary for reliable tag classification in noisy environments. Beyond accuracy, a detailed analysis of classification metrics reveals a model that is both precise and balanced. As shown in Table 4.1 below, the F1-scores are identical for both classes (True and False) at 0.96, indicating that the model maintains an excellent trade-off between precision and recall regardless of whether the tag key refers to an application or not.

	Precision	**Recall**	**F1-Score**	**Support**
False	0.94	0.98	0.96	60
True	0.98	0.93	0.96	58
Accuracy	-	-	0.96	118
Macro Avg	0.96	0.96	0.96	118

Table 4.1: Tag Keys Classification Report using XGBoost

This is critical in our context, as both false positives (irrelevant tags wrongly classified as application-related) and false negatives (missing actual application tags) could significantly distort downstream analyses. The trained model was then used to classify every unique tag key in our dataset. Only keys predicted as application-related were retained, and corresponding tag values were interpreted as application names (referred to as application-related dataset). This step served as a critical filtering mechanism, ensuring that downstream analyses focused solely on relevant application-level data. This process could be criticized but we only had this dataset to base ourselves on. In-fine, the model was trained, tested and deployed on the same dataset. Results before and after the classification can be observed in Table 4.2. When using the Naïve Bayes classifier, tag keys such as app_owner and app_leader were classified as True, because of the "app" sequence. Thanks to the particularities of the XGBoost model, no such misclassification happened

Tag key	**Classification**
environment	False
application	True
app_owner	False
applicion	True
app_leader	False

Table 4.2: Sample of classified tag keys. Small variations are understood by the model and only application-related tag_keys are kept.

By resolving the ambiguity and inconsistency inherent in user-defined tags through supervised machine learning, this work successfully mitigates a key technological lock-in: the lack of standardization in tagging practices across Azure tenants. The approach is scalable, adaptive to new data, and demonstrates strong generalization capabilities in the face of real-world noise such as typos, naming variability, and inconsistent formatting. Ultimately, it enables more reliable and efficient exploitation of billing metadata for intelligent cloud infrastructure management.

4.4 Normalizing Tag Values

Following the successful classification of tag keys, the next logical step was to focus on the normalization of tag values. These tag values, representing application names associated with the now-identified relevant tag keys, originated from a dataset derived from 70 companies. Specifically, we utilized the subset of this data where tag keys were classified as application-related by our previously developed model. The challenge at this stage was to address the inconsistency in naming conventions, as different string representations of the same application were often used. For instance, a single application-related tag key could be linked to multiple, slightly varied tag values (e.g., "citricx_server," "citrix," "citrix_prd", all correspond to the “citrix” application). The objective was to consolidate these variations into standardized names for more accurate downstream analysis.

4.4.1 Initial Approach and Challenges

Initially, to group these semantically similar but textually distinct tag values, we considered employing a combination of advanced dimensionality reduction and clustering techniques. Uniform Manifold Approximation and Projection (UMAP) was evaluated as a candidate for dimensionality reduction. UMAP is a non-linear technique adept at learning the manifold structure of data and creating low-dimensional embeddings that preserve both local and global relationships. The intention was to convert high-dimensional vector representations (e.g., TF-IDF or pre-trained sentence embeddings) of the tag values into a dense, lower-dimensional space (typically 2D or 3D) where similar values would ideally be positioned closely.

Subsequently, Density-Based Spatial Clustering of Applications with Noise (DBSCAN) was considered for the clustering phase. DBSCAN is advantageous as it can identify clusters of arbitrary shapes and does not require the number of clusters to be specified beforehand; it groups together points that are closely packed (points with many nearby neighbors within an epsilon radius) and marks as outliers points that lie alone in low-density regions. However, this approach proved computationally prohibitive due to the sheer volume of unique tag value instances present in

the dataset. UMAP's initial step involves constructing a k-nearest neighbor graph, which, even with optimizations, scales at least super-linearly (often cited around $O(N \log(N))$ to $O(N^{1.14})$ with approximate methods) with N the number of data points.

DBSCAN in its naive form, has a time complexity of $O(N^2)$ due to pairwise distance calculations, though this can be reduces to $O(N \log(N))$ on average with spatial indexing. Given the potentially hunderds of individual tag value instances to process, the computational cost on the machines we had at work for both UMAPs' graph construction and DBSCANs' density estimation became a significant bottleneck, rendering this combined strategy impractical for our need. In fact, some execution times for a simple clustering of poor quality took up to 30 minutes. In addition to this, the use of predefined vocabularies for UMAP to embed our tag values was unnecessary as we work with a very niche type of data which didn't justify the use of external data to compute extensive embeddings.

4.4.2 Simplified Approach with Unsupervised Clustering

As a result, we pivoted to a more practical and straightforward solution: unsupervised hierarchical clustering based on string similarity, specifically using the Levenshtein distance (edit distance). The Levenshtein distance quantifies the similarity between two strings, say str1 of length m and str2 of length n, by calculating the minimum number of single-character edits (insertions, deletions, or substitutions) required to change str1 into str2.

The choice of Levenshtein distance over other text-based metrics was deliberate, driven by its suitability for the specific types of variations observed in application names – typically minor misspellings, presence/absence of separators (hyphens, underscores), case differences (though often handled by pre-processing), or slight variations in abbreviations. Also it is the most widely used when it comes to clustering values that have a common baseline (e.g. "qlick-cloud", "qlick-server" belong to "qlick" cluster).

This direct string metric was chosen over the embedding-based UMAP and DBSCAN approach primarily because our analysis indicated it was significantly more computationally efficient for the task at hand. While calculating a pairwise Levenshtein distance matrix for N strings has a complexity related to $O(N^2 \times L^2)$ (where L is the average string length), this proved more tractable than the complex graph construction of UMAP and the neighborhood searches of DBSCAN when applied to the vast number of tag value instances.

