INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA DE INTELIGÊNCIA ARTIFICIAL




Bridging Automation and Customization: MLOps in Recommender System Development

MIGUEL JOSÉ RIBEIRO JORDÃO
Setembro de 2025

P.PORTO

Bridging Automation and Customization: MLOps in Recommender System Development

Miguel José Ribeiro Jordão

Student No.: 1201487

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Artificial Intelligence Engineering

Supervisor: Isabel Cecília Correia da Silva Praça Gomes Pereira, Associate Professor, Institute of Engineering, Polytechnic of Porto

Evaluation Committee:

President: António Constantino Lopes Martins, Associate Professor, Institute of Engineering, Polytechnic of Porto

Members:

Joaquim Filipe Peixoto dos Santos, Associate Professor, Institute of Engineering, Polytechnic of Porto

Isabel Cecília Correia da Silva Praça Gomes Pereira, Associate Professor, Institute of Engineering, Polytechnic of Porto

Porto, September 30, 2025

Abstract

Recommender systems have become essential in modern digital platforms, supporting decision making and personalization across domains such as e-commerce, media, and enterprise applications. At BMW Group, MyWorkplace (MWP) is a centralized hub managed by Critical Techworks (CTW) that provides access to hundreds of internal tools. Discoverability remains challenging given the size and heterogeneity of the tools catalog. This creates inefficiencies, highlighting the need for a scalable, reliable, and auditable recommendation solution.

This project presents an MLOps-first approach for a recommender grounded in the CRISP-ML(Q) process model. It characterizes the recommendation problem, available data sources, and success criteria, and proposes a reference architecture integrating automated ETL, feature preparation, containerized training and serving, and CI/CD for continuous delivery. Several content-based approaches are implemented and evaluated under realistic data constraints using established ranking metrics; collaborative and hybrid extensions are outlined for future phases once interaction feedback becomes available.

The contributions of this work are both technical and methodological: the design and validation of a recommendation strategy for the hub platform; an assessment of operational and governance requirements, including security and compliance, and the demonstration of the system in a real-world industrial environment. In addition to the deployment within BMW Group, this project advances the understanding of how MLOps principles can be applied to balance automation and customization in recommender systems.

Results indicate that an MLOps-first design improves scalability, maintainability, and auditability, and lays the groundwork for collaborative filtering, feedback loops, and, when governance permits, large language model components. The system and methodology are applicable to enterprise-scale recommendation scenarios with similar operational constraints.

Keywords: Recommender Systems, MLOps, Data Engineering, Kubernetes, ETL Pipelines, Machine Learning

Resumo

Os sistemas de recomendação tornaram-se essenciais nas plataformas digitais modernas, apoiando a tomada de decisão e a personalização em domínios como o comércio eletrónico, os media e as aplicações empresariais. No BMW Group, o myWorkplace (MWP) é um hub centralizado, gerido pela Critical Techworks (CTW), que disponibiliza acesso a centenas de ferramentas internas. A capacidade de descoberta continua a ser desafiante, dada a dimensão e a heterogeneidade do catálogo de ferramentas. Isto gera ineficiências, evidenciando a necessidade de uma solução de recomendação escalável, fiável e auditável.

Este projeto apresenta uma abordagem "MLOps-first" para um sistema de recomendação, alicerçada no modelo de processo CRISP-ML(Q). Caracteriza o problema de recomendação, as fontes de dados disponíveis e os critérios de sucesso, e propõe uma arquitetura de referência que integra ETL automatizado, preparação de características (features), treino e serving conteinerizados, e CI/CD para entrega contínua. Implementam-se e avaliam-se várias abordagens baseadas em conteúdo, sob restrições de dados realistas, recorrendo a métricas de ranking estabelecidas; extensões colaborativas e híbridas são delineadas para fases futuras, assim que exista feedback de interação.

As contribuições deste trabalho são simultaneamente técnicas e metodológicas: a conceção e validação de uma estratégia de recomendação para a plataforma hub; uma avaliação dos requisitos operacionais e de governação, incluindo segurança e conformidade; e a demonstração do sistema num ambiente industrial real. Para além da implementação no BMW Group, este projeto aprofunda a compreensão de como os princípios de MLOps podem ser aplicados para equilibrar automação e customização em sistemas de recomendação.

Os resultados indicam que um design "MLOps-first" melhora a escalabilidade, a manutenibilidade e a auditabilidade, e estabelece as bases para filtragem colaborativa, ciclos de feedback e, quando a governação o permitir, componentes de modelos de linguagem de grande escala (LLM). O sistema e a metodologia são aplicáveis a cenários de recomendação à escala empresarial com restrições operacionais semelhantes.

Palavras-chave: Sistemas de Recomendação, MLOps, Engenharia de Dados, Kubernetes, Pipelines ETL, Aprendizagem Automática

Acknowledgement

I would like to begin by extending my profound gratitude to all those who have played a vital role in my academic journey and contributed to the successful completion of my project at Critical Techworks. This experience has been immensely enriching, and I owe it all to the unwavering support and guidance of those involved.

My heartfelt thanks go out to entire project team at Critical Techworks, myWorkplace team. Their assistance and feedback were invaluable and played a critical role in bringing this project to fruition. The professionalism and dedication displayed by all have been both contagious and inspiring.

Furthermore, I would like to express my sincere appreciation to my ISEP advisor, Isabel Praça, and to João Vitorino for their invaluable guidance and advice. Their feedback and suggestions were fundamental in fostering my personal and professional growth.

Finally, I would like to extend my heartfelt gratitude to all my friends, colleagues, and family for their unwavering support, encouragement, and emotional sustenance, especially Valérie Duarte for this last push. Their support has been an essential source of motivation throughout this journey, and I am deeply grateful for their presence in my life.

Contents

List of Figures

List of Tables

List of Listings

Acronyms and Symbols

ACR Azure Container Registry

AF Azure Function

AI Artificial Intelligence

AKS Azure Kubernetes Service

ANOVA Analysis of Variance

API Application Programming Interface

AP@K Average Precision at K

ArgoCD GitOps Continuous Delivery for Kubernetes

CB/CBF Content-Based (Filtering)

CF Collaborative Filtering

CI/CD Continuous Integration / Continuous Delivery

CRISP-DM Cross Industry Standard Process for Data Mining

CRISP-ML(Q) Cross-Industry Standard Process for ML (Quality)

CSI Container Storage Interface

CTR Click Through Rate

DCG Discounted Cumulative Gain

DevOps Development and Operations

DLRM Deep Learning Recommendation Model

EDA Exploratory Data Analysis

ETL Extract, Transform, Load

FCP Fraction of Concordant Pairs

FR Functional Requirement

GDPR General Data Protection Regulation

GraphQL Graph Query Language (application query protocol)

HDF5 Hierarchical Data Format version 5

HPA Horizontal Pod Autoscaler

IDF Inverse Document Frequency

JWT JSON Web Token

K8s Kubernetes

KB Knowledge-Based (Recommender)

KEDA Kubernetes Event-Driven Autoscaling

KPI Key Performance Indicator

LLM Large Language Model

Loki Log aggregation system (Grafana Loki)

MAE Mean Absolute Error

MAP@K Mean Average Precision at K

MCRS Multi-criteria recommender systems

MF Matrix Factorization

ML Machine Learning

MLOps Machine Learning Operations

mTLS Mutual Transport Layer Security

MWP MyWorkplace

NDCG@K Normalized Discounted Cumulative Gain at K

NFR Non-Functional Requirement

OAuth Open Authorization

ONNX Open Neural Network Exchange

PoC Proof of Concept

PRISMA Preferred Reporting Items for Systematic Reviews and Meta-Analyses

PSO Particle Swarm Optimization

PV Persistent Volume

PVC Persistent Volume Claim

RAG Retrieval-Augmented Generation

RBAC Role-Based Access Control

REST Representational State Transfer

RMSE Root Mean Squared Error

RS Recommender System

RTM Requirements traceability map

SLI Service Level Indicator

SLO Service Level Objective

SLR Systematic Literature Review

SQL Structured Query Language

TF–IDF Term Frequency–Inverse Document Frequency

User–Item Generic RS pair denoting a user and an item with an interaction or potential interaction

Chapter 1

Introduction

This chapter introduces the motivation and scope of a recommender system for the My-Workplace (MWP) application at Critical Techworks (CTW) and highlights the need for an MLOps-first approach in the implementation. First, it establishes the broader role of recommender systems, then narrows to the specific CTW context, discussing current limitations and their operational impact. The integration of MLOps following a CRISP-ML(Q) approach enables a reliable, scalable system. The chapter then sets the objectives that guide the project and the research questions it aims to answer. Finally, it explains the remainder of the document across all chapters.

1.1 Contextualization

Recommender systems (RS) are one of cornerstones of e-commerce, media, and social platforms, supporting the users decisions and personalizing their experiences. Classical approaches include content-based filtering (CBF), which takes in consideration item attributes to match user profiles, and collaborative filtering (CF), which takes advantage of user–item interaction patterns to infer preferences. At an industrial scale (e.g., YouTube, Netflix), production systems typically adopt multi-stage pipelines strategy with candidate generation followed by learning-to-rank, subject to constraints such as scalability and near-real-time serving (N. Li et al., 2024; Xie et al., 2020).

The effectiveness of RS hinges on robust data collection, processing, and analysis that turn raw interaction traces into reliable suggestions. This need is especially pronounced in hub platforms that centralize many different tools (applications, widgets, links, and others) into a single workspace. In such settings, an RS boosts productivity by surfacing relevant items, shortcuts, and resources from within a large, evolving catalog. These settings pose distinctive challenges: integrating heterogeneous data sources, responding in (near) real time, and scaling as usage grows. With that, adopting Machine Learning Operations (MLOps) helps meet these challenges by automating pipelines, monitoring quality, and enabling continuous improvement, while process models like CRISP-ML(Q) emphasize repeatability and quality assurance across the ML lifecycle (De la Rua Martinez et al., 2024; INNOQ, 2025).

Each classical RS algorithm approach has limitations. CF suffers from sparsity and cold start for new users/items, while CBF tends to over specialize and may struggle when content descriptors are weak or multimodal (e.g., images, audio). As a result, hybrid architectures (weighted, switching, feature combination, model based) are widely adopted to balance diversity and accuracy to mitigate sparsity and cold start. Motivated by these challenges, recent work blends deep models (e.g., Neural CF) and context aware factorization to capture richer signals (Julianti et al., 2022), and compares similarity functions to trade off ranking

quality and catalog coverage in practical media domains (Vemberly et al., 2023). Beyond CF/CBF, large language models (LLMs) open a complementary pathway for text-heavy or schema light domains: they capture semantics in unstructured descriptions and, with Retrieval-Augmented Generation (RAG), can ground responses in fresher, domain-specific knowledge. However, LLM-based recommenders require the same (or stronger) operational guardrails such as versioned data and models, drift detection, explainability, and security by design. This will reduce risks like prompt injection, data leakage, and unsafe outputs, but, will also require a RAG integration with a different approach (B. Long et al., 2025).

Critical Techworks (CTW) develops software for BMW Group, including MWP, a centralized web platform used daily by a diverse internal BMW Group user base (manufacturing, IT, analytics, and more). MWP aggregates hundreds of internal tools into customizable dashboards (public and private). Because the catalog is both large and heterogeneous, users struggle to discover relevant resources, creating friction in onboarding and daily workflows and missing out on the full capacity of the hub itself. Therefore, stakeholder feedback surfaced a clear need, which is to recommend the right applications to the right departments and users inside MWP whenever a menu is opened to search for applications, improving productivity and tooling adoption.

The operational impact of suboptimal discovery is nontrivial: time lost searching, underutilization of strategic tools, and uneven diffusion of best practices. While MWP aggregates applications, widgets, and dashboards, this project deliberately restricts the proof of concept to application recommendations. This focus reflects the availability of more data (with better quality) for applications, which enables a fair evaluation of content-based and hybrid methods under operational constraints. Concretely, the data pipeline, model interfaces, and deployment approach are designed to generalize, so the same solution can later be extended to widgets and dashboards with minimal changes to schemas and serving models based on the needed resource.

To be useful in production, the RS must be repeatable, observable, robust to drift, and easy to evolve. Concretely, this means automated ETL and feature pipelines; versioned, containerized model artifacts; Continuous Integration/Deployment (CI/CD) for training and serving; pre-deployment validation with guardrails (e.g., metrics thresholds and coverage constraints); and runtime monitoring/alerting for data, model, and service health. Together, MLOps and security by design enable a scalable, auditable evolution from content-only recommendations toward hybrid and, potentially, LLM-assisted flows, without sacrificing reliability or compliance. This project therefore explores how to bring MLOps practices into the RS lifecycle.

Recommendation pipelines must handle enterprise traffic with minimal downtime, comply with internal security requirements, and remain auditable under strict governance. These constraints make MLOps integration not optional, but essential.

1.2 Objectives

This project designs and validates a recommender system strategy for MWP, grounded in CRISP-ML(Q), that evolves from content-only to hybrid models as richer data becomes available. To achieve this, the following objectives are pursued:

1. **Review the state of the art.** Search for existing approaches of recommender systems and MLOps best practices, with emphasis on industrial scale applications (or approaches), operational constraints, and quality assurance. This review provides the conceptual foundation for the model algorithms approach and architecture experiments that follow in Chapters 5 and 6.
2. **Characterize the problem and success criteria.** Formalize the recommendation approach necessary inside MWP. Describe current data constraints (e.g., department-level logs and coarse item metadata), and define measurable offline success criteria and prospective online KPIs.
3. **Build and document the dataset.** Design an extraction and preprocessing workflow that consolidates interaction logs and metadata into a reproducible dataset. Ensure that schemas, filtering, and transformations are transparent and auditable, enabling future extensions.
4. **Propose a reference architecture and lifecycle.** Present an end-to-end, MLOps-aware blueprint (data readiness, model training/validation, deployment, and monitoring) tailored to MWP's constraints.
5. **Investigate and evaluate content based, collaborative and hybrid approaches under realistic constraints.** Study content-based models and collaborative filtering and explore hybrid variants aligned with available attributes that balance accuracy with catalog coverage and novelty. Establish an evaluation protocol using standard ranking metrics (Precision@K, Recall@K, MAP@K, NDCG@K) and coverage, with leakage-safe splits.
6. **Specify operational and governance requirements.** Derive requirements for automation, versioning, data quality and drift checks, observability, privacy, and security by design so that future iterations remain reliable, auditable, and compliant.

1.3 Research Questions

The main objective of this project is to develop a recommendation system for a centralized hub that aggregates various applications following an MLOps approach that aims for customer satisfaction inside the BMW group, and therefore focuses on their area of work and expertise. Based on this overall goal, this project addresses three research questions:

RQ1: How can MLOps support the development and deployment of recommender systems across the data preparation, training, deployment, and monitoring phases, and what challenges arise?

RQ2: How does the integration of MLOps practices, such as CI/CD, automated feature engineering, model monitoring, data versioning, and containerization affect the scalability, reliability, and maintainability of recommender systems?

RQ3: What are the trade-offs between automation and customization in recommender systems developed using MLOps workflows?

In Chapter 2, these same questions will be revisited and addressed through the systematic review of the collected articles.

1.4 Contributions

This project makes several key contributions to the field of recommender systems, particularly in the context of integrating MLOps practices, summarized as follows:

1. **Development of a Recommendation System with MLOps Integration:** The project introduces a robust architecture for a recommendation system tailored to the MWP application. By leveraging MLOps methodologies, the system is designed to provide personalized recommendations to users based on their daily tasks, areas of expertise, and department.
2. **Implementation of a Comprehensive Data Pipeline:** A detailed Extract-Load-Transform (ETL) workflow is developed to preprocess and aggregate data from multiple sources.
3. **Automation and Monitoring for Continuous Improvement:** The proposed system incorporates CI/CD pipelines and observability tools to automate key processes, including model retraining, model evaluation, deployment, and performance monitoring. This automation minimizes manual intervention while ensuring scalability and reliability.
4. **Addressing Challenges in MLOps for Recommender Systems:** The project identifies and addresses unique challenges in applying MLOps to recommender systems, such as handling diverse data sources, managing data drift, active monitoring of the application and others. These insights contribute to the broader understanding of MLOps in specialized domains such as MWP.
5. **Real-World Application within BMW Group:** The system is implemented as a part of the MWP ecosystem, providing great value to BMW internal employees improving user experience, more specifically productivity. This practical implementation demonstrates the feasibility and effectiveness of the proposed solution.
6. **Contribution to Academic Knowledge:** By combining state-of-the-art practices in recommender systems and MLOps, this project bridges a critical gap in the literature. The findings provide a foundation for future studies exploring the combination between these two different fields.
7. **Framework for Future Enhancements:** The design of the system allows for future scalability and maintainability. The incorporation of advanced machine learning techniques, such as deep learning models, real-time analytics, or adaptive user interfaces will guarantee the effectiveness and stability of the deployed application.

1.5 Document Structure

This project is organized into seven chapters, each addressing different aspects of the research.

Chapter 1 introduces the context of the project and explains the need for the proposed RS. It also outlines the study's contributions and the research questions the project seeks to investigate.

Chapter 2 presents a scientific background, providing context on recommender systems, MLOps, and ETL pipelines. It surveys relevant technologies and methodologies, followed by a systematic review of studies in the area.

Chapter 3 outlines the design approach and requirements analysis, providing a high-level technical overview of the problem and presenting an initial design for its potential solution.

Chapter 4 delves into the data analysis, focusing on the data available for ingestion. This chapter includes an initial exploratory analysis, along with the feature engineering and preparation processes.

Chapter 5 outlines the different models to be trained on the processed data. It presents algorithms and approaches to achieve the best possible model for the recommendation system.

Subsequently, Chapter 6 explains the implemented architecture that supports the end-to-end ETL pipeline (following the MLOps approach) and how the RS is deployed to serve recommendations. It also includes a final architecture diagram to illustrate the overall landscape.

Finally, Chapter 7 presents the final conclusions drawn from the development and analysis of results. It also describes the limitations encountered and proposes directions for future work.

Chapter 2

State of the art

This chapter reviews the base concepts and related work necessary to develop the proposed project within the broader literature. First, it starts by outlining the scientific background, covering recommender systems, machine learning operations, and extract-transform-load pipelines. These concepts are essential to rationalize the design decisions and results presented in the following Chapters. Lastly, this chapter explores related works to this project, before concluding with a Systematic Review that directly addresses the research questions introduced in Chapter 1.

2.1 Scientific Background

Taking into consideration a recommender system (RS) that fits the needs of this project, it will be needed a solid grounding in three domains: RS, MLOps, and Extract-Load-Transform (ETL) pipelines. Conceptually, RS provide the foundation for the algorithms and strategies explored later, including traditional, multi-criteria, and soft computing-based approaches. Operationally, MLOps establishes the principles for robust machine learning systems. Moreover, ETL pipelines ensure that data is reliably extracted and prepared to be used.

2.1.1 Recommender Systems

The rapid growth of online information has intensified the need for filtering mechanisms, giving rise to RS as a central component of modern digital platforms. Classical paradigms include collaborative filtering, content-based, and hybrid methods, later extended to multi-criteria recommenders capable of modeling aspect-level preferences (Julianti et al., 2022). Over time, the field has increasingly integrated machine learning, and more recently, deep learning techniques that capture complex user–item relationships beyond usual linear interactions.

From an engineering perspective, scalability has become a defining challenge. Deep neural network, based RS, while powerful, are often memory intensive and computationally expensive, especially when trained on million-scale catalogs. To meet these demands, parallel and distributed architectures have emerged to reduce training bottlenecks, enabling hybrid RS that exploit both model- and data-parallelism for large-scale deployments (Stergiopoulos et al., 2024). Similarly, embedding compression methods like RecJPQ adapt quantization techniques from information retrieval, reducing embedding tensor size by up to 48$\times$ while maintaining or even improving sequential recommendation quality (Petrov and Macdonald, 2024). These directions reflect an industrial push toward systems that can efficiently operate at billion-user and billion-item scale.

Recent work has also explored automation and adaptability in RS design. AutoRec introduced one of the first AutoML frameworks for recommender models, automating architecture search and hyperparameter tuning within TensorFlow. It supports both sparse and dense inputs across rating prediction and CTR tasks, demonstrating the feasibility of modular and searchable pipelines for recommendation (T.-H. Wang et al., 2020). In parallel, meta-learning has been proposed as a strategy to personalize sub-models per user or entity. At LinkedIn, LiMAML applies model-agnostic meta-learning to generate meta-embeddings deployable in production, yielding consistent gains over strong baselines for CTR and job recommendation scenarios (R. Wang et al., 2024).

The rise of Large Language Models (LLMs) has opened new opportunities for recommendation, taking advantage of their semantic reasoning and generative capabilities. Emotion-aware music recommenders such as LEMON employ LLMs to extract nuanced affective signals from lyrics and align transient emotional states with long-term preferences, achieving superior accuracy in large-scale A/B testing (S. Wang et al., 2025). In the context of serendipity, SERAL aligns an LLM-based component (SerenGPT) with human judgments of unexpectedness and integrates nearline generation with online ranking to overcome latency constraints, successfully deployed in Taobao's production environment (Xi, Weng, et al., 2025). Beyond single-task modeling, maintaining LLM-powered RS in dynamic environments requires update strategies. Hybrid approaches that combine periodic fine-tuning with frequent Retrieval-Augmented Generation (RAG) updates have been shown to balance cost, agility, and knowledge freshness, significantly improving user satisfaction in billion user video platforms (C. Meng, Ling, et al., 2025).

Alongside modeling advances, concerns for fairness, beyond accuracy, and trustworthiness have become central. With that, surveyed work maps the space across group vs. individual criteria, calibration based notions, and counterfactual tests, highlighting tensions between procedural and resulting equity. These frameworks reveal persistent inconsistency in recommendation accuracy across user groups and exposure opportunities for minority items, motivating algorithmic interventions and overall fairness objectives (Y. Wang et al., 2023). Complementary studies explore personal calibration of non-accuracy criteria, finding that users vary in desired novelty/diversity. Conceptually, personalized re-ranking can then tune recommendations to each user's target mix (Naghiaei et al., 2024). In news recommendation, diversity has further been conceptualized as a normative principle, with metrics such as RADio extending descriptive diversity measures toward democratic theory and rank-aware evaluation (Vrijenhoek et al., 2024).

These concerns converge in the broader notion of Trustworthy Recommender Systems (TRS) (Ge et al., 2024). TRS research enforce that focusing solely on accuracy is insufficient meaning that models must be transparent, robust to adversarial manipulation, respectful of user privacy, fair to all stakeholders, and responsive to user control. In particular, explainability, for example, improves the trust and engagement, while controllability mitigates risks of recommendation loops by allowing users to influence their recommendations. Together, these dimensions outline a shift from accuracy-centric optimization towards a human-centered RS design.

In summary, recommender systems research has evolved from neighborhood and matrix factorization methods to highly scalable deep and large-model approaches, automated pipelines, and trustworthiness aware frameworks. For example, in a e-commerce application where a RS is implemented to suggest new products to the user, in case the catalog changes and a new product is implemented, the RS will not be prepared to handle that specific product.

With a MLOps approach the system will be prepare to, in a sequentially train and serve, to suggest this new product. However, current challenges are not only in achieving a good state-of-the-art accuracy but also in ensuring a robust recommendation ecosystem.

Traditional Approaches

Early RS relied on collaborative filtering and content-based paradigms. In, matrix factorization and neighborhood methods learn latent user–item structures from historical interactions, enabling the possibility of recommendations even in limited rating environments. In contrast, CB systems build user profiles based on consumed/used items and match them against item metadata such as text, categories, or attributes. Moreover, knowledge-based systems extend these ideas by encoding explicit domain rules or constraints, while hybrid systems combine signals from CF, CB, and KB to mitigate cold-start and sparsity problems. Empirical evidence shows that hybrid CF+CB models on datasets like MovieLens 100K can balance accuracy and diversity, alleviating user cold-start while improving RMSE and FCP performance under simple evaluation splits (Vemberly et al., 2023). However, evaluation of these early models has a history centered on top-K metrics such as Precision@K, Recall@K, MAP@K, and NDCG@K, and, as well as, catalog-oriented measures like coverage, with offline validation (Argyriou et al., 2020).

Multi-Criteria Recommender Systems

Multi-criteria recommender systems (MCRS) generalize single-score feedback by modeling multiple criteria such as quality, price value, or usability, offering richer preference signals and more controllable trade-offs during recommendation. Methodologically, soft computing technique such as fuzzy logic, evolutionary algorithms, particle swarm optimization, and neural models are often adopted in MCRS due to their robustness against uncertainty and incomplete data. For instance, in personalized e-learning, MCRS estimate similarity between course materials and learner goals to generate adaptive learning paths that reduce cold-start effects and dropout while improving alignment to heterogeneous learning styles and prerequisite structures (Julianti et al., 2022). These systems highlight the interplay between item-level facets, user profiles, and outcome-oriented objectives such as competency attainment in minimal time, making them particularly suitable for domains where multi aspect utility is critical.

Soft Computing and Deep Learning in RS

Modern recommender systems increasingly adopt deep architectures for representation learning and sequential modeling, including CNN, RNN, and Transformer backbones. To meet industrial constraints, these models are often adapted with efficiency mechanisms such as sparsity, token merging, and hybrid attention, enabling histories of up to 10^4 events to be modeled with GPU feasible training and serving (Chai, Ren, et al., 2025). Despite these gains, deployment pressures have further motivated compact designs, for example through distillation and pruning. Building on this idea, StuGAN integrates knowledge distillation into GAN based recommenders to reduce parameters while maintaining accuracy on real-world datasets (Y. Zhao et al., 2022), while Shaver applies Shapley value guided pruning to retain performance under tight memory budgets with field-aware codebooks (Tran et al., 2025).

Large Language Models (LLMs) have also been embedded into domain-specific RS. LEMON, for example, employs LLM-driven emotion extraction from lyrics and dual-temporal encoders

to balance transient emotions with long-term preferences, improving music recommendations in production (S. Wang et al., 2025). At a systems level, AutoRec demonstrates the potential of AutoML for RS, providing modular pipelines for sparse and dense inputs across tasks and automatically discovering near–state-of-the-art configurations without expert tuning (T.-H. Wang et al., 2020). Complementarily, LiMAML applies meta-learning for personalization, generating meta-embeddings that allow LinkedIn to serve adaptive CTR and job recommenders at scale (R. Wang et al., 2024). Together, these directions show how these techniques deal with industrial challenges.

Traditional Evaluation Metrics

The quality of recommendation lists is often assessed offline with top K ranking metrics. With that, Precision@K measures the fraction of recommended items that are relevant to the user, while Recall@K captures the fraction of all relevant items retrieved in the top-K. Mean Average Precision at K (MAP@K) averages the precision at each rank where relevant items appear, rewarding implementations that obtain correct items earlier in the list. Normalized Discounted Cumulative Gain (NDCG@K) further incorporates graded relevance and applies logarithmic discounting to lower-ranked items, emphasizing both accuracy and position. Beyond ranking, catalog-level metrics such as Coverage@K evaluate the proportion of the item space exposed by the recommender, which is important for ensuring long-tail visibility and novelty. In production, these offline evaluations are typically complemented with online A/B testing, where business-facing metrics such as CTR, conversion, revenue, or dwell time provide direct measures of user impact (Argyriou et al., 2020).

Beyond Accuracy: Additional Evaluation Dimensions

While accuracy metrics, mentioned in the previous subsection, remain central, recent literature gives importance of additional dimensions that capture more aspects of a good user experience and system behaviour. Complementarily, Diversity ensures that recommendations are not overly redundant, while novelty emphasizes the inclusion of less popular or previously unseen items. From a user experience perspective, Serendipity goes further by rewarding recommendations that are both relevant and surprising. Calibration assesses how well the distribution of recommended items aligns with user preferences, preventing deviations toward the dominant items. Fairness addresses whether different users, groups, or items are equitably represented, mitigating the systematic bias in exposure (Y. Wang et al., 2023). Finally, Robustness evaluates how stable and reliable the system remains under noisy or adversarial conditions.

These different perspectives are recognized in academic and industrial frameworks, with some studies proposing personalized calibration and normative diversity measures for domains such as news recommendation (Naghiaei et al., 2024, Vrijenhoek et al., 2024). For this project, the evaluation described in Chapter 5 is scoped to ranking and catalog-oriented metrics, while these broader dimensions are noted as future directions for expanding recommender assessment.

2.1.2 MLOps in Recommender Systems

The deployment of RS in any environments requires more than algorithmic efficiency, it demands various aspects such as reproducible, scalable, and auditable processes that can handle rapidly evolving data and dynamic usage contexts. These needs have enforce the

need of MLOps, an extension of DevOps that adapts its principles to, also, the need of a data-driven context of a machine learning ecosystem. While early work on recommender systems gives importance to collaborative filtering, content-based strategies, or hybrid modeling, modern approaches reveal that algorithmic progress alone is not enough. The true bottleneck lies in ensuring that data pipelines, training workflows, and deployment environments are robust to drift, resource efficiently. In this sense, MLOps is no longer optional but a cornerstone of industrial recommendation systems.

From DevOps to MLOps

The transition from DevOps to MLOps reflects a shift from uniform software releases to a multi-staged model lifecycles. In DevOps, Continuous Integration (CI) and Continuous Delivery (CD) enable software to be built, tested and deployed at in different environments meanwhile in MLOps, these practices are extended with Continuous Training (CT), where models must be cyclically trained and deployed as new data arrives (Moreschini et al., 2023). This extension is non-trivial because machine learning artifacts are inseparable from the data distributions they depend on. Unlike the traditional software code, models can decay in performance without any change in their logic, this is due to possible shifts in user behaviour, or a different item catalog, or, even contextual signals. This makes reproducibility and versioning a must where both code and data are tracked, validated, and audited throughout the system lifecycle.

Industrial recommender systems show these challenges noticeably. For example, Large-scale deployments, such as those at Netflix, LinkedIn, or Microsoft, demonstrate that training and serving environments must be integrated, enabling models to adapt to fast-changing logs while ensuring rollback capability in case of regressions. Microsoft's open-source Recommenders framework has this convergence integrated, combining algorithmic baselines with tools for data preparation, evaluation, and deployment in heterogeneous environments (Argyriou et al., 2020). The shift illustrates that recommender pipelines cannot be treated as static since they must continuously ingest, transform, train, validate, and redeploy under operational governance.

MLOps Lifecycle and CRISP-ML(Q)

To provide a structured basis, lifecycle frameworks such as CRISP-ML(Q) extend the classic CRISP-DM (Wikipedia contributors, 2025a) methodology by integrating quality assurance and risk management checkpoints trough the ML pipeline. CRISP-ML(Q) highlights the need for iterative loops where business objectives, data engineering, modeling, and monitoring are directly tied to reproducibility and governance Heck, 2024. This is particularly relevant for RS, where offline training and online serving can easily diverge due to data schema mismatches, inconsistent feature transformations, or even possible item catalog updates.

Feature stores like Hopsworks reinforce this idea by centralizing feature computation guaranteeing that training and serving share the same transformations (De la Rua Martinez et al., 2024). Similarly, OSSARA illustrates how an MLOps pipeline can be designed as an extension of the DevOps infinite loop, integrating ML developers into the development cycle alongside software engineers and operations teams. In this model, continuous training and deployment are core components, supported by open-source tools such as DagsHub,

DVC, and MLflow for data and model versioning, enabling automated cyclical training and deployment when performance thresholds are met (Moreschini et al., 2023).

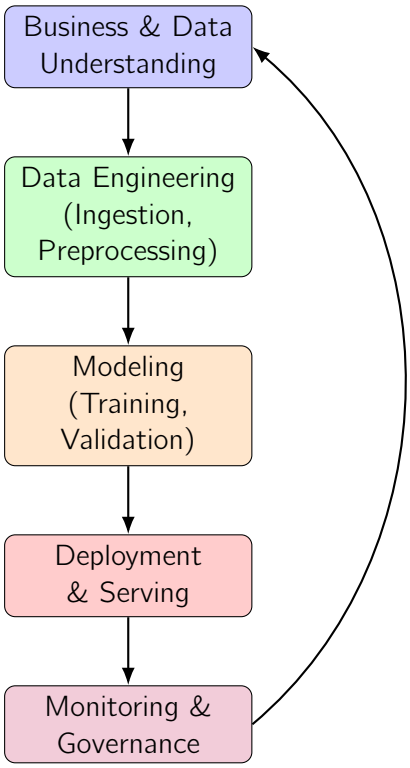

Figure 2.1: CRISP-ML(Q) lifecycle adapted to recommender systems.

Figure 2.1 shows how the CRISP-ML(Q) lifecycle aligns with RS implementation. Business and Data understanding are defined as recommendation objectives (e.g., personalization, diversity). Moreover, Data engineering takes in consideration ingestion of interaction logs and catalog metadata. Sequentially, Modeling covers algorithmic training and validation, from collaborative filtering to deep learning approaches. In addition, Deployment and serving refer to how the integration of models into user-facing applications can be done, while monitoring and governance ensure data compliance and possible drifts detection. This closed loop structure highlights how each stage must be structurally implemented in MLOps pipelines to sustain RS performance.

Beyond the mentioned frameworks, a recurring theme in the literature is that, usually, data rather than algorithms and their implementation is often the true bottleneck for the pipelines. This perspective, captured by data-centric AI (DCAI), highlights that data quality, governance, and reuse determine the performance of machine learning systems (Heck, 2024). The boundary between DataOps and MLOps is blurred, both focus on automation, traceability, and collaboration, but MLOps adds continuous training and deployment.

Studies in AI data engineering demonstrate that without systematic mechanisms pipelines remain fragile and models are at risk of silent failure (De la Rua Martinez et al., 2024; Moreschini et al., 2023). Feature stores exemplify this integration, as they centralize feature definitions across training and serving mitigating offline/online deviation. In this way, data engineering is not a supporting role but a foundational dimension of MLOps maturity, especially in domains such as RS where data quality and consistency are important.

Challenges in Industrial Recommender Systems

Despite methodological progress, industrial-scale recommenders continue to surface structural challenges that define the boundaries of MLOps. Leading among these is concept drift as user preferences, seasonal trends, or catalog updates evolve, model accuracy can deteriorate rapidly if the cyclical training is not sufficiently frequent or adaptive. Approaches such as ReLoop2 introduce test-time adaptation modules that leverage error memory to mitigate drift without full retraining, demonstrating promising responsiveness in dynamic environments (Zhu et al., 2023).

Scalability further constrains recommender pipelines. Embedding tables in deep learning recommenders may span terabytes, requiring distributed parameter servers, hybrid memory layouts, and optimizer strategies that balance throughput with efficiency. Kraken exemplifies this frontier by trisecting embedding memory costs while maintaining accuracy through sparsity-aware optimization and adaptive storage policies (Xie et al., 2020).

Latency also remains a pressing concern. Real-time recommenders must serve results within tens of milliseconds, leaving little room for computationally intensive pipelines. This forces system designers to adopt tiered architectures, lightweight retrieval and filtering stages followed by more complex scoring and ranking, as codified in NVIDIA Merlin's four-stage pipeline (Higley et al., 2022). Finally, governance and compliance requirements increasingly change the industrial deployments where explainability, fairness, auditability, and privacy are no longer optional but mandatory in supervised domains. Together, these challenges demonstrate that industrial recommenders require MLOps not just for efficiency but for trustworthiness and sustainability.

