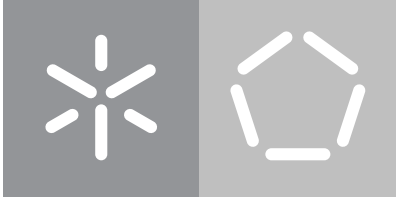
Universidade do Minho
Escola de Engenharia

Afonso Rafael Carvalho Sousa

Orchestrator selection process for cloud-native machine learning experimentation

February, 2022



Universidade do Minho
Escola de Engenharia

Afonso Rafael Carvalho Sousa

Orchestrator selection process for cloud-native machine learning experimentation

Master's Dissertation
Integrated Master's in Informatics Engineering

Work supervised by
João Miguel Lobo Fernandes
André Leite Ferreira

February, 2022

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

_____, _____

(Location)

Assinado por : **Afonso Rafael Carvalho Sousa**
(Num. de Identificação: BI14732180
Data: 2022.02.22 18:56:42+00'00'

(Afonso Rafael Carvalho Sousa)

Acknowledgements

I would like to acknowledge the contributions of all people who helped me during the course of this masters degree. First, I must give my thanks to my university supervisor Professor João Miguel Lobo Fernandes and co-supervisor Professor André Leite Ferreira from University of Minho for their support and availability during the course of this dissertation.

I also want to express my gratitude to the people at Bosch Car Multimedia S.A. for their welcome and support, and to Sascha Lange for his guidance and suggestions, resourcefulness, and ensuring I was well integrated with the teams at B52 and had access to any material and compute infrastructure necessary for this thesis. I must also express my thanks to Raja from Bosch India (RBEI) for this support over several frustrating weeks during the first deployment of Kubeflow, and to all from across company locations who took out of their time and participated during the interviews.

Lastly, a big thank you to all the professors and colleagues from the University of Minho who guided me through my academic course, with a special mention to Professor José Nuno Oliveira for his availability and support over the years, plus for the opportunity to take part in the organization of the 2019 Formal Methods World Congress, and giving me a ride to the venue after my car broke down.

Abstract

Machine learning (ML) model development is a very experimental, repetitive, and error prone task, because ML is itself very obscure - there is no way to know what model works best for our goals beforehand, so practitioners have an incentive to experiment with as many models, approaches and techniques as they can. Additionally, going from raw data to a well adjusted model is a delicate process that often requires complex, multi-step pipelines. Combine the two factors and it becomes apparent how easy it is to get lost within a sea of artifacts and results without a well defined process, hindering the development process with poor reusability, lots of technical debt, and integration-hell. This makes adherence to best practices - MLOps - paramount.

However, with the recent boom experienced in this field came a plethora of different tools and services, all trying to satisfy different subsets of needs of the model life cycle, meaning that, more often than not, ML practitioners do not know what the best set of tools for their use case might be. The experimental nature of ML means we should indeed try different tools, but there is a high risk that it might not fit the necessary requirements, generating needless costs. One particularly relevant type of tool is the orchestrator - a central piece of the experimentation process which controls the communication and execution of the components of a model pipeline.

This work follows the creation process for an enterprise ML cloud environment, with particular focus on the selection of an adequate orchestrator for cloud-native setups. Additionally, it presents MetaTool, a web application designed to speed up future tool selection processes by leveraging knowledge gathered during previous instances.

Finally, it reaches two key conclusions: first, broader organizational factors that might seem out of scope can influence or even alter the final choice, and second, although using a tool like MetaTool might speed up the decision-making process, it requires significant organizational commitment.

Keywords: Kubeflow, Kubernetes, Machine learning, MLOps, Orchestration.

Resumo

O desenvolvimento de modelos de machine learning (ML) é uma atividade muito experimental, repetitiva e propícia a erros, porque ML é muito obscura - não há forma de saber de antemão qual o modelo mais adequado para os nossos objetivos, pelo que os praticantes têm um incentivo para experimentar com o maior número possível de modelos, abordagens e técnicas que conseguirem. Adicionalmente, passar de dados para um modelo bem ajustado é um processo delicado que frequentemente requer pipelines complexas e com vários passos. Combinando os dois fatores fica aparente o quão fácil é ficar perdido num mar de artefactos e resultados sem um processo bem definido, dificultando o processo de desenvolvimento com fraca capacidade de reutilização, muita *technical debt*, e *integration hell*. Isto torna a adesão às melhores práticas - MLOps - imperativa.

Contudo, com o recente avanço verificado neste domínio veio uma abundância de diferentes ferramentas e serviços, todos tentando satisfazer diferentes subconjuntos de necessidades do ciclo de vida dos modelos, pelo que os praticantes de ML acabam frequentemente na dúvida de qual poderá ser o melhor conjunto de ferramentas para os seus casos de uso. A natureza experimental de ML faz com que se devam experimentar diferentes ferramentas, mas há um grande risco de escolher algo não satisfaça os requisitos necessários, levando a custos desnecessários. Uma categoria de ferramentas particularmente relevantes são os orquestradores - uma peça central no processo de experimentação que controla a comunicação e execução dos componentes da pipeline do modelo.

Este trabalho acompanha a criação dum ambiente cloud industrial para ML, com particular foco na escolha do orquestrador adequado para ambientes na nuvem. Adicionalmente, apresenta MetaTool, uma aplicação web pensada para acelerar futuros processos de tomada de decisão empregando conhecimento adquirido durante processos anteriores.

Finalmente, alcança duas conclusões chave: primeiro, fatores organizacionais aparentemente irrelevantes podem influenciar ou até alterar a escolha final, e segundo, apesar de ferramentas como MetaTool poderem acelerar o processo de tomada de decisão, requerem um empenho da organização.

Palavras-chave: Kubeflow, Kubernetes, Machine learning, MLOps, Orquestração.

Contents

List of Figures

List of Tables

Glossary

cluster Set of computers that work together so that they can be viewed and used as a single, atomic system.

continuous integration (CT) Process of automating the build and testing of code and artifacts every time a team member commits changes to version control.

continuous delivery (CD) Process to build, test, configure, and deploy from a development or build environment to a production environment.

continuous training (CI) Process of automating the retraining of models to adapt to changes that occur in the data.

domain specific language (DSL) Programming language created to be used for a specific application domain. Uses concepts and rules from its application domain.

grid computing Decentralized computing resources executing non-interactive workloads to reach a common goal.

manifest Specification of the desired state of a Kubernetes API object in JSON or YAML format.

mutual TLS Method for two-way authentication, ensures parties at each end of a network connection are who they claim to be.

Acronyms

AAD Azure Active Directory

AKS Azure Kubernetes Service

API Application Programming Interface

AWS Amazon Web Services

CI/CD Continuous Integration/Continuous Delivery

CRUD Create, Read, Update, Delete

DAG Directed Acyclic Graph

ETL Extract, Transform, Load

GDPR General Data Protection Regulation

KCD Kubeflow Central Dashboard

KFP Kubeflow Pipelines

MADM Multi-Attribute Decision Making

MCDM Multi-Criteria Decision Making

ML Machine Learning

MLaaS Machine Learning as a Service

MLOps Machine Learning Operations

MODM Multi-Objective Decision Making

OIDC OpenID Connect

PoC Proof of Concept

SIMD Single Instruction Multiple Data

WSM Weighted Sum Model

Chapter 1

Introduction

This chapter looks at the evolution of the field of machine learning, the pain points currently being experienced by practitioners and how choosing the right set of tools is such an important decision, followed by an overview of the scope of this thesis and an analysis of what objectives can be define in order to evaluate its success.

1.1 Context

This masters thesis aims to address a problem being experienced by a recently created machine learning and data science team for the RideCare project being developed at the Cross-Computing division at Bosch Car Multimedia Portugal S.A., in Braga, Portugal. In particular the team's goal is to develop machine learning models capable of identifying violence situations in car interiors. Those models will take data from multiple sensors placed throughout the car interior such as audiovisual feeds, and process those inputs locally, embedded on a device, to throw alerts in real time.

Being subject to the limitations of an embedded device with limited computing power makes model development particularly challenging, as compute power will be very limited. It also makes the adoption of best practices absolutely fundamental to optimize what is expected to be an arduous project. Additionally, instead of the company's traditional on-premises approach, they are now moving to a cloud native approach i.e. moving its development process to the cloud as much as possible. Due to Bosch's partnership with Microsoft for their cloud operations and infrastructure, this masters thesis will reveal a preference for Azure services.

One problem raised concerns how to create a cloud environment - Gen2Cloud - that is well adjusted to the needs of this project because, among other reasons:

- Unlike with software, model development is highly experimental and there needs to be some way for the team to keep track of everything associated with it, e.g. sensor data, ground truth, models,

results, tests, among others.

- The raw data that is going to be sued for model training contains videos from actors whose identities must be protected in the context of General Data Protection Regulation (GDPR), requiring very strong and safe data handling.

- The processes used by the team need to be well integrated with other teams' also working on the same project.

This masters thesis is focused on the first point, attempting to create an adequate cloud environment by figuring out the right tools to be used for the use case at hand. Decisions made during the course of this internship were subject to any and all restrictions imposed by the company.

1.2 Motivation

Back in the 1990s, when the Internet began to really reach the masses, the amount of data available began to grow rapidly, and, naturally, processes that could produce useful tools or bring relevant insights from this newly found data source soon experienced high demand. From then until now, the field of machine learning has grown immensely, with its techniques now taking part on a wide variety of software applications from many diverse fields, ranging from image, audio and natural language processing, all the way to stock market investing and weather forecasting, passing through network intrusion detection systems and recommendation systems (e.g. social media feeds). Although impossible to accurately predict the future of Machine Learning (ML), we can make some educated guesses by comparing the evolution of ML with that of other, more mature software domains.

Let us consider the following example: just a decade ago, setting up an e-commerce store was a non-trivial and time-consuming task, requiring teams to spend time and effort on technicalities and tasks ranging from setting up an environment and necessary infrastructure, to one of choosing from a set of options and tools which should better fit their use case, all of this in addition to the programming, deployment and operation of the e-store itself. Nowadays, launching an e-commerce store can be done in just a few minutes as easily as a few clicks on a web browser. All that is left is the operation of the store i.e. adding products, setting prices, registering stock, etc.

Even by judging this ease of use by itself we can clearly see that, after many years of maturing, the remaining tasks are the ones that sit at the core of the business and reside mostly around the human aspect. This reality is probably somewhere in the future for the field of machine learning.

Like software (Ozkaya, 2019), machine learning has become pervasive, greatly advancing computer vision, speech, language understanding, and many other fields, and it promises to help with the grand challenges facing our society (Dean, Patterson, & Young, 2018), ranging from modelling fracture growth of brittle materials (Moore et al., 2018) to classification of internet traffic (Nguyen & Armitage, 2008). However, because ML is not just code, like in traditional software development, but code plus data, care

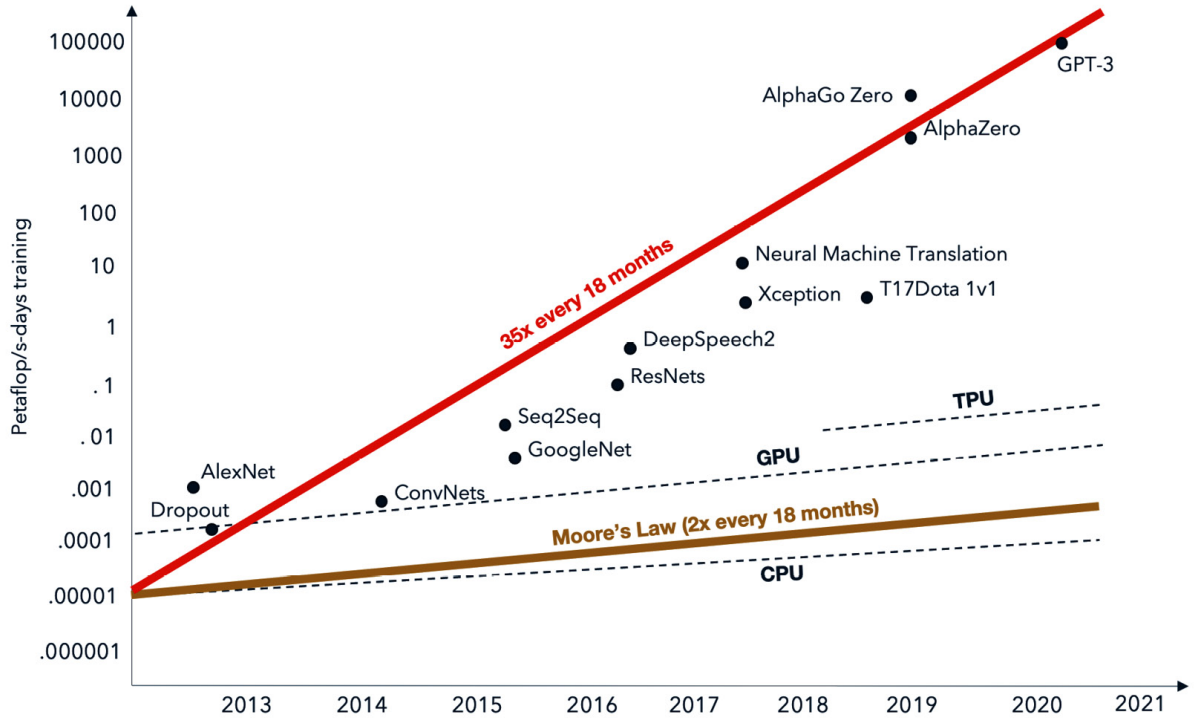


Figure 1.1: Demand for deep learning over time supported by computation (Raj, 2021)

must be taken to bridge the two together in development so they evolve in a controlled way, towards a robust and scalable ML system (Raj, 2021). This is easier said than done, and the difficulty in achieving such a system has real world impact. For example, a 2020 survey indicates that 55% of companies using ML have not deployed a single machine learning model to production (Algorithmia, 2020; Raj, 2021).

The demand for ML is out there (Makridakis, 2017), there are more papers being published, more applications being developed, and more funding and investment being provided (Algorithmia, 2020; Perrault et al., 2019), but its foundation is still riddled with very important questions that must be addressed in order to truly bring out its potential (Sculley et al., 2015).

Although machine learning will undoubtedly reach a point of maturity sometime into the future, some difficulties still stand. On the computational front, the end of Moores law and Dennard scaling has led to the end of rapid improvement in general-purpose program performance. The computations at the core of ML, and in particular deep learning, are low-precision linear algebra, known for its significant data-level parallelism. Advances in Single Instruction Multiple Data (SIMD) architectures and some domain-specific processing units, namely Googles Tensor Processing Units (Amodei & Hernandez, 2018), plus the adoption of approaches that allow greater algorithmic parallelism such as huge batch sizes, architecture search, and expert iteration, have greatly increased the amount of compute available for training and inference, at least for some applications (Amodei & Hernandez, 2018), as shown in figure 1.1.

In terms of development, some pain points can be addressed with the application of DevOps principles, originally meant for software development, in the context of model development, but others are inherent to model development, to the uncertain nature of data quality and its correlativity (Kanwarpal et al., 2021). One such pain point comes from the large number of stages and tools used during the model lifecycle,

and concerns the management and integration of all those pieces in a way that streamlines the process of model development as much as possible. Such tools, called orchestrators, are meant to satisfy this need, and in this masters dissertation, we will look at such tools, and undergo a decision-making process to select one for an enterprise use-case.

This is an important step for large scale projects - because of the role it plays in coordinating the many programs that fulfill the use cases, the orchestrator interacts with most components of the workflows of the project, thus any features and restrictions implied by the chosen solution can have an impact, be it positive or negative, on project development and maintenance. Sub-optimal selection increases the likelihood of an eventual migration to the newly-found, more adequate tool, generating a hidden cost during development (Sculley et al., 2015).

Additionally, this masters dissertation will move towards a tool that lets practitioners focus their efforts on a reduced set of recommended options, obtained from a set of desired functionalities and given preferences, e.g. cost, licence, performance, compatibility with some specific tool or cloud infrastructure, etc. This will allow for more meaningful analysis and reduce efforts wasted analyzing and experimenting with sub-par or unfeasible solutions.

1.3 Objectives

This masters thesis is centered around the development of a machine learning pipeline in the cloud, which leads to the production of three artifacts: the pipeline itself, a knowledge base with learnings from the pipeline development, plus a web application that can leverage this knowledge base and accelerate future projects.

With this in mind, the objectives can be categorized into two blocks: model pipeline and web application. For the pipeline, the main goals are:

1. Support the team during the comparison of the three options being considered for pipeline orchestration - Apache Airflow, Kubeflow and MLflow, while annotating the comparison criteria.

2. Prepare an Azure Kubernetes Service (AKS) instance with adequate configurations for the deployment of the chosen orchestrator.

3. Deploy and configure the chosen orchestrator on the AKS instance, while ensuring compliance with company policies and best practices.

4. Develop a functional, reproducible and well documented Proof of Concept (PoC) pipeline for model inference and result validation.