More importantly, this unsupervised learning approach directly addressed our specific need: identifying clusters of textually similar strings without relying on predefined labels or complex feature transformations into an embedding space. The Levenshtein distance provides an intuitive and direct measure of orthographic variation appropriate for capturing minor naming inconsistencies, and hierarchical clustering allows for an interpretable grouping based purely on this textual similarity. This made it not only less compute-intensive but also highly appropriate for the nature of our data and the goal of standardizing application names based on their surface-form variations.

4.4.3 Data Processing and Clustering Process

Before proceeding with clustering, we applied a preprocessing pipeline to the tag values to standardize the data. This involved converting all text to lowercase, removing accents, eliminating non-alphanumeric characters, and trimming excess whitespace, resulting in the creation of preprocessed tag values. Additionally, values that contained special characters often associated with non-application names (such as email addresses or URLs) were filtered out. Strings exceeding 20 characters post-preprocessing were also discarded to eliminate potential noise from overly lengthy entries.

Hierarchical Clustering

The clustering process started by computing the Levenshtein distance between every unique pair of preprocessed tag values. This exhaustive pairwise comparison yielded a symmetric N×N distance matrix, where N is the number of unique tag values. To optimize memory usage and streamline input for the subsequent clustering algorithm, this full matrix was transformed into a condensed 1D array using scipy.spatial.distance.squareform.

This condensed form efficiently stores only the $\frac{N(N-1)}{2}$ unique distances from the upper (or lower) triangle of the symmetric matrix, which is the standard input format for SciPy's linkage function. This condensed distance vector was then fed into the hierarchical clustering algorithm, specifically using the 'average' linkage method (UPGMA - Unweighted Pair Group Method with Arithmetic Mean). This method iteratively merges the closest pair of clusters, where the distance between two clusters is defined as the average of all pairwise distances between elements in one cluster and elements in the other. This agglomerative process builds a dendrogram, a tree-like diagram that visually represents the hierarchical relationships and the distances at which successive fusions occurred. Finally, this dendrogram was 'cut' at a predefined distance threshold to derive a flat set of distinct tag value clusters.

Over our application-related dataset, to determine the appropriate clusters, a distance threshold was applied to the dendrogram. In this case, we specified a normalized absolute edit distance of 0.7 as the cutoff for cluster formation. After multiple tries, this threshold yielded the best results, it ensured that only tag values that were sufficiently similar to each other were grouped together. In fact a threshold set to 2 or 3 output clusters that were too granular where they could have been grouped in a single application. For example, cluster 1 $C_1 = \{$solulabo-champ$\}$ and cluster 2 $C_2 = \{$solulabo2mc$\}$, both contain information about one single application "solulabo", and should be grouped together. On the other hand, high values like 8 or 9 output clusters that were too general. For example cluster 1 $C_1 = \{$citrix, cegid$\}$. The presence of letter "c" and "i" combined with a high threshold can lead to incorrect clustering.

Once the clusters were formed, a representative name for each cluster was chosen by selecting the cleaned tag values member that exhibited the lowest average normalized Levenshtein distance to all other members within that cluster. This value served as the cluster app, a standardized name representing each application for each row in our dataset.

Final Mapping

Tag value	Cluster app
OWP	owp
citrix	citrix
dotcom	dotcom
palo-alto	paloalto
paloalto	paloalto
DOTCOM	dotcom
citrixserver	citrix
Q-lick	qlick
Dtcom	dotcom
qlick	qlick

Table 4.3: Example of a few tag value / cluster app associations

In our dataset with only application-related tag keys, each original tag value was mapped to its corresponding cluster app name, effectively normalizing the data. This process consolidated variations in application naming and allowed for grouping and analysis based on consistent, standardized labels. The outcome was a set of normalized application names that reduced the complexity and inconsistencies inherent in the raw data, paving the way for more reliable downstream analysis. Some results of that mapping can be seen in Table 4.3.

4.4.4 Results and Evaluation

Evaluating unsupervised clustering is inherently more challenging than supervised classification because there are typically no "ground truth" labels to compare against directly. However, we can use several intrinsic metrics, along with qualitative analysis. One of those metrics is called the Silhouette Score. This score measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The score ranges from -1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If the score is close to 1, the clustering is dense and well separated, if the score is close to 0, then some clusters overlap and could be better separated (not considered bad though). However, when close to -1, it indicates that some samples have been assigned to the wrong cluster.

Normalized Threshold Value	Silhouette Score
0.1	0.086
0.2	0.083
0.3	0.217
0.4	0.226
0.5	0.249
0.6	0.295
0.667	**0.304**
0.7	0.298
0.8	0.248
0.9	0.157

Table 4.4: Test of the normalized threshold value. The best performing threshold is 0.667 outputting a score of 0.304.

When calculating this metric on our application-related dataset we obtain a score of 0.298, which is obtained with a threshold of 0.7 (cf. Table 4.4). We can see that small threshold variations can lead to important silhouette scores.

A positive score means our clustering algorithm is finding some level of grouping that's better than random. Points within the same cluster are, on average, more similar to each other (based on Levenshtein distance) than they are to points in other clusters. And, the low positive value suggests that many data points might lie close to the decision boundary between neighboring clusters. This means some tag values could plausibly belong to more than one cluster. However, for text data, especially when using edit distances like Levenshtein, achieving very high Silhouette Scores can be challenging. String variations can be gradual, and clear-cut boundaries are less common than in, say, numerical data with distinct distribution. Given this score and a manual cross check across more than 400 clusters, we can confirm that this

is a good enough score and the process seems to yield interesting results for a first version.