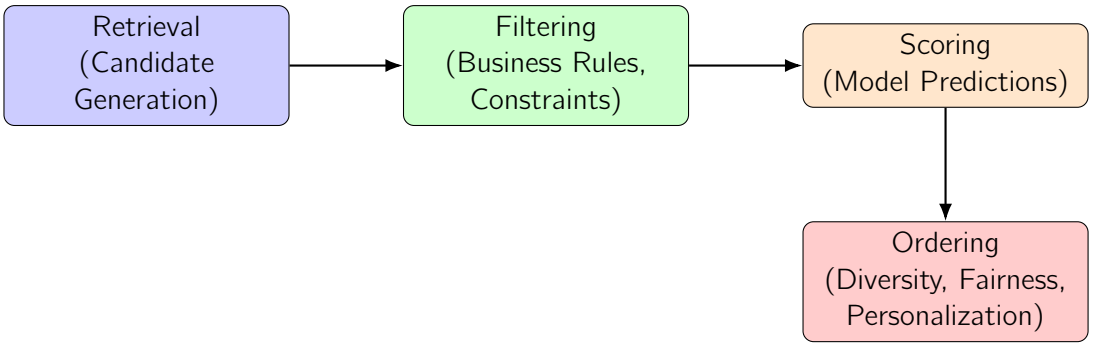

Figure 2.2: Four stage recommender pipeline.

Figure 2.2 represents a four-stage pipeline. The first step, Retrieval, filters the full catalog to a compact candidate set (e.g., via co-visitation, embeddings, or even popularity priors). After that, Filtering applies business rules and constraints (like eligibility, compliance, and availability) to remove inadmissible items. Subsequently, Scoring evaluates the remaining candidates with specific models and feature signals where low-confidence items can be removed from the list using thresholds or guardrails. Finally, Ordering performs a final list level ranking to balance the items relevance producing the top items for the user.

MLOps Maturity Levels

The maturity of MLOps adoption is described in per automation levels, where the degree of CI, CD, and CT integration defines the robustness of the system pipeline. At Level zero,

model development is basically manual where data scientists run experiments locally, deployment is ad hoc, and monitoring is also manual. Level one introduces automation in data gathering, training, and deployment, usually supported by feature stores and model registries, enabling repeatable pipelines and active monitoring. Finally, the most complete, Level two represents full automation, where a single code commit (git commit for example) can trigger the whole process from data ingestion to deployment into a production environment with integrated CI/CD practices (Amit Chauhan, 2025).

Few organizations currently achieve Level two, but it represents the aspirational benchmark for industrial recommender pipelines. By situating research within this maturity framework, it becomes clear that progress in tooling, governance, and automation is as central to recommender success as progress in algorithms themselves.

A key connection between MLOps maturity and RS evaluation is in the reliability of the performance metrics over time. In low maturity levels, metrics such as Precision@K, Recall, MAP, or NDCG may quickly degrade in production due to data drift or unmonitored feature inconsistencies. Level one automation already enables partial mitigation through monitoring and retraining pipelines, helping maintain catalog coverage and long-tail exposure. Only at Level two, where continuous training and CI/CD integration are in place, can organizations ensure that evaluation metrics measured offline are consistently sustained in dynamic online environments. This demonstrates that operational maturity is not only about engineering efficiency but also about preserving the validity and stability of recommender performance as discussed in Chapter 5.

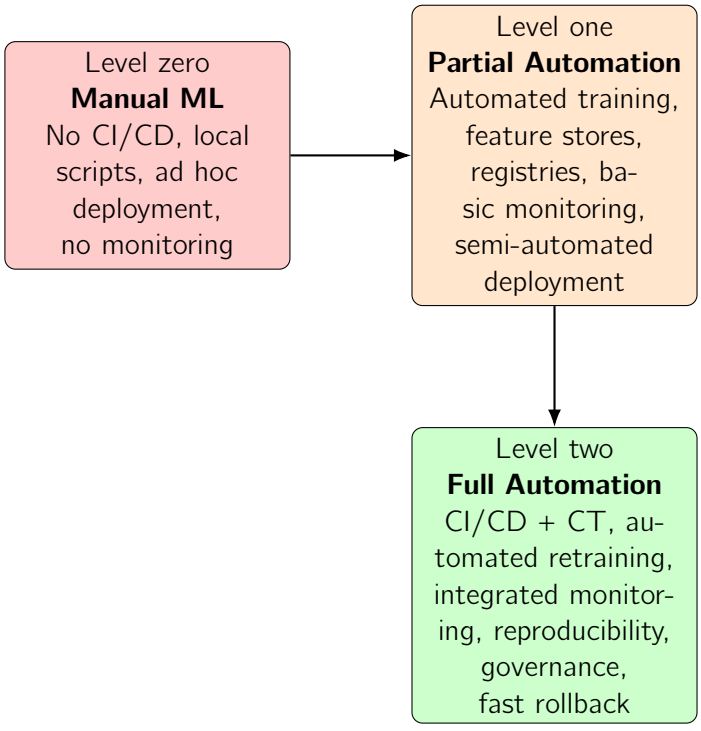

Figure 2.3: MLOps maturity levels.

Figure 2.3 summarizes three MLOps maturity levels. Level zero (Manual ML) relies on local scripts and ad-hoc releases, with no CI/CD or monitoring. Level one (Partial automation) introduces automated training, feature stores/registries, basic monitoring, and

semi-automated deployment, enabling repeatable pipelines. Level two (Full automation) implements CI/CD with continuous training, integrated monitoring, governance and reproducibility controls, fast rollback, and scheduled/triggered retraining, aligning with CRISP-ML(Q) and modern DevOps practices.

Key Tools and Frameworks

The tooling ecosystem for MLOps in recommender systems reflects the convergence of automation, scalability, and heterogeneity. For example, Reinforcement learning optimizers such as InTune dynamically allocates resources during DLRM training, improving, therefore, ingestion and training through factors exceeding 2× with production settings (Nagrecha et al., 2023). RePlay provides an experimentation library that extends from small-scale prototypes in Pandas to distributed Spark execution, enabling transitions from research to industrial deployment (Vasilev et al., 2024). For example, NVIDIA Merlin implements the mentioned four-stage pipeline facilitating the deployment of production-grade recommenders that scale with billion-interaction workloads (Higley et al., 2022).

Complementary these frameworks demonstrate the importance of the integration across the whole lifecycle. OSSARA demonstrates how open-source components (DagsHub, MLflow, DVC) can be orchestrated to create an end-to-end pipeline with CD/CT (Moreschini et al., 2023). Meanwhile, feature engineering platforms such as Hopsworks show control and adaptability by extracting data pipelines from model logic (De la Rua Martinez et al., 2024).

The previous mentioned tools are not evaluated only on algorithmic novelty but also on their ability to integrate without any problem into heterogeneous infrastructures in order to guarantee reproducibility, and support fast deployment cycles.

Positioning in Recommender Research

Within RS research, MLOps has transitioned from a supporting role to a defining dimension of production need. Academic work increasingly acknowledges that accuracy-focused evaluations are insufficient in practice, as operational resilience, lifecycle automation, and compliance ultimately determine how robust the system is. Pipelines such as OSSARA and frameworks such as Hopsworks, ReLoop2, and Merlin illustrate a new paradigm where methodological rigor in ML is inseparable. For RS in particular, this shows that advances in collaborative filtering, embeddings, or LLM-powered personalization cannot be implemented without also considering automation, monitoring, and governance. In this way, MLOps builds not only a bridge from research to practice but also is a research point itself, defining how RS can remain trustworthy in dynamic environments.

Finally, MLOps should be understood not in isolation but as one of three interdependent pillars of enterprise-ready RS. While recommender algorithms (Chapter 2.1) provide the core, and ETL pipelines (Chapter 2.1.3) ensure the reliable preparation of interaction data, MLOps delivers the operational backbone that integrates both into a robust end-to-end system. By taking in consideration automation, monitoring, and governance into the lifecycle, MLOps enables RS to evolve perpetually with user interactions while remaining verifiable and compliant. In this alignment, the advances in algorithms, data engineering, and MLOps maturity reinforce one another, defining the conditions under which industrial recommender systems can achieve both scalability and trustworthiness.

2.1.3 ETL Pipelines

In data intensive machine learning, the reliability and timeliness of upstream data flows often bound end-to-end system quality more tightly than model architecture does. Classical ETL emerged from data warehousing to shuttle records from heterogeneous sources into a consolidated store, emphasizing schema harmonization and batch output. As digital platforms shifted toward continuous logging, multimodal sources, and near real-time decisioning, ETL evolved into programmable, orchestrated data pipelines that serve both analytics and operational ML serving paths (Heck, 2024). In contemporary recommender systems, this evolution manifests as two intertwined requirements: (i) robust, low-latency paths that feed fresh features to online ranking components (Section 2.1); and (ii) governed, reproducible pipelines that guarantee consistency between training and serving, reducing offline/online deviation (De la Rua Martinez et al., 2024).

Traditional ETL

Traditional ETL is based on three stage process, having as the first step the batch workflow where is extracted structured data from operational systems, after that the transformation of it by cleansing, normalizing, and possible aggregations, and, finally, the load step where the results are sent to a warehouse (or other data structures) for downstream reporting. This model privileged daily (or in another time interval) replenishment windows, visual transformation tooling, and stable schemas. While effective for historical reporting, two frictions arise in ML settings. First, the frequency is misaligned with continuous data drift and currency needs typical of recommenders (e.g., trending items, short-lived contexts). Second, ETL stacks assumed a small set of predefined sources and formats, struggling with semi-structured logs, streaming data, or cross-organizational data exchange (Mondal et al., 2020; Zarate et al., 2024). These gaps motivated a shift from point tools to programmable, orchestrated pipelines with first-class support for streaming, metadata, and automation.

Modern Data Pipelines

Current modern data pipelines generalize ETL along two dimensions. On the temporal dimension, systems combine micro-batch or streaming ingestion with incremental transformation to shrink data staleness from days to seconds and, on the operational dimension, CI/CD principles are applied to the regular data flows: declarative orchestration, data validation gates, lineage and versioning, and, finally, automated deployments (Heck, 2024; Mondal et al., 2020).

From an AI data engineering perspective, the pipeline lifecycle covers ingestion, transformation, serving, and storage, and, also, must have in consideration both the offline (training) and online (inference) loops (Heck, 2024). In production DW/BI settings, automated pipelines reduce human intervention and shorten time-to-insight; integrating change data capture, schema migration, and ML-assisted preprocessing has been shown to improve data freshness and quality while supporting near real-time reporting (Mondal et al., 2020). In cross-organization settings (e.g., data spaces such as Gaia-X), pipelines also need to materialize *data products* (datasets packaged with contract metadata, quality descriptors, and access policies) so that data can be shared and monetized under sovereignty constraints (Zarate et al., 2024).

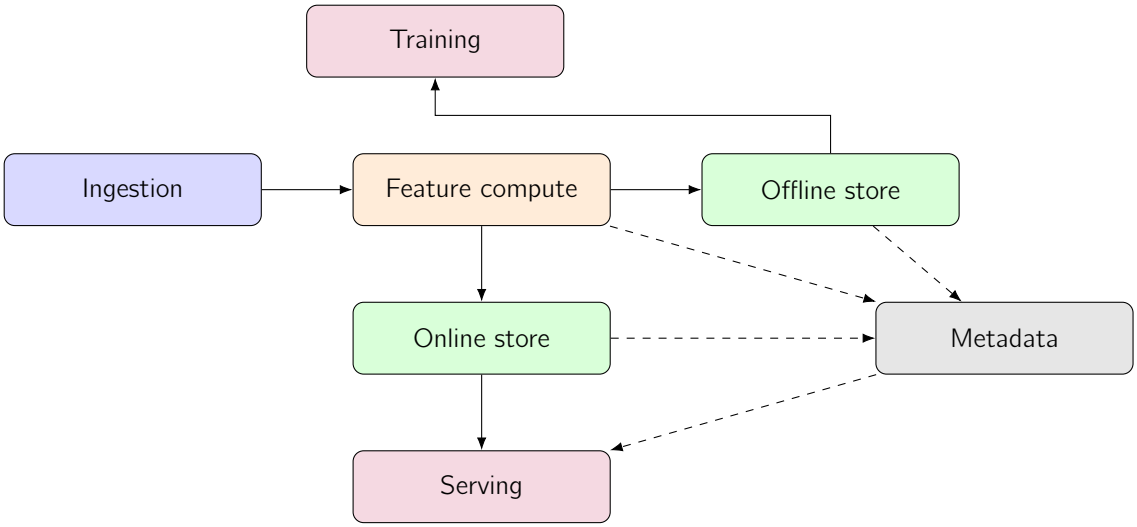

Figure 2.4: Feature pipeline layout.

Figure 2.4 shows a classic batch ETL with streaming/micro batch pipelines. In both, orchestration, validation, lineage, and versioning function as a data CI/CD layer that improves pipelines for ML use cases (Heck, 2024; Mondal et al., 2020).

Feature Stores and Data Versioning

Feature stores sit at the intersection of ETL and MLOps, unifying training and serving by centralizing feature computation, storage, and access semantics. From a design perspective, systems such as Hopsworks adopt a dual store, a columnar offline store for efficient historical reads and a low-latency online store for inference, with APIs to construct point-in-time correct training sets and to retrieve coherent feature vectors online (De la Rua Martinez et al., 2024). This design promotes feature reuse at scale (popular features are shared by many models), reduces write amplification from model-specific transforms, and mitigates training/serving deviation via shared, versioned transformation code paths (De la Rua Martinez et al., 2024; Heck, 2024).

In RS, the feature store becomes the bridge between the four-stage serving pipeline and the incoming upstream data, ensuring that currency constraints and access patterns are explicit (Higley et al., 2022).

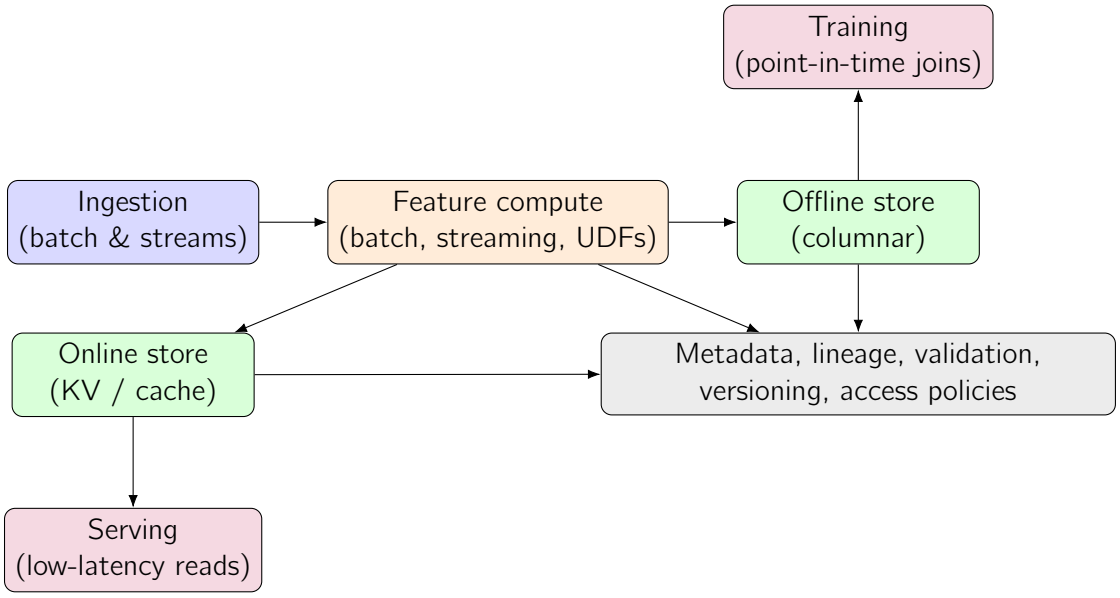

Figure 2.5: Feature store pattern.

Figure 2.5 highlights the dual serving pattern and the governance layer (metadata, lineage, validation, policies) emphasized in recent AI data engineering surveys (Heck, 2024) and feature store systems (De la Rua Martinez et al., 2024).

ETL Beyond the Firewall, Data Products and Data Spaces

A parallel evolution extends ETL from *internal* curation toward *external* exchange. Initiatives such as International Data Spaces and Gaia-X call for pipelines that do more not only to prepare tables but also to synthesize *data products* (datasets packaged with standardized metadata (like DCAT and DQV) so organizations can share and monetize data under sovereignty and interoperability constraints Zarate et al., 2024.

The DATAMITE approach positions ingestion, discovery, and metadata unification as first-class pipeline concerns, using orchestrators capable of batch and streaming (e.g., Mage) and catalog backends (e.g., Apache Atlas) to produce products that can be onboarded into Gaia-X via verifiable presentations Zarate et al., 2024.

For recommenders, these mechanisms enable trustworthy incorporation of third-party catalogs or context streams while preserving governance.

Automation, Validation, and Near Real-Time ETL

With respect to pipeline engineering, aligning ETL with MLOps means schema migrations under version control, CI triggers on data and pipeline code changes, automated unit/integration tests for transformations, and, eventually, rollout to different environments (Mondal et al., 2020). For instance, case studies in retail, marketing, and finance show that integrating these techniques into a database release automation with ETL orchestration and ML assisted preprocessing reduces latency between transactional changes and analytical availability, improving, therefore, data quality (Mondal et al., 2020).

In recommender systems, these practices directly affect evaluation fidelity and business KPIs where fresher features reduces offline/online divergence, while validation gates curb silent

failures that would otherwise confound model performance (De la Rua Martinez et al., 2024; Heck, 2024).

Implications for Recommender Pipelines

For the four-stage serving pattern (Higley et al., 2022), ETL pipelines will supply: (i) new candidates and user/item feature snapshots for the retrieval; (ii) policy and catalog signals (e.g., stock, geography) for filtering; (iii) point-in-time correct versioned features for the scoring; and (iv) diversity/fairness signals for the ordering. Feature stores then enforce a single transformation code path across offline evaluation and online serving, pulling together the link between the metrics used later in Chapter 5 and the data contracts that underpin them (De la Rua Martinez et al., 2024). In this sense, ETL is not merely a feeder system but a measurable, governed substrate whose automation level materially influences experimental validity and production reliability (Heck, 2024; Mondal et al., 2020; Zarate et al., 2024).

Cloud ETL Ecosystems

Public clouds converge on a similar reference stack for modern ETL where there is object storage for raw and curated zones, managed streaming for log/event ingress, declarative orchestration for batch and micro-batch jobs, and catalog/lineage services for governance. Within this template, platform choice is largely an operational trade-off (cost, latency, skill sets) rather than a conceptual one. In ML settings, a recurring pattern is to pair these data services with a feature store that exposes (i) point-in-time–correct training sets from the offline store and (ii) low-latency reads from an online store for inference (De la Rua Martinez et al., 2024).

From an AI data-engineering perspective, the cloud role is to standardize the lifecycle across ingestion, transformation, serving, and storage, while also enabling data oriented CI/CD, with validation gates, lineage, versioning, and automated rollouts (Heck, 2024). In cross-organization scenarios (e.g., data spaces), clouds also host the pipelines that synthesize *data products* (datasets packaged with DCAT/DQV-style metadata and verifiable credentials) so that organizations can publish under sovereignty constraints (e.g., Gaia-X onboarding flows) (Zarate et al., 2024).

2.2 Systematic Review

This review follows the guidelines for systematic reviews, including clearly defining the search criteria, the inclusion and exclusion parameters, and a detailed evaluation of the quality of selected studies, ensuring a reliable foundation for the findings and discussions section. This systematic review aims to provide a comprehensive analysis of the existing literature relevant to this project.

2.2.1 Methodology

According to Budgen and Brereton (Verner et al., 2012) and Kitchenham (Kitchenham et al., 2011), a systematic literature review (SLR) is a method for collecting and synthesizing all relevant research on a specific area, topic, research question, or phenomenon of interest. There is limited literature that reviews recommender systems using an MLOps approach. Therefore, a systematic review was conducted to explore existing works in this area. Following Kitchenham's methodology (Kitchenham et al., 2011), a review protocol was developed

outlining the necessary steps for conducting the SLR. This protocol includes formulating the research questions, defining the search strategy, and specifying the data sources and search terms to be used. It then conducts a search for relevant studies, screens and selects them based on defined inclusion and exclusion criteria, and evaluates the quality of the included studies using established quality assessment criteria. Next, it performs data extraction and analysis, compiles the evidence, interprets the results, and presents the findings to answer the proposed research questions.

Data Sources

In order to perform the systematic review, three different databases were selected to search for relevant surveys specifically IEEE Xplore Digital Library, ScienceDirect, and ACM Digital Library. IEEE Xplore, managed by the Institute of Electrical and Electronics Engineers (IEEE), is well known for its vast collection of resources in electrical engineering, computer science, and related subjects. ScienceDirect, an Elsevier database, offers a wide range of scientific and technical research, particularly in the physical and life sciences. Finally, the ACM Digital Library, provided by the Association for Computing Machinery, is distinguished by its focus on computing and information technology, containing a extensive collection of literature, including influential and pioneering research.

Search Terms

The choice of the search query to be used in the database involved selecting the keywords related to the search domain. The objective is to improve the knowledge about RS following an MLOps approach, so the following keywords were selected:

"Recommender Systems" OR "Recommendation Systems"; these keywords are interchangeable, and both are present in the literature.

"MLOps" OR "Machine Learning Operations OR "ML Operations" OR "DevOps for Machine Learning" OR "CI/CD for ML"; these terms have the key concepts of automating and operationalizing machine learning workflows.

"Scalability" OR "Maintainability" OR "Automation" OR "Customization" OR "Data Integration" OR "Model Monitoring" OR "Feature Engineering" OR "Deployment" OR "Observability" OR "Reproducibility"; these keywords represent important factors in the development and deployment of RS, focusing on system performance, adaptability, and the ability to handle large datasets reliably.

All the previous keywords mentioned were used to create three different queries. These were ran against the selected sources in order to fetch relevant surveys to the project. Table 2.1 displays the number of results found for each query in the different sources.

Table 2.1: Literature search hits by query and source.

Query	IEEE	ACM	ScienceDirect
("Recommender Systems" OR "Recommendation Systems") AND ("MLOps" OR "Machine Learning Operations" OR "ML Operations") AND ("Scalability" OR "Maintainability" OR "Automation")	2	50	113
("Recommender Systems" OR "Recommendation Systems") AND ("MLOps" OR "DevOps for Machine Learning" OR "Machine Learning Operations")	7	40	124
("Recommender Systems" OR "Recommendation Systems") AND ("Model Deployment" OR "Pipeline" OR "CI/CD")	164	1177	2819

Table 2.1 shows that a lot of entries were collected and, in order to fetch only the most relevant ones, the following Sections will be important to define how that will happen.

Inclusion and Exclusion Criteria

Given the previous results, the following Inclusion criteria was used:

- Studies that specifically address RS and their integration with MLOps practices, such as machine learning pipelines, CI/CD, model monitoring, and model deployment.
- Research papers discussing MLOps, including only those related to scalability, automation, customization, data integration, and model monitoring.
- Studies published within the last 5 years (e.g., from 2020 onwards), to ensure the literature is up-to-date and relevant to current MLOps and RS practices.
- Studies written in English (since the review is focusing on the global literature, and most of the research in this field is published in English).
- Peer-reviewed sources limited to the computer engineering and computer science domains, including journal articles, conference papers, book chapters, workshops, and technical reports that provide empirical research, case studies, or systematic reviews related to the integration of MLOps in recommender systems.
- Studies that specifically evaluate or propose techniques for integrating MLOps in recommender systems, discussing challenges, frameworks, methods, or case studies of practical applications.

The following criteria excluded sources from the selection process:

- Studies that do not focus on recommender systems or MLOps. For example, research on recommender systems without any connection to MLOps or studies on MLOps in unrelated domains (e.g., computer vision, natural language processing).
- Opinion papers, editorials, and theoretical papers that do not present new empirical findings or practical applications related to MLOps in recommender systems.

- Studies published before 2020, as they may be outdated in terms of MLOps practices, tools, and technologies, given the rapid evolution of both fields.
- Studies published in languages other than English, unless they provide an English summary or abstract sufficient for understanding.
- Studies with unclear or insufficient methodological details, such as those that do not explain the MLOps approach used, lack data, or provide insufficient details on the recommender system used.
- Non-peer-reviewed sources such as preprints, blog posts, and white papers, which may not meet the quality standards required for the review (exclusion applies only to the literature review subset).

Quality Assessment

After the paper passed the inclusion and exclusion criteria, its quality was assessed regarding these aspects:

- The study must directly address one or more research questions related to MLOps, scalability, maintainability, automation, and customization in recommender systems. If the study does not address these themes, it will be excluded.
- The study should describe a rigorous methodology, including clear descriptions of the data sources, models, and techniques used. Studies that do not provide adequate details about their methodology or leave important aspects ambiguous will be considered low quality.
- The study must be empirical (quantitative, qualitative, or mixed methods) or a well-conducted case study.

2.2.2 Data Extraction and Synthesis

After conducting the three research queries across the selected data sources (IEEE Xplore, ACM Digital Library, and ScienceDirect), a large set of results was obtained. To systematically analyze and filter these results, the PRISMA 2020 framework was applied, ensuring that only the studies relevant to this dissertation were retained.

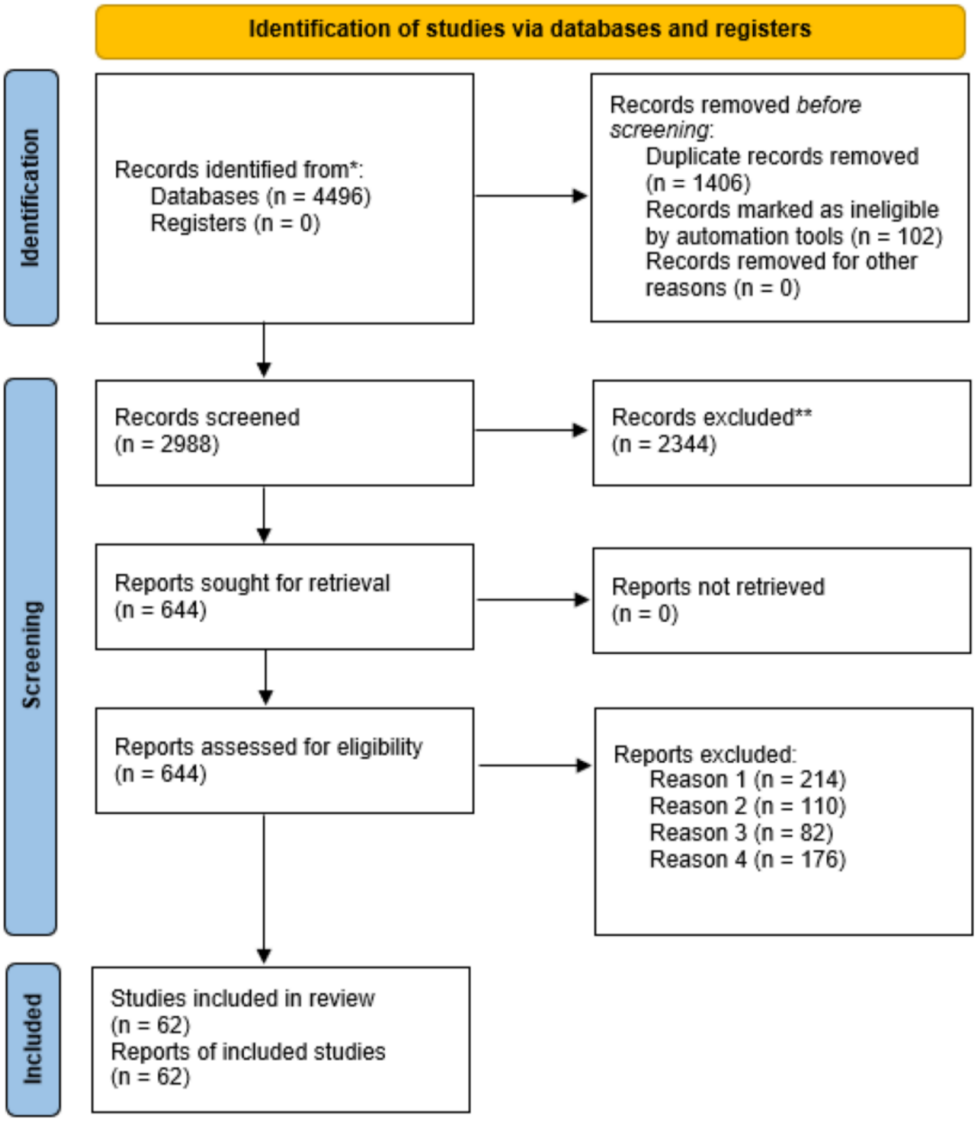

Figure 2.6: PRISMA flow diagram illustrating the study selection process.

Figure 2.6 shows the flow of records through the different phases of the review. In total, 4496 sources were collected from the database searches: 173 from IEEE, 1267 from ScienceDirect, and 3056 from ACM (Page and McKenzie, 2021).

Duplicate references were identified and removed using a Python script that compared titles, DOIs, and other metadata fields. This step eliminated 1406 duplicates. In addition, 102 records were automatically excluded by filters, mainly due to being published before 2020, not peer-reviewed, or not written in English. The script can be seen in Listing B.1.

The remaining 2988 records were screened at the title and abstract level. In this phase, 2344 records were excluded because they did not focus on recommender systems combined with operational machine learning practices (MLOps), or they belonged to unrelated domains (e.g., computer vision, natural language processing, or biomedical applications without connection to recommender systems). To achieve this, an LLM assisted screen was performed, followed by human confirmation.

The full metadata of 644 records was then assessed for eligibility. At this stage, 582 reports were excluded for the following reasons:

- **Reason 1: No integration of RS and MLOps (n = 214):** papers that presented recommender system algorithms but did not address deployment, CI/CD, monitoring, or other MLOps aspects.
- **Reason 2: Not empirical / no clear methodology (n = 110):** conceptual or vision papers that lacked case studies, experiments, or systematic validation.
- **Reason 3: Insufficient methodological detail (n = 82):** studies mentioning RS and operational aspects but without enough information to evaluate the approach.
- **Reason 4: Insufficient methodological detail (n = 176):** studies mentioning RS and operational aspects but not providing enough information to evaluate the proposed approach (e.g., missing data description, unclear pipeline, or absent evaluation protocol).

After this refinement, 62 studies were retained as the final set for in-depth analysis. These studies directly informed the discussion of challenges, frameworks, and trade-offs in integrating MLOps into recommender systems.

2.2.3 Findings and Discussion

The literature presents a progressive alignment between RS and MLOps, focusing on the lifecycle continuity, governance, and operational resilience as the main factors of impact. Versatile pipelines that extend DevOps with CT and ML artifact management (e.g., OSSARA) show how experiment tracking, versioned data, and automated promotion turn unstable handovers into closed feedback loops that connect offline progress to online performance Moreschini et al., 2023. At the same time, feature stores such as Hopsworks target training and serving consistency by enforcing shared, versioned transformations and point-in-time correct reads, reducing, with this, offline/online deviation (De la Rua Martinez et al., 2024).

On the serving side, production stacks tend to adopt multi-stage architectures that separate high-selectivity retrieval from learned ranking and ordering. Patterns popularized in industrial toolkits (e.g., NVIDIA Merlin) couple this staging with canarying, rollback, and latency budgets, embedding observability and control points at stage boundaries Higley et al., 2022. Algorithm collections such as Microsoft Recommenders advance the baseline library of models and utilities, yet their focus lies more on algorithmic breadth and templates than on end-to-end lifecycle automation and governance, which must be supplied by the surrounding MLOps fabric Argyriou et al., 2020. Complementary data-engineering guidance further elevates data contracts, validation gates, lineage, and quality checks to first-class lifecycle steps, aligning recommender pipelines with broader AI engineering practice Heck, 2024.

The benefits claimed across these strands are consistent. CI/CD/CT pipelines amortize frequent updates and reduce deployment risk; feature stores and data CI/CD improve reproducibility and reduce silent failures; modular orchestration stabilizes collaboration between data, model, and platform workstreams De la Rua Martinez et al., 2024; Heck, 2024; Higley et al., 2022; Mondal et al., 2020; Nagrecha et al., 2023; Vasilev et al., 2024; Zarate et al., 2024. At scale, production systems report that resource-aware training (e.g., reinforcement-learning allocation of I/O and compute in DLRM-style setups) can more than double throughput, shortening iteration cycles and enabling larger, fresher models Nagrecha et al., 2023. In serving, staged designs with explicit SLOs enable capacity planning and

autoscaling with predictable tail latencies Higley et al., 2022. Collectively, these practices contribute to scalability, reliability, and maintainability in ways that are measurable and repeatable.

The challenges surfaced by the same body of work are primarily systemic rather than algorithmic. Concept drift remains structural, even with continuous training, behavioural changes can outpace retraining cadences. Methodologically, test time adaptation with error memory (e.g., ReLoop2) offers responsiveness without full retraining but shifts complexity into inference and increases the need for careful instrumentation to avoid masking upstream data issues Zhu et al., 2023. At the other end of the spectrum, embedding scale and sparsity management drive systems complexity in large recommenders; storage- and sparsity-aware techniques can reduce footprint while preserving accuracy, yet they expand the MLOps surface that must be monitored and governed Xie et al., 2020. Governance and compliance cut across all layers: audit trails, fairness considerations, and reproducibility must be embedded in data contracts and monitoring rather than bolted on after deployment Heck, 2024; Higley et al., 2022.

These tensions crystallize the trade-offs between automation and customization. Highly automated retraining and promotion reduce operational toil and time-to-fix but risk overreacting to short-term fluctuations, with potential side effects on long-tail exposure and beyond-accuracy goals if untempered by policy-aware re-ranking and catalog-level guardrails Heck, 2024; Higley et al., 2022. Conversely, customized systems-level optimizations (e.g., memory tiers for sparse embeddings) unlock efficiency and headroom while increasing the surface area of failure modes that the platform must observe and control Xie et al., 2020.

Ecosystem choices amplify these trade-offs: general-purpose, open pipelines (OSSARA) afford transparency and portability at the cost of additional integration work, whereas vertical stacks (e.g., Merlin or Microsoft Recommenders) accelerate path-to-baseline yet may constrain governance or introduce coupling Argyriou et al., 2020; Higley et al., 2022; Moreschini et al., 2023. In cross-organization settings, publishing data as products (DATAMITE) streamlines exchange under FAIR principles and verifiable credentials, but sovereignty and compliance requirements can slow rapid iteration when metadata obligations are stringent Zarate et al., 2024.

Positioned against this landscape, the present work focuses on a recommender-specific pipeline that integrates these strands under realistic constraints: limited content features at inception, department-level signals, and an emphasis on reproducibility, governance, and safe iteration. The review indicates that the largest gains at this stage arise from lifecycle continuity (CT pipelines and validation gates), training–serving consistency (feature definitions and filtered evaluation), and operational guardrails (coverage floors, canarying, rollback) rather than from exotic modeling. Accordingly, the system architecture in Chapter 3 foregrounds data contracts and validation gates; Chapter 4 operationalizes EDA-driven feature selection and drift checks; Chapter 5 compares content-based models under leakage-safe evaluation with macro/micro reporting; and Chapter 6 implements the end-to-end pipeline with observability, promotion thresholds, and rollback. Collaborative filtering and large-language-model components are scoped as subsequent enhancements once richer interaction signals and metadata become available, aligning the deployment trajectory with the trade-offs documented in prior work.