For the web application, ensure it:

5. Speeds up the tool discovery process, allowing users to obtain a filtered set of options that satisfy indicated parameters, e.g. ”license must not have copyleft”, and ”available REST API”.

6. Facilitates the comparison of admissible options in relation to criteria that describe the needs of the use case, e.g. ”quality of documentation”.

7. Is easy to use, maintain, deploy and modify, while remaining a low-cost operation.

8. Promotes the retention of developers' decision-making and procedural knowledge in a central knowledge base, thus building organizational knowledge.

1.4 Document Structure

The following chapters of this masters dissertation have the following structure: First, chapter 2 presents the current state of the art for machine learning pipelines in cloud-native environments. Section 2.1 performs a quick overview of the challenges for in ML development, helping better understand the purpose and value of the artifacts to be presented in this document. Then, section 2.2 does an introduction to the concept of Machine Learning Operations (MLOps), the domain of this document, in order to explain how the traditional ML activities have many others that support them, linking them together into a lifecycle. With this context, section 2.3 looks at one of those support activities - orchestration, and its purpose in the model lifecycle. Finally, section 2.4 shows analysis for the three reference orchestrators in focus for this master dissertation.

Next, chapter 3 follows the decision-making process used in the company to determine which orchestrator to use and how it gets deployed. Starting with section 3.1, it begins with a brief contextualization and description of the use case for this deployment. Then, section 3.2 goes over the requirements imposed to possible orchestrators, and criteria used to differentiate them Section 3.3 then studies the process used to take those criteria and produce a decision, as well as the adopted option. Section 3.4 presents a demo pipeline aimed at providing first-hand experience with Kubeflow, the deployment process and associated challenges Closing this chapter, section 3.5 reveals how the team feels in relation to this solution. Due to force majeure reasons, it was not possible to perform model experimentation beyond this small demo.

Chapter 4 follows with MetaTool, a web application created to store knowledge from decisions made by the team, and take advantage of it to speed up that process with precise and accurate tool recommendations. It begins with a requirements analysis listing what the team wants from this application, split between must-haves and nice-to-haves. Then, section 4.2 introduces the architectures for both MetaTool and the knowledge base. Section 4.3 concerns itself with the functional aspects of the application, conveying the core of MetaTool's user experience with an application walkthrough.

With all the artifacts presented, chapter 5 analyzes the outcome of this masters dissertation. First, section 5.1 does a follow up on the selection for an orchestrator to understand how the activities in chapter 3 impacted the project. It also provides a reflection on the currently available orchestrators, their features, and what could be done to promote their adoption. Then, section 5.2 discusses the pros and cons of MetaTool, and what kind of impact it can have at the company. Afterwards it goes over the lessons learned and future work.

Chapter 2

State of the art

2.1 Machine Learning Model Lifecycle

Machine learning provides mechanisms for the extraction of models (or patterns) from data (Ashmore, Calinescu, & Paterson, 2021). On a high level, the steps taken during the model lifecycle can be grouped into four major stages:

1. Data Collection - Obtain, analyze, preprocess and augment ground truth, storing it in a standardized format.
2. Model Production - Load data from storage, use it to train and validate a model, storing it in a model registry.
3. Deployment - Using a developed model from the model registry, ensure proper integration with the production environment. If adequate, deploy into production.
4. Monitoring - Obtain and inspect operational data e.g. model performance. Act accordingly, possibly by tweaking production environment or updating requirements for future models.

Ground truth is first collected and analyzed, possibly with the help of an expert in the domain in question, to determine whether or not the data is meaningful. Then, steps are taken to process the data in its raw form into something usable. Such changes can include cleaning, removal of outliers, and encoding. When the data is in a usable format, it is stored for later use. These processes are usually called Extract, Transform, Load (ETL).

Next, some data is loaded from data storage, and further processed if necessary, into a dataset. Paired with a selected model, the two are then used during the learning process to produce a trained model, the main object of model development. The newly trained model is then validated with a subset of data unused during training to simulate real-world input, indicating the quality of its results. Further tuning

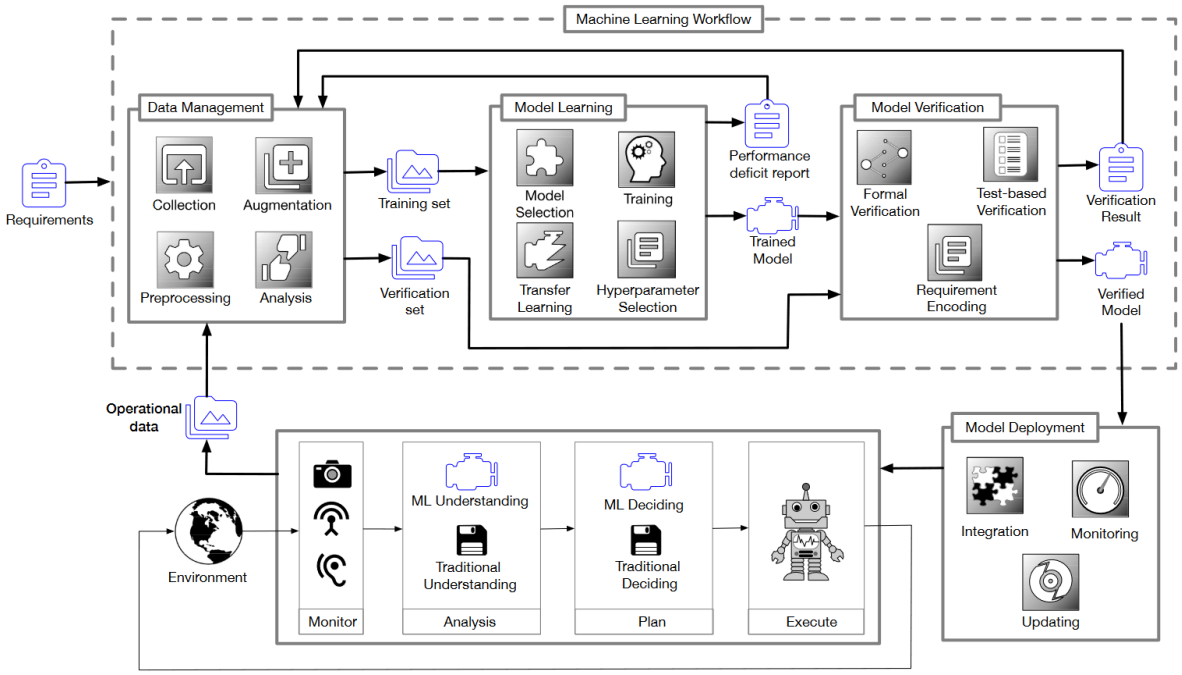


Figure 2.1: Machine learning lifecycle (Ashmore, Calinescu, & Paterson, 2021)

may be necessary to comply with requirements. With a satisfactory result, the model is then packaged and stored in a model registry, helping keep track of these artifacts and their properties.

An adequate, trained model is then chosen from the model registry to be rolled out into production. Before this occurs, it is first integrated into the current software solution which makes use of this model, and tested to ensure it is working adequately and satisfies all requirements such as computational load, inference times and GDPR compliance, just to name a few.

Lastly, thanks to careful preparations built into the production environment or software solution, the developers obtain and analyze operational data describing the performance of the application, including model performance metrics and input drift.

There are other activities that support ML projects such those commonly associated with project management e.g. problem definition, stakeholder consulting and infrastructure setup, but these stand out of the scope for model lifecycles.

2.2 Challenges in ML development

ML is code plus data, and although subject to similar difficulties as software development in terms of requiring version control, unit testing, integration testing, among others, model development is an iterative, costly, work-intensive process (Alla & Adari, 2021), subject to additional difficulties and with specific need, namely:

- Human: Practitioners who focus on exploratory data analysis, model development, and experimentation, might not be experienced software engineers who can build production-class services

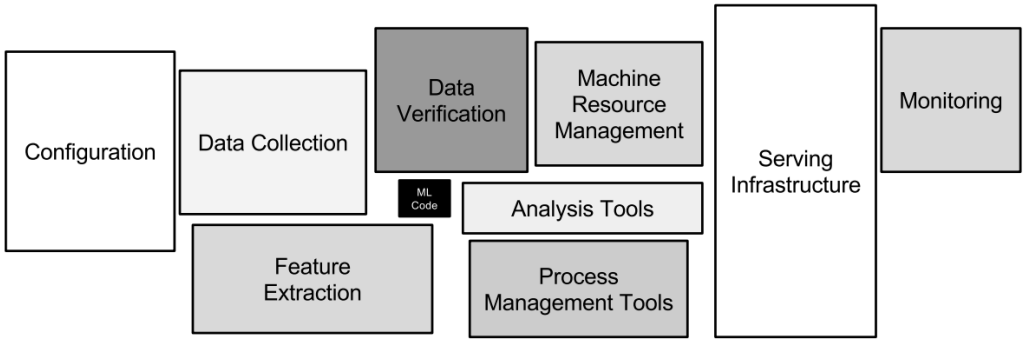


Figure 2.2: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex. (Sculley et al., 2015)

(Google, 2020). Also, machine learning teams require domain experts to support the ML specialist to analyze and synthesize the data, or qualify the data for use (Kanwarpal et al., 2021).

- Data collection and drift: Raw data requires processing to remove data points such as null values and faulty data values. Alternatively, we may need to process the raw data to extract only the useful information among all of the noise (Alla & Adari, 2021). Also, in many practical applications, changes in the relationship of the response and predictor variables are regularly observed over time, resulting in the deterioration of the predictive performance of these models, often called data drift.

- Model production: Models are subject to overfitting, and it is not always possible to solve it without fixing the training set by introducing more variety or increasing the number of data points (Alla & Adari, 2021), which can require a lot of effort to obtain. Manually iterating through the ML lifecycle two or three times might be doable, but over time, with dozens or hundreds of iterations, it becomes unfeasible. Developers become tired of doing the same thing and, since everything is manual, there is a chance for human error, inducing the need for more complex pipelines with higher adherence to the MLOps principles.

- Configuration and Scale: Bringing one model to life is just a small part of the many steps and efforts involved in developing a real-world machine learning system, which often consume most of the effort. Overlooking these activities can be incredibly costly in the long term (Sculley et al., 2014). Just not keeping track of the surrounding components and how they are configured raises the difficulty in perceiving the consequences from small configuration changes, an afterthought which Sculley et al. noted between researchers and engineers, raising the problem of configuration debt. On top of this, as pointed out by TA and NGUYEN, training complex ML models with large datasets often requires powerful computing infrastructure, which is costly to acquire and maintain. As a result of this, ML developers move their operations to the cloud in order to take advantage of the on-demand and elastic resource provisioning capabilities. Poorly configured cloud environments can be a serious hurdle, leading to low performance, poorly trained models and risk significant cost (TA & NGUYEN, 2018).

- Deployment and Maintenance: ML systems can require the deployment of multi-step pipelines to automatically retrain and deploy model. Pipelines add complexity and requires automation of steps that are manually done before deployment by data scientists to train and validate new models (Google, 2020). Also, machine learning requires a greater operational effort, since unlike traditional software systems, a machine learning solution is constantly at risk of degradation because of its dependency on data quality, raising the need to collect operational data, track summary statistics of our data and monitor the online performance of the model (Google, 2020; Kanwarpal et al., 2021) to take proper action when values deviate from our expectations.

- Technologies: ML practitioners keep trying new tools and approaches on their quest for better accuracy, unlike with traditional software development, where teams select one tool for each phase (Zaharia et al., 2018). Even at academic level, students are encouraged to build the same model type on different frameworks and tools, e.g. TensorFlow[1], Caffe[2], PyTorch[3], Knime[4], and compare their outcomes. This exploration, combined with the increasing popularity of ML, led to hundreds of software tools for each phase of ML development, most of which neither me nor the reader will never even known of their existence. Also, there is often no guarantee that any two tools might interface correctly together, increasing the difficulty of the tool selection process. ML developers made attempts at solving this problem by building ML oriented development platforms, namely Google's TFX[5] and Databricks[6], plus all the Machine Learning as a Service (MLaaS) from public cloud service providers, e.g. Amazon Web Services (AWS) Sagemaker, Azure ML and Google AI Platform. However, the framework may be too closed, to the point where it becomes difficult to integrate with other options and technologies, which can be a deal breaker for projects requiring custom or legacy tools. MLaaS can be particularly problematic for enterprise solutions as cost, vendor lock-in and data privacy concerns must also be factored in.

- Experiment Tracking and Reproducibility: Keeping track of experiment results and all the necessary artifacts to reproduce the obtained results is one of the most important, yet most difficult things to achieve when producing ML models. Even though a model might feature high levels of accuracy, its creators might need to re-implement it, as is usually the case, due to some reason that might not be obvious during development, as described in (O'Leary & Uchida, 2020). One way to mitigate this is by using adequate tools to keep track of performed experiments, training runs, hyperparameters and training data. Another approach would be to make ML developers produce pipelines instead of models, or pieces of code to produce models. It should be noted that, according to O'Leary and Uchida, a pipeline-centric workflow is challenging because it requires formalizing programs before and after model training in ways that can be properly unit tested and deployed. However, model

[1] https://www.tensorflow.org
[2] https://caffe.berkeleyvision.org/
[3] https://pytorch.org
[4] https://www.knime.com
[5] https://www.tensorflow.org/tfx/
[6] https://databricks.com/solutions/machine-learning

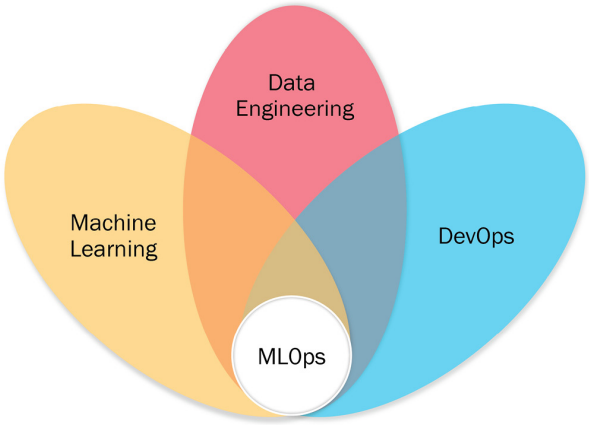


Figure 2.3: MLOps intersection (Alla & Adari, 2021)

development with a formal pipeline program is slower than with an iterative environment such as notebook, due to the overhead of added software abstractions. This aspect is a major reason for the popularity of tools capable of generating pipelines from notebooks, such as Kale[7], enabling the best of both worlds.

2.3 What is MLOps

The DevOps methodology has improved the efficiency of development teams (Virmani, 2015) by seeking all the pieces of a software delivery lifecycle work like a well oiled machine with rapid and automated software delivery and quality assurance from source code changes with CI/CD pipelines (Humble & Farley, 2010), feedback with production monitoring, and configuration consistency with infrastructure-as-code or data (Mäkinen et al., 2021)(Jabbari, bin Ali, Petersen, & Tanveer, 2016). It is center around (Treveil et al., 2020):

- Robust automation and trust between teams
- Collaboration and increased communication between teams
- The end-to-end service life cycle (build, test, release)
- Prioritizing continuous delivery and high quality

MLOps has emerged as the result of applying DevOps methodologies and best practices to machine learning models in place of software (Alla & Adari, 2021). It unites the developmental cycles followed by data scientists and machine learning engineers with that of operational teams to help ensure continuous delivery of high-performance machine learning models (Alla & Adari, 2021).

MLOps streamlines the development, deployment, and monitoring pipeline for ML applications, unifying the contributions from the different teams involved and ensuring that all steps in the process are

[7] https://kubeflow-kale.github.io/

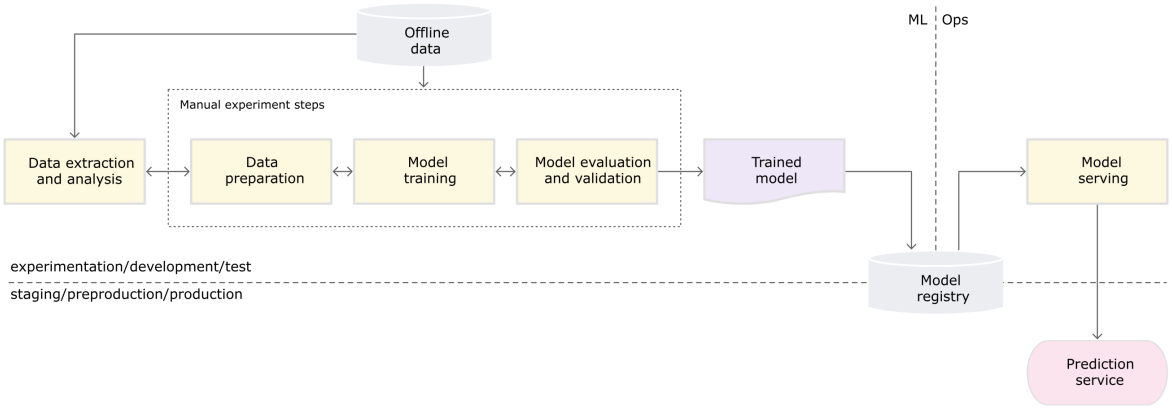


Figure 2.4: Manual ML steps to serve the model as a prediction service. (Google, 2020)

recorded and repeatable (Raj, 2021) by using tooling and workflows to automate one or more of the phases of the lifecycle (Mäkinen et al., 2021). This is done through the creation of pipelines, an infrastructure that contains a sequence of components manipulating information as it passes through (Alla & Adari, 2021) to automatically build, test, release and configure software release (Humble & Farley, 2010) aiming to streamline the practice of model changes and updates. Also, the deployment of machine learning models into production is fundamentally different from software code. Data is always changing, which means machine learning models are constantly learning and adapting — or not — to new inputs (Treveil et al., 2020), making continuous performance monitoring and adjusting essential.