4.5 Normalizing Environments

As mentioned in the introduction of this chapter, optimizing cloud resource utilization is very important for cost efficiency and performance. Azure Reservations offer significant savings for long-term Virtual Machine (VM) commitments, but their true value relies on the predictable usage of the underlying workloads. A critical factor influencing this predictability is the stability of the application on which these VMs are running. However, it is even more interesting to know what kind of environment the virtual machine is running on. To corroborate this environment as well as the application, to later compute their joint stability represents a very interesting insight for recommendation. Assessing environment/application stability provides crucial insights, enabling more granular and reliable reservation recommendations; recommending a reservation for a VM running on an unstable environment like “testing” or “sandbox" may not yield the expected benefits. Furthermore, understanding stability serves as a vital implicit feature for accurately modeling VM usage fluctuations over time.

However, deriving consistent environment information directly from Azure VM tags is quite tricky due to inconsistent naming conventions, ambiguous tag_keys, and variations in tag_values or resource ids. This section addresses the question: How can we normalize Azure environments? The primary objective of this methodology is to accurately and consistently identify the operational environment (e.g., Production, Development, Staging) for each Azure resource represented in our dataset. This is essential for subsequent analyses, such as cost allocation, stability assessment (in our case), and resource optimization. A significant challenge arises from the inherent inconsistency in how environment information is recorded within Azure; it may be present in resource naming conventions, resource group names, specific tags, or dedicated metadata fields, often with non-standardized nomenclature and varying degrees of reliability. Therefore, a robust methodology is required to systematically analyze multiple potential sources, resolve ambiguities, and derive a single, standardized environment identifier for each resource instance.

4.5.1 Extracting Relevant Environment Information

In the Azure environment, execution environments are something quite tricky to properly identify. Similarly for the application information introduced in the previous sections, there are no standardized naming conventions that exist, because environments can be set automatically or manually. Both of these versions often

don't match, which makes the analysis even harder. Our initial thought to extract relevant information from the environment-related columns was to make a classification model using supervised learning. This would be a multi-class classification, and the model would be trained on manually labelled data.

Data Sources

Our model would be based on the following source columns within our dataset:

- *subscription_name*: The name of the Azure Subscription which includes organizational, environment and, occasionally, application naming conventions.
- *resource_group*: The name of the Azure Resource Group, which often follows organizational naming conventions that include environment indicators.
- *resource_id*: The unique Azure Resource ID. Specifically, the terminal segment corresponding to the resource name (e.g., Virtual Machine name) is extracted, as it frequently incorporates environment codes
- *tag_key_unfiltered*: The key associated with an Azure tag.
- *tag_value*: The value associated with an Azure tag. This is considered a primary source when the corresponding tag_key_unfiltered explicitly indicates an environment context (e.g., key is 'environment').
- *environment*: A dedicated prebuilt environment column. If present in the source data, intended to store the environment explicitly. However, it presents inconsistencies with the resource group information.

As it turns out, this approach was going to be extremely difficult. The structure of the information contained in these columns is extremely complex, and the labeling task would have taken a lot of time and expertise. The reason for this was that we couldn't anticipate the number of classes that were to be expected, for the simple reason that there is no convention. Theoretically, there could be as many classes as the person inputting the environment would want to. Then, it would mean an extensive processing and labeling step in order to have training-ready data. Thus, keeping

the same data, we decided to fall back on a pseudo-exhaustive regular expression approach. Doing this would lose the "intelligence" of a machine learning approach, but would simplify the task greatly, saving lots of time in labeling, processing and tuning the model while remaining data-driven.

Environment Lexicon and Standardization

To handle variations in nomenclature, a controlled vocabulary and mapping strategy were established:

- Canonical Environment Identifiers: A definitive list of standardized, lowercase environment identifiers used as the final output values (e.g. ‘prd’, ‘tst’, ‘sta’, ‘dev’).
- Canonical Environment Map: A dictionary mapping various observed lowercase variations and synonyms (e.g., ’prod’, ’production’, ’stg’, ’staging’, ’development’, ‘test’) to their corresponding canonical target. This ensures that different input strings representing the same concept are normalized.
- Environment Pattern Recognition: A dictionary where each key is a canonical target identifier, and the value is a pre-compiled regular expression pattern. The patterns are applied case-insensitively, as we normalize all values by converting them to lowercase.
- Valid Canonical Environments: A set derived directly from the canonical identifiers, used for final validation of identified environments.

All these elements are used to build an exhaustive filter, where by processing combined information from our columns we are able to extract the environment information in an elegant way.

Candidate Extraction and Cleaning

Following the development of the rule-based regular expression system for identifying environment indicators (referred to as "pattern-catcher"), a structured methodology was required for its application to the dataset. A primary consideration was the potential for multiple source columns within a single record to yield environment-related matches (e.g. column resource group contains ‘prd’ and column tag key contains ‘dev’ on the same row) This necessitated a strategy to resolve instances where conflicting environment indicators were detected across different attributes, a scenario anticipated due to the previously identified inconsistencies in environment naming conventions.

Consultation with domain experts led to the establishment of a hierarchical prioritization scheme for analyzing the source columns. This hierarchy dictates the order in which columns are interrogated for environment information and the precedence given to a match from a higher-priority source. The rationale for this prioritization is based on the perceived reliability and explicitness of the environment information contained within each source. The process for identifying potential environment

indicators (candidates) from the designated source columns within each data record involves several distinct stages, designed to ensure systematic and contextually relevant extraction.