While the surveyed evidence is largely drawn from systems papers, industrial toolkits, and

case studies, and is therefore context-dependent, the recurring patterns on lifecycle continuity, feature consistency, and governance provide a robust basis for the design choices adopted here. Reported metrics across prior work are often not directly comparable due to dataset heterogeneity, different cutoffs (K), and varying evaluation protocols, which motivates the standardized macro/micro@K and coverage reporting adopted in Chapter 5.

Notably, few sources describe an end-to-end, auditable promotion policy that explicitly guards catalog exposure (e.g., coverage floors and tie-breaks) under insufficient content features in an enterprise setting. The pipeline developed in this dissertation addresses that gap by coupling leakage-safe offline evaluation with coverage-aware validation thresholds and controlled rollout, aligning automation with governed diversity requirements.

Most sources surveyed are systems or industry papers and toolkits, so empirical gains are context-dependent. Reported metrics are often non-comparable across datasets and cutoffs, which motivates the standardized macro/micro@K and coverage protocol adopted in Chapter 5.

Few papers demonstrate an auditable, coverage guarded promotion policy for enterprise recommenders under insufficient content and constrained metadata. So, basically, the pipeline designed in this project need to target this gap with explicit coverage floors, train-seen filtering, and the possibility of having canary/rollback controls.

The literature shows a clear convergence between recommender systems and MLOps, where lifecycle continuity, governance, and operational resilience matter as much as algorithms. End-to-end stacks couple CT/CI/CD with experiment tracking, versioned data, and feature stores to keep training and serving consistent, while multi-stage serving (retrieval/ranking/ordering) adds canarying, rollback, and latency budgets for predictable operations. Consequentially, the benefits include faster, safer iteration and fewer silent failures. However, systemic challenges persist (concept drift, embedding scale/sparsity, and cross-cutting compliance) shifting effort from model tweaks to platform discipline. These pressures expose trade offs where automation accelerates recovery but can overreact without policy aware guardrails, while deep systems optimizations buy efficiency but widen the surface to monitor and govern. Against this backdrop, the present work prioritizes lifecycle continuity (CT with validation gates), training (serving consistency with shared feature definitions and leakage-safe evaluation), and operational guardrails (coverage floors, canary/rollback) over exotic modeling, targeting an auditable promotion path suited to enterprise constraints and coarse content features.

Chapter 3

Design Approach and Requirement Analysis

This chapter conducts the design approach for the project, covering requirement analysis, data protection and privacy, security posture, and the selection of methods and tools that will be needed during the implementation. It identifies all key elements needed to meet the project objectives while meeting technical robustness for the recommender system and user needs.

3.1 Requirement Analysis

Requirement analysis is the first step and serves as the foundation of the project. This section explores the various functional and technical needs essential for implementing the proposed project. By analyzing the objectives and constraints, this phase ensures that all components of the system are well-defined, paving the way for a seamless integration of the design approach.

3.1.1 Stakeholder Requirements

The initiative to develop a RS emerged directly from a request provided by MyWorkplace (MWP) end users. These users highlighted the need for a feature or tool within the application to present to them with relevant applications, widgets, dashboards, and links that could streamline their daily workflows. This requirement comes from the fact that MWP integrates a lot of BMW applications and widgets. For new users, navigating this ecosystem can be overwhelming and time-consuming.

As an internal tool for the BMW Group, MWP caters to a wide variety of end-user groups. These range from manufacturing personnel in factories to DevOps teams responsible for integrating applications into the system. Among the primary user groups, the IT departments and car manufacturing teams constitute the most significant proportion of users. For example, manufacturing personnel rely on applications within the system to support factory operations, while IT professionals use MWP for development and operational tasks.

Given the diverse range of end-users and their distinct needs, the recommender system must cater to various roles and scenarios. This makes the development of a personalized and efficient recommendation feature a priority to enhance user experience therefore also increasing the productivity across departments.

While stakeholders requested suggestions for applications, widgets, dashboards, and links, the initial scope is deliberately limited to applications to validate the approach with the available data. Based on the outcome of the project, a new strategy can be considered to improve the data collection for widgets and dashboards in order to extend the project to those artifacts.

3.1.2 Functional Requirements

This section outlines the functional requirements of the proposed system for MWP. These requirements are derived from the business needs, stakeholder expectations, and the technical capabilities necessary to ensure the recommender system delivers value. They describe what the system must do to fulfill its intended purpose.

The following functional requirements (FR) were defined:

- **Personalized Recommendations**: The core FR is to generate personalized application recommendations for users based on their department and activity history. The system must learn continuously from user interactions to obtain the best possible results.
- **Integration with MWP**: Recommendations must be integrated into the MWP application, displayed inside of already existing menus that contain the applications and widgets. This recommendations will be provided through REST or GraphQL APIs for easy integration with the landscape.
- **Feedback Loop**: Users can react to recommendations (e.g., relevant / not relevant). Feedback is stored with minimal context (model version, rank, department) and aggregated for retraining. To avoid echo chambers, a small exploration budget (e.g., occasional re-ranking of candidates) may be introduced in later iterations.
- **Model Training**: The system must include a mechanism to periodically train models with updated data to capture new usage patterns and maintain recommendation quality.

3.1.3 Non-Functional Requirements

Besides functionality (FR), the system must meet quality attributes that ensure usability, maintainability, and long-term adoption. The following non-functional (NFR) requirements were formulated following the FURPS+ framework (Wikipedia contributors, 2025b).

To make NFR verifiable, service level indicators (SLIs) are defined as the measurements that are observed (e.g., latency, error rate), and service level objectives (SLOs) as the targets those SLIs need to meet (Beyer et al., 2016).

- **Functionality**:
  - The RS must support both content-based and collaborative approaches, with the possibility of hybrid extensions.
  - The system must work within the MWP current ecosystem without disrupting existing services.

- **Usability**:
  - Recommendations should appear directly in the MWP interface, without requiring additional user input/actions.
  - The UI should provide simple controls for feedback (e.g., thumbs up/down, relevance buttons).
  - The interface must comply with BMW Group UX/UI guidelines for layout, colors, and interactions.
  - Accessibility must be ensured, including contrast ratios and screen reader compatibility as stated in WCAG 2.1 (W3C, 2025a, 2025b).
- **Reliability**:
  - The system must provide high availability (HA) at all times (target uptime $>$ 99.5%).
  - Failover mechanisms and self-healing must be implemented to ensure CRISP-ML(Q) guidelines.
  - CI and scheduled training pipelines must ensure that models remain up to date at all time.
- **Performance**:
  - Response latency for generating recommendations should remain below a predefined threshold (established in performance testing). P50 $\leq$ 80 ms, P95 $\leq$ 200 ms, P99 $\leq$ 400 ms (measured at service ingress).
  - The system must sustain high throughput, capable of serving concurrent recommendation requests under peak load. Sustain $\geq$ 150 RPS at $<$ 0.5% error rate during 10-minute peaks.
  - ETL and feature engineering processes must complete within time limits that support near real-time updates. Nightly ETL completes in $<$ 40 minutes (P95); scheduled training completes in $<$ 20 minutes (P95).
- **Supportability**:
  - Maintenance should be simplified via CI/CD pipelines supporting rolling updates or canary deployments.
  - Observability must include logs, metrics, and alerts through the existing MWP monitoring stack (Loki, Prometheus, Grafana).
  - Documentation must be maintained for developers and operators, covering system maintenance, troubleshooting, and model updates.
- **Security and Privacy**:
  - All traffic data must be encrypted.
  - Access to the system must be restricted to the BMW intranet, with also proper role-based access control (RBAC).

– Compliance with GDPR via data minimization, pseudonymization, documented retention, and a lawful basis (e.g., legitimate interest). Explicit consent is required only where the chosen lawful basis demands it.

These SLOs/SLIs are enforced by the validation and rollout policy described in Chapter 6, ensuring that model promotion remains a controlled, observable process.

Table 3.1: Requirements traceability map (RTM).

Objective / RQ	**Key FR/NFR (Ch. 3)**	**Realized in (Ch. 6)**	**Measured in (Ch. 5 and 7)**
Improve discoverability (related to RQ1)	FR: Personalized recs; integration with MWP. NFR: latency/availability SLOs.	Serving API; AKS Deployments; CI/CD; HPA	NDCG@10, Recall@10, Coverage@10; latency/availability SLOs; business outcomes
Operate safely and audibly (related to RQ2)	NFR: observability (logs/metrics/alerts); security/RBAC; rollbacks	Loki/Prom/Grafana; validation gates; rollback path	Error rates; gate pass/fail; incident notes
Ensure pipeline scalability (related to RQ3)	FR: scheduled ETL+training. NFR: ETL/train windows; versioning stance	CronJobs; PV/PVC; training jobs; container registry (ACR)	ETL < 40 m; Train < 30 m; reproducibility evidence
Enable path to hybrid methods (related to RQ3)	FR: feedback loop; fairness guardrails; roadmap to CF/hybrid	Feedback ingestion; staged hybridization	Engagement proxies; dept-stratified metrics; future-work targets

Table 3.1 summarizes how Chapter 1 objectives and research questions are mapped to the FR/NFR defined in Sections 3.1.2 and 3.1.3, meanwhile the concrete implementation to them are available in Chapter 6), and, finally, the metrics and outcomes that are used to verify them are present in Chapters 5 and 7).

3.2 Design Approach

Based on the requirements detailed on Section 3.1.1, achieving the desired project solution requires designing and integrating a robust approach to address the problem.

3.2.1 Methodology Selection

The project will follow, as previously mentioned, to the CRISP-ML(Q) guidelines, ensuring that the entire RS can regenerate the model and deploy a new version based on newly trained artifacts. This approach aligns with the dynamic requirements of ML projects, where constant experimentation, evaluation, and feedback loops are essential for success. Additionally, the project will include mechanisms to roll back to a previous ML model in case any issues are detected.

3.2.2 Selection of Methods and Tools

The proposed project must integrate seamlessly into the existing MWP ecosystem, which follows a micro-frontend and monolithic backend architecture (with some additional backend microservices in the landscape). The solution system will be deployed on Azure Kubernetes Service, with Kubernetes serving as the orchestration layer. Kubernetes provides scalability, fault tolerance, and resilience, making it the natural choice for hosting and managing workloads in both development and production environments (Cloud Native Computing Foundation, 2025; Microsoft, 2025).

The Extract, Transform, Load (ETL) process will be automated using Kubernetes (K8s) CronJobs, enabling the periodic ingestion and preprocessing of data. The scheduling frequency (daily vs. every 2–3 days) will be evaluated to balance freshness with cost efficiency. Data will be extracted from two main sources: (i) OpenSearch, which stores user interaction logs (e.g., applications opened, widgets accessed), and (ii) the MWP domain PostgreSQL database, which provides authoritative application metadata and categories. OpenSearch is chosen for its distributed indexing and fast retrieval capabilities in log analytics scenarios (Linux Foundation, 2025), while PostgreSQL is already the established metadata source within MWP. During transformation, data will be cleaned, merged, and enriched into a unified dataset, ensuring that the features required for modeling (e.g., department–application usage matrices) are consistent and reliable. The processed dataset will then be stored in Kubernetes Persistent Volumes (PV/PVC), which ensure durability and cluster-wide accessibility.

For serving recommendations, the solution system will employ a lightweight and portable application framework. Concretely, Flask has been selected as the base API layer (Pallets, 2025) due to its simplicity and flexibility in exposing REST endpoints. While alternatives such as FastAPI were considered, Flask's maturity and ecosystem alignment make it better suited to CTW's landscape. To future-proof the interface, support for GraphQL queries will also be evaluated, enabling more flexible interaction patterns if required. Both the application and trained models will be containerized with Docker and stored in Azure Container Registry (ACR) for reproducibility and version control.

Deployment and lifecycle management of these containers will be handled by Kubernetes Deployments and Services. A ClusterIP Service will restrict access to internal backends, ensuring security by design. Resource definitions (e.g., CPU and memory requests/limits) will guarantee predictable scheduling and support horizontal scaling via the Horizontal Pod Autoscaler (HPA). For declarative configuration, Helm charts will be used to package Kubernetes manifests with parameterized values (Helm Authors, 2025). To align with BMW's GitOps practices, ArgoCD will synchronize git repositories with the cluster state, ensuring that any manifest change is automatically propagated to AKS with auditability and rollback support (CNCF GitOps Working Group, 2021; Intuit, 2025). For local development and validation, Minikube will provide a lightweight test cluster, enabling rapid iteration before promotion to enterprise environments.

The choice of model framework will depend on the complexity of the algorithms and from the collected data quality. For lightweight approaches (e.g., TF–IDF similarity, mixture models), Python libraries such as scikit-learn and pandas will be sufficient for rapid prototyping (Jérémie du Boisberranger, 2025; NumFocus, 2025). If later iterations explore deep learning (DL) or embeddings, TensorFlow or PyTorch will be more suitable due to their scalability and ecosystem support (Google Brain team, 2025; The Linux Foundation, 2025). In all cases,

trained models will be exported in reproducible formats, containerized, and versioned in the MWP ACR to ensure compatibility with CI/CD workflows and long-term reproducibility.

Finally, observability will be critical for production readiness. Logging will be centralized with Loki (Grafana Labs, 2025), while Prometheus and Grafana will monitor both system and model-level metrics such as inference latency, request volume, coverage, and engagement proxies. ArgoCD's audit trails, combined with alerting and rollback strategies, will ensure that failures (e.g., data drift or degraded metrics) can be detected and mitigated promptly. Together, these tools provide an MLOps-ready environment that emphasizes reliability, scalability, and transparency INNOQ, 2025.

Each training run is packaged with its container image digest, dataset manifest, and parameters, so that the same inputs always recreate the same model.

3.2.3 Proposed Architecture

Before model selection and automation, an EDA is conducted to quantify sparsity, skew, and feature reliability. As shown in Chapter 4, the department–application matrix is highly sparse and long-tailed; application descriptors are coarse (category-level) with limited text; and user-level signals are not yet available. Under these conditions, collaborative filtering is fragile (cold start and sparsity), and text-embedding models are underpowered. Therefore, the design starts with content-based methods over department and category signals, while the pipeline is prepared to evolve toward hybrid strategies when richer signals (text, user feedback) become available. Chapter 4 visualizes these constraints via a department$\times$application usage heatmap (sparsity), per-category support distributions (long tail), and coverage@K by department (catalog exposure), which motivate both model choice and the evaluation guardrails.

These findings shape the first iteration: a content-based model that recommends applications at login. Accuracy thresholds and guardrails are evaluated iteratively, but always against the same validation contract introduced in Chapter 6. When those thresholds are met, the model is bundled into a lightweight web service (REST/GraphQL) and readied for controlled rollout.

The MWP application consists of five environments, of which only three involve real users. Consequently, this deployment will only exist in three of the five namespaces within the cluster to serve end users. Once the pods hosting the web application are healthy and ready to handle traffic from the MWP domain, additional monitoring pods will ensure the application's continued reliability. These monitoring pods will include a Loki stack to verify the application's status. Alerts will also be configured to detect issues such as data drift in the model or pod health problems.

If a new model underperforms or SLOs are breached, the system falls back to the last known-good recommendations (or, if needed, a popularity-by-department baseline). Alerts are raised, dashboards point to the failing gate, and ArgoCD performs a clean rollback.

The cron jobs will run daily (or in a 42-72 hours interval) to train the model with new data that may have been added to the data sources. Furthermore, MWP will collect user feedback on the recommendations provided. This feedback will serve as a third data source, feeding into the cron jobs for retraining the model. This enables the integration of a collaborative filtering approach, transforming it into a hybrid recommendation system. This feedback

supplies the first explicit user signals, enabling a careful transition from content-only ranking to hybrid strategies when governance and data quality allow.

Figure 3.1 provides a high-level overview of the entire system, highlighting its potential implementation approach.

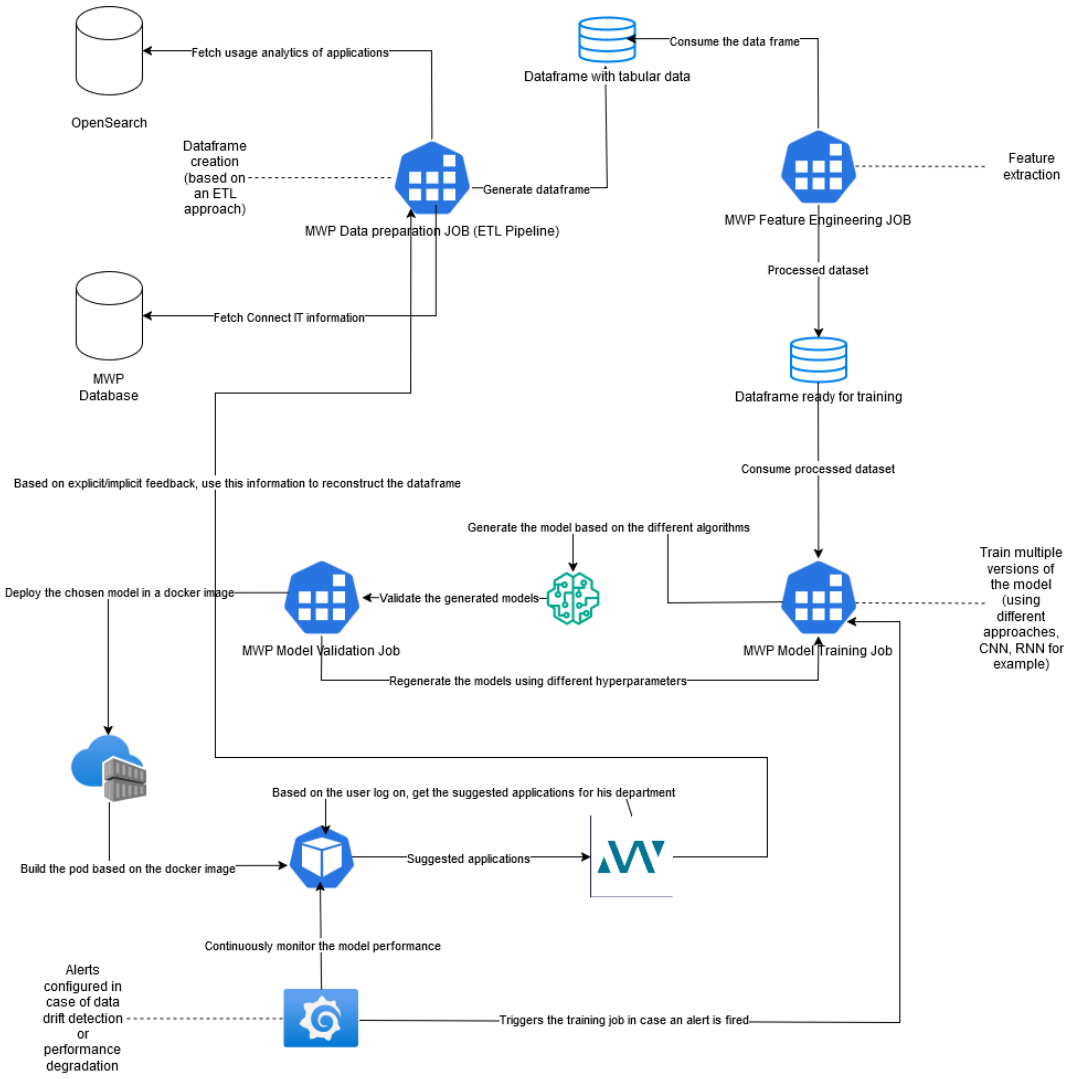

Figure 3.1: Proposed architecture for the recommender system.

3.3 Data Protection and Privacy Considerations

To ensure the solution follows legal and regulatory standards, compliance with relevant data protection regulations must be carefully described, including frameworks such as the General Data Protection Regulation (GDPR).

The ETL pipeline reads from OpenSearch (user logs/events) and the domain database (application metadata). To minimize risk, processing operates on department-level aggregates and pseudonymous identifiers, where no names are required for the first iteration. Log fields are reviewed against BMW internal policies, with data minimization and retention limits applied.

At authentication time, MWP can resolve a user's department and internal number. The first project iteration uses only departmental context and application attributes and, in the future, will include other type of data as stated in Section 7.3.

3.4 Security Analysis

Security is addressed here at the architectural policy level (isolation, least-privilege, encryption), while Chapter 6 details the concrete enforcement in Kubernetes and AKS.

In production, service-to-service authentication uses short-lived JWTs (using a private enterprise authentication system to generate them). With that, the Flask API should also implement this tokens mechanism and enforces RBAC on "/recommend" endpoint. Sequentially, the project will include secrets that will live in Azure Key Vault having rotation policies and expiration dates (to one year since it was created). Moreover, Kubernetes mounts expose only non-sensitive configuration. Finally SLA-based mechanism (using network policies) are default-deny to deny all incoming traffic. With that, the project remains cluster-internal and only accessible through different security mechanisms.

The recommender system will be integrated as an application with its own responsibilities and monitoring tools. It will be isolated from the public network and accessible exclusively through the BMW Intranet. The application will only communicate with the MWP backend, with all other intranet domains restricted. For the project, the service remains cluster-internal and relies on network isolation and service-to-service credentials. In the production path, the REST API must enforce bearer tokens end-to-end implemented, as outlined above. This maintains a clear hardening path without over-engineering the project. With these measures in place, the system aligns with BMW's internal security policies and provides a secure operational environment.

At the data layer, the ETL pipeline outputs are stored on Azure Data Disks, mounted into the cluster as Persistent Volumes (PVs) and accessed via Persistent Volume Claims (PVCs). This ensures durability, seamless integration with Kubernetes workloads, and restricted access to authorized applications inside the BMW intranet environment. Role-based access controls (RBAC) further enforce fine-grained permissions. Data at rest is encrypted by default within the Azure storage layer, providing an additional layer of protection.

Finally, daily geo-redundant backups safeguard against data loss, enabling recovery even in failure scenarios. In addition, rollback procedures allow the system to revert to previously validated model versions if newly trained models fail to meet the required thresholds, ensuring operational reliability.

3.5 Ethical Considerations

Ethical considerations are a critical aspect of any machine learning system and are treated here as first-class requirements alongside functionality and performance. The intent is to ensure that the recommender operates responsibly and equitably, preserves confidentiality, and remains auditable throughout its lifecycle from data ingestion to serving.

The data used in this project contain department identifiers and application attributes. Aggregation at department level reduces individual level risk but does not eliminate bias or privacy concerns. Fairness is therefore monitored by department where exposure and ranking quality are reported on stratified splits that preserve department identity, mentioned

in Chapter 5, and promotion gates include coverage guardrails to avoid popularity collapse into a few highly active departments. When a deviation is detected (e.g., persistent exposure imbalance or popularity drift), a calibrated re-ranking or a small exploration budget will be applied to maintain a balance between relevance and diversity. The objective is to have recommendations reflect legitimate usage patterns without amplifying incidental biases introduced by data sparsity or logging artifacts.

However, privacy and data protection obligations apply even when operating on aggregate logs. The solution adheres to data minimization and purpose limitation principles having only attributes required for analysis and modeling where temporal and device fields without discriminative value are dropped, discussed in Chapter 4, and personal identifiers are neither ingested nor persisted. Storage is limited to a rolling window sufficient for analysis and evaluation, with retention explicitly configured for both datasets and model artifacts, described in Chapters 4 and 6. All transfers occur over encrypted channels; at rest, storage uses encrypted volumes. Access is controlled through role-based policies and segregated namespaces so that only designated services can read or write model and data artefacts. Operational logs avoid printing raw payloads or secrets, and configuration separates secrets from non-sensitive parameters. These measures reduce the likelihood of reconstructing individual behaviour from aggregate signals and support compliance with internal governance.

Automation introduces additional ethical responsibilities. The MLOps workflow incorporates explicit validation thresholds, canary promotion, and rollback, present in Chapter 6, which together act as safeguards against the unintentional deployment of underperforming or skewed models. All training, validation, and serving events are recorded with immutable artefact versions, configuration hashes, and metrics summaries to support post-hoc auditability. Promotion is blocked when guardrails are not met, in that case the prior model remains active, preventing deterioration from propagating to users. These controls ensure that model evolution remains accountable and reversible.

Transparency and interpretability are addressed through model documentation and operator-facing explanations. Each promoted model is accompanied by a short model card describing inputs, evaluation protocol, known limitations (e.g., long-tail departments), and the business rules applied during re-ranking. For operations, per-request logs record which high-level signals (e.g., department–category similarity) contributed to a recommendation, without exposing sensitive raw data. This level of transparency facilitates internal review while respecting confidentiality constraints. Importantly, recommendations are presented as decision support: they do not enforce access or performance assessment and carry no punitive consequences. Users retain autonomy to ignore or down-vote suggestions, and those feedback signals are used strictly to improve utility in subsequent iterations, seen in Chapter 6.2.5.

Finally, governance processes frame the system's scope and boundaries. Changes to data sources, features, or thresholds follow documented review, and any proposal to introduce richer content (e.g., textual metadata or user-level histories) requires prior assessment of privacy impact and consent pathways before ingestion, as discussed in Chapter 7. The system avoids sensitive inferences (e.g., attempting to deduce personal traits) and refrains from cross-context use of data beyond the stated purpose of application discovery. Together, these practices promote proportionality, respect for stakeholder expectations, and long-term accountability in an automated enterprise setting.

Chapter 4

Data Analysis and Preparation

This chapter presents the process of collecting, preparing, and exploring the data that underpins the recommender system. Focusing on the exploratory data analysis (EDA), where several statistical and visualization techniques are used to understand the data quality, detect possible anomalies, and, also, understand the structure and distribution of key variables. These analyses are important to ensure that only relevant and representative features are retained for model training.

Finally, the Chapter ends with all exploratory findings that explained preprocessing choices, clarifying which variables were retained, transformed, or discarded ahead of model development in Chapter 5.

4.1 Initial Data Identification and Collection

This section focuses on the identification and collection of data necessary for building the solution system. It outlines the relevant data sources, the OpenSearch and MyWorkplace (MWP) domain database, the methods used to acquire data while adhering to privacy and security guidelines defined previously, and an initial examination of the data's characteristics. These foundational steps are critical to ensure that the data is suitable for further analysis and model development.

4.1.1 Data Sources

The ETL pipeline will initially process data from two primary data sources. The first is OpenSearch, which contains user logs, such as application accesses, widget usage, and other activities. The second is the primary domain database, which provides detailed information about the applications themselves, aiding in their categorization to enable accurate recommendations tailored to user needs. In the final phase, a third data source will be integrated: the feedback from users to recommendations generated by the system. This will serve as an additional data source, seamlessly incorporated into the ETL pipeline to further refine and enhance the recommendation system's performance.

4.1.2 Data Acquisition Methods

For exploratory analysis, the data was extracted from two sources: OpenSearch user interaction logs (application openings, widget usage, and related actions) organized in monthly indices, and the myWorkplace (MWP) domain database containing application metadata used for categorization. To assess volume and potential seasonality, up to six months of logs were inspected. However, the figures and statistics reported in this chapter use a four

month slice (approximately 1.7M rows), which provided stable distributions while keeping the analysis tractable. Listing B.2 and Listing B.3 (present in the B) illustrate how monthly indices can be enumerated and queried, subsequently, authentication details and exact index names are omitted for brevity. Retrieved documents were JSON objects that, for EDA purposes, were parsed into a structured tabular format and materialized in columnar Parquet to support efficient filtering, grouping, and joins at this scale.

In parallel, application categorization was retrieved from the primary domain database (PostgreSQL Flexible Server) via parameterized SQL, as illustrated in Listing B.4 (also present in B). The resulting categorical metadata were joined to the OpenSearch sample on the `action_info`, yielding a unified dataset with explicit links between departments, applications, and their categories. This normalization followed two straightforward steps: first, parsing and restructuring the OpenSearch JSON logs into Parquet; second, enriching that table with domain categories from SQL to map each application to its respective category or department. As a result, the unified dataset serves as the basis for the summaries and visualizations presented next.

4.2 Data Analysis

After analyzing the generated dataset, it became clear that many columns contain irrelevant information for the purposes, such as the user's method of accessing the application (e.g., browser), device type (mobile or desktop), timestamp of the log, and others. These columns are not necessary for the analysis. Four months of logs (1.7M rows) provided a sufficient and representative basis for EDA. For completeness, up to six months were initially inspected to gauge volume and potential seasonality, however, the four-month slice reported here provided stable distributions and sufficient coverage for all EDA figures and statistics in this chapter.

As a first step, it was conducted standard quality checks prior to visualization and summaries: (i) null analysis and per-column missingness profiling; (ii) uniqueness and duplicate detection at the (`department`, `action_info`, timestamp) level; (iii) format validation of department identifiers via a regular expression matching the internal naming convention; and (iv) basic range and domain checks for categorical fields. Eight malformed department identifiers were flagged by regex and removed accordingly.

To avoid hindsight bias during summary statistics, temporal ordering was preserved whenever relevant. The formal evaluation procedure and splits are detailed in Chapter 5.

The generated dataset tracks user interactions with the MWP application, recording data on various user actions. The logs, for instance, which application or resource was opened, the user's department, the device type, browser used, timestamp of the interaction, and other information. The data is stored in indexed logs that capture user activity on a granular level (based on the log collection logic, meaning, what is being tracked inside MWP). Nonetheless, a big portion of the data includes irrelevant events such as navigation through menus, opening dialogs and notifications, and accessing other private items, which are not pertinent to the goals of the recommendation system.

4.2.1 Dataset Overview

The dataset's key features include:

- **Category**: The type of log collected. It refers to the main action associated with the log.
- **Action**: The specific action conducted by the user, related to the category (like opening an application).
- **Action_info**: MWP internal application identifier. This is the unique identifier (ID) for MWP.
- **Department**: The department to which the user belongs (e.g., HR, IT, Marketing).
- **Device_type**: The type of device used by the user (e.g., desktop, mobile).
- **Strong_auth**: Indicates whether the user was authenticated using a stronger authentication method (e.g., OAuth 2.0).
- **Business_role**: The internal BMW role assigned to the user.
- **User_session**: A value object containing the user's session information when the log was recorded (encrypted).
- **Source**: The source application where the log was tracked (e.g., other applications hosted within myWorkplace).
- **Create_timestamp**: The timestamp indicating when the log was recorded.
- **Http_origin**: Indicates whether the log was recorded from a user inside the BMW intranet or outside (internet).
- **Browser_version_major**: The major version of the browser used by the user when the log was recorded.
- **Browser_name**: The name of the browser used when the log was created.
- **Os_name**: The operating system used by the user when the log was recorded.
- **Screen_width**: The screen width of the user's device when the log was recorded.
- **Screen_height**: The screen height of the user's device when the log was recorded.
- **Employee_status**: Indicates whether the user is an internal BMW Group employee or an external resource (e.g., outsourcing).
- **Timezone**: The timezone in which the log was recorded.
- **Timezone_offset**: The timezone offset when the log was recorded.
- **Locale**: The region or locale of the user when the log was recorded.
- **Feature_info**: Complements the action_info with additional details.
- **Prime_name**: Category of the application (BMW Internal Categories).

In order to understand better the data structure, a small portion of the generated dataset, showcasing user interactions with the applications, is present in the following Tables:

Table 4.1: Mocked-up OpenSearch index dataset: User Interactions (Part 1).

Category	Action	Action_info	Department	Device_type
Application	Open	User opened Application X	IT	Desktop
Widget	Open	User opened Widget Y	IT	Desktop
Application	Open	User opened Application Z	HR	Mobile
Application	Open	User opened Application X	HR	Mobile
Dashboard	Access	User accessed Dashboard 1	Marketing	Desktop
Application	Open	User opened Application X	Marketing	Desktop
Application	Open	User opened Application Y	HR	Desktop

Table 4.2: Mocked-up OpenSearch index dataset: User Interactions (Part 2).

Business_role	User_session	Source	Create_timestamp	Http_origin
Developer	Session123	MWP	2025-01-10 08:05:23	Intranet
Developer	Session124	MWP	2025-01-10 08:06:01	Intranet
HR Manager	Session125	MWP	2025-01-10 09:15:45	Internet
HR Manager	Session126	MWP	2025-01-10 09:17:01	Internet
Marketing Analyst	Session127	MWP	2025-01-10 10:45:10	Intranet
Marketing Analyst	Session128	MWP	2025-01-10 10:47:35	Intranet
HR Manager	Session129	MWP	2025-01-10 11:05:55	Intranet

Table 4.3: Mocked-up OpenSearch index dataset: User Interactions (Part 3).

Browser_version_major	Os_name	Screen_width	Screen_height
120	Windows	1920	1080
120	Windows	1920	1080
95	iOS	375	812
95	iOS	375	812
120	Windows	1920	1080
120	Windows	1920	1080
120	Windows	1920	1080

Table 4.4: Mocked-up OpenSearch index dataset: User Interactions (Part 4).

Strong_auth	Prime_name	locale	Browser_name	Feature_info
Yes	Category 1	DE	chrome	Additional details
Yes	Category 2	DE	chrome	Additional details
No	Category 1	DE	chrome	Additional details
No	Category 2	DE	chrome	Additional details
Yes	Category 4	PT	edge	Additional details
Yes	Category 1	ZA	edge	Additional details

All data presented in the four Tables above is artificially generated for illustrative purposes. It does not correspond to the actual data collected or utilized during model training and prediction phases.

4.2.2 Data Research Objectives

The objective of EDA is to gain a comprehensive understanding of the created dataset structure for this project and its distribution/relations between data. This step is crucial to identify data quality issues, such as missing values, inconsistent or anomalous values within columns, imbalanced classes and even potential outliers due to malformed log entries.

With that, the EDA has the following objectives:

- Verify the distribution and frequency of key features such as user actions, application, device types, and authentication methods and others.
- Investigate correlations between user behaviour and contextual attributes (e.g., department, session origin, or device configuration).
- Identify categorical variables and potential candidate features for one-hot encoding or embedding layers.
- Detect and address anomalies or inconsistencies that may influence model performance.
- Guide the feature selection process by evaluating feature importance heuristically.
- Validate whether the collected data provides sufficient variability and representativeness for training a recommender model.

These objectives helped inform subsequent preprocessing steps and ensured that the dataset was both technically sound and semantically aligned with the goals of automation and customization in the recommender system pipeline.

4.2.3 Data Preprocessing and Cleaning

As mentioned before, the initial step of EDA is the preprocessing step. The first action was to decompose the `create_timestamp` column into multiple temporal features that show way more granular insights, such as: `day_of_week`, `day_of_month`, `month`, `year`, `hour`, `minute`, and a categorical `time_interval`. The `time_interval` was defined as follows: *night* ($0 \leq$ hour < 6), morning ($6 \leq$ hour < 12), afternoon ($12 \leq$ hour < 16), and *evening* (otherwise). These created features provide valuable temporal context in order to understand the application usage patterns (particularly in relation to departmental behaviour).