This methodology manifests itself in the types of setups used for model development, usually distributed into three (Alla & Adari, 2021; Google, 2020) or five (Microsoft, 2021) distinct categories, each corresponding to increasing stages of automation and adherence to MLOps principles.

2.3.1 Manual workflow

With this approach, both stages - experimentation and production - are done manually. This process is usually driven by experimental code that is written and executed in notebooks by data scientist interactively, until a workable model is produced (Google, 2020). If used for more serious purposes, manual workflows tend to be inefficient as chances are high that practitioners will have to update it to maintain its performance on the new data, forcing the model development team to update the model and repeat the entire process again by hand (Alla & Adari, 2021). Although the process may not be that bad the second time around, there is still the issue of fixing any potential bugs that may arise from the new model. This makes it difficult to manage the machine learning model lifecycle (Microsoft, 2021). Refer to figure 2.4.

These are some of the characteristics with no MLOps adherence:

- When the entire process of analyzing new trends, training, testing, and validating data is manual, this may require significant resources over time, which may become unfeasible for a company without the resources to spare (Alla & Adari, 2021).

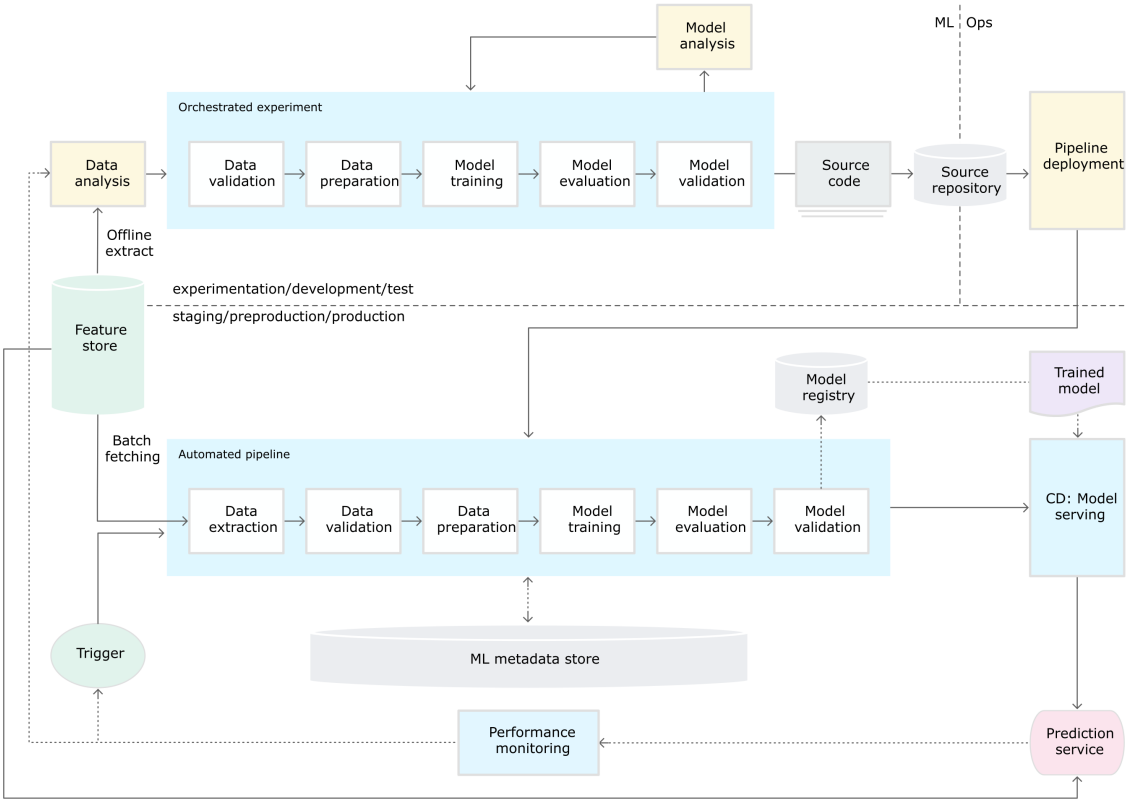


Figure 2.5: ML pipeline automation for CT. (Google, 2020)

- The process separates data scientists who create the model and engineers who serve the model as a prediction service, which can lead to train-serving skew (Google, 2020; Microsoft, 2021).
- Heavily reliant on data scientist expertise to implement new models (Microsoft, 2021).
- Lack of continuous integration (CT)/continuous delivery (CD) leads to longer integration cycles, resulting in so called ”integration hell”, a situation where delivery is in halt for a long-period of time (Mäkinen et al., 2021). Usually, testing the code is part of the notebooks or script execution.
- Predominant use of notebooks leads to lack of proper experimentation tracking and version control (Microsoft, 2021), difficult to keep track of different results and reproduce the experiments.

2.3.2 Workflow with automated model pipeline

The next step is to move the training process away from manual operation with the automation of the ML pipeline to achieve continuous training of the model (Alla & Adari, 2021; Google, 2020; Microsoft, 2021), enabling continuous delivery of model prediction service. This is a more advanced setup which is more difficult to implement due to the additional services it relies on.

These are some of the characteristics for setups capable of continuous training (CI):

- Artifact being manually deployed is now an entire, automated pipeline to automatically and recurrently run to serve the trained model as the prediction service (Alla & Adari, 2021; Google, 2020). This shift from model to pipeline production is arguably the most important step of MLOps.
- Code modularization and use of containers to make code reproducible between development and production environments (Google, 2020).
- Automated collection of performance metrics and new user data, plus automated feature extraction.
- Training code and resulting models are now version controlled (Microsoft, 2021).
- Triggering the training and retraining of new models can be manual or automatic - based on triggers from operational data e.g. data drift reaches a certain threshold (Google, 2020).
- Integration of extra services to reach continuous training (CI), such as model registries, experiment trackers, and feature or data stores.
- Still heavily reliant on data scientist expertise to implement new models (Microsoft, 2021).

2.3.3 Full MLOps Automated Retraining

The last level makes use of all MLOps principles and achieves full Continuous Integration/Continuous Delivery (CI/CD) of pipelines. Alla and Adari sees this as a big advantage, as it means businesses can now keep up with significant changes in the data requiring the creation of new models and new pipelines, and can also capitalize on the latest machine learning trends and architectures all thanks to rapid pipeline creation and deployment combined with continuous delivery of model services from the previous setup. Refer to figure 2.6.

These are some of the characteristics for setups fully compliant to MLOps principles:

- Iterative experimentation of new models with orchestrated experiment steps (Google, 2020).
- CI/CD process with unit and integration tests for model and application releases (Microsoft, 2021).
- Semi-automated deployment to a pre-production environment, for example, a deployment that is triggered by merging code to the main branch after reviewers approve the changes (Google, 2020).
- The pipeline and its components are built, tested, and packaged when new code is committed or pushed to the source code repository (Google, 2020).

2.4 Machine learning workflow orchestration

Data processing, model training and inference are very heavy workloads, so it is natural that businesses will tend to use large computational infrastructures to fulfill their business problems. Arguably the single,

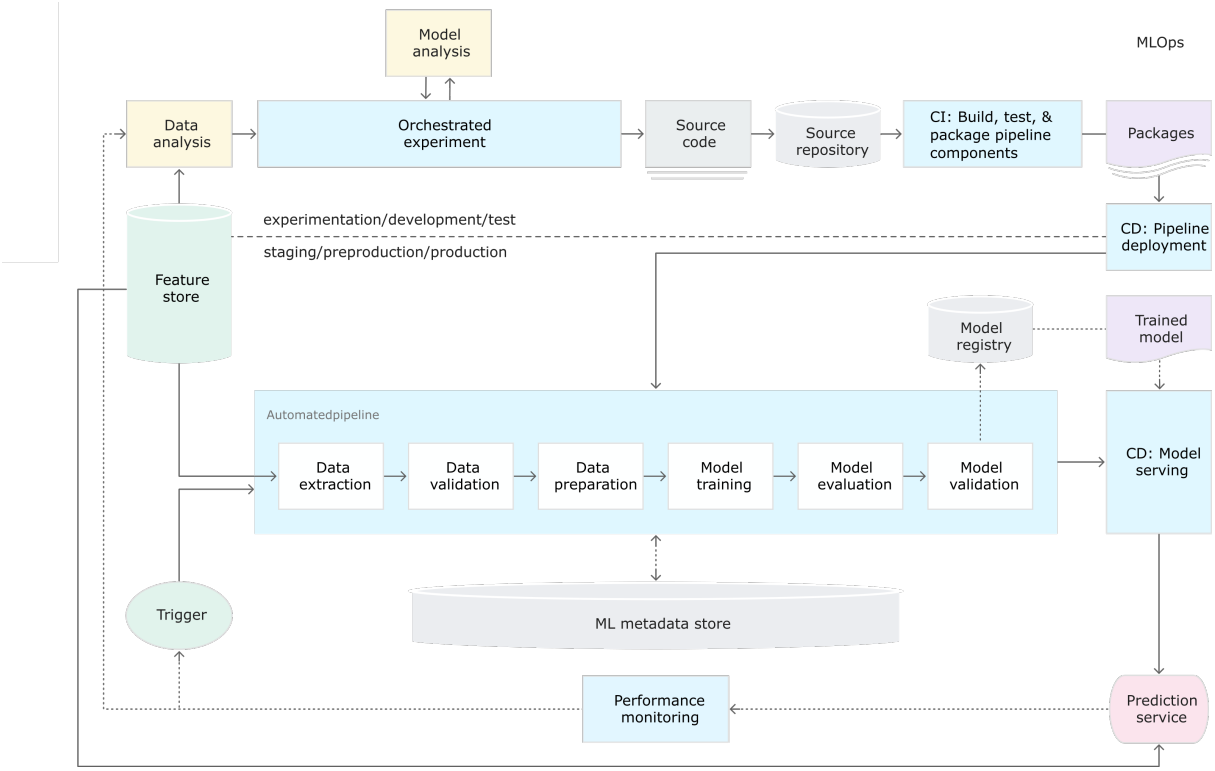


Figure 2.6: CI/CD and automated ML pipeline. (Google, 2020)

biggest advantage provided by cloud service providers is the abstraction of compute infrastructure - it enables the amortization of infrastructure costs through the many customers, as opposed to in-house infrastructure and compute clusters, often only available to large entities.

Existing mainstream cloud service providers have prevalently adopted container technologies in their distributed system infrastructures for automated application management (Zhong, Xu, Rodriguez, Xu, & Buyya, 2021), which brings clear benefits, including reusability, scalability, and ease of change (Atamel, 2020). Having these huge, elastic (Treveil et al., 2020) - they automatically adapt to an increase in usage without the need to permanently provision resources that are rarely required - microservice ready infrastructure behind a pay-as-you-go business model lowers the barrier to access the necessary computing power for very complex and computationally expensive workloads. This makes cloud services very attractive for heavy workloads, including ML workloads like training deep neural networks and inference on random tree forests (Treveil et al., 2020). With microservices, developers are pushed into making applications that run through many small components. These can have hundreds or thousands of components to keep track of (Lohr et al., 2021), all of which need to be coordinated for many reasons e.g. resource allocation, even if they do not integrate the same application.

As previously mentioned, in the context of ML experimentation, we want to produce pipelines which, when executed, output a usable model plus any desired metadata. Observing figure 2.5 we can identify five generic steps which constitute such a pipeline:

On both cases, the components, although independent from each other, need to be coordinated in order to achieve some broader goal. There are two major approaches (Atamel, 2020) one can take: either

Step	Description	Inputs	Outputs
Data validation	Ingest and validate data into the pipeline, ensuring that the data is meaningful and well adjusted to the purpose of the experiment.	Data	Validated dataset
Data preparation	Do any necessary preprocessing e.g. remove outliers, normalize, encode, and split the data into train, test and validation sets.	Validated dataset	Training set, test set, validation set
Model training	Train model with the data allocated for model training to obtain model parameters that fit the data.	Training set, test set	Trained model
Model evaluation	Test the model parameters on data it has never seen i.e. holdout test set, evaluating how well the model fits the data.	Trained model, validation set	Verified model, verification results
Model validation	Confirm the model predictive performance is better than a certain baseline, as well as other desired characteristics e.g. model size and inference times.	Verified model	Validation results

Table 2.1: Common steps in experimentation pipelines

we take all components and implement a publisher-subscriber system where each component registers for and emits events as they need, or we take a more centralized approach and add a new service dedicated to defining the interactions and controlling the flow of communications between components (Lohr et al., 2021). These two approaches are choreography and orchestration, respectively.

Container orchestration is already widely prevalent within cloud service providers, with Kubernetes standing as the go-to solution. It is offered by major cloud providers and is often offered as a managed service. However, in the context of ML experimentation, there isn't a dominant, industry-standard solution to turn a set of components into an actual pipeline. Instead, there are many alternatives, each with different approaches, each better suited to different use cases. Some well known options are Apache Airflow[8], Argo[9], Luigi[10], MLflow[11], Kubeflow Pipelines[12], and Prefect[13].

2.4.1 Understanding workflows

Before analyzing and looking into what it means to be an orchestrator and what it does, it is important to build an understanding of the concepts used by them. One such concept is the one sitting at the heart of workflow orchestration: the workflow itself.

Workflows are often described as a set of repetitive activities occurring in a particular order. IBM describes them as *"the mechanism by which people and enterprises accomplish their work, whether*

[8]https://airflow.apache.org/
[9]https://argoproj.github.io/
[10]https://github.com/spotify/luigi
[11]https://mlflow.org
[12]https://www.kubeflow.org/docs/components/pipelines/
[13]https://www.prefect.io/

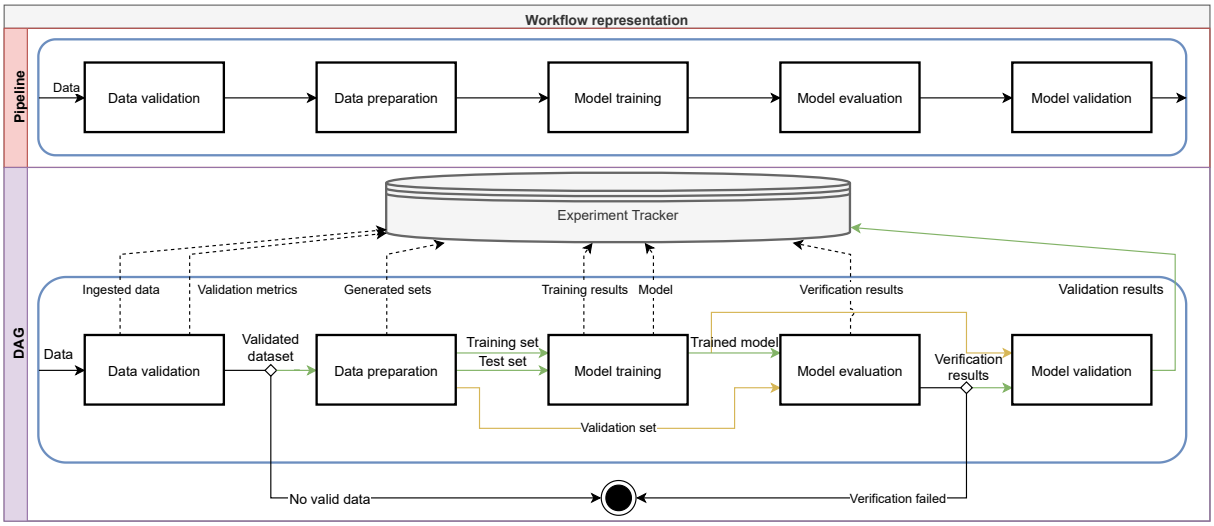


Figure 2.7: Representing workflows as pipelines versus DAGs.

manufacturing a product, providing a service, processing information or any other value-generating activity" (IBM, 2020). In the context of ML, a workflow is how ML engineers abstract a multi-step operation through which data can pass through, with dependencies dictating the order of execution.