The extraction of environment identifiers follows a multi-step, standardized process to ensure accurate and consistent labeling. First, all textual inputs undergo normalization by converting them to lowercase and trimming whitespace. Attributes explicitly designated to represent environments—such as user-defined tags—are analyzed only if their keys match a predefined list (e.g., 'env', 'environment') after cleaning. For structured identifiers (e.g., resource names), regular expressions isolate semantically meaningful segments, which are then normalized. A curated set of regular expression patterns is applied to these normalized strings to detect canonical environment labels (e.g., 'production', 'development'). Finally, all matched identifiers are consolidated into a structured set representing the environment context of each data record, enabling further analysis and conflict resolution.

To resolve conflicting environment identifiers extracted from multiple sources, a strict prioritization and selection logic is applied. Sources are ranked by reliability: user-defined environment tags are considered most explicit, followed by resource group names, subscription names, VM names, and finally a default environment column. For each resource, the first valid canonical environment found in this priority order is selected. If multiple identifiers are found within a top-priority source, the first match or a deterministic fallback (e.g., alphabetical order) is used.

4.5.2 Results of the Environment Extraction

After executing our environment extraction procedure on our dataset, we can observe that our method outputs the actual environment that was imputed in the columns such as resource_group or subscription_name as seen in Table 1. We observe that most of the time, because the tag key column has empty values (or non environment-related values), the priority of the environment comes back to what was found in the column resource group or subscription name.

The key takeaway here is that our method is able to get the correct environment out of the analyzed columns, respecting a priority order to have a much better and granular understanding of the environments used on various resources. To this day, this method has proved itself better than the initial one that created the environment column. And we can see from Table 4.5, that it is consistent. Evaluation of the environment extraction method on a test dataset comprising 100 samples with defined ground truth labels yielded an overall accuracy of 95%. This indicates a high level of agreement between the extracted and true environments, with a total of 5 misclassified instances observed within the analyzed subset. Performance analysis,

sub_name	res_group	res_id	tag_key	tag_value	env	real_env
...-DEV	...-DEV	...-dv	non_prd	dev	...-DEV	dev
...-DEV	...-DEV	...	non_prd	dev	...-DEV	dev
...-PRD	...-PROD	...	...	...	non_prd	prd
...-STG	...-STG	...-st	...	...	non_prd	sta
...-NONPROD	...-QA	...-qa	non_prd	qa	non_prd	qa

Table 4.5: Sample of the dataframe containing the extracted environment. In red we display the correct environment when it is found in the analyzed columns. Elements that are not environment-related have been set as "...". The columns called "res" are short for "resource", the one called "sub" is short for "subscription".

supported by the classification report, demonstrates robust classification capability for the most frequently occurring environments. While performance on less common environments varies, some achieved perfect classification scores, whereas others exhibited instances of reduced recall or precision. Specific analysis of misclassifications highlights patterns including the incorrect assignment of samples to labels not present in their true distribution, and the misclassification of some less frequent true environments as other categories.

Not only are we able to keep a computationally viable option, we don't need to leverage clustering or supervised learning to achieve such results. This method using multiple columns for the analysis allows us to keep a larger amount of data, in fact, if we only wanted to focus on user input tag keys that referred to environments, and then focus on extracting the environment, we would discard a substantial amount of data from our analysis. Additionally, the initial environment column covered around 93% of the data, whereas our method covers 97% of the data with increased precision and reliability.

4.6 Estimating Application-Environment Stability

To quantify the stability of applications within Azure environments, the study introduces a metric α, ranging from 0 (unstable) to 1 (highly stable), computed for each application-environment pair based on historical usage data.

4.6.1 Initial Approaches

Several preliminary methods were considered:

- Standard Deviation (σ): Simple variability measure, but lacked normalization relative to usage level leading to over-sensitivity to outliser and underrepresentation of meaningful shifts.

- Min-Max Normalized Standard Deviation: This normalized usage to the $[0, 1]$ range before applying standard deviation, yielding a more relative measure of volativity. However, it failed to capture absolute shifts in usage levels and was insensitive to longer-term trends.
- Hybrid CV + Trend Model: Combined the Coefficient of Variation (CV) with Relative Trend Strength (T) in an exponential decay function to compute α. While promising, it was overly sensitive to hyperparameter tuning and lacked responsiveness to recent behavioral changes - often missing stabilization patterns in more recent data.

Due to these limitations - particularly the inability to differentiate between recent and historical stability and the high sensitivity to configuration- we sought a more adaptive model.

4.6.2 Refined Approach: Dual Horizon Model

The final model introduces a temporal dual-horizon strategy to capture both short-term and long-term stability dynamics:

- Two Horizons:
  - Short-Term (e.g. 15-60 days)
  - long-Term (e.g. 180-360 days)

  For each horizon the following procedure was applied:

- Trend Filtering: Fit a linear regression to the vCPU daily usage intensity time series of the application cluster $I_{cluster}$. If the slope β is below a threshold (e.g. β_{thresh}, -0.01 to -0.05), the usage is considered to be in decline, and the horizon stability score $\alpha_{horizon}$ is set to 0.
- Volatility Scoring: If $\beta > \beta_{thresh}$, compute σ, the average of the standard deviations over sliding windows (e.g. 7-14 days). The per-horizon stability score is then calculated via an exponential decay function parametrized by a deccay constant γ (≥ 1) penalizing erratic usage.

Finally, the alpha score is computed by merging the two horizon scores in a weighted average:

$$\alpha_{final} = w.\alpha_{short} + (1 - w).\alpha_{long}$$

where $w \in [0, 1]$ (e.g. $w = 0.667$) prioritizes recent activity while still incorporating historical consistency.

This refined model successfully balances recent trends with long term consistency, while remaining tunable yet robust to noise, making it a superior solution for application-environment stability.

4.6.3 Results and Evaluation

Following the estimation of application-environment stability scores (α), the results are compiled into a structured Alpha Catalog, a precomputed lookup table that associates each (application, environment) pair with a corresponding α score. This catalog forms a core component of downstream analytical pipelines, particularly those related to Virtual Machine (VM) Reservation recommendations in the Azure cloud.