Subsequently, several columns were deemed irrelevant for model training and removed, based on an initial heuristic evaluation of their predictive value. For instance, device-related information such as operating system, screen dimensions, and browser type are unlikely to influence application access within the same department. A user on a mobile device or desktop, for exa

Subsequently, several columns were deemed irrelevant for model training and removed, based on an initial heuristic evaluation of their predictive value. For instance, device-related information such as operating system, screen dimensions, and browser type are unlikely to influence application access within the same department. A user on a mobile device or desktop, for example, may still access the same applications, and these variables do not carry semantic weight for the recommendation task. mple, may still access the same applications, and these variables do not carry semantic weight for the recommendation task.

The most informative features retained at this stage were Department and `prime_name`, with others subject to further analysis to determine their impact on application selection.

The following columns were removed for the reasons outlined below:

- `create_timestamp`: redundant after decomposition into derived temporal features.
- `http_origin`: lacked variability and offered limited relevance to the prediction task.
- `browser_name`, `browser_version_major`: browser type and version showed no meaningful correlation with application or departmental behaviour.
- `os_name`, `screen_width`, `screen_height`: device-related features were excluded based on the assumption that application usage is platform-independent in this context.
- `timezone`, `time_zone_offset`, `locale`: time zone and regional settings were considered irrelevant given the globally consistent usage patterns and the inclusion of derived time features.
- `feature_info`: contained metadata not directly related to the act of opening an application, and therefore not valuable for modeling user intent.
- `source`: always set to MWP, offering no discriminative power.
- `user_session`: excluded due to privacy and GDPR concerns; aggregation was performed at the department level instead.
- `category`: typically contained the verb open-app, which did not contribute meaningful variance or correlate with application preferences across departments.
- `strong_auth`, `employee_status`: removed due to limited relevance for department-level application choice and to simplify the feature space.

Following the initial column elimination process, a further refinement was carried out by analyzing the proportion of missing values within each remaining column where if it was missing more than 20% data then that column was considered for removal. Moreover, `business_role` column, which had approximately 32% missing values across the dataset was deleted. Although the `business_role` is potentially informative (since it reflects the user's internal business role, which may influence their application usage) its incomplete coverage significantly undermined its utility. Concretely, the absence of values in this column was primarily due to the fact that some users that use MWP do not have a business role associated with them. As a result, the column was excluded from further analysis to prevent potential negative impact on model performance and generalization.

Additionally, rows with missing values in columns with lower levels of nullity (between 0% and 5%) were also examined. It was observed that 1.23% of the records lacked an associated Department value. Since every log entry is expected to be linked to a valid user or technical user (each of whom should be associated with a department) these missing values were treated as data inconsistencies. Therefore, all rows lacking a value in the Department column were removed from the dataset to maintain data integrity.

4.2.4 Exploratory Findings and Data Understanding

Following the previous preprocessing and cleaning steps, a series of different visualizations were conducted to explore the data and its patterns. These visual analyses aim to support early hypothesis formulation regarding the relationship between different features and application usage, particularly in the context of potential recommendation strategies.

Using interaction logs from the most recent six-month period, an initial inspection of the Department column revealed a total of 1,839 unique entries. However, this number does not correspond to the actual count of valid internal departments within the BMW organization. Upon further examination, it was noted that department identifiers within BMW follow a specific naming convention. A regular expression pattern was constructed based on this convention, which enabled the identification of 8 anomalous department entries that did not conform to the expected format. These entries were considered malformed, likely due to erroneous logging or system-level inconsistencies, and subsequently removed. After this filtering step, the dataset retained 1,831 valid and unique department values.

Regarding the `action_info` column, which identifies is the unique identifier for the applications, a total of 515 unique values were detected. Since these values represent internal application IDs from the MWP domain, they are important for the recommendation itself.

The first visualization conducted was precisely focused on the distribution of logs across defined time intervals (e.g., morning, afternoon, evening, night). This allowed for a temporal understanding of the data collected and its usage trends.

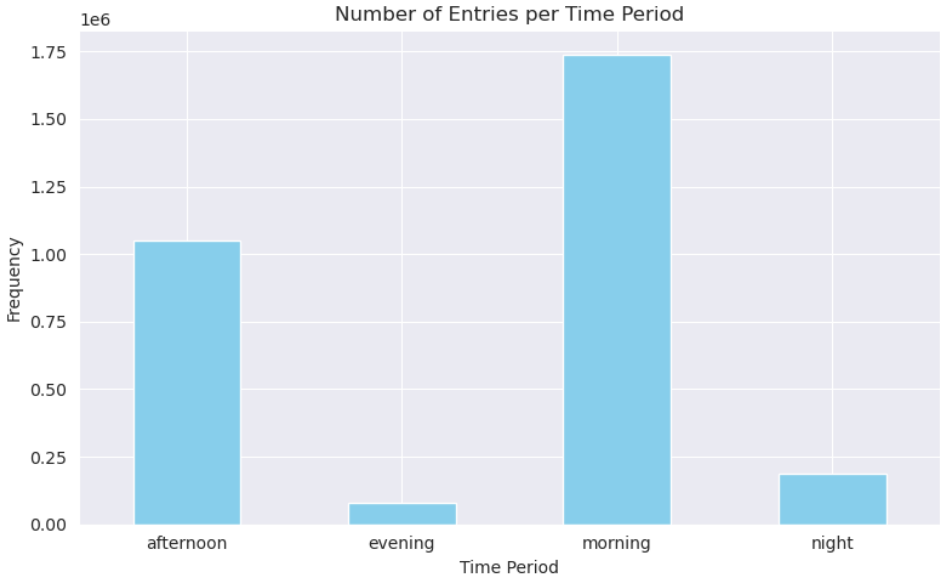

Figure 4.1: Distribution of user logs across time intervals.

Figure 4.1 illustrates the distribution of the entries across different time intervals through the day. As expected, due to the work hours, the majority of entries collected were generated during the morning, followed by the afternoon. This pattern aligns with standard working hours within the organization, where users are more likely to interact with applications during the morning. Additionally, this feature, supports the rationale for engineering time-based correlations, which may influence the RS's effectiveness by having the possibility to adapt to different usage periods. Moreover, outliers or unexpected usage during less active intervals

(e.g., night) could also show logging errors or unusual usage patterns that motivate for a further investigation.

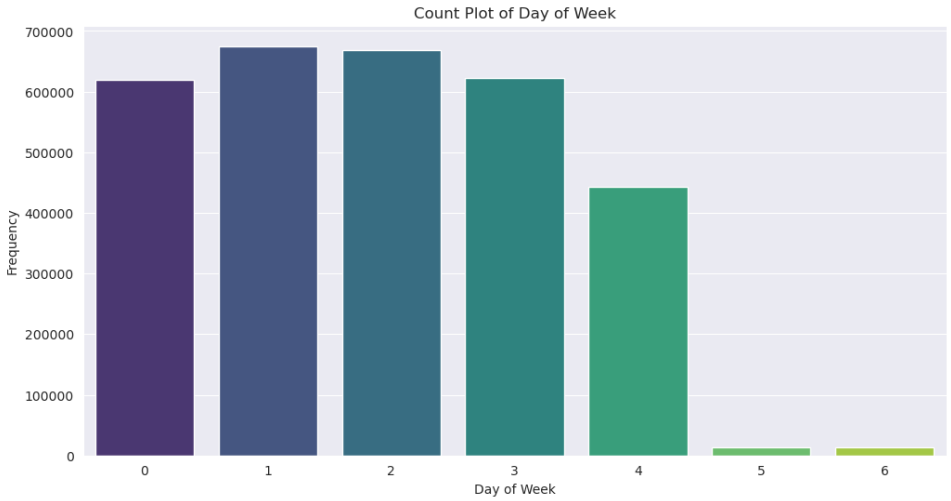

Figure 4.2: Distribution of user logs across day of the week.

Following the previous Figure, a second temporal analysis, this time more granular, was conducted to identify the distribution of user patterns across different days of the week. As present in Figure 4.2, interactions peak during Tuesdays and Wednesdays and have a lower value towards weekends.

This analysis reinforces the relevance of this feature feature created during preprocessing since it may serve as an important temporal indicator when modeling user behaviour or generating application recommendations having a good correlation with other features in the dataset.

Following the analysis of log distributions across different time intervals to identify peak activity periods, an additional graph was generated to explore the number of applications per `prime_name`. This visualization aimed to assess whether the dataset exhibited any imbalance in the categorization of application usage across the defined primes.

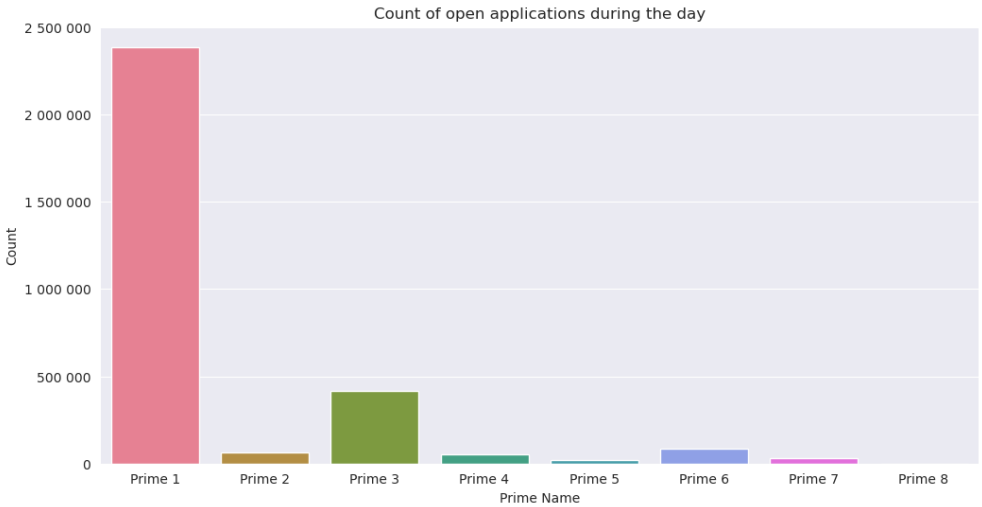

Figure 4.3: Distribution of entries per prime.

As illustrated in Figure 4.3, there is a noticeable imbalance in the distribution of data across the `prime_name` categories. This presents a significant challenge for the model training phase, as the goal is to develop a model capable of recommending applications across all available `prime_name` categories used within BMW. As seen in the Figure, the number of entries for each corresponding class is: `prime 1` has 2,386,135 entries, `prime 2` has 62,503, `prime 3` has 417,886, `prime 4` has 50,546, `prime 5` has 21,507, `prime 6` has 85,899, `prime 7` has 29,887, and `prime 8` has only 947 entries. This shows how imbalance the current `primes` are.

Given this, it is expected to affect downstream model behaviour (e.g., popularity bias and uneven performance across departments). With that said, Chapter 5 will investigate various possibilities to mitigate this problem.

To gain a deeper understanding of the `prime_name` distribution, since there is a big imbalance problem, two additional visualizations were generated, one showing the relationship between application and `prime_name`, and another between department and `prime_name`. These charts help understand whether the distribution of applications across different `prime_name` categories is balanced or not and whether the departments that accessed the MWP system over the past six months are evenly represented across the various `prime_name` classes.

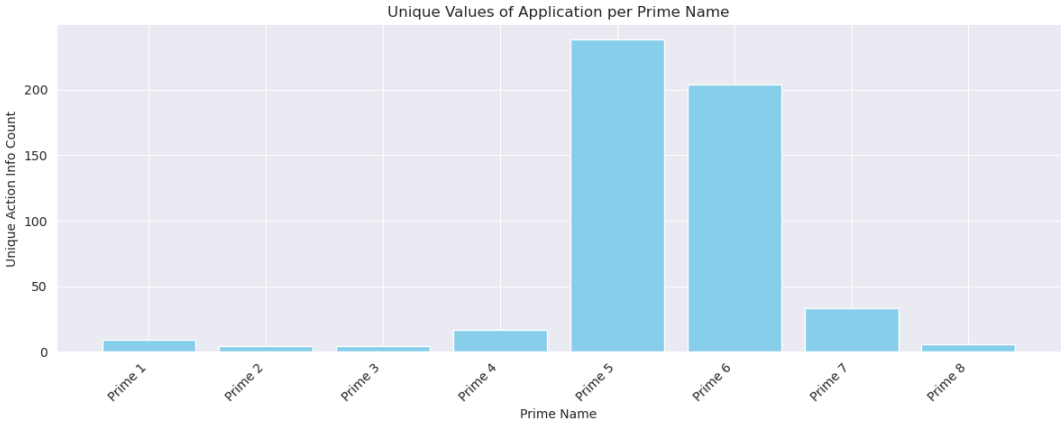

Figure 4.4: Distribution of unique values of application per prime.

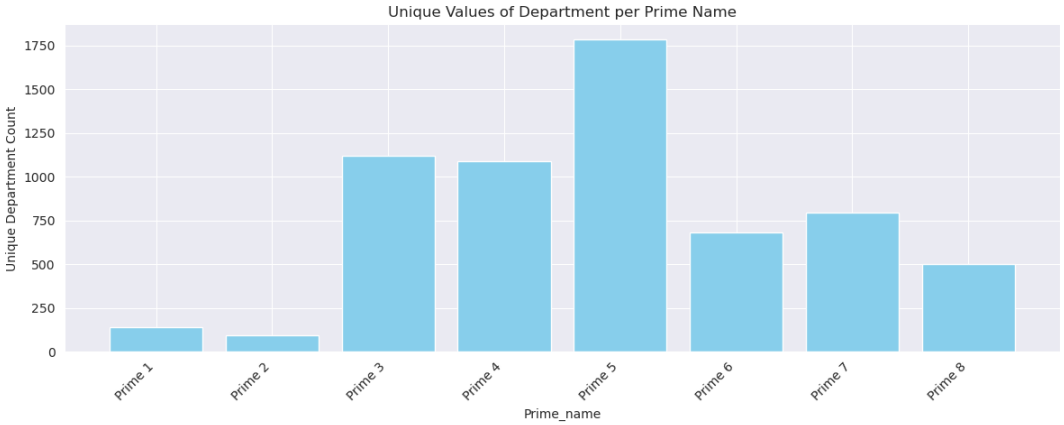

Figure 4.5: Distribution of unique values of department per prime.

As shown in Figures 4.4 and 4.5, the `prime_5` present in these Figures contains a significantly higher number of entries compared to the other categories, confirming the previously identified class imbalance. Since the real prime category label was omitted we are using "prime 1,2,3.." so the `prime_5` here present would reflect the `prime_1` from the Figure 4.3. This imbalance may introduce challenges during model training and validation by biasing the system toward the most frequently used categories. Nonetheless, other `prime_name` categories such as `prime_6`, `prime_4`, and `prime_3` also present a meaningful volume of interactions, which can still support effective learning if appropriate techniques are applied.

At the same time, departments or applications with very few recorded interactions represent potential cold-start cases. This is a well-documented challenge in recommender systems, as models trained primarily on historical data often struggle to provide accurate recommendations for users or items with limited representation. Although cold-start cannot be fully resolved at the EDA stage, acknowledging its presence underscores the need for model design and evaluation strategies that remain robust in such scenarios.

The distribution of values within the `device_type` column was also visualized to evaluate its potential relevance as a feature for model training.

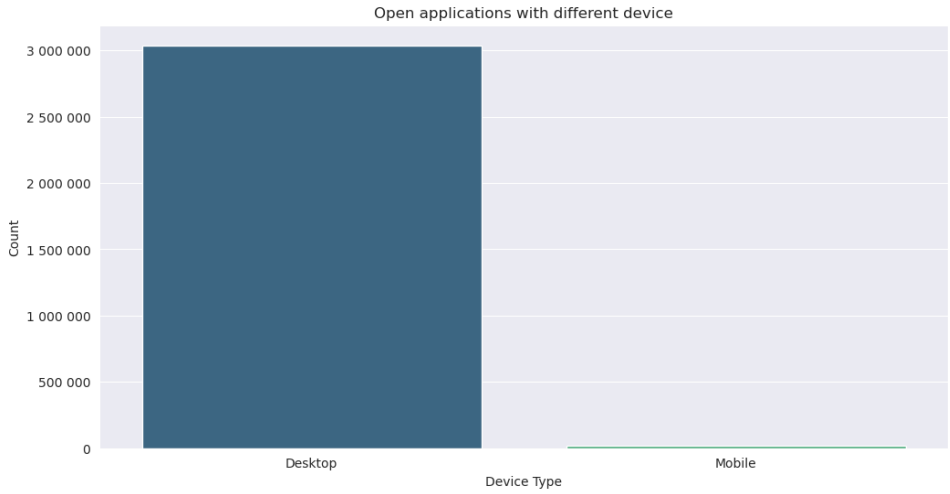

Figure 4.6: Distribution of entries per device type.

As shown in Figure 4.6, desktop devices dominate the `device_type` distribution, leaving little variability. Given this imbalance, the column was excluded from training, as it adds minimal predictive value.

A related analysis was conducted for the `strong_auth` and `employee_status` columns to examine their value distributions and assess their potential utility as features in the model.

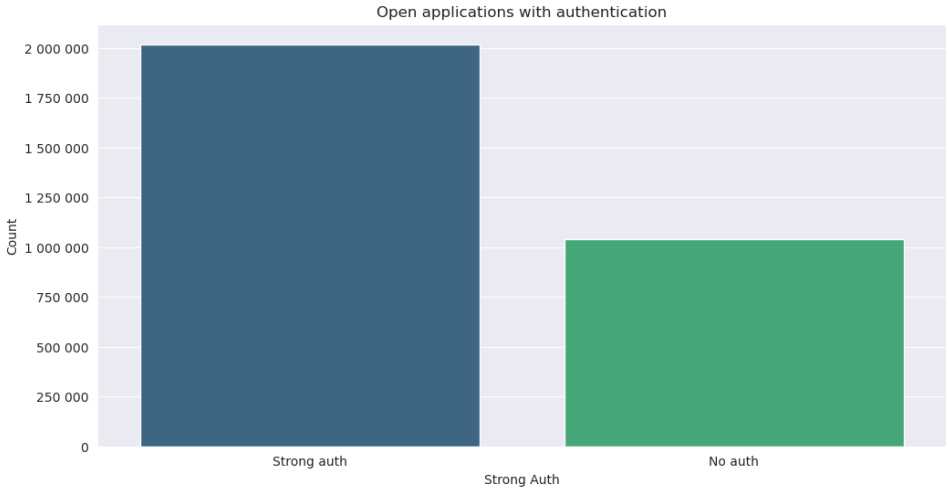

Figure 4.7: Distribution of entries per strong auth.

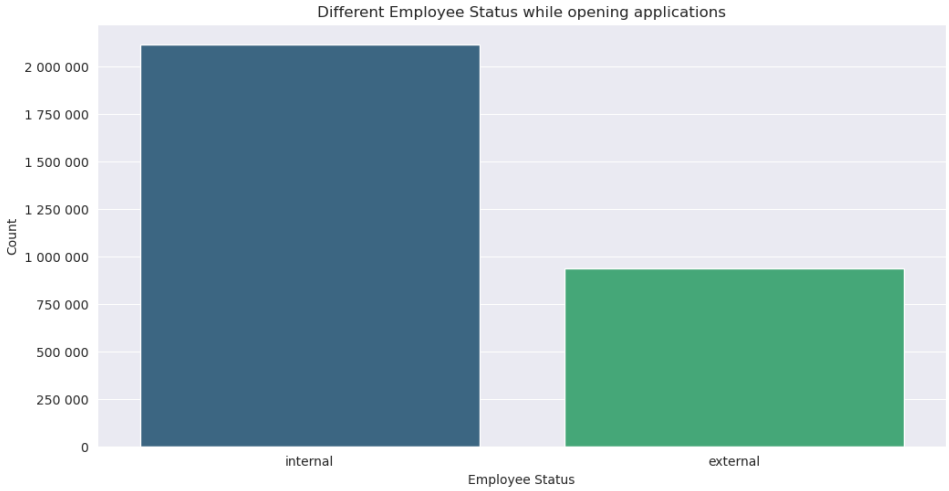

Figure 4.8: Distribution of entries per employee status.

As illustrated in both charts (Figures 4.7 and 4.8), the distributions are reasonably balanced. However, exploratory tests and the correlation study indicate that `strong_auth` carries no material signal for application choice once department and `prime_name` are accounted for. It is therefore excluded from the modeling feature set and retained only in the raw logs for monitoring or access-policy auditing. Meanwhile, `employee_status` shows a mild association with department but did not yield consistent gains in offline metrics it is treated as optional metadata and is not used as a predictor in the first iteration. This keeps the feature space minimal and aligned with operational semantics, reducing noise and overfitting risk.

It is also essential to assess whether the department and application columns exhibit class imbalance. The following visualizations were generated.

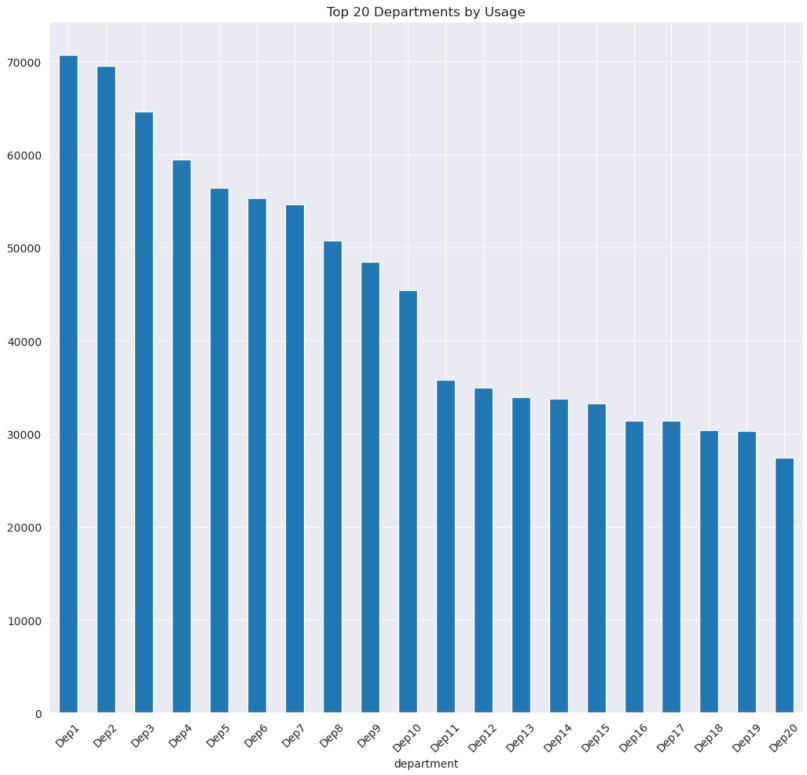

Figure 4.9: Distribution of the top 20 departments with more entries.

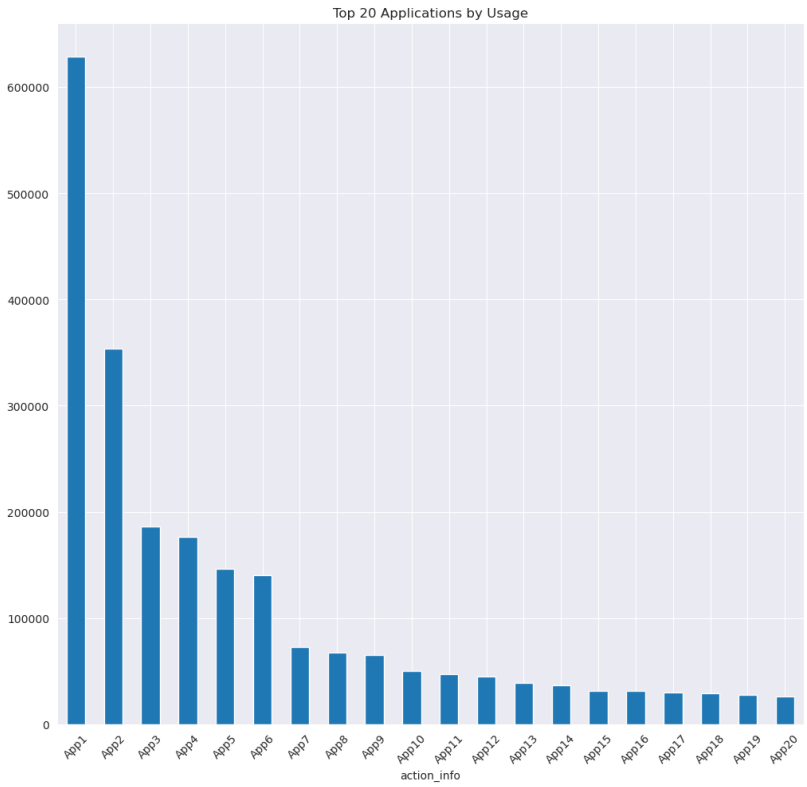

Figure 4.10: Distribution of the top 20 applications with more entries.

Figures 4.9 and 4.10 reveal a noticeable imbalance, particularly among the top two departments and applications. This skewed distribution may lead the model to overfit to these dominant classes, thereby reducing its generalization ability across underrepresented departments and applications.

At the same time, it is important to note that these imbalances, while significant, are not unexpected. Within the organizational context of BMW, certain departments naturally generate more activity logs due to their size and operational responsibilities, and certain applications or application categories are inherently more widely used. Therefore, artificially rebalancing the dataset (e.g., via oversampling or weighting) would risk distorting the distribution of real-world behaviour. Instead, these characteristics are treated as domain-driven properties that the recommendation models must account for, particularly with respect to mitigating excessive popularity bias while still maintaining realistic usage patterns.

4.2.5 Correlation Study

Before proceeding to the next phases, a correlation study was conducted to understand the relationships between columns and to identify possible patterns within the dataset. Given that the dataset contains both categorical and numerical variables, it was conducted categorical-categorical associations with chi-square tests and report Cramér's V as an effect size. Methodologically, where a numeric variable is compared across categories, it was used ANOVA. (Agresti, 2019).

Table 4.5: Cramer Correlation Matrix (Part I)

	action	action_info	department	time_period
action	1.000	NaN	NaN	NaN
action_info	NaN	1.000	0.188	0.116
department	NaN	0.188	1.000	0.323
time_period	NaN	0.116	0.323	1.000
employee_status	NaN	0.463	0.645	0.047
prime_name	NaN	1.000	0.601	0.057

Table 4.6: Cramer Correlation Matrix (Part II)

	employee_status	prime_name
action	NaN	NaN
action_info	0.463	1.000
department	0.645	0.601
time_period	0.047	0.057
employee_status	1.000	0.263
prime_name	0.263	1.000

Table 4.7: ANOVA Results (Part I)

Categorical	Numerical	F	p
time_period	device_type	4.53×10^3	0.000000e+00
time_period	strong_auth	8.95×10^3	0.000000e+00
time_period	day_of_week	2.28×10^3	0.000000e+00
time_period	day_of_month	5.21×10^1	1.089674e-33
time_period	month_of_year	1.39×10^2	2.375905e-90
time_period	year	1.32×10^2	8.287978e-86
time_period	hour_of_day	4.62×10^6	0.000000e+00
time_period	minute_of_hour	6.12×10^3	0.000000e+00
employee_status	device_type	7.27×10^3	0.000000e+00
employee_status	strong_auth	2.42×10^4	0.000000e+00
employee_status	day_of_week	4.19×10^1	9.665832e-11

Table 4.8: ANOVA Results (Part II)

Categorical	Numerical	F	p
employee_status	day_of_month	1.58×10^2	3.380615e-36
employee_status	month_of_year	8.32×10^3	0.000000e+00
employee_status	year	8.03×10^3	0.000000e+00
employee_status	hour_of_day	4.80×10^3	0.000000e+00
employee_status	minute_of_hour	2.60×10^1	3.461256e-07
prime_name	device_type	6.52×10^3	0.000000e+00
prime_name	strong_auth	9.92×10^3	0.000000e+00
prime_name	day_of_week	1.24×10^2	3.718056e-183
prime_name	day_of_month	6.64×10^1	2.633864e-96
prime_name	month_of_year	5.38×10^2	0.000000e+00
prime_name	year	5.30×10^2	0.000000e+00
prime_name	hour_of_day	2.14×10^3	0.000000e+00
prime_name	minute_of_hour	1.12×10^1	2.558428e-14

As seen in Tables 4.7 and 4.8 the F column represents the F-statistic, which is a indicator of the ratio of variance explained by the model relative to unexplained variance while the p column represents the p-value, which directly tests the statistical significance of the conducted relationship. Smaller values (typically $p < 0.05$) indicate stronger evidence against the null hypothesis.

The correlation analysis conducted reveals notable associations between several categorical variables, such as between `employee_status` and `department`, as well as strong correlations involving `prime_name`. The ANOVA results indicate statistically significant differences across categories for numerical variables related to time and device usage.

However, the temporal columns, such as `hour_of_day`, `day_of_week`, and `month_of_year` do not exhibit a direct impact on the selection of applications or department associations in this dataset. In this context, these features may introduce noise or unnecessary complexity to the model, potentially reducing its predictive performance. Therefore, these temporal variables will be removed from the dataset during preprocessing.

Alternatively, if temporal patterns become relevant in future improvements (for example in case there is a necessity to detect peak usage times), these features could be engineered in a different way or modeled separately.

The anomalies and class imbalances observed here (e.g., highly dominant primes, long-tail departments) inform two downstream choices: (i) the evaluation design in Chapter 5, which reports both micro and macro ranking metrics to surface cross-department equity, and (ii) the operational data checks in Chapter 6, which translate these EDA findings into schema validations and distribution monitors. In order to maintain the defined focus for the current Chapter operational thresholds will not be mentioned here.

4.2.6 Feature Engineering

After concluding the data understanding phase and analyzing the graph results, analysis showed that certain columns needed to be removed. Specifically, the `device_type` column, all temporal variables, and the action column were dropped. The action column was removed because the dataset was already filtered to include only the "open-app" action, rendering this column constant and thus uninformative for the model.

After removing these columns, only three columns remained: `action_info`, `department`, and `prime_name`. Since all three are categorical, encoding strategies suited to their cardinalities are required. Given that department and application contain many unique values (1839 and 515 respectively), appropriate methods that can handle high cardinality will be necessary.

The dataset contains several categorical variables with varying cardinalities, which required tailored encoding approaches to prepare the data for model training.

Target variable (`action_info`). The target comprises 515 unique application identifiers. For downstream modeling, label encoding is a natural baseline representation to index classes compactly. Moreover, alternative item representations (e.g., hashed identifiers or learned embeddings) can be considered and are evaluated in Chapter 5.

Department Encoding. The `department` column has high cardinality, with 1,839 unique values. A two-step encoding procedure was applied:

- **Label Encoding**: Each department was assigned a unique integer label.
- **Target Encoding (candidate).** A smoothed target-mean encoding is considered to capture department–target association while reducing variance for low-frequency departments. The effectiveness is evaluated in Chapter 5.

Departments appearing only in the validation set were encoded with the global mean target value to avoid introducing bias from unseen categories.

The `prime_name` column, having only eight categories, was encoded using one-hot encoding. This creates binary indicator variables for each prime category, enabling the model to learn distinct effects without imposing an ordinal relationship.

Although `strong_auth` showed a reasonably balanced distribution during EDA, it did not exhibit discriminative value for department-level application choice and was therefore excluded to reduce noise and simplify the feature space.

Based on the analyses above, the following columns are retained as primary candidates for modeling in Chapter 5:

- `department` (high cardinality; categorical),
- `prime_name` (8 categories),
- `action_info` (target; 515 applications).

This combined encoding approach leverages the advantages of different encoding methods adapted to each column's characteristics, providing an informative representation for subsequent model training. The following Table summarizes the final features and the number of unique values.

Table 4.9: Summary of Encoding Methods and Feature Dimensions

Column	**Unique Values**	**Encoding Method**
`action_info` (target)	515	Label Encoding + Unseen class handling
`department`	1,839	Label Encoding + Target Encoding
`prime_name`	8	One-Hot Encoding

No missing values were introduced after encoding, and the training and validation sets maintained consistent feature dimensions, meaning it was ready for model development.

It should be noted that while the current chapter focuses exclusively on the exploratory analysis of the dataset. The identification of imbalances, anomalies, and malformed indexes guided early preprocessing and feature selection decisions. For instance, training the initial recommender on all prime categories while considering category specific models if performance proved insufficient, and discarding device-related and temporal variables that showed no discriminative value. At the same time, these findings informed the design of MLOps components such as structural and statistical drift detection, as well as automated data cleaning procedures, which build directly upon the observations made during EDA.

Furthermore, the exploratory analysis has direct implications for the evaluation methodology. The presence of highly dominant classes and underrepresented departments requires validation strategies that move beyond aggregate accuracy. As discussed in Chapter 5, stratified metrics such as Precision@K and Normalized Discounted Cumulative Gain (NDCG) will be employed to ensure that the recommender is not only accurate in aggregate but also fair and robust across heterogeneous usage patterns.

Chapter 5

Model Development and Evaluation

This chapter describes the development and evaluation of content-based recommenders systems (RS) for application suggestion across departments. Currently, the available inputs are limited to department (who defines the user internal BMW department), `prime_name` (application category following BMW internal categories), and `action_info` (application unique identifier inside the MWP landscape). However, departments are significantly different between each other and also in application usage patterns, even though adjacent department codes (e.g., AA-71 vs. AA-72) will have unrelated business logic, and, sometimes, departments with different prefixes share more in common than those with similar names (e.g., AA-71 and XX-99). All of this combined with the absence of explicit feedback directly from the users/departments will motivate an initial phase focused on content only methods.

5.1 Content-Based Models

This section will introduce the trained models considered in the initial phase, together with the evaluation method and metrics used. It emphasizes interpretable content-based signals that recommends the applications per department, having collaborative components to a later phase when explicit feedback becomes available.

5.1.1 TF-IDF + Cosine (Dept-Category)

The first evaluated model is a TF-IDF + Cosine approach. Department-category usage profiles are constructed and stabilized with Laplace smoothing to account for small departments, preventing zero-probability estimates when few or no interactions exist. Cosine similarity is then used to identify peer departments. Although departments differ substantially in business logic and usage, Cosine similarity over category profiles serves as a first-order signal of shared patterns. To reduce spurious matches in this coarse feature space (eight categories), an overlap-shrinkage term down-weights pairs with very small intersections.

Application scores are obtained by aggregating usage from the most similar departments (weighted by similarity), penalizing applications already used by the target department, and re-ranking with a diversification step.[1] This diversification ensures that the final recommendation list balances relevance with variety, reducing redundancy among recommended items and avoiding over-concentration on a narrow subset of applications. To mitigate the dominance of ubiquitous categories, a TF-IDF variant of the profiles is also considered:

[1] Diversification is a standard practice in recommender systems to balance accuracy with coverage and novelty, ensuring that the final recommendation list is not dominated by very similar or redundant items.

$$\text{tfidf}(d, c) = \frac{\text{count}(d, c) + \alpha}{\sum_{c'}(\text{count}(d, c') + \alpha)} \times \left(\log \frac{1+N}{1+\text{df}(c)} + 1\right),$$

where N is the number of departments and $\text{df}(c)$ counts departments using category c (Manning et al., 2008).