If we recall the experimentation pipeline example in figure 2.5 and the artifacts listed in table 2.1, it should become apparent that pipelines, although easy to understand, are not an accurate description of the structure underneath because the flow between each step is not solely and strictly dependant on one preceding step i.e. not a chain of execution. While the input of any step is dependant on the output of the previous, it can also be dependant on other outputs. Refer to figure 2.7 and observe how step 4 - Model evaluation depends on two artifacts, the trained model from step 3 - Model training, but also a validation set from step 2 - Data preparation. This dependency would be impossible to describe with a pipeline. Thus, workflows, although often interpreted as a pipeline, have their dependencies described as a Directed Acyclic Graph (DAG).

2.4.2 Orchestration as a role

ML orchestrators have two key purposes in an ML workflow (Lohr et al., 2021):

1. Define the interactions between the different components that implement the model pipeline;
2. Ensure the flow of communications between components according to the dependencies between them.

These tools take in an explicit description of the workflow, usually some code in a domain specific language (DSL) or a YAML specification, indicating tasks i.e. code to run or commands to execute, the dependencies between those tasks, and possibly additional workflow control logic - conditions, loops, etc. From this, the orchestrator can schedule these tasks for sequential or parallel execution (Gubala & Hoheisel, 2007).

In order to operate, orchestrators need to be in contact with all components in the pipeline. This centrality makes orchestrators very important for the general ease of use of the entire setup, and the presence - or absence - of features can significantly impact the overall usability.

When describing the needs for scientific workloads with grid computing, a very complex system to orchestrate, Gubala and Hoheisel highlights the importance of:

- having the workflow language and the infrastructure that surrounds it support more than one level of abstraction in the resource description;
- use the dynamic scheduling for as-late-as-possible decisions to allow scheduling based on the most recent infrastructure information;
- introduce a certain level of fault-tolerance so the workflow engine can recover from a failure;
- allow the user to modify the workflow during runtime, e.g., dependent on intermediate workflow results.

Given that, as previously discussed in section 2.4, ML workloads are taking advantage of containers to improve modularity and reusability, the orchestrator does not need to, at this level, manage the infrastructure and its resources, e.g., GPU access. This is handled by a container orchestrator, e.g. Kubernetes.

2.5 Analysis of ML orchestrators

When machine learning on the cloud began to gain traction, there were no tools dedicated to orchestrating ML pipelines, leading to the adoption of generic workflow managers for ML activities, as was the case with Apache Oozie and Airflow, Luigi and others. Later and on opposite direction we saw tools from the field of ML branching out to perform the role of workflow manager. This is the case of Kubeflow Pipelines (TensorFlow Extended, itself a TensorFlow pipeline manager, and Argo CD, a Kubernetes-only GitOps tool), currently one of the most prominent ML workflow managers for cloud deployments.

The tools used for ML pipeline orchestration can generally be classified in terms of their approach with one of two labels: dedicated orchestrator, and re-purposed tool.

Dedicated orchestrators represent tools developed to be, and used as, workflow orchestrators. They may include some useful functionalities, but their scope is strictly aimed at this role. Popular, free and open-source tools of this category include Apache Airflow, Luigi and Kubeflow. Re-purposed tools are considered to be pieces of software which were not created with orchestration in mind, but developers were able to leverage their functionalities and use them as such. MLflow is a great example of this.

For this analysis we will focus on three particular options, the ones considered for the project as they satisfied the use case requirements. Refer to 3.2 for further details.

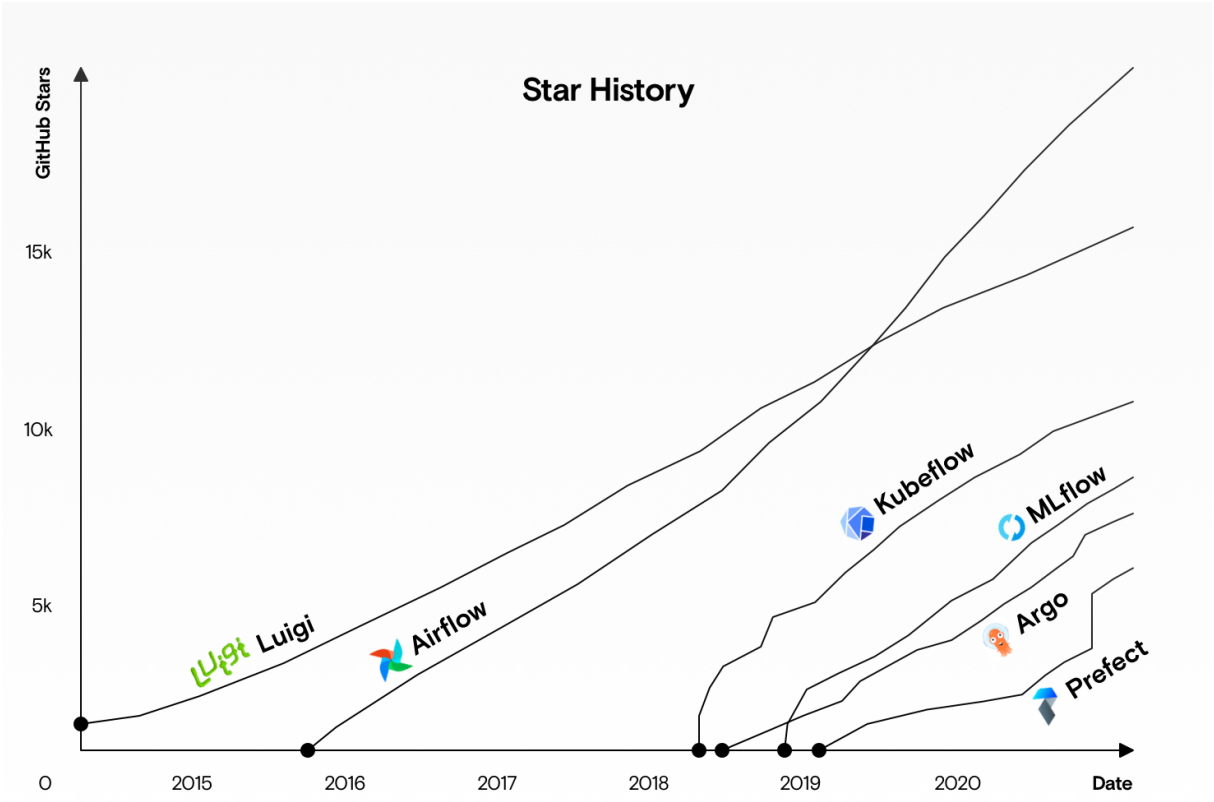


Figure 2.8: Star history comparison(Schmitt, 2020).

2.5.1 Apache Airflow

According to its creator Maxime Beauchemin, Airbnb was experiencing increasingly complex workflows back in 2014, which lead to the creation of Airflow. It was born as a framework with which to schedule and monitor such kinds of jobs and run the pipelines smoothly, for consistent output (Singh, 2019). Although it can and does orchestrate ML workflows, the use cases it is meant to fulfill are not just those of ML. In fact, Airflow is capable of orchestrating all kinds of generic workflows, and run them on top of several execution backends, which it abstracts as *Executors*. These include:

- Local Executor - runs tasks by spawning processes in a controlled fashion in different modes.
- Dask Executor - enables Airflow tasks in a Dask Distributed cluster.
- Kubernetes Executor - creates a new pod for every task instance.

According to its documentation ("Quick Start — Airflow Documentation," n.d.), Airflow pipelines are configuration as Python code, allowing for dynamic pipeline generation. Thanks to its modular architecture - message queues to any number of workers, the only limit to Airflow's scalability is the ammount of available computational resources. Being capable of running on top of a Kubernetes cluster on a public cloud, its scalability is virtually infinite. This makes it appealing for enterprise solutions, where the necessary scale can often be a key factor.

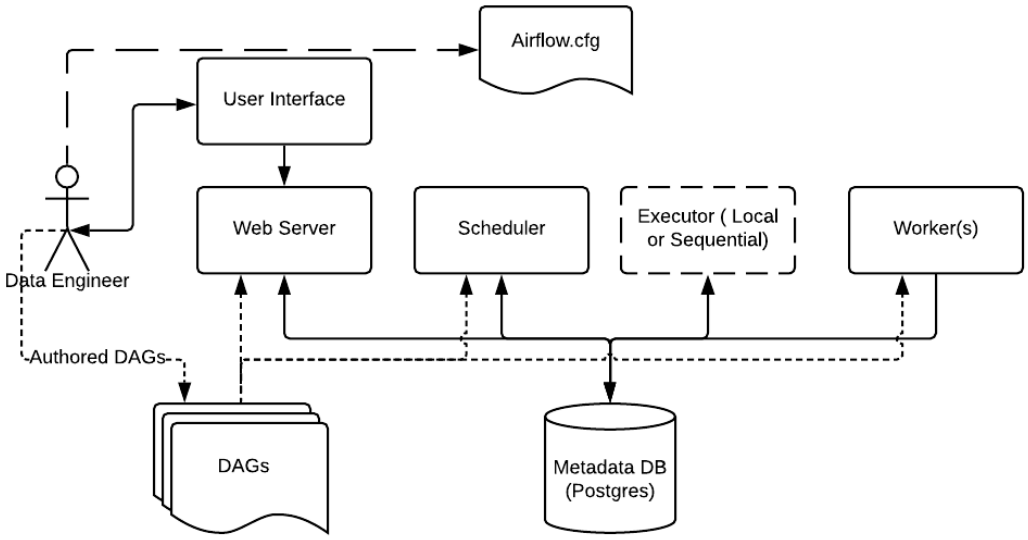


Figure 2.9: Basic Apache Airflow architecture. ("Quick Start — Airflow Documentation," n.d.)

Its pure Python base, with no extensions added to the language, makes it very user friendly for machine learning engineers and data scientists.

2.5.2 Kubeflow and Kubeflow Pipelines

Kubeflow[14] is a free and open source, Kubernetes-native platform, made to enhance and simplify the process of deploying machine learning workflows on top of Kubernetes clusters. Kubeflow Pipelines (KFP) is the workflow orchestrator at the core of this platform. It provides a simple way for building and deploying containerized machine learning workflows on top of Kubernetes, without bothering on the low-level details of managing a Kubernetes cluster (Bisong, 2019).

Kubeflow bundles software necessary to create end-to-end ML pipelines and manage the entire life cycle, capable of fulfilling all needs and performing all activities, from development to deployment, and has a large and vibrant multi-vendor community behind it.

The Kubeflow Pipeline platform consists of (Authors, 2020; Bisong, 2019):

- A user interface (UI) for managing and tracking experiments, jobs, and runs;
- An engine for scheduling the ML pipeline activities, described on the multi-step workflows;
- An SDK for defining and manipulating pipelines and components;
- Notebooks for interacting with the system using the SDK.

[14] https://www.kubeflow.org/

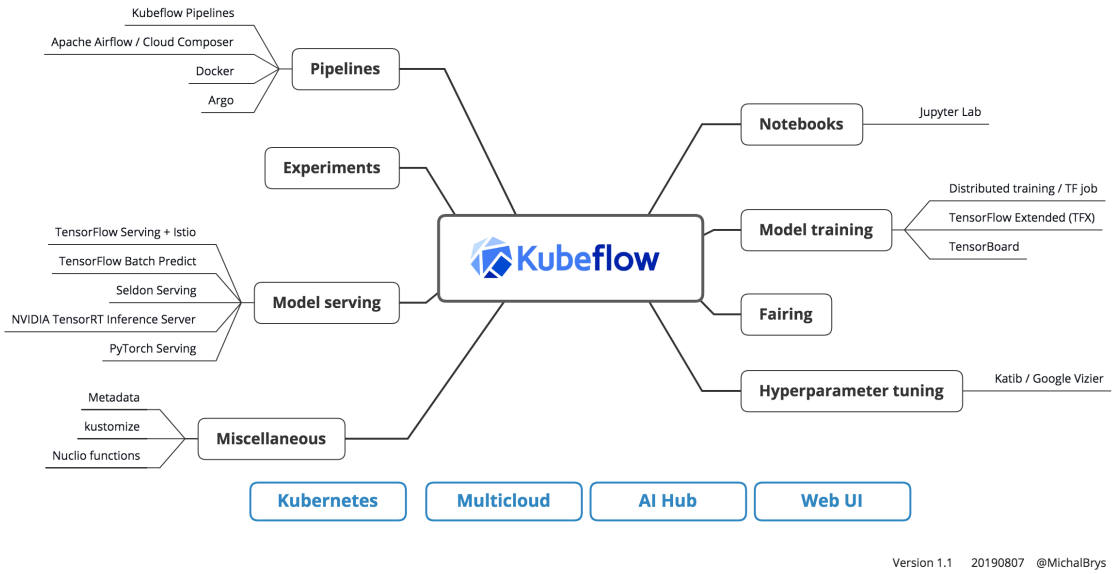


Figure 2.10: Kubeflow map (Brys, 2019)

It is well suited for orchestration of common machine learning tasks such as experiment tracking, model training and serving, but imposes a high user burden to design technically sophisticated components or common ML steps (O'Leary & Uchida, 2020). ML teams seem to be interested in out-of-the-box components and component templates, that could make it easier to adopt and customize workflow steps. In situations requiring the use of custom components, these must be implemented in particular ways to properly interface with the out-of-the-box components, e.g. integrating an application with the Kubeflow Central Dashboard (KCD).

Kubeflow requires an underlying Kubernetes cluster and interfaces directly with the cluster via the Kubernetes API. Opting for this tool reduces the spectrum of admissible options for cluster management to Kubernetes-based solutions, but fortunately most, if not all of the public cloud providers offer the possibility to deploy Kubernetes on their infrastructure, some even providing managed Kubernetes environments.

2.5.3 MLflow

Developed by Databricks, MLflow[15] represents a different take on workflow orchestration. Although it is not an orchestrator, nor is this a role it aims to fulfill, it is sometimes used for this purpose. As its initial paper points out (Zaharia et al., 2018), it was created as an out-of-the-box way to manage ML experiments and deployments, allowing developers to log information such as model parameters, as well as the resulting trained models. According to its creators, it was designed to allow for maximum flexibility by defining general interfaces for each abstraction, that "allow users to bring their own code or workflows".

[15] https://mlflow.org/

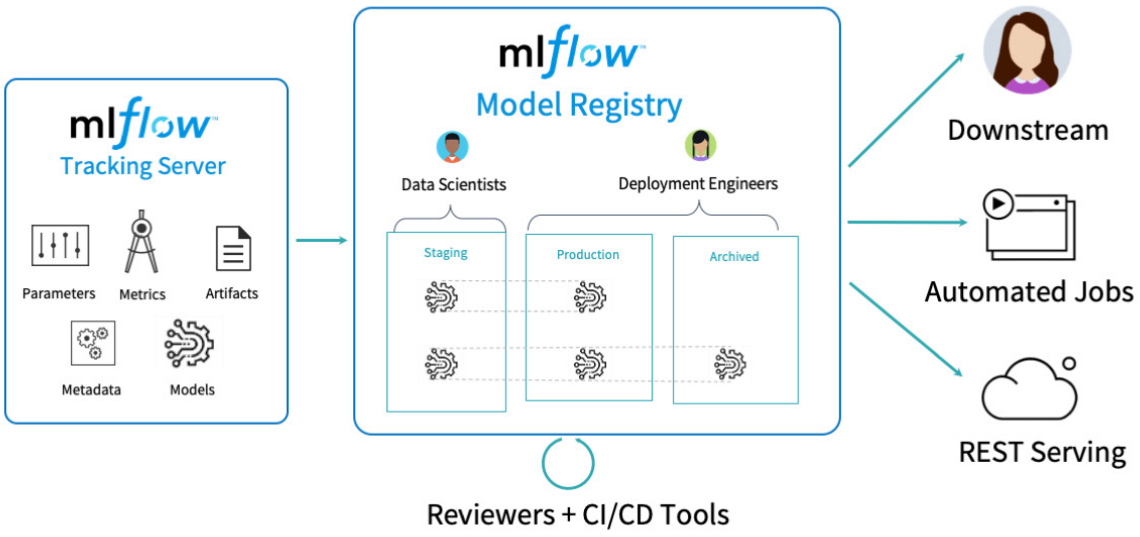


Figure 2.11: Overview of the CI/CD tools, architecture and workflow of the MLflow centralized hub for model management. (Parkhe, Ann Hong, Damji, & Mewald, 2020)

MLflow began with three core components, according to Zaharia et al., with Model Registry being added later (Parkhe, Ann Hong, Damji, & Mewald, 2020):

- MLflow Tracking, which is an API for recording experiment runs, including code used, parameters, input data, metrics, and arbitrary output files. These runs can then be queried through an API or UI.
- MLflow Projects, a simple format for packaging code into reusable projects. Each project defines its environment (e.g., software libraries required), the code to run, and parameters that can be used to call the project programmatically in a multi-step workflow or in automated tools such as hyperparameter tuners. MLflow projects can be submitted to cloud platforms for remote execution.
- MLflow Models, a generic format for packaging models (both the code and data required) that can work with diverse deployment tools (e.g., batch and real-time inference).
- MLflow Model Registry, a centralized model store, set of APIs, and UI, to collaboratively manage the full life cycle of an MLflow Model.