Purpose and Integration

The Alpha Catalog is designed to enable efficient reuse of stability insights across multiple analytical contexts without recomputing the α metric in real-time. It is built to enrich cost optimization pipelines with contextual, behavior-driven features that go beyond raw utilization metrics. The Alpha Catalog is a DataFrame that has 3 columns "application_cluster", "environment", "alpha_score". It is built only once after the applications have been clustered, and the environments normalized.

Figure 4.1 illustrates the process of creating the catalog. The creation of the Alpha Catalog, which quantifies the stability of an application within a given environment, follows a multi-stage pipeline originating from the client billing data. The process bifurcates into streams focused on identifying application identity and extracting environment information, respectively.

The application identity stream begins with tag key classification, employing a binary classification model (specifically, an XGBoost classifier) trained to distinguish tag keys related to applications from others. This filtering step yields a set of application-related tag keys. These keys are then used to isolate the corresponding application values (names) from the billing data. Subsequently, these raw application names undergo clustering, utilizing a syntactic approach (as detailed in Section X, referring to your clustering methodology) to group variations of the same application name into standardized clusters.

In parallel, the environment extraction stream applies a rule-based approach, likely using regular expressions, to identify and extract environment information from various source columns within the billing data (as detailed in Section Y, referring to your environment extraction methodology).

The pipeline converges by combining the outputs from the application name clustering (the standardized application clusters) and the environment extraction (the identified environments). A final process, takes these paired (application cluster, environment) combinations. For each unique pair, this process calculates a stability metric (alpha), which quantifies the historical consistency and predictability of the workload associated with that specific application cluster running in that particular environment. The resulting Alpha Catalog is a repository mapping these (application, environment) pairs to their calculated alpha stability values.

The catalog is used in the following way:

- Billing Data Annotation: During the preprocessing, each record in the billing dataset is associated with its corresponding standardized environment and application.
- Lookup and Retrieval: The precomputed α score for the given (application, environment) pair is retrieved from the catalog.

The Alpha Catalog enhances operational reliability by enriching forecasting models with historical consistency indicators, enabling more precise VM reservation recommendations that avoid unstable workloads and reduce financial risk. Beyond raw usage, the α score also provides behavioral context about workload maturity. To ensure performance and scalability, computationally intensive processes—such as tag classification and clustering—are decoupled from real-time analysis and executed periodically or upon specific triggers, maintaining up-to-date stability insights without impacting the whole systems' latency.

Application-Environment Stability Evaluation

A visual evaluation and cross-check with expert knowledge was the only way to evaluate our approach. For instance, a classic case was analyzed by observing the aggregated daily usage of a sample application across two distinct environments: staging ('sta') and production ('prd'). The staging environment exhibits substantial temporal variability, characterized by significant fluctuations and a cyclical pattern in daily VCPU intensity, reflecting high volatility. This profile is associated with a calculated stability score of 0.000. In contrast, the production environment displays a stable usage pattern, reaching a consistent level after an initial unstable period, resulting in a significantly higher stability score of 0.806.

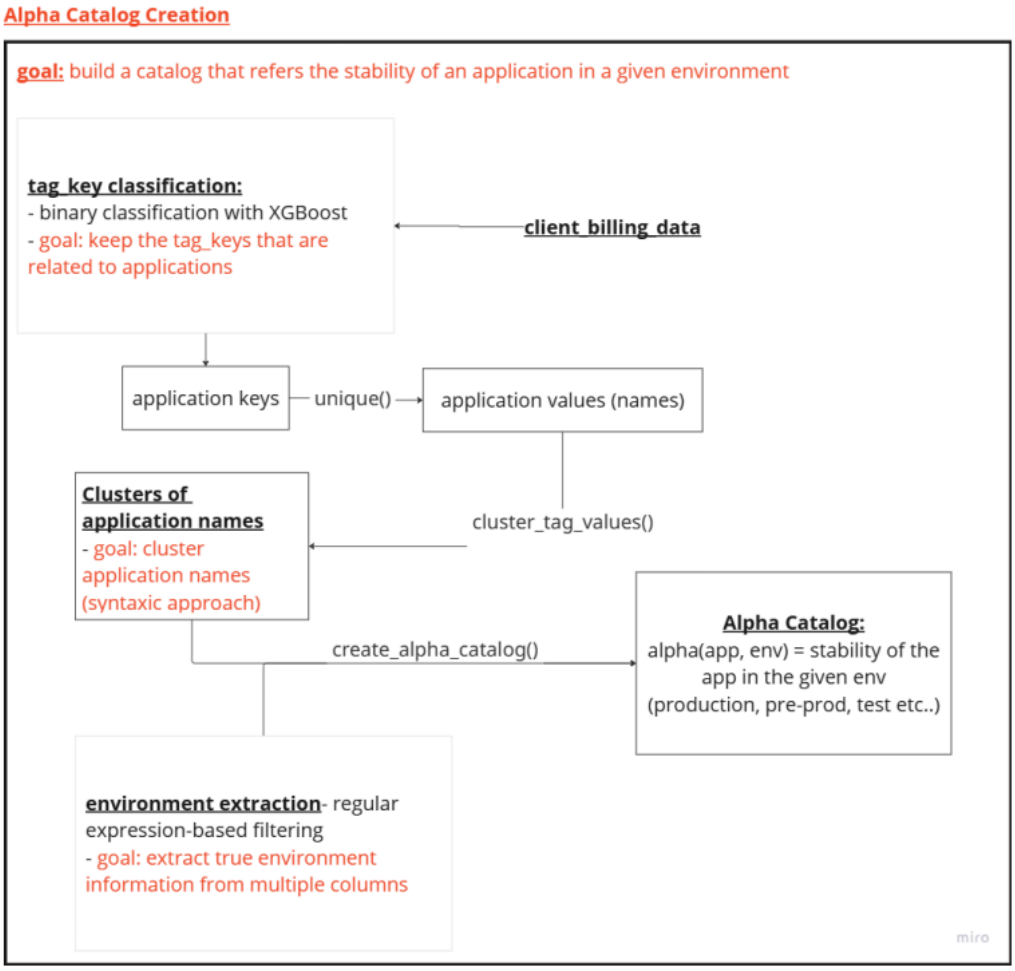

Figure 4.1: Diagram of the pipeline, starting with the client billing data, ending with the alpha catalog. It leverages the tag key classification, tag value clustering, and environment extraction.