5.1.2 Category-Mixture (Pure Content)

The mixture model avoids explicit neighbor matching and scores each app by marginalizing over categories:

$$\text{score}(i \mid d) = \sum_{c} P(c \mid d)\, P(i \mid c).$$

Both $P(c \mid d)$ and $P(i \mid c)$ use Laplace smoothing, optionally, IDF^{γ} is applied to de-emphasize ubiquitous categories. By removing explicit neighbor matching and working only with category distributions, this approach is more stable under sparse data conditions, though personalization remains limited by the coarse category space.

5.1.3 Simple Baselines

Two baselines calibrate expectations and quantify absolute lift: (i) Global Popularity recommends the most-used applications overall (with train-seen filtering), and (ii) Category Popularity recommends the most popular apps within a department's top categories. These baselines serve as reference points: Global Popularity captures the intuition that widely adopted applications may also be useful across departments, while Category Popularity adds a minimal degree of personalization by conditioning on a department's main categories. As such, they provide lower-bound performance estimates (simple to compute and easy to interpret) against which the gains of more sophisticated models can be clearly measured in the evaluation.

5.1.4 Content-Only Hybrid

A content-only hybrid combines a neighbor signal (TF-IDF Cosine) with the mixture score, followed by diversification in re-ranking:

$$s_{\text{hyb}}(i \mid d) \;=\; \alpha\, s_{\text{nn}}(i \mid d) \;+\; (1 - \alpha)\, s_{\text{mix}}(i \mid d).$$

Here, α controls the blending weight, γ (in IDF^{γ}) controls category informativeness; and λ controls the relevance, diversity balance in re-ranking. Diversification reorders the top-K list so that recommended items are not overly redundant, for example, preventing the system from suggesting several very similar applications from the same category. This ensures a better trade-off between accuracy (recommending relevant items) and coverage (surfacing a wider variety of the catalog), which is a standard practice in modern recommenders.

5.1.5 Hybrid Approach

The design targets a two-stage architecture. The first stage is candidate generation using content signal cosine similarity with TF-IDF, category mixture, or embeddings and later collaborative filtering such as co-visitation and matrix factorization. The second stage is

re-ranking using a learning-to-rank model once feedback is available, or a weighted ensemble with business constraints such as novelty and coverage until then. The current content-only hybrid serves as a stepping stone toward this pipeline and is evaluated under the same dataset and protocol described in Section 5.2.2.

5.1.6 Random Baseline

For reference, a uniform random recommender over the application catalog is included. For each department, the candidate set comprises all applications observed during training, excluding those already interacted with by that department (train-seen filtering). Items are sampled without replacement to produce the top-K list. No features (such as categories) are used, and a fixed random seed ensures reproducibility.

This baseline anchors performance against chance and quantifies the absolute gains of the structured content-based models. Its main drawback is the high variability of results which means that without fixing a seed, metrics such as Precision or Recall fluctuate unpredictably across runs. Moreover, even when stabilized with a seed, the random baseline provides no meaningful personalization or interpretability, serving only as a minimal reference point.

5.2 Evaluation of Models

This section presents the evaluation methodology and results for the developed recommendation models. The goal is to establish a consistent and leakage-safe protocol, compare candidate algorithms under standardized metrics, and identify the most promising configurations for deployment. The discussion is organized as follows: Section 5.2.2 describes the evaluation protocol and metrics; Section 5.2.3 details the hyperparameter optimization strategy; Section 5.2.4 reports the main results and discussion with, also, plots to illustrate the performance.

5.2.1 Evaluation Metrics and Averaging

Ranking quality is evaluated using standard information-retrieval metrics, with both micro and macro aggregates reported having each metric, a concise definition provided. Coverage@K is also tracked to measure how widely recommendations span the catalog, reflecting diversity beyond popular items. Following common practice, micro-NDCG@10 is treated as the headline metric, with macro scores reported to surface cross-department equity (Bauer et al., 2024; W. X. Zhao et al., 2023).

Let G_d denote the set of distinct held-out applications (ground truth) for department d in the test split, and let $R_d@K$ denote the top-K recommendations for d after filtering apps seen for d in training.

Precision@K and Recall@K

Recommendation quality is summarized with two complementary metrics. Precision@K captures how accurate the top-K list is (the share of the K recommended items that are relevant) while Recall@K captures how complete it is (the share of all relevant items that appear within the top-K). The results of this are reported per department, following common practice in top-N evaluation (W. X. Zhao et al., 2023).

$$\text{Precision@K}(d) = \frac{|G_d \cap R_d@K|}{K}, \qquad \text{Recall@K}(d) = \frac{|G_d \cap R_d@K|}{|G_d|}.$$

- G_d is the list of relevant items for each department d,
- $R_d@K$ is the set of top-K items recommended to department d,
- $|\cdot|$ denotes the set cardinality (number of elements).

Average Precision@K and Mean Average Precision@K

Average Precision@K (AP@K) builds on top of Precision@K by rewarding recommendation lists that rank relevant items higher. Rather than simply counting how many relevant items appear within the top-K, AP@K computes the precision at each position where a relevant item occurs and then averages these values. As a result, relevant items that appear earlier in the list contribute more, reflecting the intuition that users are more likely to examine higher-ranked recommendations (Bauer et al., 2024).

Formally, for a given department d:

$$\text{AP@K}(d) = \frac{1}{\min(|G_d|, K)} \sum_{i=1}^{K} \text{Precision@i}(d)\, r_i,$$

where:

- G_d is the set of relevant items for department d,
- $r_i \in \{0, 1\}$ indicates whether the item at position i in the ranking is relevant,
- Precision@i(d) is the precision of the top-i list,
- the denominator $\min(|G_d|, K)$ normalizes by the smaller of the number of relevant items or K.

Mean Average Precision@K (MAP@K) aggregates this measure across all departments, providing a single global metric:

$$\text{MAP@K} = \frac{1}{|D|} \sum_{d \in D} \text{AP@K}(d),$$

where D is the set of departments. MAP@K thus captures both correctness and ranking quality across the entire evaluation set.

If multiple departments are evaluated, MAP@5 would be the average of all their AP@5 scores.

This formulation highlights how AP@K and MAP@K do not just measure whether relevant items appear in the recommendation list, but also reward systems that prioritize placing them closer to the top.

Normalized Discounted Cumulative Gain@K

Normalized Discounted Cumulative Gain@K (NDCG@K) is a ranking metric that compares the obtained recommendation list to an ideal ordering in which all relevant items are ranked before any irrelevant ones. Unlike Precision@K or Recall@K, NDCG@K takes into account

not only whether relevant items appear, but also where they appear in the ranking, by applying a logarithmic discount to lower-ranked positions. This reflects the realistic assumption that users pay more attention to items ranked higher (W. X. Zhao et al., 2023).

For a given department d, first compute the Discounted Cumulative Gain (DCG) as:

$$\text{DCG@K}(d) = \sum_{i=1}^{K} \frac{r_i}{\log_2(i+1)},$$

where $r_i \in \{0, 1\}$ is a binary relevance indicator for the item at position i (1 if relevant, 0 otherwise).[2] Then normalize by the Ideal DCG (IDCG), which is the maximum possible DCG obtained by ranking the relevant items (or, in the graded case, the highest-gain items) first:

$$\text{NDCG@K}(d) = \frac{\text{DCG@K}(d)}{\text{IDCG@K}(d)}.$$

Coverage@K

Coverage@K measures how broadly the recommender system uses the available item catalog. While accuracy-oriented metrics (Precision, Recall, MAP, NDCG) evaluate recommendation quality, Coverage@K evaluates diversity across the catalog.

A system with low coverage may over-recommend only the most popular items, whereas high coverage indicates that recommendations explore more of the catalog, exposing users to a wider variety of items. In practice, Coverage@K may also vary temporally, as application usage patterns shift seasonally or new applications enter or leave the catalog. This makes it an especially valuable complementary metric to track over time in addition to static evaluation (Kaminskas and Bridge, 2016).

Formally:

$$\text{Coverage@K} = \frac{|\bigcup_{d \in D} R_d@K|}{|\mathcal{I}|},$$

where:

- $R_d@K$ is the top-K recommendation list for department d,
- $\mathcal{I}$ is the full set of items in the training catalog,
- the numerator counts how many unique items appear across all top-K lists.

High coverage is desirable in many recommendation settings, as it supports catalog exploration, reduces popularity bias, and increases fairness of item exposure.

Macro vs. Micro Averaging

When reporting evaluation metrics across multiple departments, it is important to distinguish between macro and micro averaging. Both approaches aggregate per-department scores into a single number, but they differ in how departments are weighted.

[2] More generally, NDCG supports graded relevance by replacing r_i with a non-negative gain $g_i \geq 0$, e.g., relevance levels between 0 and 3. The binary formulation is adopted in this project.

Macro averaging treats all departments equally, regardless of their size or activity level:

$$\text{Macro}(\cdot) = \frac{1}{|D|} \sum_{d \in D} (\cdot)(d).$$

Here, $(\cdot)(d)$ denotes the value of a given metric (e.g., Precision@K, Recall@K, NDCG@K) for department d. This formula gives the same importance to every department, even if some are much smaller or less active than others. As such, macro averaging highlights fairness across departments.

Micro averaging weights departments by the number of relevant items they contain, effectively favoring larger or more active departments:

$$\text{Micro}(\cdot) = \frac{\sum_{d \in D} |G_d| \, (\cdot)(d)}{\sum_{d \in D} |G_d|},$$

where $|G_d|$ is the number of relevant items for department d. This formula captures the overall utility of the recommender system when deployed in the organization, as it reflects the aggregate performance over all items, not just per-department averages (Henriques and Pinto, 2023).

Due to departments in the dataset highly heterogeneity and the substantially difference in size and activity, both macro and micro averaged results are reported. Micro averaging reflects overall utility for the organization, whereas macro averaging highlights performance disparities across departments. Following common practice, micro-NDCG@10 is used as the headline metric, with macro results reported alongside for completeness.

5.2.2 Evaluation Protocol and Metrics

On 3,055,310 interaction rows (515 apps, 1,839 departments, 8 categories), it was used a per-department split yielding 2,444,328 train and 610,342 test records, with 1,531 departments common to both. Moreover, per-department record splits are adopted so that train and test share departments in order to avoid leakage, applications present in a department's training interactions are filtered from recommendation lists during evaluation.

5.2.3 Hyperparameter Optimization

A hyperparameter grid search is employed to explore model configurations with the objective of maximizing ranking quality while maintaining a minimum level of catalog coverage. Without a coverage safeguard, configurations may converge to accurate but narrow solutions that compress exposure.

Execution is parallelized across multiple threads so that several configurations can be evaluated concurrently. In addition, early stopping prunes branches that fall clearly below target thresholds according to micro-NDCG@10. Accordingly, a coarse-to-fine procedure is adopted: an initial broad scan identifies promising regions of the space, followed by finer sweeps on sensitive parameters, namely the blending weight α, the category informativeness exponent γ in IDF^{γ}, the diversification strength λ, and the neighbor pool size. Meanwhile, parameters that proved inert under filtered evaluation (e.g., the seen-item penalty α_{seen}) are fixed to reduce the dimensionality of the search.

A coverage constraint enforces Coverage@10 ≥ 0.30, ensuring that high-ranking configurations with narrow exposure are discarded. In the presence of effective ties, defined as an absolute difference within ± 0.005 in micro-NDCG@10, tie-breaking first favors higher Coverage@10 and, secondarily, higher macro-Recall@10. This policy is aligned with the promotion thresholds discussed in Chapter 6.2.3.

By the grid being representative rather than exhaustive, expanding it would substantially increase runtime with limited expected gains given the observed flat regions of the search space in sensitive regions.Empirically, on the evaluation hardware used, the sweep completed in approximately six hours.

In this way, the hyperparameterization process balanced thoroughness with efficiency, producing robust model candidates without incurring prohibitive computational cost.

The constrained grid for each family is summarized in Table 5.1.

Table 5.1: Hyperparameters (grid values and selected best) per model.

Model	**Hyperparameter**	**Grid values**	**Best value**
Category Popularity	C	$\{1, 2, 3, 4, 5\}$	2
Category Popularity	Laplace	$\{0.5, 1.0, 2.0\}$	0.5
Dept–Category Cosine (neighbor)	use_tfidf	{`False`, `True`}	`False`
Dept–Category Cosine (neighbor)	m	$\{10, 20, 30, 50\}$	20
Dept–Category Cosine (neighbor)	shrink	$\{0, 3, 5\}$	3
Dept–Category Cosine (neighbor)	α_{seen}	$\{0.3, 0.5, 0.7\}$	0.5
Dept–Category Cosine (neighbor)	λ	$\{0.7, 0.8, 0.9\}$	0.9
Dept–Category Cosine (neighbor)	cand_mult	$\{2, 3, 4\}$	3
Category Mixture	γ	$\{0.0, 0.3, 0.6, 0.9\}$	0.0
Category Mixture	Laplace	$\{0.5, 1.0, 2.0\}$	1.0
Hybrid (neighbor + mixture + diversification)	Mixture γ	$\{0.3, 0.6, 0.9\}$	0.3
Hybrid (neighbor + mixture + diversification)	blend α	$\{0.5, 0.65, 0.8\}$	0.65
Hybrid (neighbor + mixture + diversification)	λ	$\{0.7, 0.8, 0.9\}$	0.8
Hybrid (neighbor + mixture + diversification)	cand_mult	$\{2, 3, 4\}$	2

Table 5.1 highlights how the Hybrid and Dept–Category Cosine models converge to very similar optimal settings, which anticipates the near-tied performance observed in the evaluation results.

Using micro-NDCG@10 as selector (with Coverage@10 as a guardrail), the Hybrid slightly edges the Cosine on macro NDCG@10 (0.0438 vs. 0.0437) and macro MAP@10 (0.0191 vs. 0.0190), while micro metrics are identical and Cosine yields marginally higher coverage (0.4355 vs. 0.4316). Varying α_{seen} did not affect the selector or coverage, consistent with train-seen filtering at evaluation time.

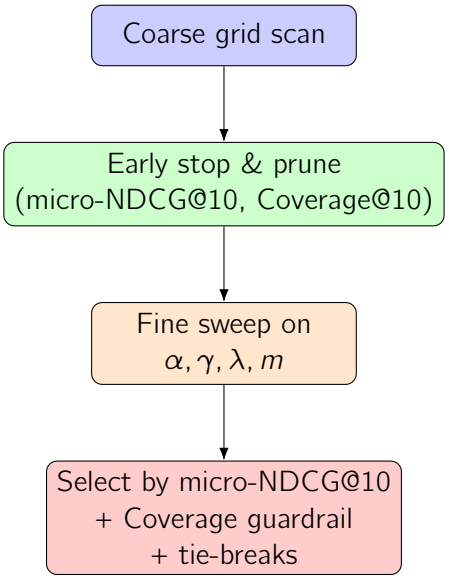

Figure 5.1: Hyperparameter search flow.

5.2.4 Results and Discussion

After hyperparameter tuning selects the best configuration for each model, six recommenders are compared:

1. **Global Popularity**,
2. **Category Popularity** (best: top–2 categories, Laplace = 0.5),
3. **Dept–Category Cosine** (best: `use_tfidf`=False, m=20, shrink= 3, α_{seen}=0.5, λ=0.9, `cand_mult`= 3),
4. **Category Mixture** (best: γ=0.0, Laplace = 1.0),
5. **Hybrid** (best: α=0.65, γ=0.3, λ=0.8, `cand_mult`= 2),
6. **Random Baseline**.

Table 5.2: Macro metrics @K=10 across models.

Model	Macro P@10	Macro R@10	Macro MAP@10	Macro NDCG@10
Random Baseline	0.0030	0.0013	0.0008	0.0028
Global Popularity	0.0217	0.0242	0.0120	0.0294
Category Popularity	0.0202	0.0192	0.0114	0.0280
Dept–Category Cosine	0.0308	0.0390	0.0190	0.0437
Category Mixture	0.0238	0.0255	0.0134	0.0330
Hybrid	0.0308	0.0390	0.0191	0.0438

Table 5.3: Micro metrics @K=10 across models.

Model	Micro P@10	Micro R@10	Micro MAP@10
Random Baseline	0.0074	0.0018	0.0020
Global Popularity	0.0297	0.0127	0.0116
Category Popularity	0.0309	0.0118	0.0132
Dept–Category Cosine	0.0446	0.0180	0.0168
Category Mixture	0.0359	0.0140	0.0150
Hybrid	0.0446	0.0180	0.0168

Table 5.4: Ranking quality and coverage @K=10 across models.

Model	Micro NDCG@10	Coverage@10
Random Baseline	0.0070	1.0000
Global Popularity	0.0334	0.0977
Category Popularity	0.0363	0.2871
Dept–Category Cosine	0.0486	0.4355
Category Mixture	0.0415	0.2461
Hybrid	0.0486	0.4316

Tables 5.2, 5.3 and 5.4 summarize performance under the selected metrics. As expected, results are modest overall: the heterogeneous nature of departments and the absence of user-level context limit achievable accuracy. Still, clear patterns emerge. The **Dept–Category Cosine** and **Hybrid** models consistently dominate across ranking metrics, while **Global Popularity** performs worst (as anticipated for such a naive baseline at department level). **Category Popularity** performs slightly better than **Global Popularity** thanks to its category-level filtering, but it lags behind **Mixture** in terms of recall and stability. The inclusion of the **Random Baseline** clarifies the absolute lift: even the simplest popularity-based models outperform chance by a wide margin.

The **Random** baseline attains Coverage@10 close to 1.0 because it samples uniformly over the item catalog, spreading recommendations broadly across applications, despite this wide item exposure, its ranking quality (e.g., Precision@K/NDCG@K) remains poor.

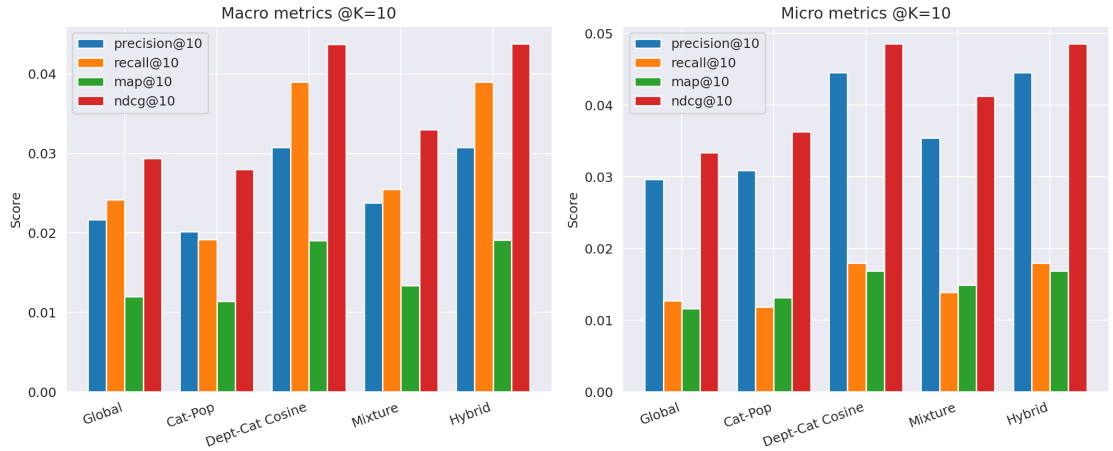

Figure 5.2: Macro and micro metrics @K=10 across models.

Figure 5.2 complements the tables with an immediate visual comparison at K=10: **Hybrid** and **Cosine** are effectively tied on ranking quality, with coverage diverging slightly in favour of **Cosine** (0.4355 vs. 0.4316). Macro P@10 values around 0.031 mean that, on average, roughly 1 in every 32–33 recommended items per position is relevant, while coverage above 0.43 indicates that recommendations are not confined to a small set of popular applications.

To assess whether observed differences are material, 95% bootstrap confidence intervals over departments and paired randomization tests over per-department NDCG@K are computed. The **Hybrid–Cosine** gaps fall within these intervals at $K \in \{5, 10, 20\}$, results are therefore treated as *statistically tied*.

When comparing across cutoffs, at K=5 both remain ahead (macro NDCG@5 $\approx$ 0.0417–0.0419), and at K=20 they retain dominance while coverage rises (Cosine: Cov@20 = 0.5723, Hybrid: 0.5605). The **Mixture** model offers lower ranking quality (macro NDCG@10 $\approx$ 0.0330–0.0342) but is more stable for low-activity departments; popularity baselines remain useful as calibration anchors.

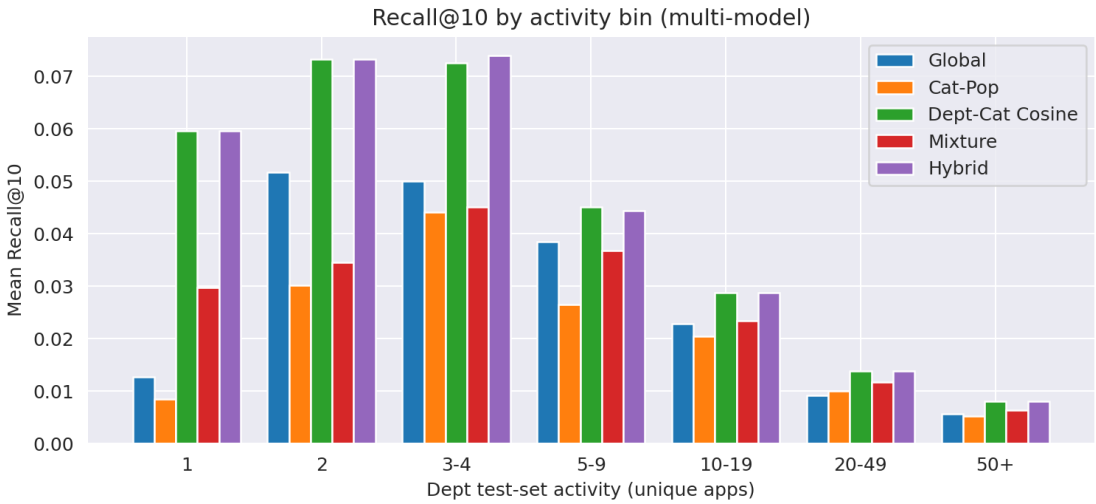

Figure 5.3: Recall@10 by department activity quartile.

Figure 5.3 shows recall rising monotonically with department activity: the more distinct apps a department used in train, the easier it is to recover its held-out apps in test. **Cosine** and **Hybrid** dominate across all quartiles, but the gap to **Mixture** narrows in the lowest-activity bin being consistent with **Mixture's** smoothing acting as a stabilizing prior when neighbor profiles are sparse or noisy.

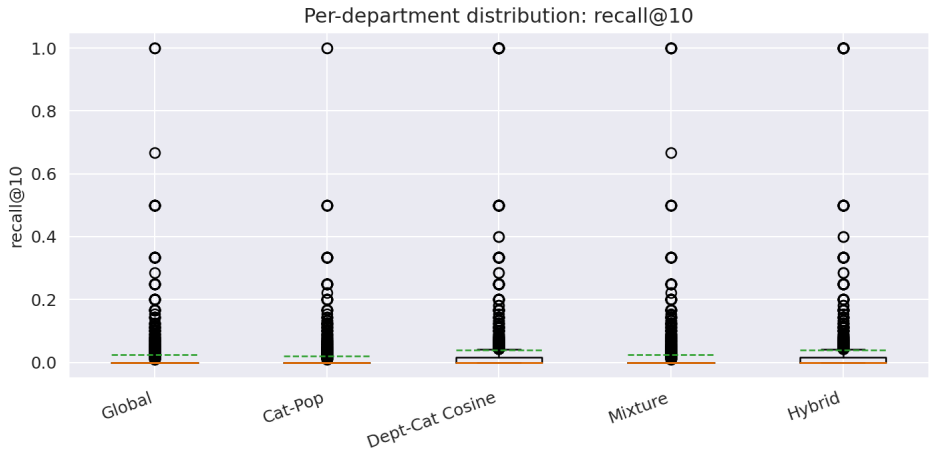

Figure 5.4: Per-department Recall@10 distribution.

As seen in Figure 5.4, recall spreads widely across departments, reflecting real operational heterogeneity. Median recall is higher for **Cosine**/**Hybrid**, and their interquartile ranges sit above those of **Mixture** and the popularity baselines. The long upper outliers indicate that some departments benefit substantially more than others.

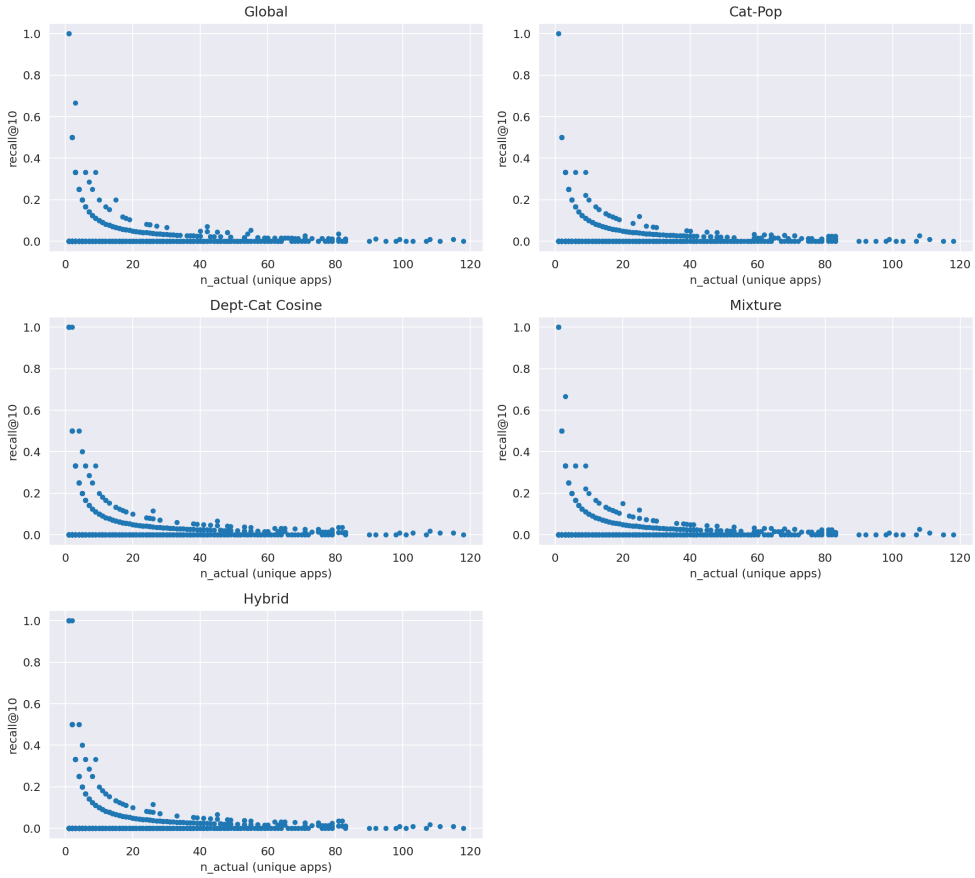

Figure 5.5: Recall@10 vs. number of ground-truth apps.

Figure 5.5 plots each department with the x-axis (ground-truth size) approximating activity and the y-axis as Recall@10. The points trend upward, showing that all models, especially **Cosine**/**Hybrid**, gain recall as the number of relevant apps grows. The cloud of low-activity departments with modest recall again motivates adding richer signals (collaborative or content) to lift performance where data is scarce.

Results therefore indicate that both the **Dept–Category Cosine** and the **Hybrid** model dominate across ranking metrics, with performance effectively tied at K=10. **Cosine** holds a slight but consistent advantage in catalog coverage (0.4355 vs. 0.4316), while **Hybrid** edges marginally in macro NDCG and MAP but within statistical noise. Given these effective ties, **Dept–Category Cosine** is selected for the first iteration (Laplace smoothing, overlap shrinkage, novelty penalty, diversification, TF–IDF optional), with the best **Hybrid** configuration (α=0.65, γ=0.3, λ=0.8, `cand_mult`= 2) retained as a production-ready backup should daily evaluation indicate better adaptation to data shifts. This choice balances performance, interpretability, and stability while keeping the system extensible.

Chapter 6

System Implementation and MLOps Automation

This chapter details the end-to-end implementation of the recommendation system using Kubernetes (K8s) and Azure Cloud Services. The design follows the pipeline logic introduced in Chapter 3 together with the data preparation insights from Chapter 4. The pipeline is realized as containerized workloads that execute the phases Extract, Transform & Load, Train, Validate, and Serve, supported by automation and observability to ensure reliability, scalability, and safe iteration. Lastly, the deployed model follows the algorithmic approach established in Chapter 5.

6.1 Infrastructure Setup

This section describes the foundational platform on which the recommendation system runs. It first details the cluster and environment setup (namespacing, security controls, deployment tooling, and the local Minikube workflow) and then specifies the persistent storage used for raw/processed datasets and versioned model artifacts. Together, these elements provide the baseline capabilities required by the pipeline described in Section 6.2.

6.1.1 Cluster Setup and Configuration

The implementation of the proposed architecture requires a Kubernetes environment capable of hosting all necessary resources. Within the existing myWorkplace landscape, multiple clusters are available (one non-production and one production). For this project, the non-production cluster is used to deploy all required components. To ensure isolation, a dedicated namespace `recommendation-system-impl` is created as the logical boundary for all resources. Communication within this namespace is handled via ClusterIP Services. Sensitive credentials (e.g., OpenSearch passwords) are stored as Kubernetes Secrets, and non-sensitive configuration parameters are externalized in ConfigMaps. Resource requests and limits are defined for each workload to ensure fair scheduling and prevent resource starvation. HorizontalPodAutoscaler (HPA) is enabled to adjust replicas dynamically.

The cluster runs on Azure Kubernetes Service (AKS) and is configured with role-based access control (RBAC), network policies, centralized logging (Azure log analytics), node pools, load balancers, and an ingress controller. Resource provisioning uses Helm charts, simplifying installation and configuration management while keeping deployments portable across environments.

Prior to deploying to AKS, a local development environment is used for testing. Minikube provides a lightweight single-node cluster for validating Helm charts and manifests before applying these resources to the enterprise cluster.

For continuous delivery, ArgoCD synchronizes the Git repository containing the Helm charts with the cluster. The "app of apps" pattern is applied so ArgoCD can automatically create, update, or delete recommendation-system components based on committed changes. Auto-sync is enabled to refresh managed applications. In the local environment, where ArgoCD is not present, resources are applied manually with Helm for test purposes.

6.1.2 Persistent Storage

The pipeline requires persistent storage for two categories of artifacts: (i) raw and processed datasets (e.g., the last four months of indexes) and (ii) trained model artifacts. Beyond that, two PersistentVolumes (PVs) are provisioned in Azure and backed by geo-redundant policies to safeguard against data loss and support recovery. These policies create scheduled snapshots in another resource group and region. Subsequently, given moderate performance requirements, Standard_LRS disks are sufficient. Access is provided via a CSI-managed StorageClass for seamless Kubernetes integration.

6.2 Pipeline Orchestration

The orchestration of the pipeline represents the core of the system's MLOps workflow. Each phase (Extract, Transform & Load, Train, Validate, and Serve) is mapped to containerized workloads scheduled and monitored in Kubernetes. These workloads follow the design principles introduced in Chapter 3 and operationalize the data-preparation insights of Chapter 4 under the CRISP-ML(Q) methodology. The following subsections describe each stage, its implementation, and the safeguards that ensure robustness, scalability, and controlled iteration.

Table A.1 presents values for placeholders used across this chapter that are injected via Kubernetes ConfigMaps.

6.2.1 Extract & Transform & Load

Following the architecture introduced in Section 3.2.3, the pipeline begins with an Extract phase that ingests daily logs from OpenSearch and enriches them with application metadata from the MWP PostgreSQL domain database. This phase is implemented as a native Kubernetes CronJob, which manages one-off executions, retries, and run history without additional wrappers, having conservative retry/backoff parameters, `startingDeadlineSeconds` for missed runs, and `successfulJobsHistoryLimit`/`failedJobsHistoryLimit` for traceability.

The extract container mounts a PersistentVolumeClaim at `/data`. Each run reads the new daily index from OpenSearch, joins rows with application "prime" (category) metadata from PostgreSQL, materializes an in-memory data frame, and writes a columnar Parquet file under `/data/loaded/`. Parquet minimizes I/O and accelerates downstream scans over multi-million-row partitions, as motivated by the EDA in Chapter 4. Ingestion follows a delta method which means that when prior data for the last four months exists, only the current

day is appended to the monthly partition. At the end of the month, the oldest month is dropped, maintaining a sliding window of approximately four months.

In addition to ingestion, structural and statistical checks align with Chapter 4. Structural checks ensure that required columns (e.g., `department`, `applicationId`) remain present and that unexpected schema changes do not propagate. Statistical checks run after dropping columns deemed irrelevant in Chapter 4 and compare the new batch against the trailing window using interpretable tests over `department` and `applicationId` distributions. When deviations exceed a hard hold (e.g., $\geq$ 40% relative change in key category mass), the batch is redirected to `/data/invalidData`, excluded from training, and an alert is raised for inspection. For medium shifts (e.g., 10–40%), the batch proceeds but also triggers an alert. This unified mechanism ensures that both structural and statistical anomalies are consistently handled and surfaced.

The daily run is scheduled for 01:00 UTC, a period of typically low activity in MWP. Listing C.2 shows the CronJob manifest specifying the container template, volume mounts, and schedule. The container image extends the extraction code from Listings B.3, B.2, and B.4 with the drift-checking logic.

Chapter 3 left ingestion frequency open (24 h vs. 48–72 h). A 24-hour cadence is adopted for three reasons: (i) meaningful day-to-day variation in MWP usage (weekday patterns, releases, campaigns) benefits from fresher data; (ii) the delta-based, four-month sliding window keeps each run lightweight at off-peak hours (01:00 UTC); and (iii) training/promotion are gated by validation thresholds, retaining the previous model if a candidate underperforms. If future cost or workload constraints arise, the same CronJob can be re-parameterized to 48–72 h or shifted to an event-driven pattern (e.g., via KEDA) without architectural changes.

6.2.2 Model Training

When the extract job finishes and writes a new Parquet file, the training phase is triggered. In the project, an Azure Function watches for file updates inside `/data/loaded` and, upon detecting a change, calls the AKS API to start a Job that performs model training. This decouples scheduling from the cluster and keeps the implementation simple. A future iteration can replace the external watcher with a Kubernetes-native controller or a KEDA-based trigger while preserving the event-driven pattern.

The training job loads the prepared Parquet data and trains the model according to Chapter 5. Since the selected model is the Dept–Category Cosine recommender, the training image implements the logic from Listing B.7. A second image implements Listing B.9 as a fallback if the primary model does not meet thresholds, present in Subsection 6.2.3. The resulting model artifact is written to a dedicated PersistentVolume that acts as a cluster-scoped model registry. Both data and model volumes apply geo-redundant backup policies. For the project, model retention is short (e.g., three days) and data retention slightly longer (e.g., seven days) to allow inspection of recent shifts or underperforming models.

Listing C.3 illustrates the Kubernetes Job resource used for model training.