By creating a project for each step, scheduling desired tasks in a programatic way, and using MLflow's Tracking API to store metadata about the state of operations instead of experiment results, MLflow can be used to orchestrate multi-step workflows of any nature, not just about machine learning. It is due to the lack of structure on how the multi-step workflows can be defined, e.g. by using DAGs, and possibility of scheduling runs inside an executing run via API, that MLflow can be taken advantage of to orchestrate ML pipelines. In return for this generic operation, the lack of rigidity leaves more opportunities for mistakes by the developers.

Unlike with Kubeflow, MLflow does not have particular requirements for the underlying environment.

Chapter 3

Selecting a tool for cloud pipeline orchestration

3.1 Gen2Cloud contextualization

Gen2Cloud is an ML deployment PoC produced in the context of the company's push towards cloud native solutions, and is meant to test the feasibility of such approach. It integrates RideCare, the in-vehicle sensing project aimed at producing ML models capable of detecting in-vehicle violence from audiovisual surveillance data, i.e. video and audio captured inside the car, for integration in embedded systems, and follows the paper by Marcondes et al. - *"In-vehicle violence detection in carpooling: A brief survey towards a general surveillance system"*.

Due to the aforementioned cloud native approach and partnership with Microsoft, Gen2Cloud was to be created with strong integration of Azure services when possible. Alongside the services to be deployed for model experimentation, the deployment was also to contain three internally developed services on the same AKS instance.

- Zeus - dataset manager that handles acquired ground truth.
- DaiVs - model training and inference engine for developed algorithms e.g. action detection (bounding boxes), semantic segmentation on input video, producing JSON formatted results.
- VisualizationUI - web application to allow easy validation, render and display ground truth i.e. labels, or algorithm results on the respective source video.

Figure 3.1 provides a representation as to how these three services integrate with Azure services.

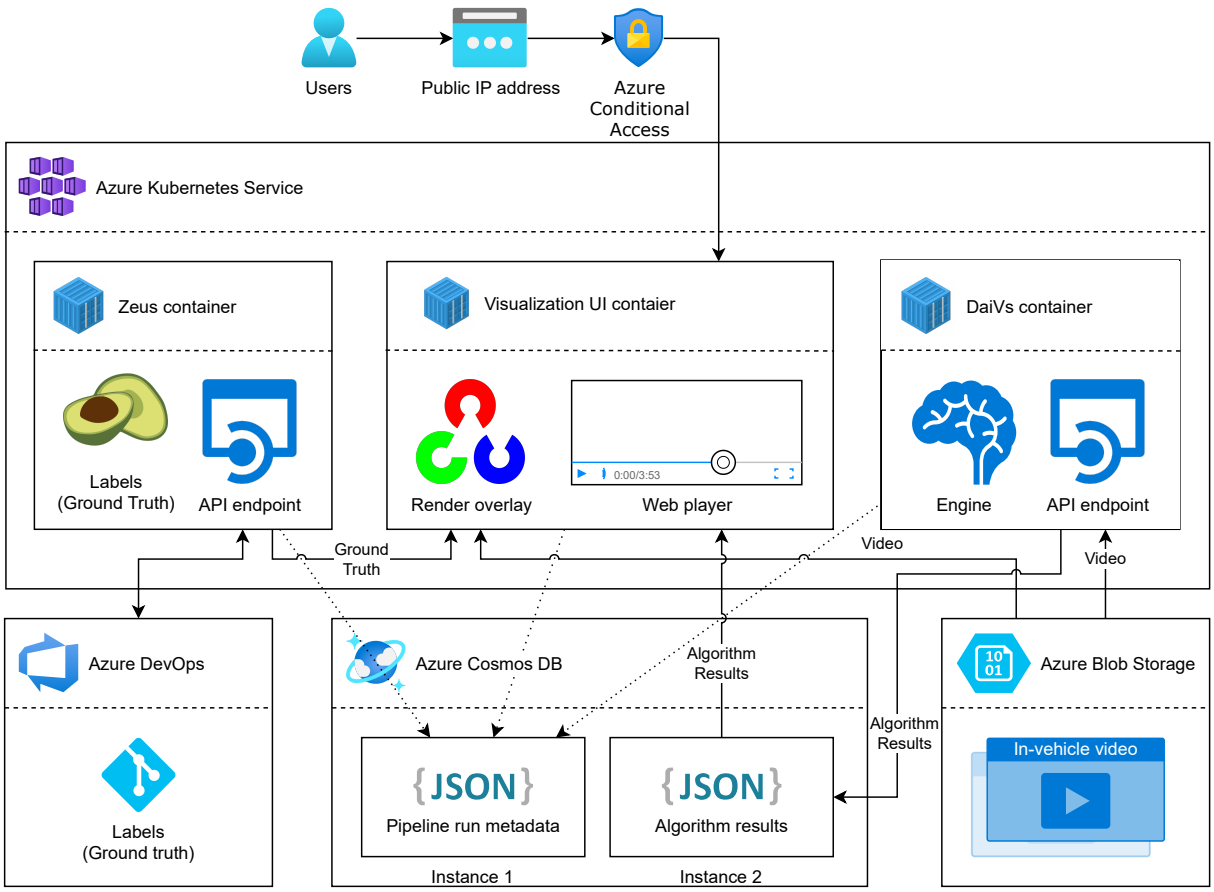


Figure 3.1: Gen2Cloud - Zeus, DaiVs and VisualizationUI architectural concept.

3.2 Orchestrator requirements

From the broader objectives for this project, three requirements were defined for an orchestrator to be considered admissible:

1. The solution must be compliant with company policies.
2. The solution must be capable of executing tasks as containers on AKS instances.
3. The solution must be able to schedule tasks that interface with internally developed dataset manager and inference engine.
4. The solution must work in other managed and unmanaged Kubernetes clusters to reduce vendor lock-in.

Verifying whether or not a solution satisfied these few requirements was done rapidly and efficiently. The first requirement was verified through information disclosed by the software vendors. Alternatively, in the cases of free and open-source software, through the license that accompanies said software. The second point was determined by checking if the tool could be deployed and used on Kubernetes-based environments, as it would assure compatibility with AKS. Doing so also satisfied the third requirement as it would always be possible to pass communications through an Application Programming Interface (API) or

container-to-container communications. The fourth requirement seemed somewhat contradictory to the stated intent of integrating with Azure services. However, it was deemed necessary as an orchestrator was itself likely to cause lock-in due to its role, making this requirement important. A major consequence of this was the exclusion of several paid options, including Azure Pipelines[1], a first-party service.

If these requirements were met, the admissible option would then be further analyzed and undergo experimentation. Web searches was by far the main collection method, providing the vast majority of options. Other methods included first-hand experience, contacts with peers and associates, and analysis of internal documents about previous projects.

At this stage, the requirements, although important, allowed for a large set of admissible and third-party solutions e.g. Apache Airflow. Notably, there was no requirement for options to be orchestrators as described in 2.4.2, leaving most automation tools as admissible. This led us to create a set of attributes that were deemed desirable and adding them to the previous requirements.

5. The solution must be aimed at ML workflow orchestration.
6. The solution must be free or low-cost to minimize project costs.
7. The solution must have good first-party documentation to accelerate deployment and familiarization with the software.
8. The solution must have widespread adoption and good community support to ease the resolution of undocumented and obscure situations.
9. The solution must have its development backed by some major entity to ensure long term support.

The new extra requirements severely reduced the number of admissible options, particularly the fifth requirement, which removes generic automation solutions such as Jenkins[2].

With this larger set of requirements, it became evident that the main direction of this process was mostly concerned with costs, either direct through software licensing, or indirectly through vendor lock-in and time spent:

- Requirement 1 concerns compliance matters e.g. GDPR, no copyleft licence;
- Requirements 2, 3, and 5 represent functional aspects;
- Requirements 4, 6, 7, 8 and 9 reduce effort and cost.

One notable omission was the desire for open-source code, which usually comes associated with the desire for free software. This was deemed insignificant beyond the increased community support because forking the software and implementing the necessary changes left the team at a vulnerable situation -

[1] https://azure.microsoft.com/en-us/services/devops/pipelines/
[2] https://www.jenkins.io

if the maintainers did not accept a pull request, maintaining the internal version up-to-date with future developments was very likely to become prohibitively costly.

Evaluating the current set of admissible options according to these further reduced its size, leaving just three solutions deemed likely to satisfy the needs of the project: Apache Airflow, Kubeflow, and MLflow. This number of orchestrators was small enough to be scrutinized in more detail.

3.3 Comparing orchestrators

Multi-Criteria Decision Making (MCDM) is a widely known branch of decision making, often associated with Multi-Attribute Decision Making (MADM), which concentrates on problems with discrete decision spaces i.e. a known finite set of alternatives (Triantaphyllou, 2000). According to the author, MCDM also includes Multi-Objective Decision Making (MODM), which studies decision problems in which the decision space is continuous e.g. linear programming problems with more than one objective function, often called multi-objective linear programs. In the context at hand, deciding which solution to adopt is a MADM problem, as there are three options being considered, and a single dimension to the problem - which to choose for orchestration. Triantaphyllou, 2000 claims a WSM is probably the most commonly used approach for these situations. This method consists of finding the option with the greatest resulting score, defined by the product of how the option evaluates according to a criterion, by the weight of the criterion (Triantaphyllou, 2000):

$$A^*_{WSM-score} = \max_i \sum_{j=1}^{n} a_{ij} \times w_j, \quad for \quad i = 1, 2, 3, \ldots, m. \tag{3.1}$$

where: $A^*_{WSM-score}$ is the WSM score of the best alternative, n is the number of decision criteria, a_{ij} is the actual value of the i-th alternative in terms of the j-th criterion, and w_j is the weight of importance of the j-th criterion.

In a decision making process, criteria can be split as objective i.e. the criterion can be independently verified, and subjective i.e. the criterion requires judgment in its application (Chou, Chang, & Shen, 2008). If the values for a given criterion are discrete and crisp, particularly objective criteria, the criterion can be used to exclude options that do not have those values, filtering them, as was done previously in section 3.2 e.g. with requirement 5, which would have Boolean values - those that are orchestrators, and those that are not. The extension of the WSM method (Triantaphyllou, 2000) with these actions results in a process shown in figure 3.2.

We used this extended method and resorted to an internal decision-matrix template built as an Excell spreadsheet to facilitate the WSM method. The criteria were weighted according to the following method:

- If the row criterion is less important than the column criterion, assign value 0.
- If the row criterion is as important as the column criterion, assign value 1.
- If the row criterion is more important than the column criterion, assign value 2.

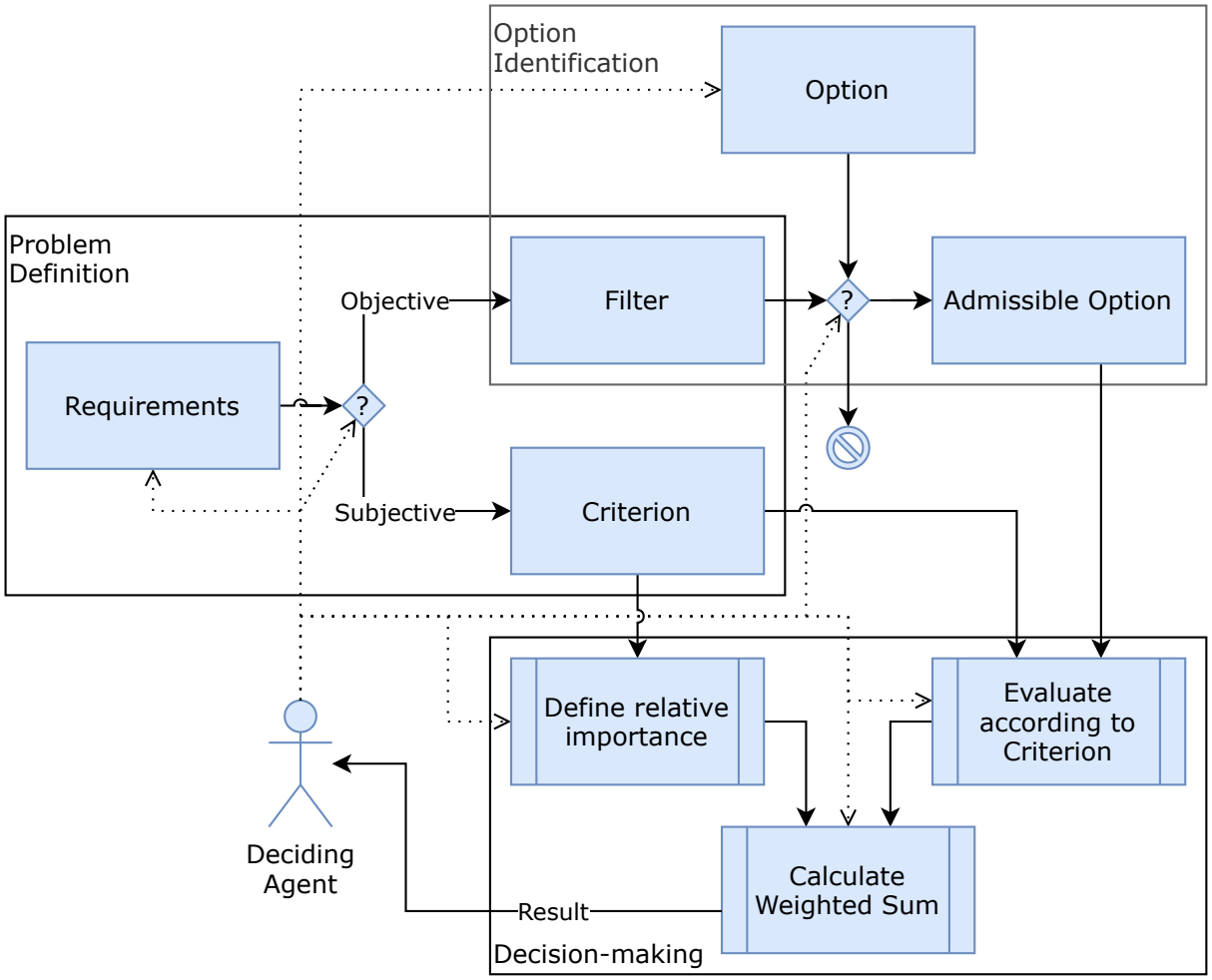


Figure 3.2: Schema of an extended WSM method.

The a_{ij} values attributed to each alternative were simple coefficients to be multiplied with the respective w_j, in orange, and did not have any restrictions as to that values could be used. Figure 3.3 shows a representation of the template applied to a fictitious example, with $a_{ij} \in [0, 10]$. Only white cells are assigned for user input.

For the criteria, usability was the most in-focus aspect. We sought something easy to deploy and manage in order to avoid dedicating someone just to maintaining the infrastructure, versatile i.e. adaptable to different use cases, and easy to program. However, it was impossible to have an accurate, albeit subjective, evaluation on how the three alternatives being considered - Airflow, Kubeflow, and MLflow - scored against the criteria without undertaking serious experimentation with all of them, as no members of the team had any experience nor familiarity with these. This was deemed unfeasible.

In order to circumvent the lack of experience with these tools, we approached other collaborators within the company and requested their opinions in relation to them. Intuitively, this approach seemed like a good course of action as the feedback given was more likely to be adjusted to the company's operations. In parallel, additional efforts were made to gather information from online sources.

In the end of this process, we decided to select the solution that had left the best impression over the course of this process and use it to develop a small demo pipeline. Kubeflow was the chosen option

Paired Comparison									Decision Table			
	Criterion A	Criterion B	Criterion C	Criterion D	Absolute Importance (W_{abs})	Relative Importance (W_{rel})	Ranking (Criterion)	Weighting (Criterion) ($W_{\%}$)	Option A	Option B	Option C	Option D
Criterion A		0	1	1	2	0.4	3	16.66%	6	6	4	3
Criterion B	2		2	1	5	1	1	41.66%	1	3	2	3
Criterion C	1	0		0	1	0.2	4	8.33%	9	6	1	1
Criterion D	1	1	2		4	0.8	2	33.33%	2	5	10	3
								100%	283	442	492	283

Figure 3.3: Decision-matrix with pairwise comparison.

for being not just an orchestrator but an entire Kubernetes-native toolkit, leading us to believe it would minimize deployment, maintenance and integration efforts. Kubeflow also seems to be the closest to a de facto cloud-native solution for MLOps (The Kubeflow Authors, 2021d).

Due to the importance associated with interoperability with the internal tools described in section 3.1, the pipeline was to run inference on a set of videos on trained models, storing back the result and the executed configuration.

3.4 Deployment and demo pipeline

3.4.1 Kubeflow deployment setup

Kubeflow has distributions dedicated for major cloud providers[3], including Azure, available through its official documentation. These are configuration YAML files meant to be applied against the Kubernetes cluster, triggering the correct deployment of all its components. We made use of these instructions during the deployment process.