This clear divergence in calculated stability scores validates the proposed hybrid quantitative approach. The method effectively differentiates these profiles by integrating trend (β) and volatility (σ) metrics over defined temporal horizons. Specifically, the high variability observed in the staging environment corresponds to a large volatility measure (σ), which, via the exponential decay function, heavily penalizes the per-horizon stability scores (α), leading to a low final weighted score. Conversely, the stable usage in the production environment indicates low volatility (α) and a non-declining trend ($\beta \geq \beta_{thresh}$), yielding high per-horizon scores that are preserved in the final weighted average, accurately reflecting its predictable and stable operational state. This demonstrates the method's robustness in capturing both consistent decline (via β) and erratic behavior (via σ), which are critical factors in assessing application stability in diverse environments.

A notable limit case was observed with the aggregated daily VCPU intensity for another application within a production environment. Over the depicted period, this second application exhibits consistent near-zero usage. Applying the described hybrid stability scoring methodology to this profile yields a stability score of 0.999. This high score is an accurate reflection of the data's characteristics: the usage trend (β) is essentially flat (near zero), which is likely above the specified negative threshold (β_{thresh}), and crucially, the daily usage fluctuates minimally around zero, resulting in extremely low volatility (σ close to 0). According to the formula $exp(-\gamma * \sigma)$, a near-zero σ leads to a stability score close to $exp(0) = 1$, thus correctly identifying this profile as highly stable in terms of temporal predictability.

However, this case represents a limit where the level of usage is negligible. While the stability metric correctly quantifies the consistency of the pattern (which is effectively no pattern of significant usage), it does not assess the utility or relevance of that usage level for decisions like virtual machine reservation. Therefore, while the high stability score is technically correct based on the method's definition of temporal predictability, the practical implication for resource planning is that this application's consistent low usage means it requires minimal or no dedicated resource reservation, despite its high stability score.

Chapter 5

Research Perspective

This chapter aims at proposing research directions for future work and improvement of the current state of what we have achieved. We will go through the improvements for the VMReservation system, and then for the Alpha Stability framework.

5.1 VMReservation

Given the current state of the system, research should be oriented towards optimizing the structure (model and logic) of the system in itself.

5.1.1 Advanced Forecasting Model Development

A major issue with our approach is the incapacity XGBoost has to extrapolate the data. In fact, in a forecasting setup, a key limitation of XGBoost is its inability to extrapolate beyond the range of values observed during training. As a tree-based model, XGBoost partitions the feature space and assigns constant predictions within each region, which makes it effective for interpolation but inherently constrained when faced with unseen or out-of-range target values. This poses a problem in forecasting contexts such as ours, where the objective is to anticipate trends that extend beyond historical patterns. For example, workloads that exhibit sustained growth may require the model to project increasing usage; however, XGBoost will instead plateau its predictions within the historical bounds, potentially underestimating future demand. This limitation reduces its ability to capture trend dynamics or seasonality, and may lead to inaccurate forecasts that affect downstream decisions such

as resource provisioning or reservation planning. Consequently, while XGBoost is powerful for pattern recognition, its lack of extrapolation capability necessitates the inclusion of trend-aware features or hybrid modeling strategies to address long-term forecasting objectives effectively.

The most significant improvement for our ML-based approach would be the adoption of the Temporal Fusion Transformer (TFT) architecture [8] (Figure 5.1). This state-of-the-art model is well suited to our forecasting challenge, as it is designed to produce high-accuracy, multi-horizon forecasts while simultaneously handling the complex mix of static, known future, and historical time-series data inherent to cloud consumption.

Practically, the model is structured to process distinct types of input data relevant to time series forecasting:

- Static Metadata (s) : information that remains constant for a given time series (in our case a Broad Combination). These attributes are fed into a Static Covariate Encoder to create embeddings that influence the entire forecasting process.
- Past Inputs : the sequence of our engineered time series features (lags, rolling stats, trend features etc.) and the historical I values. These are processed by a LSTM Encoder to capture sequential dependencies up to the present time.
- Known Future Inputs: features known for the future horizon (future date parts, time index, cyclical features, planned events if available). These are processed by an LSTM Decoder.

The core Temporal Fusion Decoder then combines outputs from the encoders and static enrichment layers. A key component is the Temporal Self-Attention mechanism, which allows the model to dynamically weigh the importance of different historical time steps when making predictions for future steps. This enables the model to identify relevant historical patterns for forecasting.

Crucially, the model's output consists of Quantile Forecasts for each future time step. That is, instead of a single point forecast, TFT predicts a range of possible values. This direct prediction of a distribution is highly valuable for our RI recommendation problem, as it inherently quantifies forecast uncertainty and can be directly used to derive statistically robust lower bounds or other risk-aware metrics for commitment decisions. Naturally, the TFT provides a powerful alternative architecture capable of leveraging our rich feature set and static combination information to produce probabilistic time series forecasts of Daily vCPU Usage Intensity.