6.2.3 Model Validation

Validation is triggered when a new model artifact is written to the model PersistentVolume. As in training, an Azure Function starts a Kubernetes Job that loads the latest artifact and computes the metrics defined in Chapter 5: ranking quality (Precision@K, Recall@K,

NDCG, MAP), optional error metrics (RMSE/MAE if ratings were predicted), sanity checks (absence of NaNs and consistent class spaces), and coverage (at least five recommendations per department).

If thresholds are not met, training is re-run up to three times, always selecting the best-performing candidate. If all attempts fail, the Hybrid image is trained to produce an alternative. If this also fails to reach thresholds, an alert is raised and no new model is deployed, keeping the previous artifact active.

When validation passes, a serving image embedding the model and a lightweight Flask app is built. Images are tagged immutably (e.g., content hash or timestamp) and rolled out with `kubectl rollout restart` to ensure deterministic updates. Due to that, using only `:latest` is discouraged due to tag drift and reduced reproducibility.

Listing C.4 shows the validation Job that loads the model, executes evaluation, and reports metrics.

On top of what was explained in this section, if any threshold fails, the candidate is rejected and the live model remains. If all pass, the artifact is versioned and promoted. With that, Canary serving uses weighted routing; any SLI breach triggers rollback and blocks further promotions.

Validation thresholds

To determine whether a newly trained model is suitable for deployment, a set of thresholds is evaluated based on the metrics discussed in Chapter 5. The evaluation is conducted at K=10 over the full validation split and verified with nonparametric bootstrap (1,000 resamples). The lower bound of the 95% confidence interval must also meet the stated requirements. These thresholds were defined using the comparative results of Section 5.2.4, where the Dep-Category Cosine and Hybrid models emerged as the most competitive candidates.

Primary ranking criteria.

- micro-NDCG@10 $\geq$ 0.047 and at least 98% of the best observed value.
- macro-NDCG@10 $\geq$ 0.042 with no decrease greater than 5-8% compared to the best observed value.
- micro-Precision@10 $\geq$ 0.043 with no decrease greater than 5-8%.
- macro-Recall@10 $\geq$ 0.037, to avoid recall collapse on smaller departments.

Diversity and exploration.

- Coverage@10 $\geq$ 0.40 and not worse than the best observed value by more than 0.02-0.08 absolute.
- At least 99% of departments must receive the full list of K recommendations.

Sanity and correctness.

- No NaNs or Infs in scores or metrics. Class spaces must remain consistent with the training metadata.
- Monotonicity checks: Precision@5 $\geq$ Precision@10 $\geq$ Precision@20. Recall and Coverage must be non-decreasing with K.

Stability and tie-breaks.

- If the candidate model is within ± 0.005 absolute of the reference value on micro-NDCG@10, it is promoted only if it improves either Coverage@10 (by at least 0.01-0.05 absolute) or macro-Recall@10 (by at least 1–5%).
- The 95% confidence interval lower bound for micro-NDCG@10 must satisfy the absolute minimum requirement.

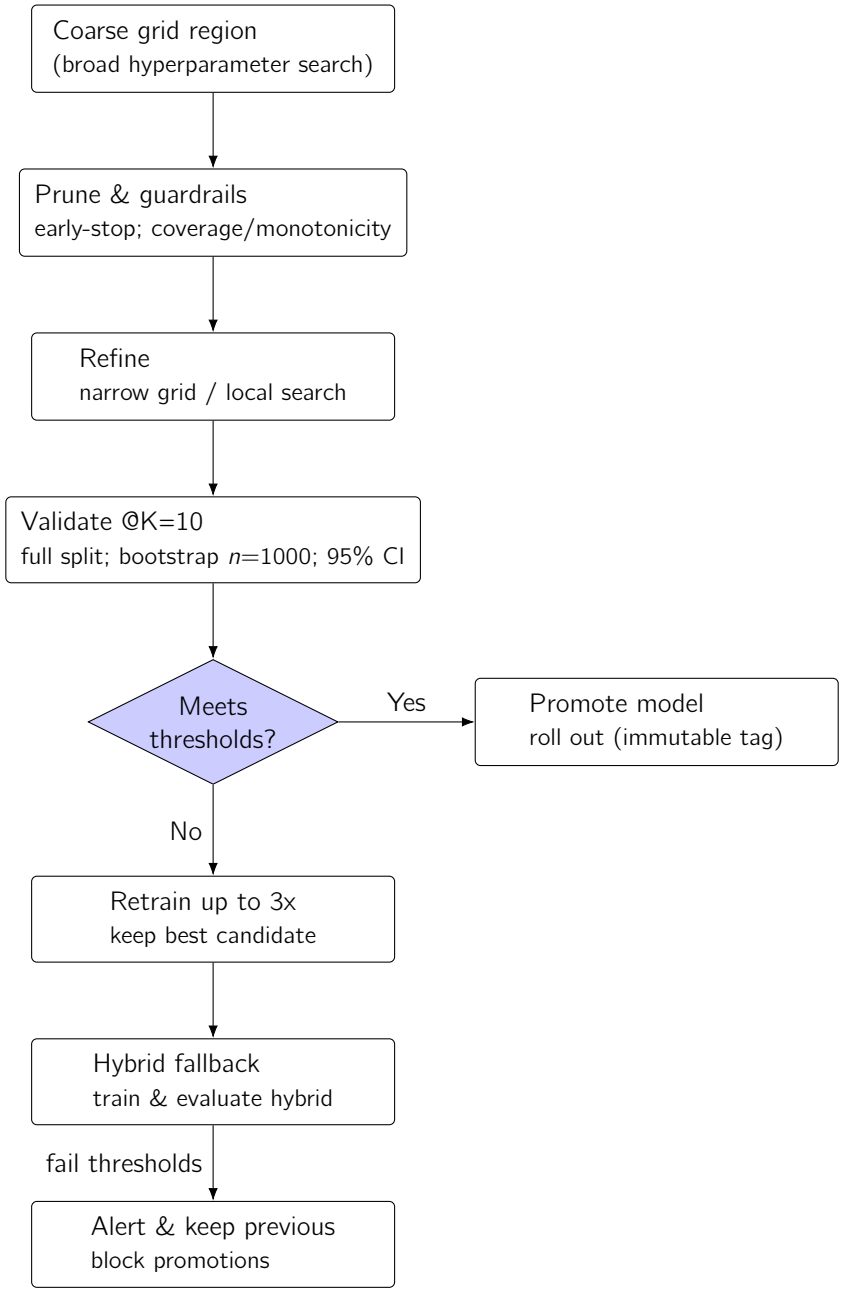

Figure 6.1: Model Selection Flow: From grid to serve.

Figure 6.1 summarizes the end to end selection workflow, from coarse grid search through pruning and refinement to validation, culminating in the promotion decision with bounded retraining, a hybrid fallback, and alerting when thresholds are not met.

In the other hand, Tables A.2 and A.3 consolidate the promotion selector, guardrails, and final hyperparameters, making the validation requirements at K=10 immediately scannable.

6.2.4 Model Export and Integration

Once the content-based model was selected (the Dept–Category Cosine recommender, with the Hybrid retained as a secondary candidate for daily evaluation), the next step was to ensure its portability and reproducibility within the broader MLOps pipeline Chapter 6. Therefore, exporting the trained model artifact into a standardized format is what allows offline experimentation to transition into production deployment.

Unlike deep learning models, which are naturally suited for export in formats such as HDF5 or ONNX, the recommenders models developed here are custom Python classes built on top of pandas and scikit-learn utilities. This distinction implications is that while HDF5 and ONNX excels at representing structured tensor weights, it is not a natural fit for serializing arbitrary Python objects, pipelines, and preprocessing logic. Attempting to force such a representation risks brittle exports or loss of critical behavior embedded in the code and there is no need for that.

For this reason, a strategy based on Joblib serialization combined with a PyFunc wrapper is adopted. Joblib preserves the full Python object, including preprocessing and parameters, for exact reloads, while a thin PyFunc wrapper exposes a stable `predict()/recommend()` interface independent of internal implementation. The resulting artifacts are versionable and registry-friendly (MLflow/Docker/Kubernetes), enabling rollback/promotion and CI/CD validation without format friction.

Compared to traditional exports (e.g., raw pickle, which is fragile and insecure, or HDF5, which is tailored to tensor-based frameworks), Joblib + PyFunc strikes a balance between Python-native flexibility and production-readiness. It avoids the brittleness of low-level binary dumps while offering a predictable interface for downstream integration.

This way, the export step becomes more than a technical necessity, it represents the bridge between experimentation and reliable, repeatable deployment. Furthermore, the chosen structure is future-proof, as hybrid models incorporating collaborative filtering are added in later phases. They too can be wrapped in the same PyFunc abstraction, guaranteeing consistency across the deployment workflow without disrupting downstream systems.

6.2.5 Model Serving

The recommender system is served by a Kubernetes Deployment that runs the Flask application bundled with the validated model artifact. The Deployment uses a rolling update strategy that when a new, immutably tagged image is pushed, a rollout restart updates pods gradually. This is automatically handled by the ArgoCD instance in the cluster. If a single replica is configured, the scheduler creates a new pod, waits for `readinessProbe` and `livenessProbe` to pass, and then terminates the old one, minimizing downtime. Additionally, an HPA scales between one and three replicas based on resource targets (e.g., add a replica when CPU usage reaches 80%).

Access to the service is intentionally constrained: a namespaced ClusterIP service exposes the application only inside the cluster. A NetworkPolicy restricts inbound traffic so that only the MWP backend (labeled selector) and Prometheus can call the API.

Listing C.5 shows the application Deployment. To adapt across environments, the ArgoCD "app of apps" references `values-<env>.yaml` files. In this project (being in a non-production environment), lower resource requests/limits are configured via Helm (using `values.yaml` file). The same strategy applies to the HPA, Listing C.6, and related resources.

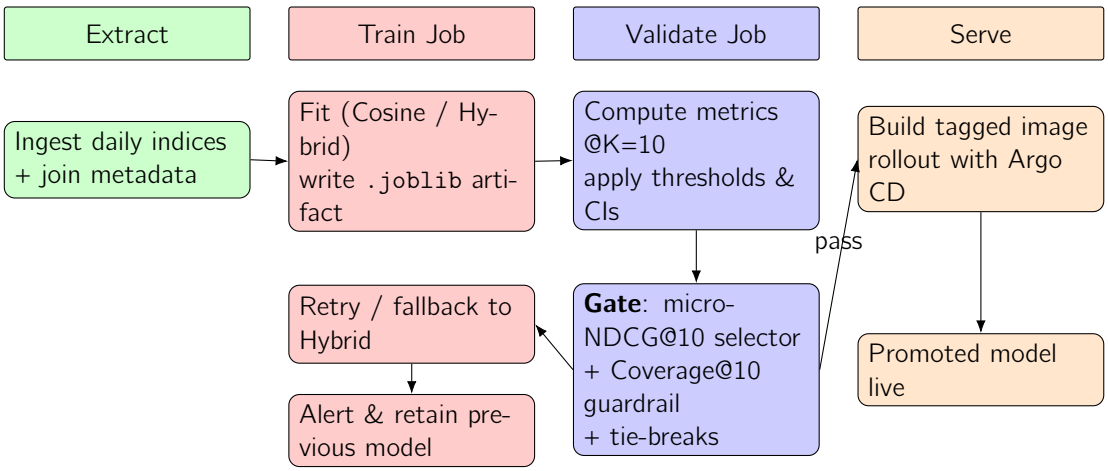

Figure 6.2: Promotion loop.

Figure 6.2 illustrates the full workflow in operational order, mentioned in this Chapter until now, serving as a practical complementary diagram to Figure 6.1, which summarizes the model selection and promotion gate.

6.3 Feedback and Online Signals Strategy

To enable transition toward a hybrid recommender system, the MWP backend exposes an endpoint that captures user interactions with the recommended list. If a suggested application is opened, that item receives a score of 5 and sibling suggestions are assigned 3 (context exit). If all suggestions are ignored, each receives 2 and explicit downvotes receive 1. These signals are persisted in a relational table and later incorporated as a third data source.

Online SLIs such as p50/p95 latency, 5xx rate, per-department Coverage@10, are tracked against SLOs defined in Chapter 3. As a guardrail, any 1-hour breach pages on-call and halts promotions. To validate user impact, click-through rate (CTR) is monitored to validate that offline gains translate to user impact.

6.4 Observability and Monitoring

The observability stack builds on the existing cluster installation of Prometheus, Alertmanager, Grafana, and Loki (via `kube-prometheus-stack` and `kube-loki`). The Flask application exposes `/metrics` for Prometheus and emits logs ingested by Loki.

Metrics include infrastructure indicators (CPU, memory, HTTP error rate, pod restarts, HPA activity) and service KPIs such as `ml_inference_requests`, `ml_inference_requests_error`, `ml_inference_latency_seconds_bucket`, `ml_recommendation_success_rate` (CTR proxy), and `ml_recommendation_coverage` (at least five per department). PromQL queries define alerting rules, integrated with Alertmanager and connected to external tools (e.g., Opsgenie), these rules trigger alerts on drift, schema failures, or KPI regressions.

If alerts indicate material KPI degradation or anomalous behaviour, a manual rollback to a prior deployment revision (e.g., via ArgoCD rollback or `kubectl rollout undo`) can be performed. This keeps rollback an explicit, human-validated action rather than an automated response to transient issues.

6.5 Security and Compliance

For this project, the recommender system endpoint is restricted to the BMW internal network and does not integrate external identity flows (e.g., OAuth 2.0/OIDC). The exposure surface is limited via network policies and in-cluster access.

Project posture

- Cluster only exposure (ClusterIP), inbound restricted by NetworkPolicy to MWP backend and Prometheus.
- Secrets via Kubernetes Secrets and non-sensitive config via ConfigMaps.
- Immutable container images, rolling updates, and basic audit logs through the existing stack.

Secret management follows rotation and least-privilege principles: credentials are stored as Kubernetes Secrets with RBAC-scoped access and rotated on a regular schedule, minimizing blast radius in case of leakage.

Detailed requirements will be finalized alongside the enhancements outlined in Chapter 7.3.

6.6 System Architecture Diagram

Figure A.3 summarizes the data and control flows from Extract to Serve, including drift checks, validation loops, and the observability stack. For simplicity, the ArgoCD and monitoring namespaces are represented only at a high level, showing how they integrate with the system by synchronizing Kubernetes resources, collecting metrics, and generating alerts. Prometheus also incorporates Alertmanager, which can connect to external tools such as Opsgenie, Microsoft Outlook, or other notification systems. To collect metrics on CronJobs, Pods, Jobs, and other resources, Prometheus communicates directly with the Kubernetes API, which is responsible for exposing this information. Azure Functions interact with the cluster by using a Service Principal recognized by the cluster's RBAC system, allowing them to authenticate and trigger Jobs as needed. Finally, the MWP backend deployment can communicate with the recommendation system deployment over HTTPS. For example:

```
http://rs-serve-clusterip.recommendation-system-impl.svc.cluster.local
/recommend?department=...
```

This request fetches application recommendations as long as the Flask application is running and healthy. Moreover, the posture aligns with the governance and security principles outlined in Chapter 3, with production-grade identity and audit controls deferred to Chapter 7.3.

Chapter 7

Conclusion

This chapter includes a summary of the objectives achieved, an overview of the challenges encountered, and the lessons learned throughout the project. It also outlines potential directions for future work and finalizes with a personal reflection on the overall experience.

Moreover, this project contributes to both the literature and practice identified in the previous chapters by framing MLOps within the context of recommender system development, demonstrating a reproducible pipeline in the BMW environment, and introducing an evaluation framework supported by appropriate metrics with defined guardrails.

7.1 Achieved Objectives

This section reviews the main objectives that were successfully accomplished through the project. With that in mind, this project was designed and implemented to address this problem by offering a faster and more efficient way to discover relevant applications based on the user's department. By analyzing user behavior and leveraging the collected data, the RS provides personalized suggestions that will also improve discoverability.

Table 7.1: Project Objectives and Completion Status

Objective	**Completion**
Review the state of the art	100%
Characterize the problem and success criteria	100%
Build and document the dataset	100%
Investigate and evaluate content-based, collaborative, and hybrid approaches under realistic constraints (limited CF due to sparse signals, LLM excluded for compliance)	70%
Propose a reference architecture and lifecycle	100%
Specify operational and governance requirements	100%
Lay the path for future enhancements	100%

The objectives defined in Chapter 1.2 served as a guide for the development of this project, and Table 7.1 summarizes their completion status. Consequently, the values obtained for each objective are based on the outcomes of the implementation described in the previous Chapters. In addition to validating that each task was carried out, these objectives directly connect back to the research questions defined in Chapter 1: `RQ1` was addressed through the systematic review and pipeline design, the `RQ2` was answered by the implementation and

evaluation of automated workflows and, finally, RQ3 was explored through the analysis of model performance and operational constraints.

This project began with a extensive literature review, which established the foundation for the RS design. As a result, content-based algorithms were identified, along with best practices for MLOps pipelines, methodologies for ETL workflows, and the most relevant Azure tools to support these requirements.

Next, stakeholder requirements were analyzed, and a high-level architecture was defined and refined, as presented in Chapter 3. This was followed by the design of an extraction and preprocessing workflow that consolidated logs and metadata into a reproducible dataset (Chapter 4).

Model development was carried out using the available features, with multiple models tested to identify the most suitable approach. Due to heterogeneity across departments and limited application metadata, collaborative filtering approaches were not feasible, and LLM-based recommenders were excluded due to compliance constraints and implementation complexity (e.g., RAG pipelines, prompt-injection protection and others). For this reason, this objective is marked as 70% complete (Chapter 5).

Finally, the system implementation was completed with the design of a full MLOps pipeline on Kubernetes and Azure. This included ETL automation, training, validation, serving, monitoring, and security considerations (Chapter 6).

In summary, the system showed how a content-based recommendation model approach, integrated into a MLOps pipeline following CRISP-ML(Q) guidelines, providing a scalable and reliable deployment in an enterprise environment.

7.2 Limitations

While the proof of concept demonstrated feasibility, this project was subject to several constraints that shaped both scope and outcomes.

First, the available dataset was limited to department-level logs and coarse application metadata, restricting the personalization capacity of content-based models. Second, collaborative filtering approaches could not be fully explored due to data sparsity and the absence of explicit feedback signals. Third, LLM-based recommenders were excluded because of compliance constraints and the lack of infrastructure for secure integration. Forth, and lastly, this project infrastructure was designed for a non-production environment, which imposed constraints on scalability and security. These factors guided the prioritization of content-based methods as the foundation for the recommendation system.

Additionally, the validity of the results and conclusions is influenced by several factors. The reduced feature space (eight application categories) constrains personalization and may underestimate the potential of richer approaches (construct validity). The evaluation relied on department-level splits and offline metrics, which may bias results toward observed patterns while overlooking unobserved dynamics (internal validity). Furthermore, since experiments were performed on BMW-specific data and infrastructure, generalization to other domains will require adaptation (external validity).

Finally, one broader limitation should also be acknowledged: this project is limited to offline evaluation. Online validation with real user interactions remains for future stages, and

would provide additional insights into the practical effectiveness of the proposed system in production environments.

7.3 Future work

While this project has successfully delivered as a proof of concept for an enterprise ready recommendation system, several opportunities remain for improvement and extension. This section outlines possible directions for future research and development, building on the foundations established in previous chapters.

The proposed modifications include enhancing the recommendation logic with additional models (e.g., collaborative filtering, embeddings, and a more complex hybrid architecture), expanding the scope of the recommended items beyond applications to include dashboards, widgets, and links while also strengthening production level features such as security integration (specifically in the endpoints exposed by the API), drift monitoring, and automated training. With that, following approaches, such as the use of Large Language Models (LLMs) for semantic profiling and re-ranking, also present promising opportunities.

Every potential future enhancement discussed in this section must also take into account both performance and system wide impact. While model improvements may positively influence the recommender system's accuracy and usefulness, it is equally important to evaluate their effect on training, deployment, and runtime operations. New approaches should be carefully assessed to ensure that the existing infrastructure does not become a bottleneck due to resource constraints or configuration limitations. Although the current setup performs effectively with the deployed components, future extensions may introduce overhead that requires rethinking or scaling the infrastructure.

These future developments, when carefully balanced with system performance and operational constraints, will help evolve the solution from a proof of concept into a robust, fully scalable system aligned with enterprise requirements and the dynamic needs of end users.

7.3.1 Collaborative Filtering Integration

Since there is significant heterogeneity across BMW departments, it is necessary, while respecting company compliance requirements, to collect additional information regarding user interactions with MWP. This includes the possibility of integrating further data sources that provide richer application metadata (e.g., detailed categories, functional scope, and related attributes), as well as incorporating implicit feedback strategies based on the currently deployed content-based recommendations. Such mechanisms would allow evaluation of application usage across departments, enabling the identification of behavioral patterns and the construction of more complex user and department profiles. The complementary strategy of explicit feedback is discussed in Section 6.2.5. Once sufficient feedback data becomes available, it can be integrated as an additional data source to refine and adapt the generated recommendations, as anticipated in Section 3.2.3.

Furthermore, as online interaction signals accumulate, collaborative filtering (CF) methods such as item co-visitation and matrix factorization can be introduced for candidate generation and hybrid recommendation. This integration is expected to increase recall, particularly for higher-activity departments.

7.3.2 Embedding Based Content Models

A natural extension of the current system is the use of embedding based models, which become feasible once richer application metadata is available. Currently, the dataset is limited to coarse attributes such as application category (prime), which constrains the representational power of content based methods. By enriching the ETL pipeline to incorporate additional metadata, such as textual descriptions, functional tags, or others, embeddings can be trained to capture semantic and structural similarities beyond simple categorical overlaps.

Techniques such as word or sentence embeddings (e.g., Word2Vec, BERT variants) and, graph embeddings, can map applications into dense vector spaces where similarity is learned directly from co-occurrence or structural patterns. Subsequently, this would improve generalization to long-tail applications and mitigate cold-start issues in departments with sparse activity, as embeddings are able to leverage shared attributes and contextual information.

In practice, embedding-based approaches could be integrated as an alternative candidate generation stage within the hybrid system. Applications would be ranked not only by explicit category frequency but also by semantic proximity in the embedding space, providing richer personalization. Over time, embeddings could be updated incrementally as new metadata becomes available, ensuring that representations evolve alongside the MWP ecosystem.

While this direction requires significant preprocessing and infrastructure adaptations (e.g., embedding training, versioning, and storage), it represents a promising avenue for scaling recommendation quality. Moreover, embeddings could serve as a foundational layer for more advanced techniques, such as re-ranking models or LLM-assisted pipelines, thereby extending the long-term flexibility and robustness of the recommendation system.

7.3.3 Multi Model per Category

Another possible direction, given the performance of the content-based models conducted, would be to train and maintain separate models for each application category (prime), following the same evaluation protocol established in Chapter 5. As illustrated in Figures 4.4 and 4.5, there are significant differences in the number of entries across categories. Since this work relies on production data (real operational logs), it was not feasible to generate synthetic data to balance these “classes”.

To address this imbalance, one option would be to design a multi-model architecture where each category has its own specialized model. After generating recommendations per prime, the system could merge results into a final compiled list, weighting each model’s output according to evaluation metrics (e.g., coverage or ranking quality). This approach could reveal whether category-specific specialization provides improved results and a better user experience, particularly in scenarios where hybrid systems with collaborative filtering are not yet integrated.

In practice, this strategy would involve training and selecting per-category models to better reflect domain heterogeneity and routing departments to prime-specific predictors. Performance could be monitored through periodic A/B tests or shadow evaluations against a global baseline, ensuring that the added complexity translates into measurable improvements.

7.3.4 LLM Assisted Approaches

Given BMW compliance constraints, the direct use of large language models (LLMs) presents significant challenges. Beyond the inherent security risks, such as prompt injection and model manipulation, it would be necessary to adopt refinement strategies, like Retrieval Augmented Generation (RAG), to ensure that responses remain contextually valid and aligned with company policies. RAG would also require embedding, vectorization, and data chunking processes to guarantee that prompts and responses are filtered through secure, auditable channels.

Developing a domain specific LLM solely for this recommendation system would be difficult given the limited metadata currently available and time constraints. However, integration with BMW's internal LLMs represents a feasible future direction, provided that appropriate safeguards and governance mechanisms are established. In such a setup, an LLM could serve as a second validation layer, evaluating whether the recommendations generated by the content-based or hybrid models are consistent with the department's context. This would enhance the interpretability of the system and potentially increase recommendation confidence by adding an additional layer of semantic reasoning.

Moreover, this approach could support explainability by justifying recommendations in natural language, aiding both end users and system administrators in understanding why certain applications are suggested. The combination of LLM driven validation with RAG based safeguards would ensure compliance with BMW's security and privacy requirements while opening the door to advanced, AI assisted personalization features.

7.3.5 Expanding Beyond Applications

In its current state, the RS is limited to handling MWP integrated applications. However, to align with evolving stakeholder needs, it will be necessary to extend its scope to include additional resources such as dashboards, widgets, and links. Achieving this requires enhancements in the host application's data collection capabilities to capture richer metadata for these new resource types. While this expansion is desirable, it must be carefully evaluated within the context of BMW's compliance and strict data protection policies, which remain a central challenge given the sensitive nature of the domain.

From a technical perspective, the required modifications should be relatively contained. If a per category model approach is not adopted, the system would simply need to manage four distinct models (applications, dashboards, widgets, and links) within the same Kubernetes deployment.

7.3.6 Recommendation System Application Framework and Security

Since this project was deployed in a non production environment, a lightweight framework, such as Flask was chosen as it requires minimal space, it is easy to containerize, and allows for a quickly exposing of an endpoint. Nevertheless, in a production environment where traffic will be significantly higher and demanding, it will be necessary to utilize a more robust alternative. In contrast, frameworks such as Django (Python based), or even Java based solutions like Quarkus, could be considered to serve REST or GraphQL interfaces with a better consideration for this liability.

Equally important is the integration of robust security mechanisms, which were beyond the scope of this project. In a production environment, authentication and authorization

should be enforced using standards such as OIDC or OAuth 2.0, combined with RBAC at the service level. In addition, centralized secrets management through Azure Key Vault and comprehensive audit logging would be essential to ensure compliance with enterprise security and governance requirements.

Scaling the solution to support multiple models and higher usage may ultimately require replacing the lightweight Flask application with a more production-grade serving framework. This transition would ensure that the system can encapsulate diverse models, handle enterprise traffic demands, and remain secure, resilient, and future-proof as its scope expands.

7.3.7 Extended Evaluation Dimensions

This initial project phase focused only on metrics such as Precision, Recall, MAP, NDCG, and Coverage, as mentioned in Chapter 5, which were aligned with the available data and defined objectives. Even so, a complete evaluation of recommender systems should also consider additional dimensions, such as diversity, novelty, serendipity, calibration, fairness, and robustness that directly affect user experience and long term adoption.

As in the other Subsections, a possible future improvement to the implemented RS is the integration of additional metadata and explicit feedback as they become available. These dimensions could be incorporated into both offline and online evaluation (during model training and within the observability stack). For instance, novelty and serendipity metrics would benefit from user level histories, fairness would require group, or item level exposure monitoring, and, finally, robustness could be implemented through temporal drift or controlled perturbation tests. Having this extra metrics and methodologies would provide a more complete view of the system performance.

7.3.8 Future Work Prioritization

Given the range of future directions outlined in the previous Subsections, it is essential to establish a clear prioritization aligned with the project's context and practical constraints. The most immediate priority is the "Recommendation System Application Framework and Security". Ensuring robust authentication, authorization, and production ready serving capabilities is fundamental to guarantee system safety and scalability, particularly under higher traffic conditions expected in a production environment.

After this improvement is in place, the next step should be the implementation of a mechanisms to enrich application metadata and also integrating explicit user feedback given the current RS implementation. These enhancements will directly improve the quality of the current RS and enable the introduction of collaborative filtering within a hybrid architecture.

Following these priorities, the next improvements can be done in parallel. Training models per category (Multi-Model per Category) may uncover further gains by addressing heterogeneity across application prime domains. In parallel, embedding-based models (Embedding-Based Content Models) can be explored if richer metadata becomes available, together with the inclusion of additional metrics such as diversity, novelty, serendipity, calibration, fairness, and robustness. These approaches represent good improvements to the current system.

With these improvements in place, the recommender system can expand its scope beyond applications to include dashboards, widgets, and links, thereby broadening its impact across the MWP ecosystem. Finally, LLM-Assisted Approaches can be considered as a long-term goal. While promising in terms of semantic validation and explainability, they demand substantial

compliance verification and infrastructure adjustments, making them the most complex and time-intensive to implement.

This prioritization ensures that system evolution proceeds incrementally, balancing technical feasibility, compliance requirements, and business value, while delivering tangible improvements at each stage. A future work diagram is illustrated in the Figure A.4.

7.4 Final Remarks

This project demonstrates that advances in recommender system algorithms cannot be separated from their operational context. Concretely, both the systematic review and the project implementation highlight that the MLOps practices are important for performance and reliability of the system as much as the model choice itself. From a scientific perspective, this suggests that future RS research should explicitly think about integrating MLOps practices into its methodological developing, moving beyond focusing only in the model benchmarks and results.

From an industrial viewpoint, the project illustrates both the advantages and the challenges of adopting MLOps-driven recommenders. While automation, CI/CD/CT and feature store-based pipelines enable scalability and maintainability, organizational factors, such as data silos, compliance requirements, and adoption barriers are super important for the system success. Subsequently, the work experience in the BMW/CTW shows that bridging research and practice requires not only technical architectures but also processes that elaborates the collaboration between different roles such as data scientists, engineers, and business stakeholders.

Furthermore, this project serves not only as a proof of concept (POC) but also as a case study of how methodological rigor can coexist with industrial practices. With that, the broader lesson is that recommender systems in enterprise contexts are socio-technical systems, where scientific innovation, technical infrastructure, and organizational culture must evolve together.

The development of this project has also been a valuable personal and professional experience. It provided the opportunity to apply knowledge gained during the Master's studies in Artificial Intelligence Engineering while offering direct exposure to the business side of software development.

Engaging with new technologies and concepts, particularly in machine learning and system infrastructure, significantly expanded the author's knowledge base. Although tackling complex tasks and processing large volumes of information was at times demanding, these challenges provided valuable learning opportunities. The hands-on nature of the work enabled a deeper understanding of both the technical and practical aspects of system development.

For CTW, the recommender system and the infrastructure developed establish a solid base point for future project ideas. Also, this implementation can serve as a reference for other projects inside the company and also provide insights for other teams aiming to integrate similar solutions into their current workflows.

Beyond the mentioned technical contributions, this project also has strengthened the author's ability to critically revise trade-offs between research innovation and enterprise constraints, being a skill that will remain essential in both academic and professional contexts for him.

Overall, this dissertation delivers not only a PoC within CTW/BMW's enterprise environment but also insights of broader relevance to recommender system research. By bridging advances in MLOps, evaluation, and content-based modeling with real-world deployment constraints, it illustrates how methodological rigor can coexist with operational feasibility. These lessons may guide future recommender system initiatives in similarly complex industrial ecosystems, where scalability, compliance, and user trust must align with cutting-edge personalization techniques.

Bibliography

Agresti, A. (2019). *An introduction to categorical data analysis* (3rd). Wiley.

Ak, R., Schifferer, B., Rabhi, S., & De Souza Pereira Moreira, G. (2022). Training and deploying multi-stage recommender systems. *Proceedings of the 16th ACM Conference on Recommender Systems*, 706–707. https://doi.org/10.1145/3523227.3547372

Amit Chauhan. (2025). *Mlops: Maturity levels for automation in machine learning*. Retrieved September 21, 2025, from https://medium.com/pythoneers/mlops-maturity-levels-for-automation-in-machine-learning-befbf6bcf01c

Argyriou, A., González-Fierro, M., & Zhang, L. (2020). Microsoft recommenders: Best practices for production-ready recommendation systems. *Companion Proceedings of the Web Conference 2020*, 50–51. https://doi.org/10.1145/3366424.3382692

Bauer, C., Zangerle, E., & Said, A. (2024). Exploring the landscape of recommender systems evaluation: Practices and perspectives. *ACM Transactions on Recommender Systems*, *2*(1), 1–31. https://doi.org/10.1145/3629170

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, *13*(null), 281–305.

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (Eds.). (2016). *Site reliability engineering: How google runs production systems*. O'Reilly Media.

Chai, Z., Lu, H., Chen, D., Ren, Q., Zheng, Y., & Zhou, X. (2025). Adaptive domain scaling for personalized sequential modeling in recommenders. *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 4234–4238. https://doi.org/10.1145/3726302.3731939

Chai, Z., Ren, Q., Xiao, X., Yang, H., Han, B., Zhang, S., Chen, D., Lu, H., Zhao, W., Yu, L., Xie, X., Ren, S., Sun, X., Tan, Y., Xu, P., Zheng, Y., & Wu, D. (2025). Longer: Scaling up long sequence modeling in industrial recommenders. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 247–256. https://doi.org/10.1145/3705328.3748065

Chen, W., Zhao, Y., Chen, L., & Pan, W. (2025). Leave no one behind: Fairness-aware cross-domain recommender systems for non-overlapping users. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 226–236. https://doi.org/10.1145/3705328.3748082

Cloud Native Computing Foundation. (2025). *Kubernetes documentation*. Retrieved January 13, 2025, from https://kubernetes.io/docs/

CNCF GitOps Working Group. (2021). *Gitops principles*. Retrieved September 21, 2025, from https://www.cncf.io/blog/2021/08/19/gitops-principles/

Davis, A. (2021). *Bootstrapping microservices with docker, kubernetes, and terraform: A project-based guide*. Manning.