Official documentation requires kfctl, a command-line interface for deploying and managing Kubeflow, exclusive to Linux based systems. It did not impact the deployment as it was performed entirely from Azure Cloud Shell, which satisfies this requirement. The deployment was performed with contributor level permissions on a dedicated AKS cluster, with consent and approval by the owner for certain actions, namely modifying network policies and cluster resources. This setup was based on Kubernetes 1.18.14, and ran Kubeflow v1.2 alongside the three containers from figure 3.1 on a single node pool of Standard_NC6 machines, scaling up to three virtual machines. Joining these to the deployment would allow for better resource optimization as the entry-level nodes with the necessary GPU acceleration (Kanchanahalli, McKittrick, & Hughes, 2021) - Standard_NC6 - were expected to have too much CPU compute power relative to the needs of this demo.

[3] https://www.kubeflow.org/docs/distributions/

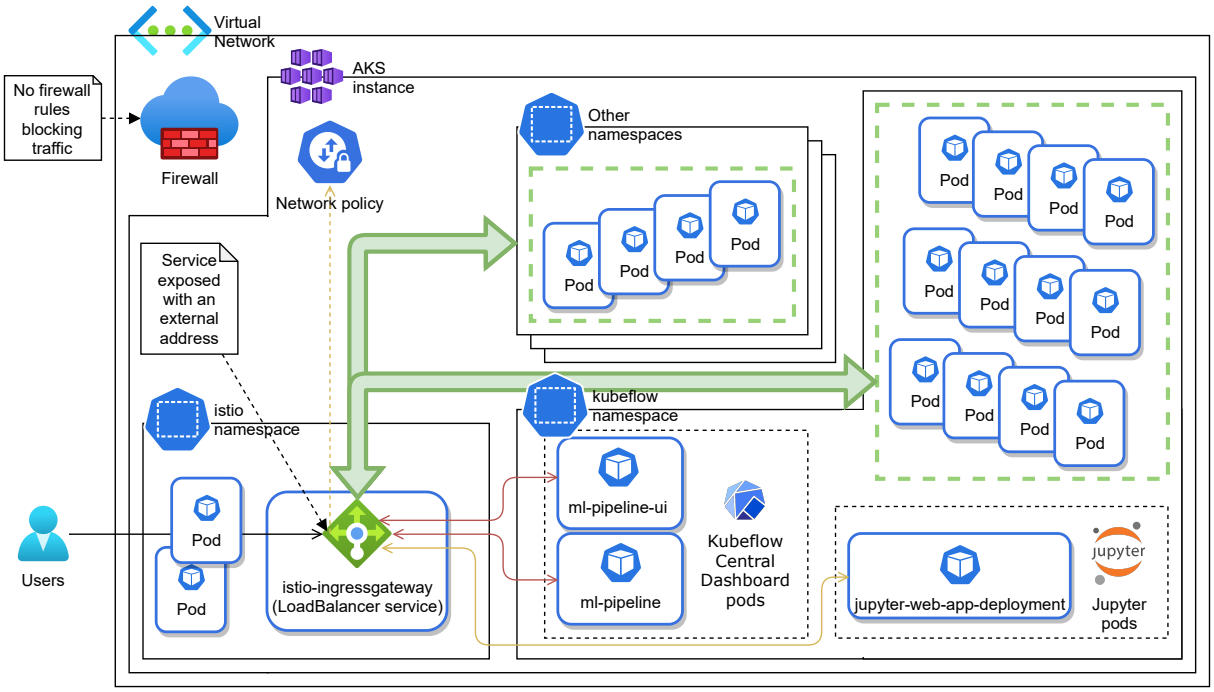


Figure 3.4: Diagram of intra-cluster network errors.

3.4.2 Challenges

Deploying Kubeflow proved itself to be significantly more difficult than expected due to severe errors within the deployment instructions and missing information. Following the available guide (The Kubeflow Authors, 2021b), we were first unable to successfully deploy Kubeflow with all of its components. It had several faults that had to be circumvented in order to make progress. Smaller situations such as referencing nonexistent kfctl releases[4] were easy to work around, whilst others were more difficult to identify and solve.

3.4.2.1 Intra-cluster network

During deployment attempts with the standard instructions (The Kubeflow Authors, 2021b), two networking problems occurred. The first was an apparent lack of connectivity between KFP and the Jupyter notebook server. Kubeflow advertises its Jupyter integration (The Kubeflow Authors, 2021d) as a feature that facilitates experimentation. Its integration would let users create pipelines for Kubeflow's orchestrator - KFP - directly from the code, either by describing a pipeline and its components with the KFP domain specific language (DSL), or by mapping each cell to a step in the pipeline. This level of ease of use was one of the factors that led us to choose this solution. Not being able to use it was a major downside. After consultation with support agents from both Microsoft Azure support and other internal teams, we discovered that the Calico network policy, an internally recommended option, was negatively impacting communications with the pod running the Jupyter server. Altering this to any of the other options - None and Azure - resolved the issue. In the second issue, the KCD was unreachable. Refer to figure 3.4. Upon running the Nmap[5]

[4] On 5th of November of 2021, The Kubeflow Authors made several corrections to the documentation. These include the referenced kubectl version, now pointing to version v1.2. instead of v1.3.

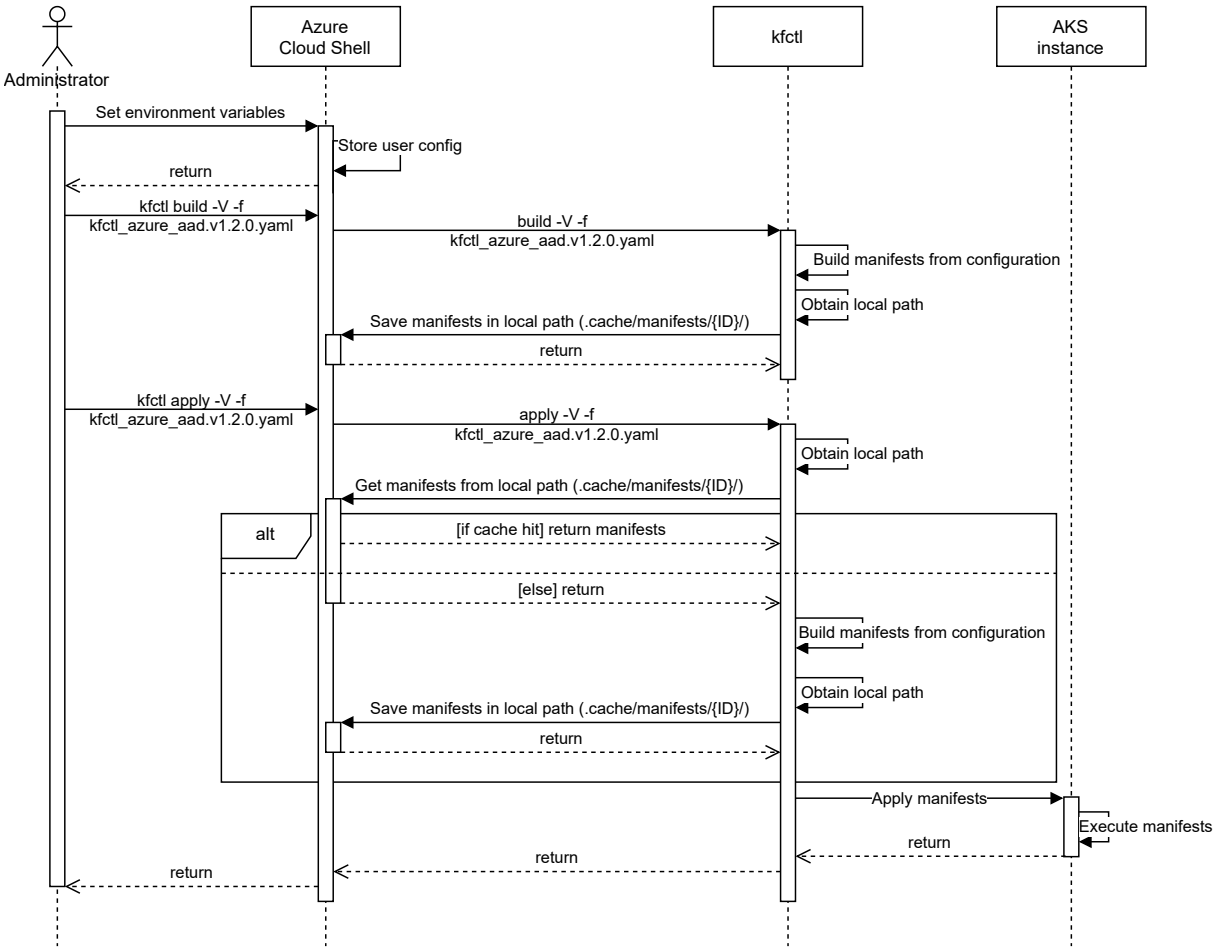


Figure 3.5: Kubeflow deployment sequence diagram.

utility against the external IP address, it was noted the connection was being filtered. Removing all the firewall rules did not alter this status. A simple NGINX[6] server was then deployed onto the same cluster and exposed to the exterior. Running Nmap against this other address produced the same result. Performing a scale up and draining the old node did not solve the issue.

After further investigation, it was discovered that Istio, a widely popular service mesh included with the deployment and responsible for network load balancing, had mutual TLS enabled by default to grant transport layer authentication. However, the Kubeflow dashboard was not authenticating properly, causing upstream connection errors. Disabling mutual TLS in the destination rules of the dashboard pods made it accessible, albeit at the cost of reduced network security.

3.4.2.2 Kfctl configuration file

Another official guide (The Kubeflow Authors, 2021a) instructs on how to deploy Kubeflow whilst using OpenID Connect (OIDC) to authenticate users which, in effect, shifted the entire burden of authentication

[5]"Nmap ("Network Mapper") is a free and open source (license) utility for network discovery and security auditing". - https://nmap.org/

[6]"NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more.- - https://www.nginx.com/resources/glossary/nginx/

to Azure Active Directory (AAD). Observe the sequence diagram in figure 3.5, describing a high level view of how a configuration would get deployed to AKS. In this process there are two types of artifacts, one configuration file describing which components are to be deployed, and the manifests to be applied on the Kubernetes cluster, these generated by kfctl. At the first application of the configuration file, or if the user used the parameter "build"instead of "apply", kfctl builds and store the Kubernetes manifests for the components described within, allowing the user to validate and modify the files. After this, all the generated manifests were applied and executed on the cluster, deploying all of Kubeflow's components. Securing access with conditional access to the KFP dashboard using AAD required, among other things, modifying two of the generated manifests in order to configure OIDC authentication, and remove "groups"from admissible OIDC scopes.

After modifying and applying these manifests with kfctl, it would always fail to find the manifests in cache. Surprisingly, kfctl was unable to determine the local path for the manifests because the contents of the YAML configuration file did not include the fields that define it, causing kfctl to build the manifests again, and place them under an apparently random path. This was an unexpected situation as the official guide (The Kubeflow Authors, 2021a) instructs the user to perform these modifications[7].

The issue was solved by appending the necessary fields. Refer to annex I.1 for the corrected configuration. Ultimately, using this configuration did not manifest the problems mentioned in 3.4.2.1, and lead to the successful deployment used for the demo pipeline.

3.4.3 Demo pipeline

The pipeline defined for this activity was fairly simple, consisting of a simple inference on a pre-trained algorithm. The project had a data storage structure that was centered around unique identifiers associated to recorded sessions i.e. videos. These would then serve to indicate the loading tasks which content needed to be obtained. In this example, these identifiers would define the videos themselves, indicate which ground truth data needed to be aggregated and loaded, and also serve as an index for the JSON file with the results. Artifacts are used between steps with a simple alias defined in the pipeline specification. KFP controls the name of the files, and maintains an association between its paths, the containers where they were produced, and their aliases, circumventing the need for extra storage options for this demo.

This pipeline was defined with KFP's domain specific language (DSL), and compiled accordingly. Its steps and objectives are as described in table 3.1 Figure 3.6 shows a representation of these dependencies.

In terms of the resulting workflow, it should be noted that even though there are two tasks which do not depend on one another in any way i.e. the two tasks under load-config, this does not test how Kubeflow handles parallel execution precisely for being executed under the same step. Performing a topological reduction of this DAG leads to a de facto pipeline without branching nor merging, as shown in figure 3.7.

[7] As of writing, the latest version of this configuration is still missing the fields relative to the local path. It is available at https://github.com/kubeflow/manifests/blob/v1.2-branch/kfdef/kfctl_azure_aad.v1.2.0.yaml.

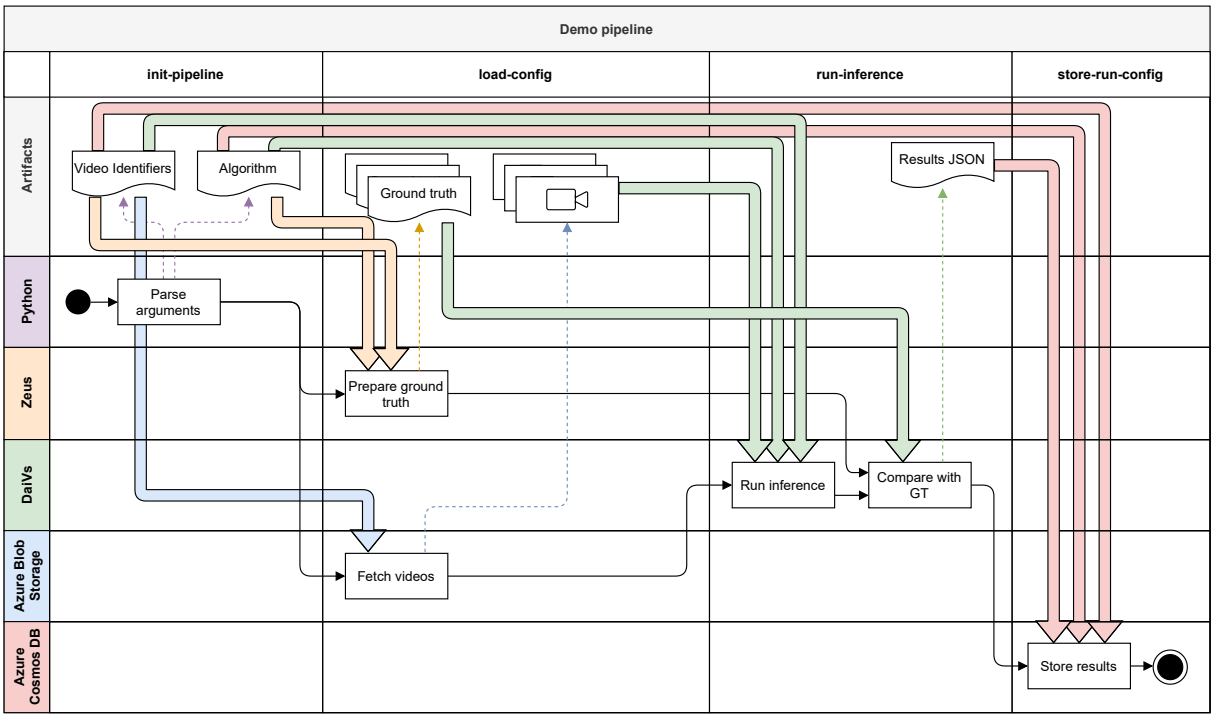


Figure 3.6: Demo pipeline diagram.

Component name	Activities
init-pipeline	Process the execution arguments, identifying which videos are to be executed, and which model is to be used, e.g. body pose estimation.
load-config	For each video identifier, Zeus fetches video's ground truth from latest branch, adding it to the dataset, and the video is requested from binary storage.
run-inference	DaiVs runs the algorithm on each video, and compares the results with those expected from ground truth.
store-run-config	The execution is stored into a database, with the identifiers and algorithm used, plus a JSON file with results produced by its application versus video ground truth.

Table 3.1: Summary of demo pipeline steps.

3.4.4 Kubeflow operation and findings

When accessing Kubeflow, the user is asked to login with the application. Given the authentication integration with company's AAD through OIDC, the user is redirected into Bosch's authentication page, returning if the user has successfully logged in. In case the user token is unknown to Kubeflow, it proceeds with the creation of a new user. This is important as it allows separation of pipelines, experiments and other functionalities according to its creator.

From here, the user is presented with the KCD, a web application that allows users to navigate between other sub-applications, including Pipelines i.e. KFP, Notebook Servers i.e. Jupyter, among others. As briefly mentioned in 2.5.2, Kubeflow allows developers to customize this e.g. to add quick access to an internal web application, a feature that could become rather useful in case of a major commitment to this solution.