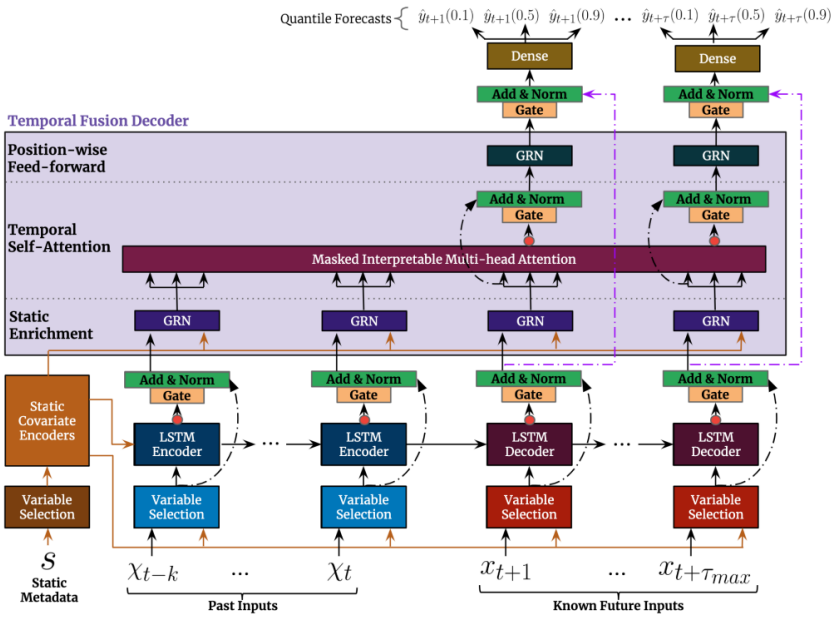

Figure 5.1: Illustration of the TFT model introduced by Lim et al. (2020) [8]

5.1.2 Model Deployment and Maintenance

To extend this research towards a robust MLOps framework, the next steps would focus on automating the forecasting pipeline. This involves implementing automated model retraining triggers, perhaps based on a fixed schedule (e.g., at the very least weekly because of data fluctuations) or performance degradation alerts (e.g., if MAPE on recent sets exceeds a threshold). Continuous model validation would be key, where newly trained models are evaluated against a holdout dataset and potentially shadow-deployed alongside the production model before promotion. For model maintenance, a version control system for models and data (like DVC or MLflow) would track experiments and artifacts, while comprehensive monitoring of input data drift, feature drift, and prediction accuracy would provide early warnings for necessary recalibration or more fundamental model redesign, ensuring the RI recommendations remain accurate and cost-effective over time.

5.1.3 Confidence of Recommendations

In terms of information outputted to the client, it would be good to develop a certainty metric, showing how confident we are in our recommendation. This can take into account the historical stability, the forecast stability and the amount of "risk" the model has taken. As mentioned in section 3.4.1, the stability estimator could be used to construct a confidence metric. I would be directly linked to the underlying workload of the virtual machine given an environment and an application justifying certain decisions.

5.2 Alpha Stability

Building on the established methodology for assessing application stability, several potential research avenues present themselves.

5.2.1 Enhancing Tag Understanding

One promising direction is to enhance the semantic understanding of tags beyond their syntactic similarity. Currently, the approach relies heavily on Levenshtein distance, which captures character-level similarity. However, incorporating Natural Language Processing (NLP) techniques, such as word or sentence embeddings by leveraging corpora of words like BERT, could improve the classification of tag keys and the clustering of tag values. This would allow the system to better capture synonymy and conceptual relationships between tags, improving both accuracy and meaningfulness.

5.2.2 Explainability and Causality

The area of model interpretability and causality is also critical for advancing the understanding of application stability. Techniques like SHAP (Shapley Additive Explanations) could be applied to interpret model predictions and explore the factors driving instability. By correlating calculated instability with external operational events or logs, we could not only identify that an application is unstable but also gain insights into why it is unstable, thus providing actionable guidance for improvements. This would represent a substantial amount of work to provide external data and verify correlations with outer events.

Chapter 6

Conclusion

This research successfully addressed the critical challenge of optimizing Azure Virtual Machine Reserved Instance (RI) commitments through a novel, data-driven approach. Recognizing the limitations of traditional forecasting in the complex cloud billing environment, this study introduced two key innovations: a methodology for quantifying application-environment stability (the "Alpha Stability" score) and a global machine learning model for forecasting VM usage, explicitly designed to leverage this stability information.

The development of a global XGBoost forecasting model was developed to predict future Daily vCPU Usage Intensity. This model transitioned from an initial, computationally prohibitive combination-specific strategy to a more scalable global approach. Methodological refinements were crucial, including consistent temporal feature engineering (globally referenced time_index), careful feature selection to exclude unstable predictors (e.g., high-order polynomial time features, overly sensitive difference metrics), the use of One-Hot Encoding for categorical identifiers (VM series, location, invoice section, environment), and the engineering of interaction features between temporal/lag components and these categorical identifiers. Data integrity measures, such as forward-filling for gaps and a focused training window on recent, representative data, coupled with time series data augmentation (jittering, time warping), enhanced model robustness and responsiveness to current usage patterns. Optimized hyperparameters, identified through Optuna, further improved predictive performance.

Concurrently, the Alpha Stability metric, derived from analyzing historical VM usage for distinct application-environment pairings, provided a robust, quantifiable measure of workload predictability. This involved sophisticated data preprocessing, including tag key classification using XGBoost and tag value normalization via hierarchical clustering with Levenshtein distance, to standardize application and environment identifiers from noisy, heterogeneous billing data. The refined hybrid approach for alpha calculation, integrating Coefficient of Variation and Relative Trend Strength over distinct temporal horizons, proved effective in capturing both short-term volatility and long-term directional shifts, offering a more nuanced stability assessment than simpler metrics. The resulting Alpha Catalog serves as a valuable lookup resource, associating each application-environment pair with an empirical stability score.