De la Rua Martinez, J., Buso, F., Kouzoupis, A., Ormenisan, A. A., Niazi, S., Bzhalava, D., Mak, K., Jouffrey, V., Ronstrom, M., Cunningham, R., Zangis, R., Mukhedkar, D., Khazanchi, A., Vlassov, V., & Dowling, J. (2024). The hopsworks feature store for machine learning. *Companion of the 2024 International Conference on Management of Data*, 135–147. https://doi.org/10.1145/3626246.3653389

De Souza Pereira Moreira, G., Rabhi, S., Ak, R., & Schifferer, B. (2021). End-to-end session-based recommendation on gpu. *Proceedings of the 15th ACM Conference on Recommender Systems*, 831–833. https://doi.org/10.1145/3460231.3473322

de Souza Pereira Moreira, G., Rabhi, S., Lee, J. M., Ak, R., & Oldridge, E. (2021). Transformers4rec: Bridging the gap between nlp and sequential / session-based recommendation. *Proceedings of the 15th ACM Conference on Recommender Systems*, 143–153. https://doi.org/10.1145/3460231.3474255

Diaz-de-Arcaya, J., Torre-Bastida, A. I., Zárate, G., Miñón, R., & Almeida, A. (2023). A joint study of the challenges, opportunities, and roadmap of mlops and aiops: A systematic survey. *ACM Comput. Surv.*, *56*(4). https://doi.org/10.1145/3625289

Eken, B., Pallewatta, S., Tran, N., Tosun, A., & Babar, M. A. (2025). A multivocal review of mlops practices, challenges and open issues. *ACM Comput. Surv.*, *58*(2). https://doi.org/10.1145/3747346

Ekstrand, M. D. (2020). Lenskit for python: Next-generation software for recommender systems experiments. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2999–3006. https://doi.org/10.1145/3340531.3412778

Fujikawa, K., Murakami, N., & Sugawara, Y. (2024). Enhancing news recommendation with transformers and ensemble learning. *Proceedings of the Recommender Systems Challenge 2024*, 42–47. https://doi.org/10.1145/3687151.3687160

Fukumoto, A., Salah, A., Shrivastava, S., Tatar, A., Schwartz, Y., Michel, V., & Xiong, L. (2025). Contrastive conditional embeddings for item-based recommendation at e-commerce scale. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 931–934. https://doi.org/10.1145/3705328.3748095

Gao, C., Li, S., Lei, W., Chen, J., Li, B., Jiang, P., He, X., Mao, J., & Chua, T.-S. (2022). Kuairec: A fully-observed dataset and insights for evaluating recommender systems. *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 540–550. https://doi.org/10.1145/3511808.3557220

Ge, Y., Liu, S., Fu, Z., Tan, J., Li, Z., Xu, S., Li, Y., Xian, Y., & Zhang, Y. (2024). A survey on trustworthy recommender systems. *ACM Trans. Recomm. Syst.*, *3*(2). https://doi.org/10.1145/3652891

Google Brain team. (2025). *An end-to-end platform for machine learning*. Retrieved July 13, 2025, from https://www.tensorflow.org/

Grafana Labs. (2025). *Loki documentation*. Retrieved January 25, 2025, from https://grafana.com/docs/loki/latest/

Gusak, D., Volodkevich, A., Klenitskiy, A., Vasilev, A., & Frolov, E. (2025). Time to split: Exploring data splitting strategies for offline evaluation of sequential recommenders. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 874–883. https://doi.org/10.1145/3705328.3748164

Heck, P. (2024). What about the data? a mapping study on data engineering for ai systems. *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, 43–52. https://doi.org/10.1145/3644815.3644954

Heitz, L., Li, R., Inel, O., & Bernstein, A. (2025). Informfully recommenders - reproducibility framework for diversity-aware intra-session recommendations. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 792–801. https://doi.org/10.1145/3705328.3748148

Helm Authors. (2025). *Helm documentation*. Retrieved January 25, 2025, from https://helm.sh/docs/

Henriques, R., & Pinto, L. (2023). A novel evaluation framework for recommender systems in big data environments. *Expert Systems with Applications*, *231*, 120659. https://doi.org/10.1016/j.eswa.2023.120659

Higley, K., Oldridge, E., Ak, R., Rabhi, S., & de Souza Pereira Moreira, G. (2022). Building and deploying a multi-stage recommender system with merlin. *Proceedings of the 16th ACM Conference on Recommender Systems*, 632–635. https://doi.org/10.1145/3523227.3551468

Hu, G., Zhang, A., Liu, S., Cai, Z., Yang, X., & Wang, X. (2025). Alphafuse: Learn id embeddings for sequential recommendation in null space of language embeddings. *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1614–1623. https://doi.org/10.1145/3726302.3729894

INNOQ. (2025). *Crisp-ml(q). the ml lifecycle process*. Retrieved September 18, 2025, from https://ml-ops.org/content/crisp-ml

Intuit. (2025). *Argocd overview*. Retrieved July 13, 2025, from https://argo-cd.readthedocs.io/en/stable/

Jérémie du Boisberranger. (2025). *Scikit-learn, machine learning in python*. Retrieved July 13, 2025, from https://scikit-learn.org/stable/index.html

Jiang, X., Wang, K., Wang, Y., Lv, F., Peng, T., Yang, S., Wu, X., Zhang, P., Yuan, S., & Zeng, Y. (2024). Deep uncertainty-based explore for index construction and retrieval in recommendation system. *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 4587–4594. https://doi.org/10.1145/3627673.3680077

Julianti, M. R., Heryadi, Y., Yulianto, B., & Budiharto, W. (2022). Recommendation system model for personalized learning in higher education using content-based filtering method. *2022 International Conference on Information Management and Technology (ICIMTech)*, 1–6. https://doi.org/10.1109/ICIMTech55957.2022.9915109

Kaminskas, M., & Bridge, D. (2016). Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems. *7*(1). https://doi.org/10.1145/2926720

Kitchenham, B., Brereton, P., Li, Z., Budgen, D., & Burn, A. (2011). Repeatability of systematic literature reviews. *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, 46–55. https://doi.org/10.1049/ic.2011.0006

Kloppers, C. M. (2025). Adding value to low-resource industrial recommender systems. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 1435–1438. https://doi.org/10.1145/3705328.3748760

Koren, Y., Rendle, S., & Bell, R. (2022). Advances in collaborative filtering. In F. Ricci, L. Rokach, & B. Shapira (Eds.), *Recommender systems handbook* (pp. 91–142). Springer US. https://doi.org/10.1007/978-1-0716-2197-4_3

Körner, C., & Alsdorf, M. (2022). *Mastering azure machine learning: Execute large-scale end-to-end machine learning with azure*. Packt Publishing.

Kubernetes Authors. (2025). *Minikube documentation*. Retrieved January 25, 2025, from https://minikube.sigs.k8s.io/docs/

Lee, H., Yoo, S., Lee, D., & Kim, J. (2023). How important is periodic model update in recommender system? *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2661–2668. https://doi.org/10.1145/3539618.3591934

Li, F., Si, X., Tang, S., Wang, D., Han, K., Han, B., Zhou, G., Song, Y., & Chen, H. (2024). Contextual distillation model for diversified recommendation. *Proceedings*

of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 5307–5316. https://doi.org/10.1145/3637528.3671514

Li, F., Li, Y., Liu, Y., Zhou, C., Wang, Y., Deng, X., Xue, W., Liu, D., Xiao, L., Gu, H., Jiang, J., Liu, H., Qin, B., & He, J. (2025). Leadre: Multi-faceted knowledge enhanced llm empowered display advertisement recommender system. *Proc. VLDB Endow.*, *18*(12), 4763–4776. https://doi.org/10.14778/3750601.3750602

Li, N., Pan, Y., Gao, C., Jin, D., & Liao, Q. (2024). Full-stage diversified recommendation: Large-scale online experiments in short-video platform. *Proceedings of the ACM Web Conference 2024*, 4565–4574. https://doi.org/10.1145/3589334.3648144

Linux Foundation. (2025). *Opensearch*. Retrieved January 25, 2025, from https://opensearch.org/

Liu, H., Gao, Q., Liao, X., Chen, G., Xiong, H., Ren, S., Yang, G., & Zha, Z. (2022). Lion: A gpu-accelerated online serving system for web-scale recommendation at baidu. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 3388–3397. https://doi.org/10.1145/3534678.3539058

Liu, R., Cao, Y., Wang, Y., Lyu, L., Chen, Y., & Chen, H. (2023). Privaterec: Differentially private model training and online serving for federated news recommendation. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 4539–4548. https://doi.org/10.1145/3580305.3599889

Liu, X., Gao, B., Suleiman, B., You, H., Ma, Z., Liu, Y., & Anaissi, A. (2023). Privacy-preserving personalized fitness recommender system p3fitrec: A multi-level deep learning approach. *ACM Trans. Knowl. Discov. Data*, *17*(6). https://doi.org/10.1145/3572899

Long, B., Wang, Y., Mehta, H., Zomnir, M., Pathak, O., Meng, C., Jia, R., Peng, Y., Hong, D., Wu, X., Gao, M., Dalal, O., & Han, N. (2025). Llm-powered nuanced video attribute annotation for enhanced recommendations. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 1002–1005. https://doi.org/10.1145/3705328.3748103

Long, J., Chen, T., Nguyen, Q. V. H., Xu, G., Zheng, K., & Yin, H. (2023). Model-agnostic decentralized collaborative learning for on-device poi recommendation. *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 423–432. https://doi.org/10.1145/3539618.3591733

Ma, R., Lian, Y., Qi, R., Song, C., & Ge, T. (2025). Valid coverage oriented item perspective recommendation. *IEEE Transactions on Knowledge and Data Engineering*, *37*(6), 3810–3823. https://doi.org/10.1109/TKDE.2025.3547968

Mahajan, K. C., Porobo Dharwadker, A., Shah, R., Qu, S., Bang, G., & Schumitsch, B. (2023). Pie: Personalized interest exploration for large-scale recommender systems. *Companion Proceedings of the ACM Web Conference 2023*, 508–512. https://doi.org/10.1145/3543873.3584656

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.

Matsui, B. M. A., & Goya, D. H. (2022). Mlops: Five steps to guide its effective implementation, 33–34. https://doi.org/10.1145/3522664.3528611

Matsui, B. M. A., & Goya, D. H. (2023). Mlops: A guide to its adoption in the context of responsible ai, 45–49. https://doi.org/10.1145/3526073.3527591

Melchiorre, A. B., Epure, E. V., Masoudian, S., Escobedo, G., Hausberger, A., Moussallam, M., & Schedl, M. (2025). Just ask for music (jam): Multimodal and personalized

natural language music recommendation. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 615–620. https://doi.org/10.1145/3705328.3748020

Meng, C., Zhai, C., Wang, X., Liu, S., Feng, X., Hu, L., Li, X., Li, H., & Gai, K. (2025). Enhancing online video recommendation via a coarse-to-fine dynamic uplift modeling framework. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 82–92. https://doi.org/10.1145/3705328.3748070

Meng, C., Ling, H., Wang, J., Liu, Y., Zhang, S., Hong, D., Gao, M., Dalal, O., Chi, E., Hong, L., Lu, H., & Han, N. (2025). Balancing fine-tuning and rag: A hybrid strategy for dynamic llm recommendation updates, 919–922. https://doi.org/10.1145/3705328.3748105

Meng, Z., McCreadie, R., Macdonald, C., Ounis, I., Liu, S., Wu, Y., Wang, X., Liang, S., Liang, Y., Zeng, G., Liang, J., & Zhang, Q. (2020). Beta-rec: Build, evaluate and tune automated recommender systems. *Proceedings of the 14th ACM Conference on Recommender Systems*, 588–590. https://doi.org/10.1145/3383313.3411524

Microsoft. (2025). *Azure kubernetes service documentation*. Retrieved July 13, 2025, from https://learn.microsoft.com/en-gb/azure/aks/

Mondal, K. C., Biswas, N., & Saha, S. (2020). Role of machine learning in etl automation. https://doi.org/10.1145/3369740.3372778

Moreschini, S., Hästbacka, D., & Taibi, D. (2023). Mlops pipeline development: The ossara use case. *Proceedings of the 2023 International Conference on Research in Adaptive and Convergent Systems*. https://doi.org/10.1145/3599957.3606211

Myakala, P. K., & Bura, C. (2025). Robust defense mechanisms for agentic news recommenders: Mitigating data poisoning attacks. *Companion Proceedings of the ACM on Web Conference 2025*, 1696–1704. https://doi.org/10.1145/3701716.3716887

Naghiaei, M., Dehghan, M., Rahmani, H. A., Azizi, J., & Aliannejadi, M. (2024). Personalized beyond-accuracy calibration in recommendation. *Proceedings of the 2024 ACM SIGIR International Conference on Theory of Information Retrieval*, 107–116. https://doi.org/10.1145/3664190.3672507

Nagrecha, K., Liu, L., & Delgado, P. (2025). Reinforcement learning for intra- & inter-node recommender data pipeline optimization. *ACM Trans. Recomm. Syst.*, *3*(4). https://doi.org/10.1145/3704923

Nagrecha, K., Liu, L., Delgado, P., & Padmanabhan, P. (2023). Intune: Reinforcement learning-based data pipeline optimization for deep recommendation models. *Proceedings of the 17th ACM Conference on Recommender Systems*, 430–442. https://doi.org/10.1145/3604915.3608778

Nguyen Thanh, T., Quach, N. D. K., Nguyen, T. T., Huynh, T. T., Vu, V. H., Nguyen, P. L., Jo, J., & Nguyen, Q. V. H. (2023). Poisoning gnn-based recommender systems with generative surrogate-based attacks. *ACM Trans. Inf. Syst.*, *41*(3). https://doi.org/10.1145/3567420

NumFocus. (2025). *Python data analysis and manipulation tool*. Retrieved July 13, 2025, from https://pandas.pydata.org/

Page, M. J., & McKenzie, J. E. (2021). The prisma 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, *372*, n71. https://doi.org/10.1136/bmj.n71

Pallets. (2025). *Flask*. Retrieved January 25, 2025, from https://flask.palletsprojects.com/en/stable/

Petrov, A. V., & Macdonald, C. (2024). Recjpq: Training large-catalogue sequential recommenders. *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 538–547. https://doi.org/10.1145/3616855.3635821

Qu, L., Tang, N., Zheng, R., Nguyen, Q. V. H., Huang, Z., Shi, Y., & Yin, H. (2023). Semi-decentralized federated ego graph learning for recommendation. *Proceedings of the ACM Web Conference 2023*, 339–348. https://doi.org/10.1145/3543507.3583337

Raschka, S., Liu, Y. (, & Mirjalili, V. (2022). *Machine learning with pytorch and scikit-learn: Develop machine learning and deep learning models with python*. Packt.

Ricci, F., Rokach, L., & Shapira, B. (Eds.). (2022). *Recommender systems handbook* (3rd). Springer. https://doi.org/10.1007/978-1-0716-2197-4

Roy, A. S., D'Amico, E., Tragos, E., Lawlor, A., & Hurley, N. (2025). Don't get bored: Enhancing scalability and diversity in session-based slate recommendation. *ACM Trans. Recomm. Syst.*, *4*(1). https://doi.org/10.1145/3733241

Ruan, Q., Mac Namee, B., Lawlor, A., Leavy, S., & Dong, R. (2025). Rewriting bias: Automated rewriting to reduce media bias polarisation in news recommender systems [Just Accepted]. *ACM Trans. Recomm. Syst.* https://doi.org/10.1145/3767327

Saito, Y., & Joachims, T. (2021). Counterfactual learning and evaluation for recommender systems: Foundations, implementations, and recent advances. *Proceedings of the 15th ACM Conference on Recommender Systems*, 828–830. https://doi.org/10.1145/3460231.3473320

Shinoda, K., Sasai, T., & Fukushima, S. (2025). Popularity-bias vulnerability: Semi-supervised label inference attack on federated recommender systems. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 660–665. https://doi.org/10.1145/3705328.3748024

Srirama, A., Gupta, S., & Saha, R. (2025). *Kubernetes for generative ai solutions: A complete guide to designing, optimizing, and deploying generative ai workloads on kubernetes*. Packt.

Stergiopoulos, V., Tousidou, E., & Corral, A. (2024). Recommender systems based on parallel and distributed deep learning, 60–66. https://doi.org/10.1145/3635059.3635069

The Linux Foundation. (2025). *Optimized tensor library for deep learning using gpus and cpus*. Retrieved August 22, 2025, from https://pytorch.org/

Tran, H. V., Chen, T., Ye, G., Nguyen, Q. V. H., Zheng, K., & Yin, H. (2025). On-device content-based recommendation with single-shot embedding pruning: A cooperative game perspective. *Proceedings of the ACM on Web Conference 2025*, 772–785. https://doi.org/10.1145/3696410.3714921

Vasilev, A., Volodkevich, A., Kulandin, D., Bysheva, T., & Klenitskiy, A. (2024). Replay: A recommendation framework for experimentation and production use. *Proceedings of the 18th ACM Conference on Recommender Systems*, 1191–1194. https://doi.org/10.1145/3640457.3691701

Vemberly, E., Ancilla, E., Anderies, A., & Chowanda, A. (2023). Enhancing movie recommendations: A hybrid filtering approach combining collaborative and content-based filtering. *2023 International Conference on Artificial Intelligence Robotics, Signal and Image Processing (AIRoSIP)*, 330–335. https://doi.org/10.1109/AIRoSIP58759.2023.10873881

Verner, J. M., Brereton, O. P., Kitchenham, B. A., Turner, M., & Niazi, M. (2012). Systematic literature reviews in global software development: A tertiary study. *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, 2–11. https://doi.org/10.1049/ic.2012.0001

Vrijenhoek, S., Bénédict, G., Gutierrez Granada, M., & Odijk, D. (2024). Radio* – an introduction to measuring normative diversity in news recommendations. *ACM Trans. Recomm. Syst.*, *3*(1). https://doi.org/10.1145/3636465

W3C. (2025a). *Contrast (minimum) (level aa)*. Retrieved September 28, 2025, from https://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum.html

W3C. (2025b). *Perceivable guidelines*. Retrieved September 28, 2025, from https://www.w3.org/TR/WCAG21/#perceivable

WANG, C., Ye, J., Wang, W., Gao, C., Feng, F., & He, X. (2023). Recad: Towards a unified library for recommender attack and defense. *Proceedings of the 17th ACM Conference on Recommender Systems*, 234–244. https://doi.org/10.1145/3604915.3609490

Wang, R., Prabhakar, P., Srivastava, G., Wang, T., Jalali, Z. S., Bharill, V., Ouyang, Y., Nigam, A., Venugopalan, D., Gupta, A., Borisyuk, F., Keerthi, S., & Muralidharan, A. (2024). Limaml: Personalization of deep recommender models via meta learning. *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 5882–5892. https://doi.org/10.1145/3637528.3671599

Wang, S., Ouyang, T., Zhou, Y., Xiao, Q., Ren, Y., Pan, Y., Li, F., & Luo, C. (2025). Enhanced emotion-aware music recommendation via large language models. *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, 4986–4994. https://doi.org/10.1145/3711896.3737212

Wang, T.-H., Hu, X., Jin, H., Song, Q., Han, X., & Liu, Z. (2020). Autorec: An automated recommender system. *Proceedings of the 14th ACM Conference on Recommender Systems*, 582–584. https://doi.org/10.1145/3383313.3411529

Wang, Y., Ma, W., Zhang, M., Liu, Y., & Ma, S. (2023). A survey on the fairness of recommender systems. *ACM Trans. Inf. Syst.*, *41*(3). https://doi.org/10.1145/3547333

Wang, Y., He, Z., Yue, Z., McAuley, J., & Wang, D. (2025). Your causal self-attentive recommender hosts a lonely neighborhood. *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, 688–696. https://doi.org/10.1145/3701551.3703587

Wei, Y., Langer, M., Yu, F., Lee, M., Liu, J., Shi, J., & Wang, Z. (2022). A gpu-specialized inference parameter server for large-scale deep recommendation models. *Proceedings of the 16th ACM Conference on Recommender Systems*, 408–419. https://doi.org/10.1145/3523227.3546765

Wikipedia contributors. (2025a). *Cross industry standard process for data mining*. Retrieved September 21, 2025, from https://pt.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining

Wikipedia contributors. (2025b). *Model for classifying software quality attributes (functional and non-functional requirements)*. Retrieved September 21, 2025, from https://en.wikipedia.org/wiki/FURPS

Xi, Y., Wang, H., Chen, B., Lin, J., Zhu, M., Liu, W., Tang, R., Wei, Z., Zhang, W., & Yu, Y. (2025). Efficiency unleashed: Inference acceleration for llm-based recommender systems with speculative decoding. *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1891–1901. https://doi.org/10.1145/3726302.3729961

Xi, Y., Weng, M., Chen, W., Yi, C., Chen, D., Guo, G., Zhang, M., Wu, J., Jiang, Y., Liu, Q., Yu, Y., & Zhang, W. (2025). Bursting filter bubble: Enhancing serendipity recommendations with aligned large language models. *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, 5059–5070. https://doi.org/10.1145/3711896.3737199

Xie, M., Ren, K., Lu, Y., Yang, G., Xu, Q., Wu, B., Lin, J., Ao, H., Xu, W., & Shu, J. (2020). Kraken: Memory-efficient continual learning for large-scale real-time recommendations. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. https://doi.org/10.5555/3433701.3433728

Xu, X., Wu, H., Yu, W., Hu, L., Jiang, P., & Gai, K. (2024). Enhancing interpretability and effectiveness in recommendation with numerical features via learning to contrast the counterfactual samples. *Companion Proceedings of the ACM Web Conference 2024*, 453–460. https://doi.org/10.1145/3589335.3648345

Xu, Z., Zeng, H., Tan, J., Fu, Z., Zhang, Y., & Ai, Q. (2023). A reusable model-agnostic framework for faithfully explainable recommendation and system scrutability. *ACM Trans. Inf. Syst.*, *42*(1). https://doi.org/10.1145/3605357

Yu, Y., Yu, F., Sheng, X., Liu, C., & Chen, X. (2023). Eaglerec: Edge-scale recommendation system acceleration with inter-stage parallelism optimization on gpus. *Proceedings of the 60th Annual ACM/IEEE Design Automation Conference*, 1–6. https://doi.org/10.1109/DAC56929.2023.10247679

Zainulabidova, Z., Borisova, J., & Hvatov, A. (2025). Learning geometry-aware recommender systems with manifold regularization. *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, 1212–1216. https://doi.org/10.1145/3705328.3759323

Zarate, G., Lopez Osa, M. J., Torre-Bastida, A. I., Iturraspe, U., Arjona, J., Navarro, B., & Gimeno, A. (2024). Evolution of extract-transform-load (etl) processes towards data product pipelines, 25–32. https://doi.org/10.1145/3685651.3686662

Zhang, Q., Teng, Z., Wu, D., & Wang, J. (2024). An enhanced batch query architecture in real-time recommendation. *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 5078–5085. https://doi.org/10.1145/3627673.3680034

Zhang, Q., Liu, J., Dai, Y., Qi, Y., Yuan, Y., Zheng, K., Huang, F., & Tan, X. (2022). Multi-task fusion via reinforcement learning for long-term user satisfaction in recommender systems. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 4510–4520. https://doi.org/10.1145/3534678.3539040

Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *52*(1). https://doi.org/10.1145/3285029

Zhao, W. X., Lin, Z., Feng, Z., Wang, P., & Wen, J. (2023). A revisiting study of appropriate offline evaluation for top-n recommendation algorithms. *ACM Transactions on Information Systems*, *41*(2), 32:1–32:41. https://doi.org/10.1145/3545796

Zhao, Y., Wang, K., Guo, G., & Wang, X. (2022). Learning compact yet accurate generative adversarial networks for recommender systems. *Knowledge-Based Systems*, *257*, 109900. https://doi.org/10.1016/j.knosys.2022.109900

Zheng, B., Hou, Y., Zhao, W. X., Song, Y., & Zhu, H. (2023). Reciprocal sequential recommendation. *Proceedings of the 17th ACM Conference on Recommender Systems*, 89–100. https://doi.org/10.1145/3604915.3608798

Zheng, Y., Wei, Z., de Hoog, F., Chen, X., Xu, H., Ye, Y., & Huang, J. (2025). Lighter-x: An efficient and plug-and-play strategy for graph-based recommendation through decoupled propagation. *Proc. VLDB Endow.*, *18*(11), 3721–3729. https://doi.org/10.14778/3749646.3749649

Zhu, J., Cai, G., Huang, J., Dong, Z., Tang, R., & Zhang, W. (2023). Reloop2: Building self-adaptive recommendation models via responsive error compensation loop. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 5728–5738. https://doi.org/10.1145/3580305.3599785

Zhu, J., Dai, Q., Su, L., Ma, R., Liu, J., Cai, G., Xiao, X., & Zhang, R. (2022). Bars: Towards open benchmarking for recommender systems. *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2912–2923. https://doi.org/10.1145/3477495.3531723

Zzh, Zhang, W., & Wentao. (2022). Industrial solution in fashion-domain recommendation by an efficient pipeline using gnn and lightgbm, 45–49. https://doi.org/10.1145/3556702.3556850

Appendix A

Extended Tables and Figures

Table A.1: Key runtime parameters (abbreviated).

Name	Purpose	Example
CRON_SCHEDULE_EXTRACT	Extract job cadence (UTC)	0 1 * * *
TH_MICRO_NDCG_AT_10	Promotion threshold (primary selector)	0.047
TH_MACRO_NDCG_AT_10	Promotion threshold	0.042
TH_MICRO_PREC_AT_10	Promotion threshold	0.043
TH_MACRO_RECALL_AT_10	Promotion threshold	0.037
TH_COVERAGE_AT_10	Coverage guardrail	0.40
TH_FULL_LIST_RATE	Absolute value of depts receiving full K recs	0.99
TIEBREAK_DELTA_MICRO _NDCG_AT_10	Tie-break window (abs)	0.005
MIN_COVERAGE_IMPROVEMENT	Tie-break: Coverage gain (abs)	0.01
MIN_MACRO_RECALL _IMPROVEMENT	Tie-break: Macro-Recall gain (rel)	0.01
BOOTSTRAP_RESAMPLES	CI computation resamples	1000
CI_LEVEL	Confidence level	0.95
P95_LATENCY_SLO_SECONDS	Online SLO (p95 latency)	0.8
PATH_RAW	Storage layout	/data/loaded
PATH_PREP	Storage layout	/data/prep
PATH_ARTIFACTS	Storage layout	/models
IMG_SERVE_TAG	Immutable image tag used for rollout	2025-03-10_build42

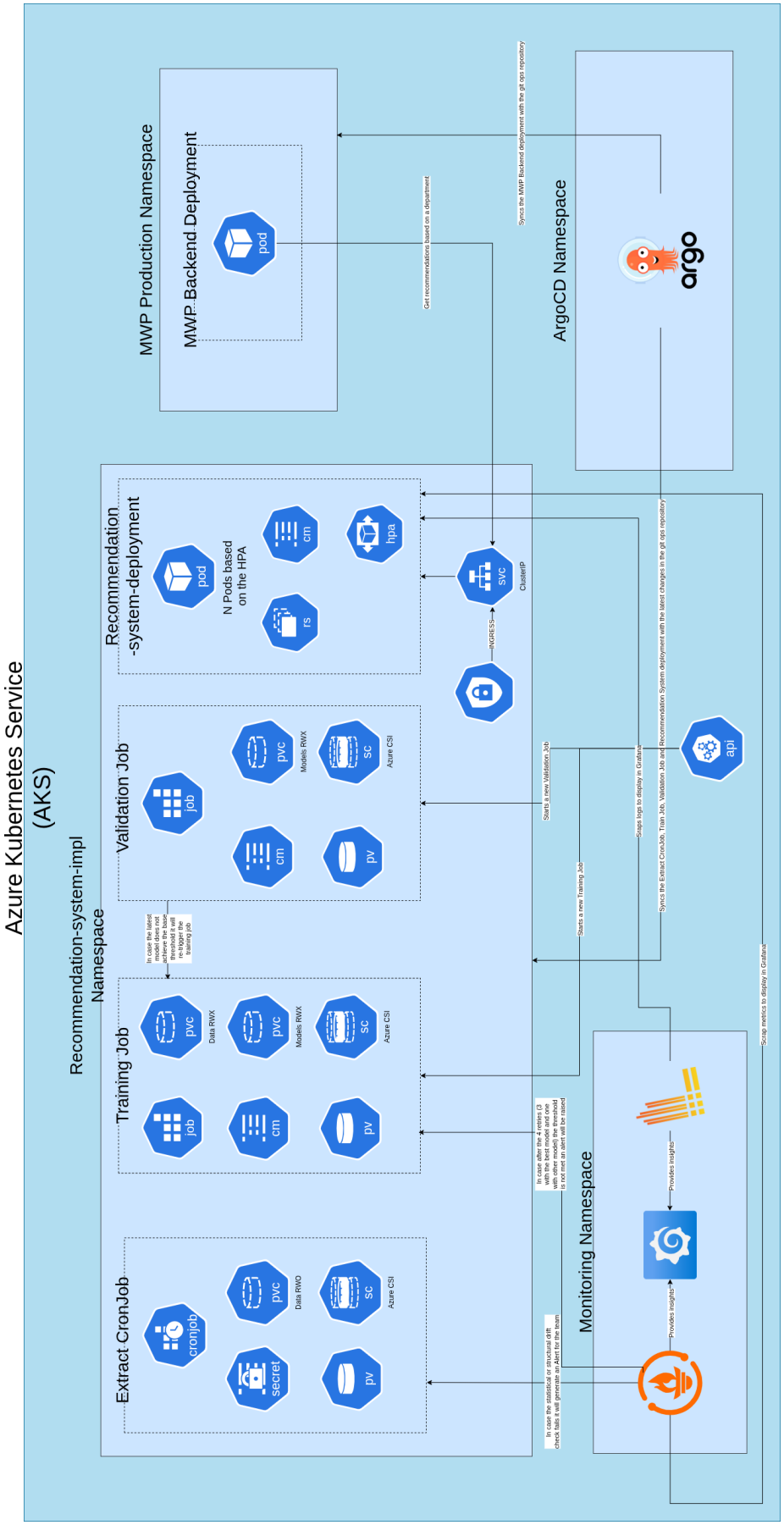

Figure A.1: Azure Kubernetes Service implementation.

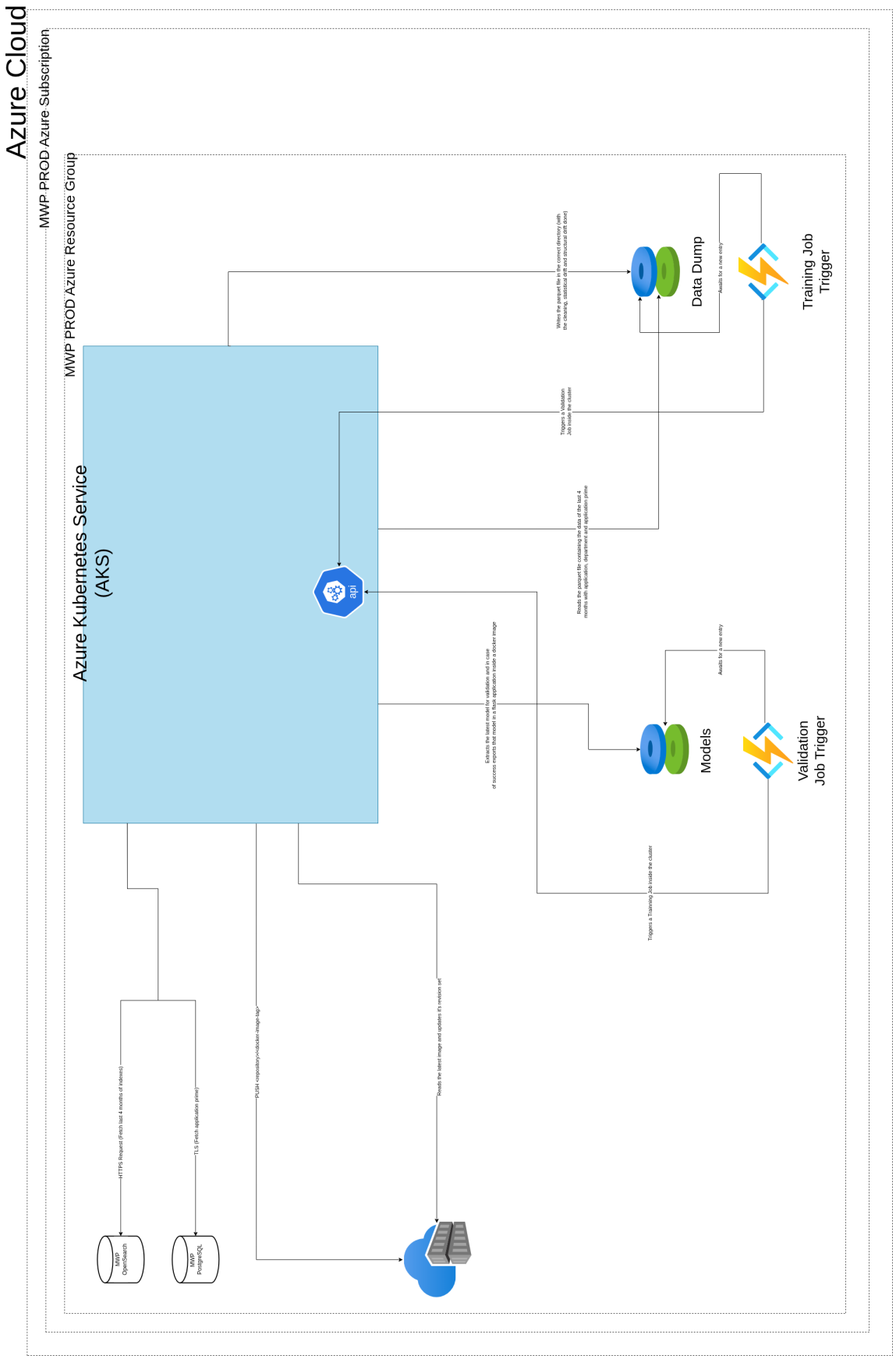

Figure A.2: Azure cloud resources.

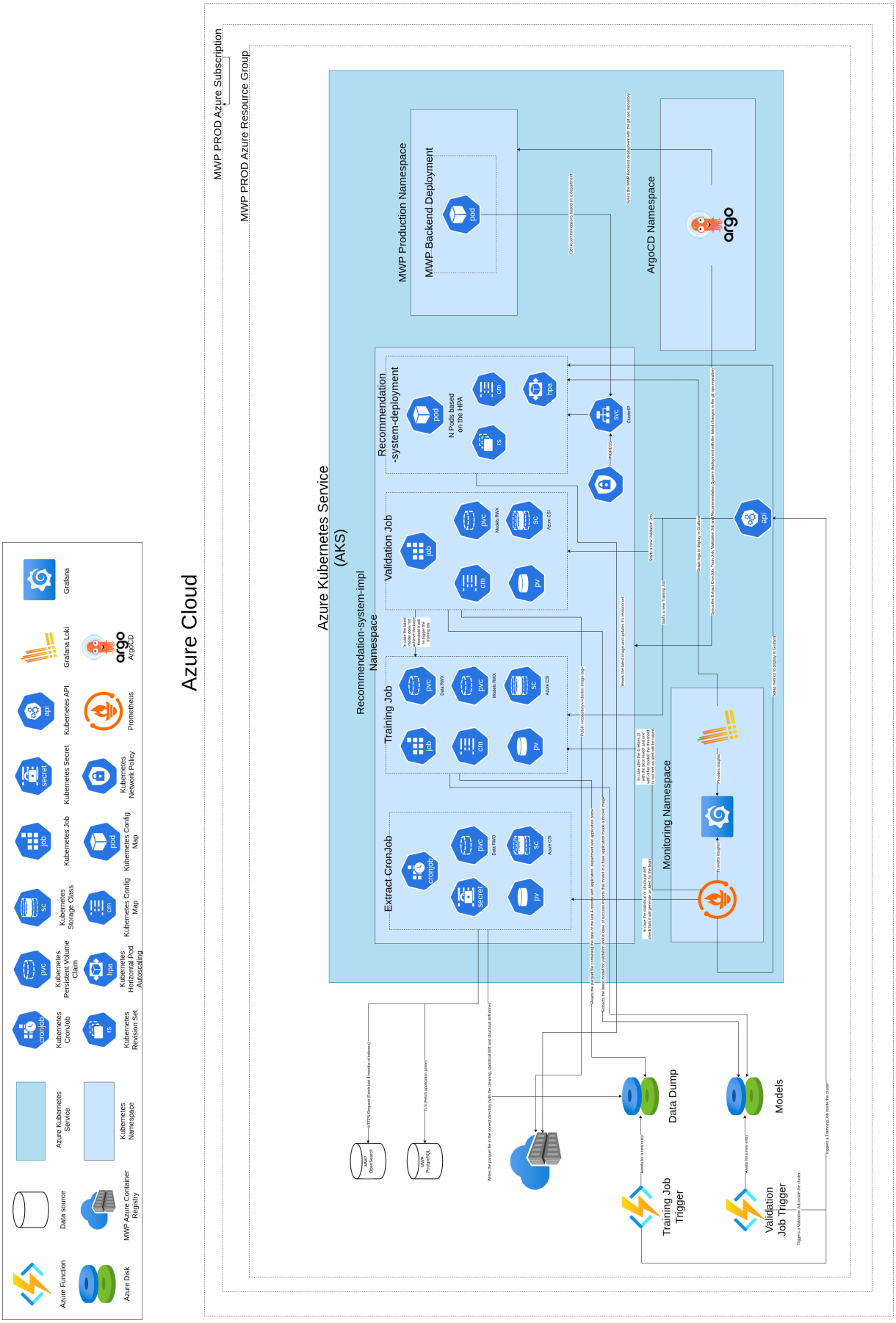

Figure A.3: Final system implementation.