According to the authors, the pipeline lifecycle, in the perspective of KFP, is as follows (The Kubeflow

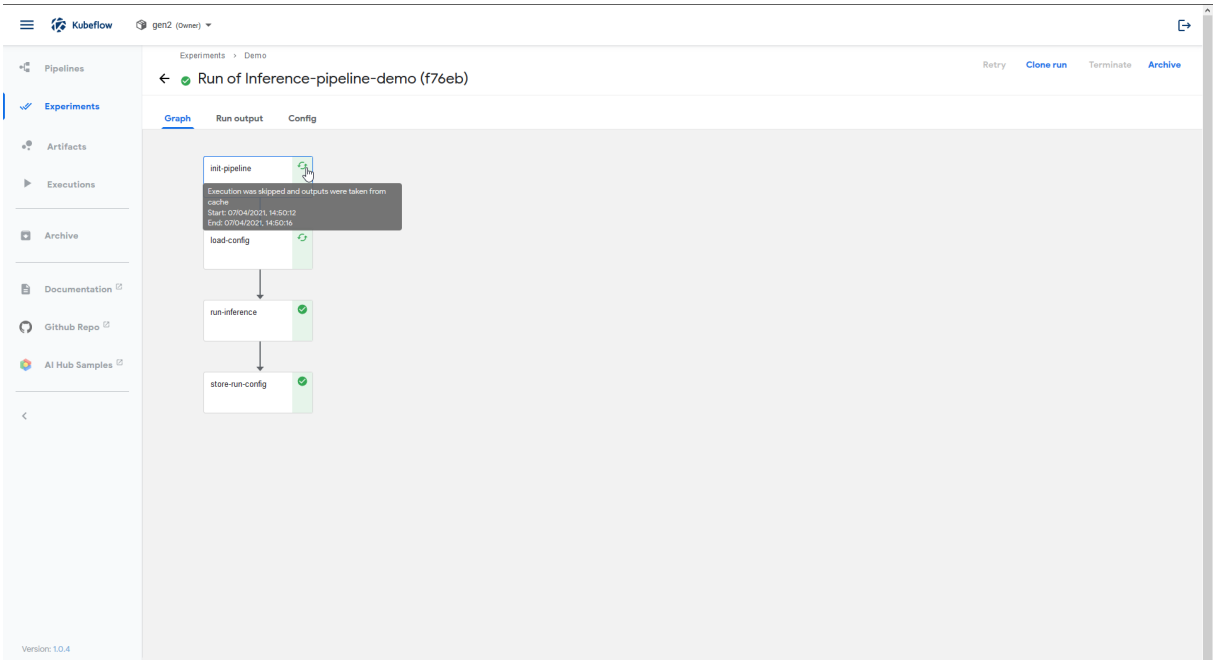


Figure 3.7: Graph view of a successful run of the demo pipeline.

Authors, 2021c):

- Create components or specify a pipeline using the Kubeflow Pipelines domain-specific language (DSL).
- Compile the pipeline with KFP's DSL compiler, transforming the pipeline's Python code into a static configuration (YAML).
- Call KFP's Pipeline Service to create a new pipeline run from the static configuration. This will trigger the creation of the necessary Kubernetes resources (CRDs) to run the pipeline.
- A set of orchestration controllers execute the containers needed to complete the pipeline. The containers execute within Kubernetes Pods on virtual machines. An example controller is the Argo Workflow controller, which orchestrates task-driven workflows, and is used by default.
- Whilst operational, KFP's Pipeline web server gathers data from various services to display relevant views: the list of pipelines currently running, the history of pipeline execution, the list of data artifacts, debugging information about individual pipeline runs, execution status about individual pipeline runs.

Architecturally, KFP relies on compiled manifests described with its domain specific language (DSL) for executing pipelines in all circumstances, be it through its API, a Jupyter notebook with its cells mapped to steps in the pipeline, or for pipelines defined as code - at some point, the pipeline went through KFP's compiler. These compiled pipelines i.e. YAML manifests that describe how Kubeflow and Kubernetes should proceed in order to execute the steps correctly, include among other information:

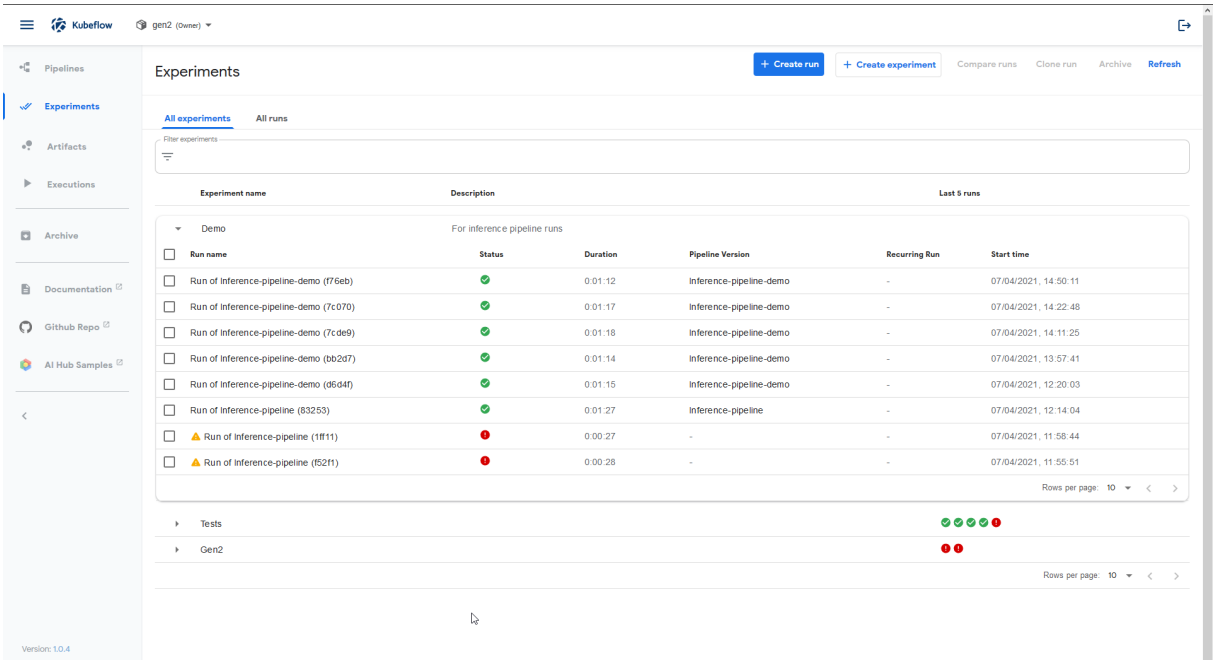


Figure 3.8: Kubeflow experiment GUI.

- KFP's API version, target namespace for execution, and some metadata e.g. name of the pipeline.
- Pipeline parameters i.e. user inputs given before starting the execution, e.g. the value for some variable that we want to test.
- Components' specifications - python code, base image, parameters, restart count, container ID.
- Network configurations - host and pod IP addresses.

Kubeflow groups pipeline executions under Experiments, granting a good overview of how the experimentation process is proceeding. One experiment can have runs from more than one pipeline. Refer to figure 3.8 for an example of this view, with the demo experiments.

3.5 Team opinions about Kubeflow

The team unanimously agreed Kubeflow was a viable solution in terms of features and how adequately it fulfills the needs, not just for model experimentation, but ML in general, including data processing pipelines, hyper-parameter tuning and training at scale. Also, having the dashboard only accessible through the company's authentication process meant complete compliance in this regard.

In terms of usability, there was consensus that Kubeflow has a non-trivial learning curve and requires lots of experience before users could define experiments with the desired level of ease, in reference to its own DSL. The flip side to Kubeflow's usage static configuration files is YAML files makes them easy to store, migrate and version. Additionally, the inclusion of Jupyter Notebooks and its integration with KFP and the remaining infrastructure facilitated the development process and allowed us, for example, to ignore any

authentication and network concerns when calling KFP's API to compile the pipeline, making the creation of new pipelines faster when performed on those notebooks rather than somewhere else, e.g. on a local editor. Also on a positive note, with these notebooks executing on a Kubernetes cluster granted a high degree of flexibility in relation to the compute needs for a given workload - if the user happened to be playing with a large model and ran out of resources, he could simply scale the deployment temporarily and provision some more to the server.

The major downside to this solution was the sheer number of hurdles faced during the deployment process - they laid bare just how much attention to detail Kubeflow still lacks. Having to resort to community forums in search for solutions was not unreasonable, but situations as those already described in section 3.4.2.1 took a rather long time to solve, even with the help of an internal support team as well as Microsoft Azure Support agents, are a strong indicator that, as an option for orchestration, Kubeflow carries some risks.

It was also noted that its *everything in one package* approach made it bloated. Whereas Apache Airflow and MLflow are small solutions aimed at solving specific pain points, Kubeflow is a big amalgamation of many different components into a single package. For context, performing a fresh deployment on a new cluster using the Azure OIDC manifest created over eighty different pods. It was not necessarily a problem in this context, as there were plenty of resources available, but with the team being part of a project that is developing models for embedded systems, there was some suspicions as to whether or not it could negatively impact the project.

In conclusion, Kubeflow seems like a good option if the team can spare enough time and effort to set everything up properly, maintain its many components, and assimilate its workflow.

Chapter 4

MetaTool

The previous chapter made clear just how much time and effort it can take in order to select which tool to use. In order to make that decision, decision-makers went through a whole process, starting with problem definition i.e. requirements, followed with identification of admissible options. Due to a lack of knowledge, there was no way to properly follow the decision-matrix process, as the evaluations of each tool in relation to the criteria was unknown. Sadly, project constraints did not allow experimentation with all alternatives, leading to experimentation with the tool deemed most likely to be adopted in order to obtain the missing knowledge. After this, the obtained knowledge could then be used for decision-making.

But the question now brought forward is how can the company avoid having to repeat this activity and not incur in needless costs? The knowledge was gathered at a great effort and used for one decision, but could we not have done it better? What will happen when others face a similar situation? Will they too incur on the same costs associated with this? Those other decision-makers could be aware somebody else had already produced knowledge that could be useful to them, and so they could approach the team for help, just like we tried to ourselves at the start of this process. But even then, what if those who held the knowledge had forgotten it or simply gone away? Any medium where one could write to would be able to solve this e.g. text files and spreadsheets, but over time they would become too cumbersome, negatively impacting its usage.

MetaTool's purpose is thus two fold: provide a central location for engineers and developers to store their knowledge about the tools they are familiar with, and allow others to access that knowledge, either as a simple catalog, or helping with the search for the tool that is best adjusted to their needs. Thus, MetaTool takes advantage of knowledge within the company, and uses it to accelerate tool discovery and selection.

4.1 Application requirements

In general, we could argue that MetaTool's usefulness is proportional to the product of how much knowledge it contains and users' trust. While more contributors lead to more contributions, having everyone contribute to the same place leaves it vulnerable to bad contributions. Contributors could, for example, have a bad experience with some particular tool due to some fault of their own, and go update its information in a negative way. This situation leaves the application in a tough spot. On one side, the whole point of the application is to enable all users to extend it with information that was previously unknown, so having some authority, i.e. curators, validate contributions could lead to stagnation as passing all operations through a central point leaves the application with a bottleneck - curators could be unable to corroborate modifications in a timely manner. On the other, ignoring the risks and letting everyone manipulate all the content would ultimately render it useless as users would not be able to trust it.

For this reason, let us assume the former approach and separate users into two categories, one for those who use the application - Users, and the other for those responsible for maintaining the application and curating the information - Maintainers.

When using large catalogs, users generally like the ability to browse it through categories e.g. searching for racing games in a mobile app store, or a particular genre of music in Spotify. Although its catalogue is not expected to reach such dimensions, the users of this application will be well versed with digital technologies, and therefore make good use of shortcuts. Thus, it makes sense for the application to provide similar mechanisms, enabling fast and accurate browsing. If, for example, the user is searching for an orchestrator, he should have some way of finding just those tools capable of orchestration. Similarly, if the user is looking for a particular factor such as being free, he should be able to reach it through its domain e.g. Costs.

Another important aspect to this application concerns the recommendation process. Recall how in section 3.3 the team sought to use a weighted decision matrix in order to determine which orchestrator to use. This process is easy to automate following the equation 3.1, using the catalogued tools as a source of options, objective criteria to filter those into a set of i alternatives for the matrix, and subjective criteria for its j criteria. The values of a_{ij} are the evaluations contained within the knowledge base. Users should be able to execute a decision matrix and obtain a recommendation simply by describing the use case through which filters should be applied, and the importance i.e. w_j for each desired criteria. In comparison to the process described in figure 3.2, this new process in figure 4.1 is much easier on the user, requiring only the description of his use case.

Finally, MetaTool is meant to be a web application accessible only to those within the company's virtual private network due to security and intellectual concerns. This could be done in several ways, most notably by applying network restrictions to the IP ranges, allowing just those associated with the VPN exit nodes, or through OIDC integration. With it being a web application, it is important that whenever a user makes some change to his current knowledge base, it should not impact those of others, unless performed by a maintainer. Also, it is fundamental to have the knowledge kept in a way that allows for easy validation,

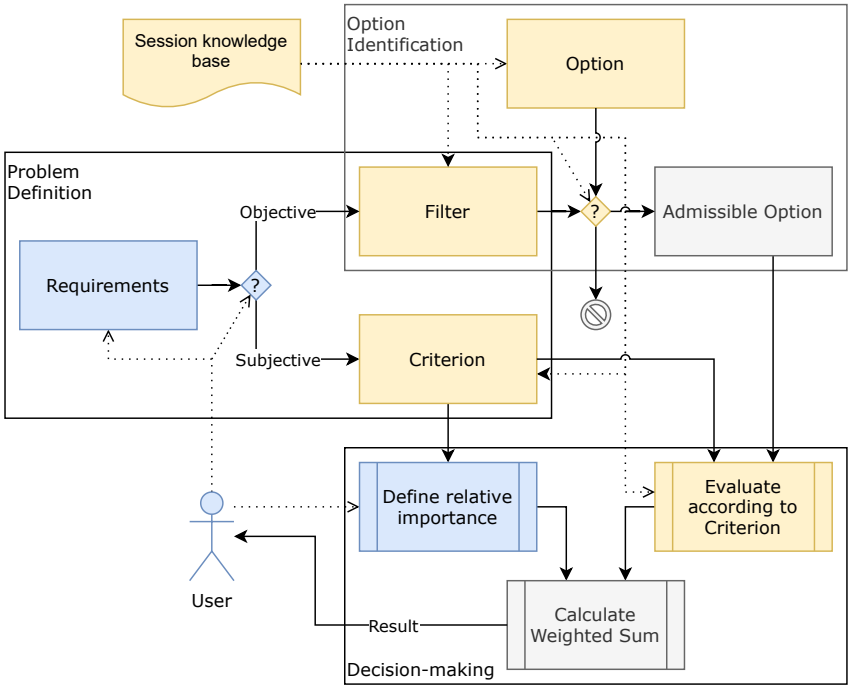


Figure 4.1: Integration of knowledge base into WSM method.

ideally automated by the application, and in a portable and versionable format for easy storage and CI/CD, e.g. with a dataframe, CSV or JSON file. MetaTool will also be subject to some non-functional objectives imposed by the company.

The aforementioned objectives induce the following set of requirements:

1. Functional requirements:

    a) Users must be able to import and export their knowledge base.

    b) Users must be able to perform Create, Read, Update, Delete (CRUD) - create, read, update, and delete - operations on their knowledge base.

    c) Users must be able to provide a set of factors and obtain a recommendation.

    d) Users must be able to consult factors available for a set of domains.

    e) Users must be able to consult options available for a set of roles.

    f) Users must be able to consult detailed information about domains and factors.

    g) Users must be able to consult detailed information about roles and options.

    h) Maintainers must be able to update the base knowledge - domains and factors.

    i) Maintainers must be able to update the base catalog - roles and options.

    j) The application must be able to validate knowledge bases.

    k) The application must be able to indicate invalid knowledge.

    l) The application must be able to import valid knowledge bases.