While initial tests showed that integrating the Alpha Stability score created a more complex recommendation model without outperforming a simple heuristic, the Alpha Catalog itself remains a valuable asset. It provides useful confidence indicators and holds potential for more nuanced models in the future, and is highly dependent on the variety of the data is deployed on. In contrast, our work on the global forecasting model was highly successful. By pairing the refined model with a robust data and feature filtering strategy, we achieved superior performance (MAE: 0.705, R^2: 0.983) with a more streamlined set of features, underscoring the benefits of a focused analytical input.

To expand the scope of optimization beyond virtual machines, the reservation of other assets, such as SQL databases, remains a future area of study. This would allow for targeting other costly services, while of course taking into account the specific characteristics inherent to each resource type. In parallel, an analysis of Savings Plans as a commitment method that is complementary or alternative to RIs is also forthcoming, in order to provide greater flexibility and cover types of spending not eligible for RIs, thereby completing the overall cloud cost optimization strategy.

This study successfully established a comprehensive, end-to-end pipeline for RI recommendation, from raw billing data processing and stability quantification to forecasting and actionable commitment derivation. The methodologies developed for application and environment normalization, stability assessment, and global forecasting provide a solid foundation for intelligent cloud cost optimization. Future research perspectives, including advanced forecasting models (e.g., probabilistic forecasting, TFTs), dynamic filtering, automated feature selection, and enhanced

MLOps practices for model maintenance and monitoring, promise further advancements in this critical domain of cloud resource management.

Appendix A: Pipeline Diagram

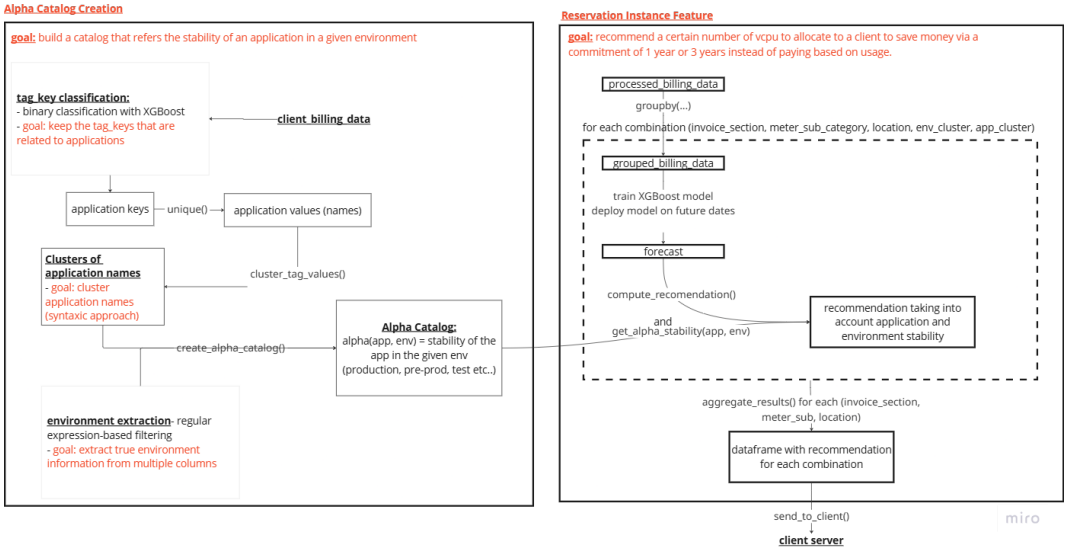

Figure 1: Diagram illustrating the entire pipeline of the work completed during this apprenticeship. On the left is the Alpha Stability framework, on the right is the VMReservation system. Both are connected during the inference of VMReservation where a stability metric will be used to enhance our predictions and reservation recommendations.

References

[1] S. Amirineni, "Leveraging machine learning, cloud computing, and artificial intelligence for fraud detection and prevention in insurance: A scalable approach to data-driven insights," *International Journal of Automation, Artificial Intelligence and Machine Learning*, 2024. [Cited on page 16]

[2] M. K. Senapaty, A. Ray, and N. Padhy, "A decision support system for crop recommendation using machine learning classification algorithms," *Agriculture*, 2024. [Cited on page 16]

[3] A. B. Rimba, A. Arumansawang, I. Putu, W. Utama, S. K. Chapagain, M. N. Bunga, G. Mohan, K. T. Setiawan, and T. Osawa, "Cloud-based machine learning for flood policy recommendations in makassar city, indonesia," *Water*, 2023. [Cited on page 16]

[4] R. Mahadik, A. S. Pawar, D. I. Navalgund, and S. Dingankar, "Machine learning-enabled medical image analysis for disease detection in the cloud," *2023 International Conference on Artificial Intelligence for Innovations in Healthcare Industries (ICAIIHI)*, vol. 1, pp. 1–6, 2023. [Cited on page 16]

[5] P. D. Chaudhari, "Air quality prediction using machine learning and cloud computing," *International Journal for Research in Applied Science and Engineering Technology*, 2023. [Cited on page 16]

[6] U. R. Saxena and T. Alam, "Systematic review on recommendation systems to select trustworthy cloud services and ensure data integrity," *2022 2nd International Conference on Technological Advancements in Computational Sciences (ICTACS)*, pp. 134–139, 2022. [Cited on page 17]

[7] S. Samadhiya and C. C.-Y. Ku, "Hybrid approach to improve recommendation of cloud services for personalized qos requirements," *Electronics*, 2024. [Cited on page 17]

[8] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," 2020. [Cited on pages 72 and 73]