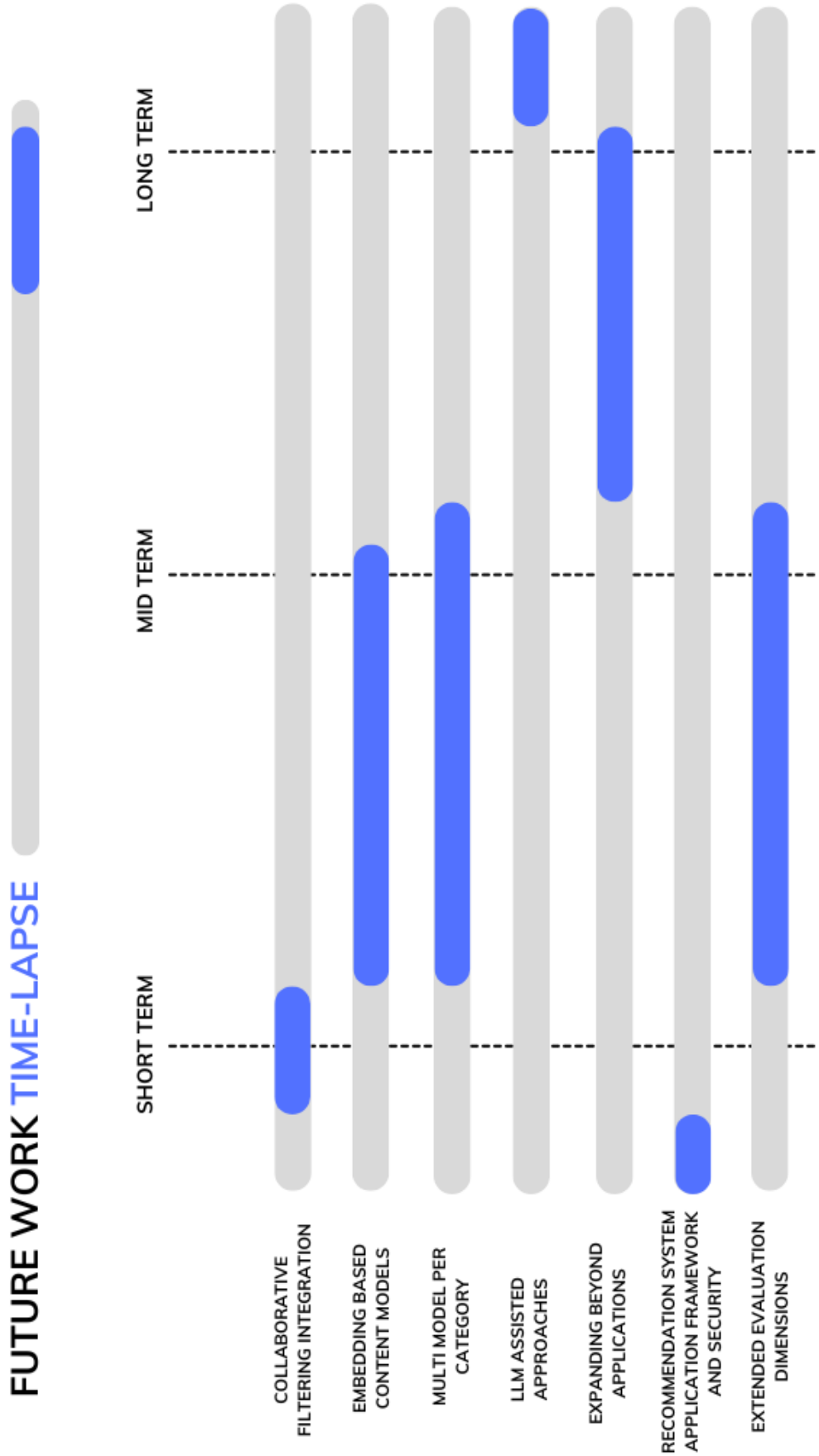

Figure A.4: Future work roadmap.

Table A.2: Promotion criteria (selector, thresholds, and tie-breaks) at K=10.

Category	**Metric**	**Requirement / Rule**
Selector & Guardrails	Primary selector	micro-NDCG@10
Selector & Guardrails	Guardrail	Coverage@10 $\geq$ 0.40
Selector & Guardrails	Tie-breaks	If within $\pm$0.005 in micro-NDCG@10, prefer higher Coverage@10 ($\geq$ +0.01 abs) or higher macro-Recall@10 ($\geq$ 1–5%).
Validation thresholds	micro-NDCG@10	$\geq$ 0.047 *and* $\geq$ 98% of best observed.
Validation thresholds	macro-NDCG@10	$\geq$ 0.042; no drop $>$ 5–8% vs best observed.
Validation thresholds	micro-Precision@10	$\geq$ 0.043; no drop $>$ 5–8%.
Validation thresholds	macro-Recall@10	$\geq$ 0.037 (avoid recall collapse on smaller depts).
Diversity / Exploration	Coverage@10	$\geq$ 0.40 and not worse than best by $>$ 0.02–0.08 abs.
Diversity / Exploration	Full list rate	$\geq$ 99% of departments must receive full K recommendations.
Sanity / Correctness	Consistency checks	No NaNs/Infs; class spaces consistent with training metadata.
Sanity / Correctness	Monotonicity	Precision@5 $\geq$ Precision@10 $\geq$ Precision@20; recall and coverage non-decreasing with K.
Stability	CI requirement	95% CI (bootstrap, n=1000) lower bound must satisfy minimal value.

Table A.3: Selected hyperparameters and promotion decision.

Model / Step	**Setting**	**Value / Notes**
Dept–Category Cosine	`use_tfidf`	`False`.
Dept–Category Cosine	Neighbors / shrink	m=20 neighbors; shrink = 3.
Dept–Category Cosine	Seen penalty	α_{seen}=0.5 (inert under filtered eval).
Dept–Category Cosine	Diversification / candidates	λ=0.9; candidate multiplier = 3.
Hybrid	Blend weight	α=0.65 (neighbor vs mixture).
Hybrid	Mixture IDF exponent	γ=0.3 (Laplace= 1.0).
Hybrid	Diversification / candidates	λ=0.8; candidate multiplier = 2.
Promotion decision	Fallback logic	If best candidate fails any threshold, retrain (up to 3 attempts). If still failing, train Hybrid as fallback; if both fail, alert and keep previous model.

Appendix B

Code Listings

```
import bibtexparser
from glob import glob

def process_bibtex_files(file_list, output_file):
all_entries = []
seen_keys = set()

for file in file_list:
print(f"Processing file: {file}")
with open(file, 'r') as bib_file:
bib_database = bibtexparser.load(bib_file)
for entry in bib_database.entries:
unique_id = entry.get('doi') or entry.get('title', '').lower()
if unique_id not in seen_keys:
seen_keys.add(unique_id)
all_entries.append(entry)

print(f"Total unique entries: {len(all_entries)}")
cleaned_database = bibtexparser.bibdatabase.BibDatabase()
cleaned_database.entries = all_entries

with open(output_file, 'w') as bib_file:
bibtexparser.dump(cleaned_database, bib_file)

print(f"Cleaned BibTeX file saved as: {output_file}")

bibtex_files = [
"file1.bib",
"file2.bib",
"file3.bib",
"file4.bib",
"file5.bib",
"file6.bib"
]

output_bibtex_file = "cleaned_references.bib"

process_bibtex_files(bibtex_files, output_bibtex_file)
```

Listing B.1: Script to remove duplicated entries

```
from opensearchpy import OpenSearch
from dotenv import load_dotenv
import os

load_dotenv()

host = os.getenv('OPEN_SEARCH_HOST')
port = os.getenv('OPEN_SEARCH_PORT')

opensearch_client = OpenSearch(
hosts=[{'host': host, 'port': port, 'scheme': 'https'}],
http_compress=True,
http_auth=(os.getenv('OPEN_SEARCH_ADMIN'), os.getenv('
 OPEN_SEARCH_PASSWORD')),
use_ssl=True,
verify_certs=False,
ssl_assert_hostname=False,
ssl_show_warn=False
)

def fetch_all_indexes():
index_pattern = "logs-myworkplace-*"
try:
response = opensearch_client.cat.indices(index=index_pattern,
 format="json", h="index")
indexes = [index['index'] for index in response]
return indexes
except Exception as e:
print(f"An error occurred: {e}")
return []
```

Listing B.2: Querying an OpenSearch index using environment variables and secure client options.

```
def fetch_data_from_indexes(indexes):
all_data = []

for index in indexes:
print(f"Fetching data from index: {index}")

query = {
  "query": { "match_all": {} },
  "size": 1000
}

response = opensearch_client.search(
index=index,
body=query,
scroll='2m'
)

scroll_id = response['_scroll_id']
total_hits = response['hits']['total']['value']
print(f"Total records in {index}: {total_hits}")
```

```
21
22  for hit in response['hits']['hits']:
23  all_data.append(hit['_source'])
24
25  while len(response['hits']['hits']):
26  response = opensearch_client.scroll(
27  scroll_id=scroll_id,
28  scroll='2m'
29  )
30  scroll_id = response['_scroll_id']
31  for hit in response['hits']['hits']:
32  all_data.append(hit['_source'])
33
34  print(f"Fetched {len(all_data)} records so far...")
35
36  return all_data
```

Listing B.3: Fetching data from multiple OpenSearch indexes with scrolling.

```
1   import psycopg2
2   import pandas as pd
3   import numpy as np
4   from dotenv import load_dotenv
5   import os
6
7   load_dotenv()
8
9   host = os.getenv('MWP_DATABASE_HOST')
10  port = os.getenv('MWP_DATABASE_PORT')
11  user = os.getenv('MWP_DATABASE_USER')
12  password = os.getenv('MWP_DATABASE_PASSWORD')
13  database = os.getenv('MWP_DATABASE_NAME')
14
15  def connect_to_db():
16  try:
17  connection = psycopg2.connect(
18  host=host,
19  user=user,
20  password=password,
21  dbname=database,
22  port=port
23  )
24  return connection
25  except psycopg2.Error as e:
26  raise Exception(f"Error connecting to PostgreSQL Database: {e}")
```

Listing B.4: Connecting to a PostgreSQL database with environment variables.

```
import numpy as np
import pandas as pd

def split_per_department(df, test_size=0.2, min_records_per_dept=5, random_state=42):
rng = np.random.default_rng(seed=random_state)
parts_train, parts_test = [], []
for dept, g in df.groupby('department', sort=False):
if len(g) < min_records_per_dept:
continue
n_test = max(1, int(len(g) * test_size))
idx = g.index.to_numpy()
test_idx = rng.choice(idx, size=n_test, replace=False)
train_idx = np.setdiff1d(idx, test_idx, assume_unique=True)
parts_train.append(df.loc[train_idx])
parts_test.append(df.loc[test_idx])
train_df = pd.concat(parts_train, ignore_index=True) if parts_train else df.iloc[0:0].copy()
test_df  = pd.concat(parts_test,  ignore_index=True) if parts_test  else df.iloc[0:0].copy()
print(f"Train: {len(train_df)} | Test: {len(test_df)} | Common depts: "
f"{len(set(train_df['department']).intersection(set(test_df['department'])))}")
return train_df, test_df
```

Listing B.5: Train/test split that preserves department overlap and minimum activity.

```
import numpy as np

def make_tfidf_dept_category(dept_cat_counts, smooth=1.0):
A = dept_cat_counts.copy()
A += smooth                                    # Laplace
tf  = A.div(A.sum(axis=1), axis=0)             # TF (row-normalize)
N   = A.shape[0]
dfc = (A > 0).sum(axis=0)                      # dept-frequency per category
idf = np.log((1 + N) / (1 + dfc)) + 1.0        # IDF
tfidf = tf.multiply(idf, axis=1)
tfidf = tfidf.div(tfidf.sum(axis=1), axis=0).fillna(0.0)  # re-normalize rows
return tfidf, idf
```

Listing B.6: TF-IDF basis for dept–category profiles; rows can replace smoothed frequencies before cosine similarity.

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
import pandas as pd

class StrongContentRecommender:
def __init__(self, laplace=1.0, sim_top_m=20,
 shrinkage_min_overlap=5,
alpha_seen_penalty=0.5, mmr_lambda=0.7, candidate_multiplier=3):
self.laplace = laplace; self.sim_top_m = sim_top_m
self.shrinkage_min_overlap = shrinkage_min_overlap
self.alpha_seen_penalty = alpha_seen_penalty
self.mmr_lambda = mmr_lambda; self.candidate_multiplier =
 candidate_multiplier

def fit(self, train_df):
self.train_df = train_df.copy()
self.dept_cat_counts = (train_df.groupby(['department','
 prime_name'])
.size().unstack(fill_value=0).astype(float))
smoothed = self.dept_cat_counts + self.laplace
self.dept_cat_norm = smoothed.div(smoothed.sum(axis=1), axis=0)
 # or TF-IDF rows
self.dept_app_counts = (train_df.groupby(['department','
 action_info'])
.size().unstack(fill_value=0).astype(float))
self.global_app_pop = train_df['action_info'].value_counts().
 astype(float)

def _similar_departments(self, target_dept):
if target_dept not in self.dept_cat_norm.index: return []
target = self.dept_cat_norm.loc[[target_dept]]
sims   = cosine_similarity(target, self.dept_cat_norm).flatten()
all_d  = np.array(self.dept_cat_norm.index)
def nonzero_cats(d):
row = self.dept_cat_counts.loc[d]; return set(row.index[row.
 values > 0])
nz_t   = nonzero_cats(target_dept)
overlaps = np.array([len(nz_t.intersection(nonzero_cats(d))) for
 d in all_d], dtype=float)
denom = np.log1p(self.shrinkage_min_overlap); shrink = np.log1p(
 overlaps) / (denom if denom>0 else 1.0)
sims *= shrink
mask = all_d != target_dept; neigh = all_d[mask]; sims = sims[
 mask]
top = np.argsort(-sims)[:self.sim_top_m]
return list(zip(neigh[top], sims[top]))

def _seen_apps(self, dept):
if dept not in self.dept_app_counts.index: return set()
row = self.dept_app_counts.loc[dept]; return set(row.index[row.
 values>0])

def _category_sim_for_apps(self, apps):
```

```
43  app2cat = (self.train_df.drop_duplicates(['action_info','
     prime_name'])
44  .set_index('action_info')['prime_name'].to_dict())
45  sim = {a:{} for a in apps}
46  for i,a in enumerate(apps):
47  ca = app2cat.get(a); sim[a][a]=1.0
48  for b in apps[i+1:]:
49  cb = app2cat.get(b); s = 1.0 if (ca is not None and cb is not
     None and ca==cb) else 0.0
50  sim[a][b]=s; sim[b][a]=s
51  return sim
52
53  def _mmr(self, candidates, rel_scores, sim_dict, top_k):
54  sel, cand = [], set(candidates)
55  while cand and len(sel) < top_k:
56  best, best_s = None, -1e18
57  for c in cand:
58  rel = rel_scores.get(c, 0.0)
59  div = 0.0 if not sel else max(sim_dict.get(c,{}).get(s,0.0) for s
      in sel)
60  s   = self.mmr_lambda*rel - (1.0-self.mmr_lambda)*div
61  if s>best_s: best, best_s = c, s
62  sel.append(best); cand.remove(best)
63  return sel
64
65  def raw_neighbor_scores(self, target_dept):
66  neigh = self._similar_departments(target_dept)
67  if not neigh: return pd.Series(0.0, index=self.dept_app_counts.
     columns)
68  agg = None
69  for d, w in neigh:
70  if w<=0 or d not in self.dept_app_counts.index: continue
71  row = self.dept_app_counts.loc[d]*w
72  agg = row if agg is None else agg.add(row, fill_value=0.0)
73  if agg is None: return pd.Series(0.0, index=self.dept_app_counts.
     columns)
74  if target_dept in self.dept_app_counts.index:
75  seen_row = self.dept_app_counts.loc[target_dept]
76  agg = (agg - self.alpha_seen_penalty*seen_row).clip(lower=0.0)
77  return agg.reindex(self.dept_app_counts.columns).fillna(0.0)
78
79  def recommend(self, target_dept, top_k=10):
80  if target_dept not in self.dept_cat_norm.index:
81  return list(self.global_app_pop.index[:top_k])
82  agg = self.raw_neighbor_scores(target_dept)
83  seen = self._seen_apps(target_dept)
84  cands = [a for a in agg.sort_values(ascending=False).index if a
     not in seen]
85  cands = cands[:max(top_k*self.candidate_multiplier, top_k)]
86  if not cands: return list(self.global_app_pop.index[:top_k])
87  sim = self._category_sim_for_apps(cands)
88  rel = {a: float(agg.get(a,0.0)) for a in cands}
89  return self._mmr(cands, rel, sim, top_k)
```

Listing B.7: Neighbor-based content recommender using dept–category profiles, cosine similarity, seen-item penalty, and MMR diversity.

```
import numpy as np
import pandas as pd

class CategoryMixtureScorer:
def __init__(self, laplace=1.0, idf_gamma=0.5):
self.laplace = laplace; self.idf_gamma = idf_gamma

def fit(self, train_df):
self.train_df = train_df.copy()
dept_cat = (train_df.groupby(['department','prime_name'])
.size().unstack(fill_value=0).astype(float))
sm = dept_cat + self.laplace
self.p_c_given_dept = sm.div(sm.sum(axis=1), axis=0)
cat_app = (train_df.groupby(['prime_name','action_info'])
.size().unstack(fill_value=0).astype(float))
sm_ca = cat_app + self.laplace
self.p_app_given_c = sm_ca.div(sm_ca.sum(axis=1), axis=0).fillna
 (0.0)
N = dept_cat.shape[0]; dfc = (dept_cat>0).sum(axis=0)
idf = np.log((1+N)/(1+dfc)) + 1.0
self.cat_weight = (idf**self.idf_gamma) if self.idf_gamma>0 else
 (idf*0+1.0)
self.categories = list(self.p_app_given_c.index)
self.seen_map = train_df.groupby('department')['action_info'].
 apply(set).to_dict()

def score_apps_for_dept(self, dept):
if dept not in self.p_c_given_dept.index:
global_cat = (self.train_df['prime_name'].value_counts() + self.
 laplace)
p_c = (global_cat / global_cat.sum()).reindex(self.categories).
 fillna(0.0)
else:
p_c = self.p_c_given_dept.loc[dept].reindex(self.categories).
 fillna(0.0)
w = self.cat_weight.reindex(self.categories).fillna(1.0)
p_c_w = p_c * w; s = p_c_w.sum()
if s>0: p_c_w /= s
scores = np.dot(p_c_w.values, self.p_app_given_c.values)
return pd.Series(scores, index=self.p_app_given_c.columns)

def recommend(self, dept, top_k=10):
scores = self.score_apps_for_dept(dept)
seen = self.seen_map.get(dept, set())
recs = [a for a in scores.sort_values(ascending=False).index if a
  not in seen]
return recs[:top_k]
```

Listing B.8: Pure content model: scores apps via P(c|dept)×P(app|c), with optional category IDF weighting.

```
class ContentOnlyHybrid:
def __init__(self, base_neighbor_model, mixture_model,
blend_alpha=0.65, mmr_lambda=0.8, candidate_multiplier=3):
self.base = base_neighbor_model
self.mix  = mixture_model
self.blend_alpha = blend_alpha
self.mmr_lambda = mmr_lambda
self.candidate_multiplier = candidate_multiplier

def _mmr(self, candidates, rel_scores, sim_dict, top_k):
sel, cand = [], set(candidates)
while cand and len(sel) < top_k:
best, best_s = None, -1e18
for c in cand:
rel = rel_scores.get(c, 0.0)
div = 0.0 if not sel else max(sim_dict.get(c,{}).get(s,0.0) for s
  in sel)
s   = self.mmr_lambda*rel - (1.0-self.mmr_lambda)*div
if s>best_s: best, best_s = c, s
sel.append(best); cand.remove(best)
return sel

def recommend(self, dept, top_k=10):
agg = self.base.raw_neighbor_scores(dept)
    # neighbor relevance
mix = self.mix.score_apps_for_dept(dept).reindex(agg.index).
 fillna(0.0)
scores = self.blend_alpha*agg + (1.0-self.blend_alpha)*mix
    # blend
seen = self.base._seen_apps(dept)
cands = [a for a in scores.sort_values(ascending=False).index if
 a not in seen]
cands = cands[:max(top_k*self.candidate_multiplier, top_k)]
if not cands: return list(self.base.global_app_pop.index[:top_k])
sim = self.base._category_sim_for_apps(cands)
rel = {a: float(scores.get(a,0.0)) for a in cands}
return self._mmr(cands, rel, sim, top_k)
```

Listing B.9: Blend of neighbor (dept–category cosine) and pure content mixture + MMR.

```
class GlobalPopularity:
def fit(self, train_df):
self.pop  = train_df['action_info'].value_counts()
self.seen = train_df.groupby('department')['action_info'].apply(
 set).to_dict()
def recommend(self, dept, top_k=10):
seen = self.seen.get(dept, set())
return [a for a in self.pop.index if a not in seen][:top_k]

class CategoryPopularity:
def __init__(self, top_cats=3, laplace=1.0):
self.top_cats = top_cats; self.laplace = laplace
def fit(self, train_df):
```

```
13  self.train_df = train_df.copy()
14  self.seen = train_df.groupby('department')['action_info'].apply(
     set).to_dict()
15  dept_cat = train_df.groupby(['department','prime_name']).size().
     unstack(fill_value=0).astype(float)
16  sm = dept_cat + self.laplace; self.p_c = sm.div(sm.sum(axis=1),
     axis=0)  # P(cat|dept)
17  self.cat_apps = (train_df.groupby(['prime_name','action_info'])
18  .size()
19  .groupby(level=0, group_keys=False)
20  .apply(lambda s: s.sort_values(ascending=False)))
21  def recommend(self, dept, top_k=10):
22  if dept not in self.p_c.index:
23  gp = GlobalPopularity(); gp.fit(self.train_df); return gp.
     recommend(dept, top_k)
24  cats = self.p_c.loc[dept].sort_values(ascending=False).index[:
     self.top_cats].tolist()
25  pool = []
26  for c in cats:
27  if c in self.cat_apps.index.get_level_values(0):
28  pool.extend(list(self.cat_apps.loc[c].index))
29  seen = self.seen.get(dept, set())
30  deduped, seen_set = [], set()
31  for a in pool:
32  if a not in seen and a not in seen_set:
33  seen_set.add(a); deduped.append(a)
34  return deduped[:top_k]
```

Listing B.10: Global popularity and category-popularity (top categories per dept).

```
1  import random
2
3  class RandomRecommender:
4  def fit(self, train_df):
5  self.items = train_df['action_info'].unique().tolist()
6  self.seen  = train_df.groupby('department')['action_info'].apply(
     set).to_dict()
7  self._rng  = random.Random(42)
8  def recommend(self, dept, top_k=10):
9  pool = [a for a in self.items if a not in self.seen.get(dept, set
     ())]
10  if not pool: return []
11  self._rng.shuffle(pool)
12  return pool[:top_k]
```

Listing B.11: Uniform over catalog minus train-seen; fixed RNG for repeatability.

```
import numpy as np

def precision_at_k(actual_set, recs, k):
recs_k = recs[:k]
return len(actual_set.intersection(recs_k)) / max(1, k)

def recall_at_k(actual_set, recs, k):
recs_k = recs[:k]
return len(actual_set.intersection(recs_k)) / max(1, len(
 actual_set))

def average_precision_at_k(actual_set, recs, k):
score, hits = 0.0, 0
for i, r in enumerate(recs[:k], start=1):
if r in actual_set:
hits += 1; score += hits / i
return score / max(1, min(len(actual_set), k))

def dcg_at_k(recs, actual_set, k):
dcg = 0.0
for i, r in enumerate(recs[:k], start=1):
dcg += (1.0 if r in actual_set else 0.0) / np.log2(i + 1)
return dcg

def ndcg_at_k(actual_set, recs, k):
ideal = sum(1.0 / np.log2(i + 2) for i in range(min(len(
 actual_set), k)))
return 0.0 if ideal==0 else dcg_at_k(recs, actual_set, k) / ideal
```

Listing B.12: Precision, recall, MAP, NDCG at K.

```
import pandas as pd
import numpy as np

def evaluate_recommender(model, train_df, test_df, k_list
 =(5,10,20)):
rows = []; coverage_sets = {k: set() for k in k_list}
item_space = set(train_df['action_info'].unique())
seen_map = train_df.groupby('department')['action_info'].apply(
 set).to_dict()

for dept, g in test_df.groupby('department', sort=False):
actual = set(g['action_info'].unique())
if not actual: continue
recs = [r for r in model.recommend(dept, top_k=max(k_list))
if r not in seen_map.get(dept, set())]
row = {'department': dept, 'n_actual': len(actual)}
for k in k_list:
row[f'precision@{k}'] = precision_at_k(actual, recs, k)
row[f'recall@{k}']    = recall_at_k(actual, recs, k)
row[f'map@{k}']       = average_precision_at_k(actual, recs, k)
row[f'ndcg@{k}']      = ndcg_at_k(actual, recs, k)
coverage_sets[k].update(recs[:k])
```

```
    rows.append(row)

    metrics_df = pd.DataFrame(rows)
    macro = {f'macro_{c}': metrics_df[c].mean()
      for c in metrics_df.columns if c.startswith(('precision@','recall@','map@','ndcg@'))}
    micro = {}
    if not metrics_df.empty:
    w = metrics_df['n_actual'].values
    for c in [col for col in metrics_df.columns if c.startswith(('precision@','recall@','map@','ndcg@'))]:
    micro[f'micro_{c}'] = np.average(metrics_df[c].values, weights=w)
    coverage = {f'coverage@{k}': len(coverage_sets[k]) / max(1, len(item_space)) for k in k_list}
    return metrics_df, {**macro, **micro, **coverage}
```

Listing B.13: Macro/micro averages and catalog coverage at K.

```
from copy import deepcopy
import pandas as pd

def run_experiments(train_df, test_df, k_list=(5,10,20)):
experiments = []

gp = GlobalPopularity(); gp.fit(train_df)
m_df, summ = evaluate_recommender(gp, train_df, test_df, k_list=k_list)
experiments.append(("Content: Global Popularity", m_df, summ))

cp = CategoryPopularity(top_cats=3, laplace=1.0); cp.fit(train_df)
m_df, summ = evaluate_recommender(cp, train_df, test_df, k_list=k_list)
experiments.append(("Content: Category Popularity (3)", m_df, summ))

sc_cos = StrongContentRecommender(laplace=1.0, sim_top_m=30, shrinkage_min_overlap=5,
alpha_seen_penalty=0.6, mmr_lambda=0.8, candidate_multiplier=3)
sc_cos.use_tfidf = False; sc_cos.fit(train_df)
m_df, summ = evaluate_recommender(sc_cos, train_df, test_df, k_list=k_list)
experiments.append(("Content: Dept-Category Cosine (Smoothed)", m_df, summ))

sc_tfidf = deepcopy(sc_cos); sc_tfidf.use_tfidf = True
m_df, summ = evaluate_recommender(sc_tfidf, train_df, test_df, k_list=k_list)
experiments.append(("Content: Dept-Category Cosine (TF-IDF)", m_df, summ))

mix = CategoryMixtureScorer(laplace=1.0, idf_gamma=0.6); mix.fit(train_df)
```

```
m_df, summ = evaluate_recommender(mix, train_df, test_df, k_list=
    k_list)
experiments.append(("Content: Category Mixture (IDF^0.6)", m_df,
    summ))

hyb = ContentOnlyHybrid(base_neighbor_model=sc_tfidf,
    mixture_model=mix,
blend_alpha=0.65, mmr_lambda=0.8, candidate_multiplier=3)
m_df, summ = evaluate_recommender(hyb, train_df, test_df, k_list=
    k_list)
experiments.append(("Content: Hybrid (Cosine TF-IDF + Mixture)",
    m_df, summ))

rows = []
for name, _, summ in experiments:
row = {'Model': name}
row.update({k: float(v) for k, v in summ.items()})
rows.append(row)
summary_table = pd.DataFrame(rows).set_index('Model')
return experiments, summary_table
```

Listing B.14: Runs baselines, cosine (smoothed/TF-IDF), mixture, and hybrid; returns a summary DataFrame.

```
{
  "category": "Application",
  "action": "Open",
  "action_info": "app_12345",
  "department": "AA-71",
  "prime_name": "prime_4",
  "create_timestamp": "2025-01-10T08:05:23Z",
  "device_type": "1",
  "browser_name": "chrome"
}
```

Listing B.15: Illustrative OpenSearch JSON document (synthetic).

```
root
|-- department: string (nullable = false)
|-- action_info: string (nullable = false)
|-- prime_name: string (nullable = false)
|-- day_of_week: integer (nullable = true)
|-- time_period: string (nullable = true)
|-- ...
```

Listing B.16: Resulting Parquet schema (excerpt).

Appendix C

Kubernetes Manifests

```
apiVersion: v1
kind: Namespace
metadata:
name: recommendation-system-impl
```

Listing C.1: Namespace definition for the PoC deployment.

```
apiVersion: batch/v1
kind: CronJob
metadata:
name: rs-extract-cron
namespace: recommendation-system-impl
spec:
schedule: "0 1 * * *" # 01:00 UTC
startingDeadlineSeconds: 600
successfulJobsHistoryLimit: 3
failedJobsHistoryLimit: 3
jobTemplate:
spec:
backoffLimit: 2
template:
spec:
serviceAccountName: <sa-extract>
restartPolicy: OnFailure
containers:
- name: extract
image: mwpAct/mwp-recommendation-system-extract-job:latest
args: ["--since=1d", "--out=/data/loaded"]
envFrom:
- secretRef:
   name: <opensearch-secrets>
- configMapRef:
  name: <extract-config>
volumeMounts:
- name: data
mountPath: /data
volumes:
- name: data
persistentVolumeClaim:
claimName: pvc-data
```

Listing C.2: CronJob for daily extract.

```
apiVersion: batch/v1
kind: Job
metadata:
name: rs-train-job
namespace: recommendation-system-impl
spec:
template:
spec:
restartPolicy: Never
containers:
- name: train
image: mwpAct/mwp-recommendation-system-training-job:latest
envFrom:
- configMapRef:
    name: rs-train-config
- secretRef:
  name: rs-db-credentials
volumeMounts:
-  name: data,
   mountPath: /data
-  name: models,
   mountPath: /models
volumes:
- name: data,
  persistentVolumeClaim:
    claimName: pvc-data
- name: models,
  persistentVolumeClaim:
    claimName: pvc-models
```

Listing C.3: Job for model training.

```
apiVersion: batch/v1
kind: Job
metadata:
name: rs-validate-job
namespace: recommendation-system-impl
spec:
template:
spec:
restartPolicy: Never
containers:
- name: validate
image: mwpAct/mwp-recommendation-system-validation-job:latest
envFrom:
- configMapRef:
    name: rs-validate-thresholds
volumeMounts:
- name: models,
  mountPath: /models
volumes:
- name: models,
  persistentVolumeClaim:
```

```
      claimName: pvc-models
```

Listing C.4: Job for model validation.

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: rs-serve
namespace: recommendation-system-impl
spec:
replicas: 1
selector:
matchLabels:
app: rs-serve
template:
metadata:
labels:
app: rs-serve
spec:
containers:
- name: api
image: mwpAct/mwp-recommendation-system:latest
ports:
- containerPort: 8000
envFrom:
- configMapRef:
    name: rs-serve-config
- secretRef:
    name: rs-api-credentials
readinessProbe:
httpGet:
  path: /healthz
  port: 8000
initialDelaySeconds: 5
periodSeconds: 10
livenessProbe:
httpGet:
  path: /healthz
  port: 8000
initialDelaySeconds: 15
periodSeconds: 20
resources:
requests:
  cpu: { .Values.requests.cpuValue }
  memory:  { .Values.requests.memoryValue }
limits:
  cpu: { .Values.limits.cpuValue }
  memory: { .Values.limits.memoryValue }
```

Listing C.5: Deployment for the API with probes and resources.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
name: rs-serve-hpa
namespace: recommendation-system-impl
spec:
scaleTargetRef:
apiVersion: apps/v1
kind: Deployment
name: rs-serve
minReplicas: { .Values.replicaSet.minimum }
maxReplicas: { .Values.replicaSet.maximum }
metrics:
- type: Resource
resource:
name: cpu
target:
type: Utilization
averageUtilization: { .Values.cpu.thresholdUtilization }
```

Listing C.6: Scale API between 1 and 3 replicas.

```
apiVersion: v1
kind: Service
metadata:
name: rs-serve-clusterip
namespace: recommendation-system-impl
spec:
type: ClusterIP
selector:
app: rs-serve
ports:
- port: 80
targetPort: 8000
```

Listing C.7: ClusterIP service for the API.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
name: rs-serve-allow-selected
namespace: recommendation-system-impl
spec:
podSelector:
matchLabels:
app: rs-serve
policyTypes: ["Ingress"]
ingress:
- from:
- podSelector:
    matchLabels:
      app: mwp-backend
- namespaceSelector:
  matchLabels:
```

```
18        name: monitoring
19    ports:
20    - protocol: TCP
21    port: 80
```

Listing C.8: Restrict API access to MWP backend and Prometheus.

```
1   apiVersion: v1
2   kind: PersistentVolume
3   metadata:
4   name: pv-data
5   spec:
6   capacity:
7   storage: { .Values.storage.dumpDiskStorage }
8   accessModes: ["ReadWriteOnce"]
9   storageClassName: managed-csi
10  persistentVolumeReclaimPolicy: Retain
11  csi:
12  driver: disk.csi.azure.com
13  volumeHandle: {{ .Values.disks.modelsDiskId }}
```

Listing C.9: Data Dump Persistent Volume.

```
1   apiVersion: v1
2   kind: PersistentVolume
3   metadata:
4   name: pv-models
5   spec:
6   capacity:
7   storage: {{ .Values.storage.modelRegistryStorage }}
8   accessModes: ["ReadWriteOnce"]
9   storageClassName: managed-csi
10  persistentVolumeReclaimPolicy: Retain
11  csi:
12  driver: disk.csi.azure.com
13  volumeHandle: {{ .Values.disks.dataDiskId }}
```

Listing C.10: Model Registry Persistent Volume.