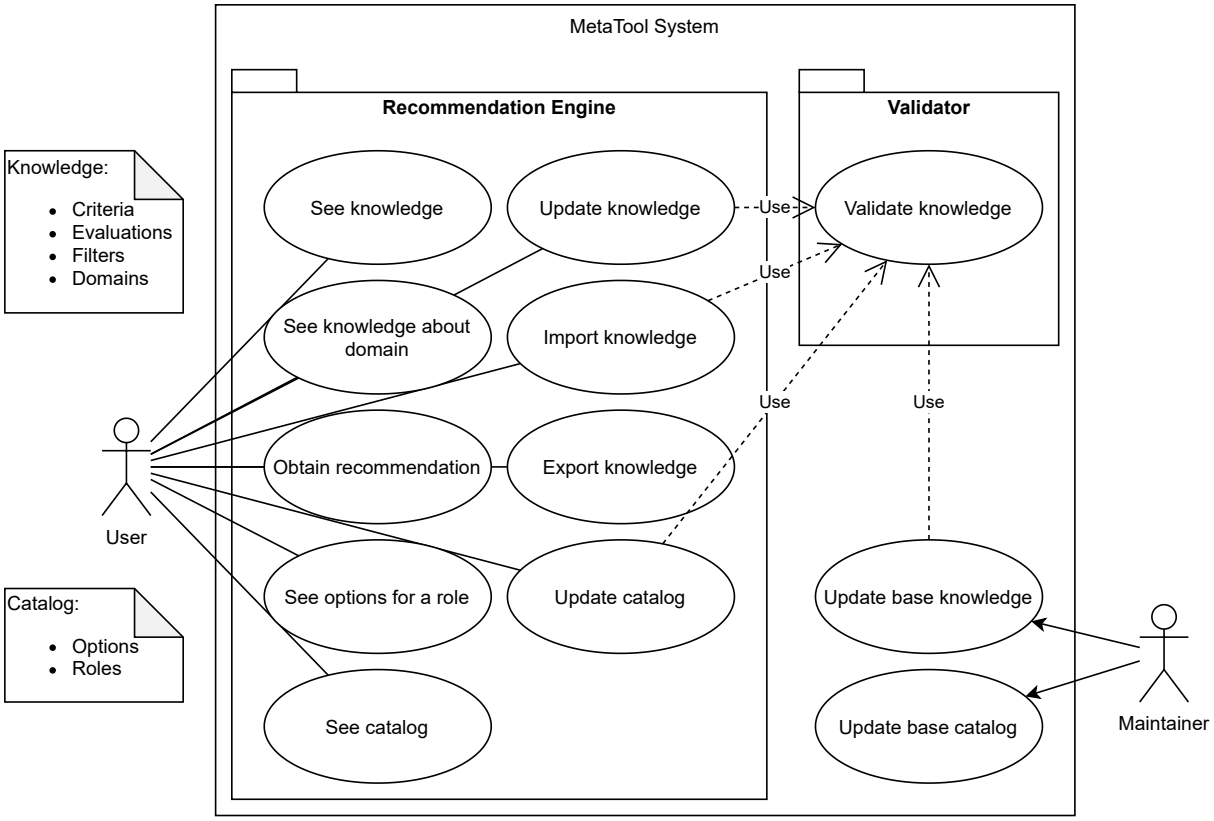


Figure 4.2: MetaTool use case diagram.

m) The application must allow different users to have different knowledge bases.

2. Non-functional requirements:

   a) The application must be a web-based application.

   b) The application must have its source code and documentation written in English.

   c) The application must only be accessible through internal corporate network.

   d) The application must be written in Python 3.

   e) The application must be deployed in an AKS instance.

Figure 4.2 models the use cases for this application from the set of functional requirements.

4.2 Architecture overview

Conceptually, the application is split into three major functional groups:

- Recommendation Engine: Component that holds all the necessary logic and data structures for the decision matrix. It implements 1c and keeps the session knowledge base.

- Validator: Parser that ensures consistency of the knowledge base, and alerts the user in case some error is found, fulfilling requirements 1j, 1k, and 1l.

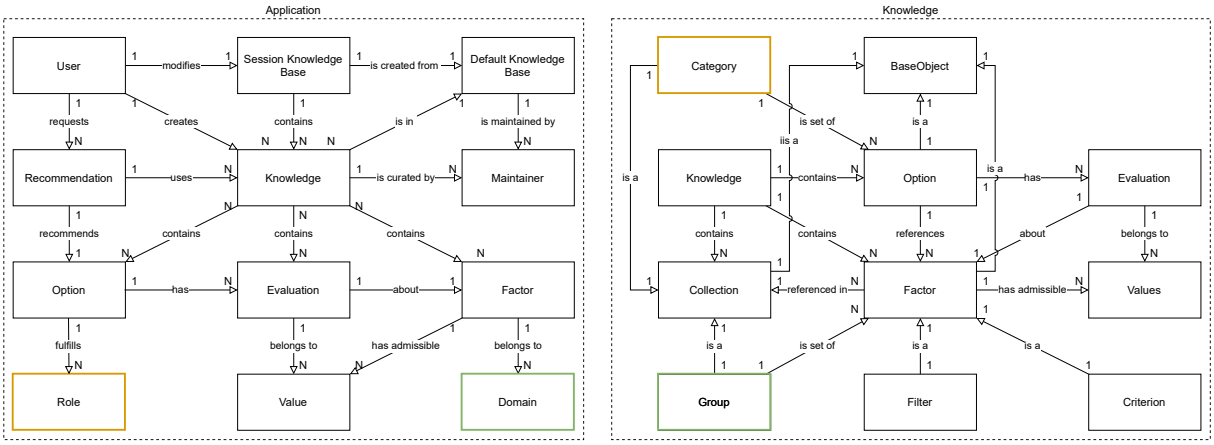


Figure 4.3: MetaTool domain diagram.

- Presentation: Front-end components that interface with the recommendation engine, allowing users to access application's functionalities such as navigate and manipulate session knowledge, provide input for a recommendation, see its result, and request knowledge import or export. This fulfills requirements 1b, 1d, 1e, 1f, and 1g.

Requirements 2a, 2c and 2e can be ensured by deploying the application, for example, as a Docker container, and using the aforementioned approach of applying an IP whitelist to the virtual network's rules, allowing through just those of the company's VPN exit nodes. This still leaves 1a, 1h, 1i and 1m unaccounted for, coincidentally all associated with the user-maintainer separation.

It was decided to make MetaTool have a default knowledge base stored in disk i.e. the one curated by the maintainers, thus modifiable by them. Then, whenever a user accesses the website, it would load a copy into his session, which would be the subject of all his modifications. Doing so grants it the user separation required by 1m. In order to comply with 1h and 1i, it was decided that users would only be able to save their contributions to memory, but not to disk. If desired, users could download their knowledge, and pass it to maintainers for verification. Lastly, requirement 2d will be met with the use of Streamlit. [1] The approach chosen took into account the fact that we are dealing with a web application, which can implement a session state and use it to hold the knowledge, and the need for knowledge to be dumped into some file format to comply with 1a. For convenience, the knowledge file would adopt the JSON format. For the sake of abstraction, when referring to knowledge and its structure, instead of "Role"and "Domain", these will be referred to as "Category"and "Group", respectively, as highlighted in figure III.7.

4.2.1 Web application

The application is split into two major blocks: a front-end layer encompassing the necessary components for the user interface, plus a logic layer with the control logic, decision matrix implementation and knowledge

[1] Streamlit is a free and open-source Python framework for building web apps for Machine Learning and Data Science. https://streamlit.io

storage, plus an implementation of session state[2]. Figure 4.5 shows this application in the light of a class diagram. Its components are as follows:

- evaluator.py - render the ‘Evaluator‘ application.
- explorer.py - render the ‘Explore Catalog‘ and ‘Explore Knowledge‘ applications.
- form.py - render the ‘Recommendation Engine‘ application.
- index.py - code entry point, define Streamlit properties and render features common to all pages: logo, MultiApp navigation.
- settings.py - render the ‘Settings‘ application.
- multiapp.py - controller that implements navigation through multiple pages, executes the app() function of the currently selected application.
- recommendation.py - implement recommendation logic and data structures.
- sessionstate.py - stores a globally available state that is kept through multiple reruns of the python code.
- utils.py - accessory functions for list and string manipulation, download button and custom styles.
- validator.py - parse knowledge to check if its contents represent a consistent knowledge base.

Index is the entry-point component, responsible for starting the web application, defining its outer elements e.g. page title, logo, and watermark, plus registering the four front-end applications - form, explore, evaluator, and settings - with a new MultiApp instance. MultiApp - MetaTool’s controller - implements a globally available navigation menu for those applications, and keeps one instance of SessionState, used to store all information associated with a user’s session, i.e. which page they are visiting, parameters e.g. content sorting criteria, color schemes, inputs e.g. criteria for a recommendation, and an instance of Recommendation, which keeps the knowledge base. SessionState is completely generic and in no way associated with the domain of this application.

The functions that implement MetaTool’s four front-end applications can be categorized into four purposes:

- *app* - main function with the purpose of preparing the base environment for that front-end application, including app-specific variables and navigation between its own pages.
- *page* - function to render exactly one view of the front-end application, receives session state, where all information is stored, and aware of the current *page_explorer_object*.

[2] MetaTool is built on Streamlit 0.82.0. This feature has since been added and is now available for release 0.84.0 and later - https://github.com/streamlit/streamlit/releases/tag/0.84.0

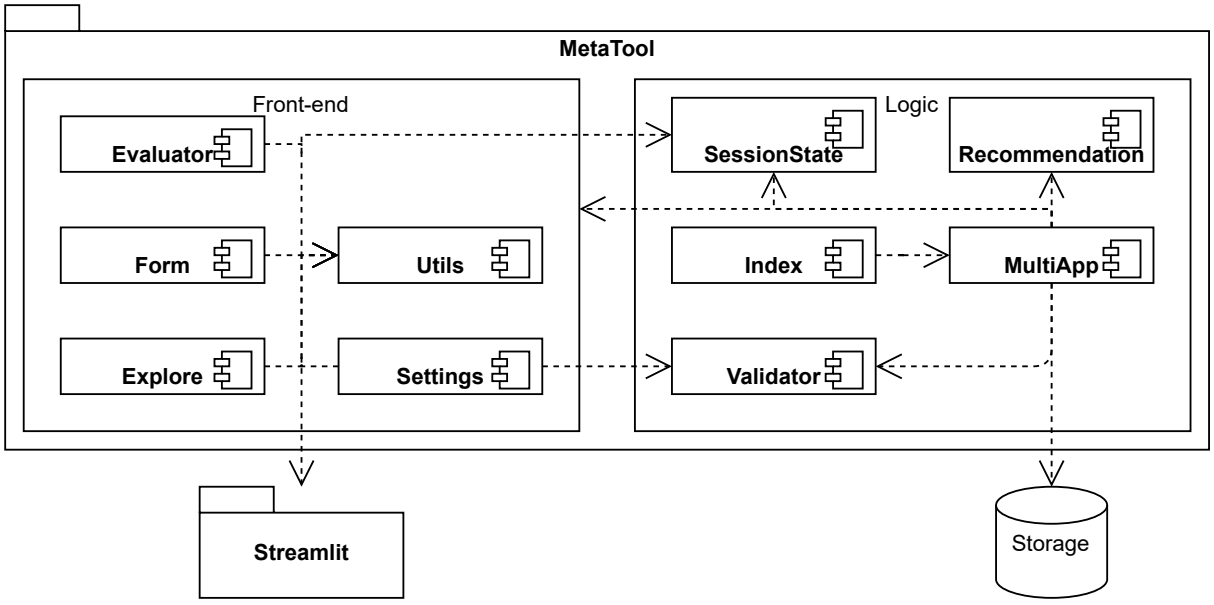


Figure 4.4: MetaTool web application component diagram.

- *render* - function to render a type of object as a row in a list, receives the current session state plus the identifier of the object to be rendered.
- *form* - function to render a view with the purpose of receiving user input for a particular object, used within a *page*.

For the purpose of example, a user consulting the detailed information of a given role e.g. orchestration would have invoked *page_details_category(s)*, which uses *category_form()*. The user would be presented with the information for that particular role and widgets to alter it. Should he perform modifications such as changing the name, the function will call for its application directly on the referenced object, which points to the category in his session state. Since the same session state is used for all of his interactions, those changes will take effect across the entire web application, ensuring consistency.

For the recommendation logic, this application relies on a single class - *Recommendation* - that operates as a single recommendation process, and is thus in charge of the yellow components at figure 4.1. MetaTool has a three-stage process for generating a recommendation:

1. User identifies the *must-have* requirements for his use-case, indicating a set of filters and their respective admissible values, allowing for the identification of admissible options.
2. User identifies criteria that describe his desires, and informs the application of how important each criteria is to him.
3. User is presented with the results of his recommendation, displaying

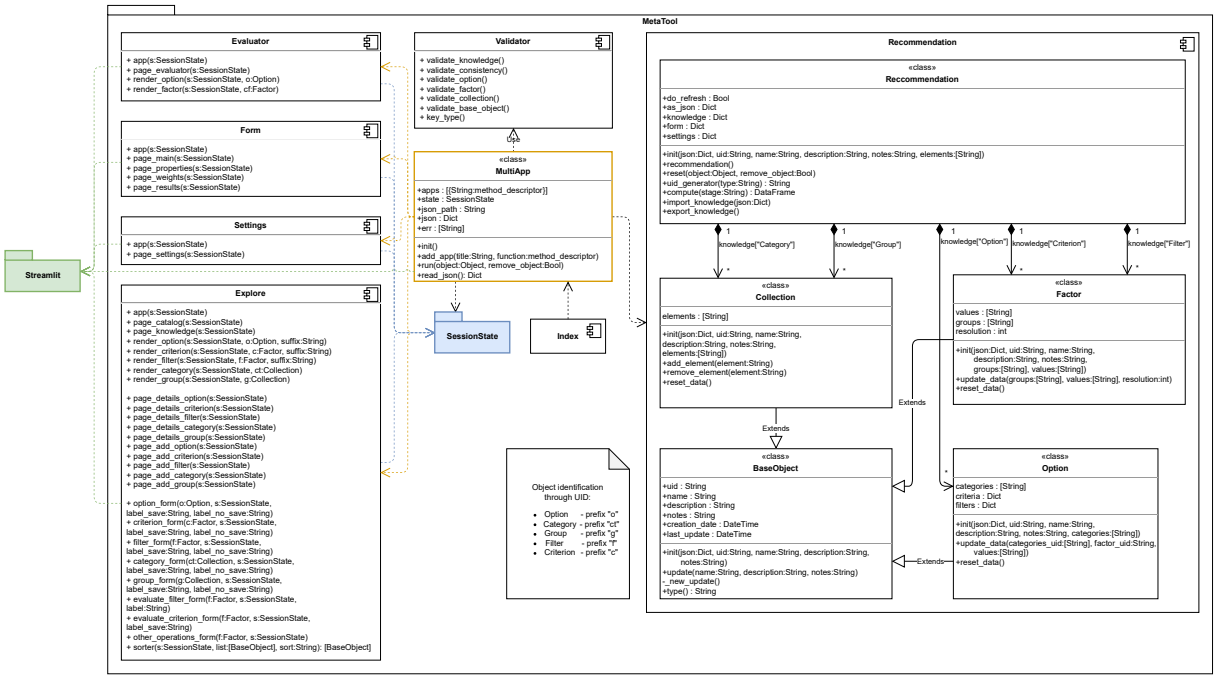


Figure 4.5: MetaTool web application class diagram.

4.2.2 Knowledge base

Due to the desire for a fast user interface, the knowledge is stored as a multi-layer dictionary, as Python implements these as hash tables. All knowledge objects are extensions of a generic class named *BaseObject*. It provides all the common attributes and methods for knowledge objects. From it, other classes extend its definition with the properties associated with each concept. This leads to the implementation of three other classes - *Collection*, *Factor*, and *Object*.

The first two classes are not then extended to produce what would be *Group* and *Category*, and *Filter* and *Criterion* respectively. Although *Group* and *Category* represent different concepts, their translation into code is exactly the same, only diverging in the types of elements they associate. *Filter* and *Criterion* would not be as similar. Even though both have a set of values that represent its scope, *Criterion* would have been more specific given that the indication of its relative weight needs to be on an axis common to all criteria.

This implementation makes use of the unique identifier inherited from *BaseObject* to differentiate between purposes for the same class. These identifiers are formatted as $prefix\_serial$, with $prefix$ indicating its type, and $serial$, a 32 digit number, differentiating between objects of the same type. The association between these abstractions, their prefixes and their concept is defined in table 4.1. Annex II shows an example of a knowledge base exported as a JSON file, featuring data about the three orchestrators featured in section 3.3, used in section 4.3 and annex III.

Class	Abstracts	Prefix	Concept
Collection	Category	c	Role
	Group	g	Domain
Factor	Criterion	c	Criteria
	Filter	f	Filters

Table 4.1: Implementation conceptual abstractions.

4.3 User interface walkthrough

The user interface is comprised of four applications, as aforementioned. Users can navigate between them on the sidebar through the radial buttons. This action triggers the corresponding *app()* function, rendering the sub-application. Extra content placed on the sidebar is always located below this navigation.

4.3.1 Consult knowledge

The user interface implements the separation between decision knowledge and options catalog as two separate pages under the *Explore* component, accessible through the radial selector. When visiting this application, users are presented with the respective list of objects added to the knowledge base, sortable as per table 4.2. Each element is rendered with the same format, with its descriptive data - name, description, notes, as well as values and collections and number of references where applicable. The text used within the *Notes* field of a *BaseObject* is interpreted with a Markdown format, but does not allow for custom HTML within it.

Sorting Method	Available to
Alphabetical	BaseObject
Last Update	BaseObject
Creation Date	BaseObject
Most Data	Option
Most Roles	Option
Most Referenced	Factor
Number of values	Factor

Table 4.2: Explorer sorting methods.

All objects displayed in this application have a button placed underneath their names for the user to open it in a more detailed page. The application also provides a shortcut menu on its lower sidebar for the user to create new objects and add them to his knowledge base. As required by requirement 1b, all knowledge objects, that is, instances of *BaseObject*, allow for CRUD operations on their descriptive data - name, description, notes, as well as values and collections, if applicable.

In the *Explore Catalog* sub-application the user is served with all tools currently known i.e. registered in his session's knowledge base, sorted alphabetically by default. For better navigation with big catalogs, it

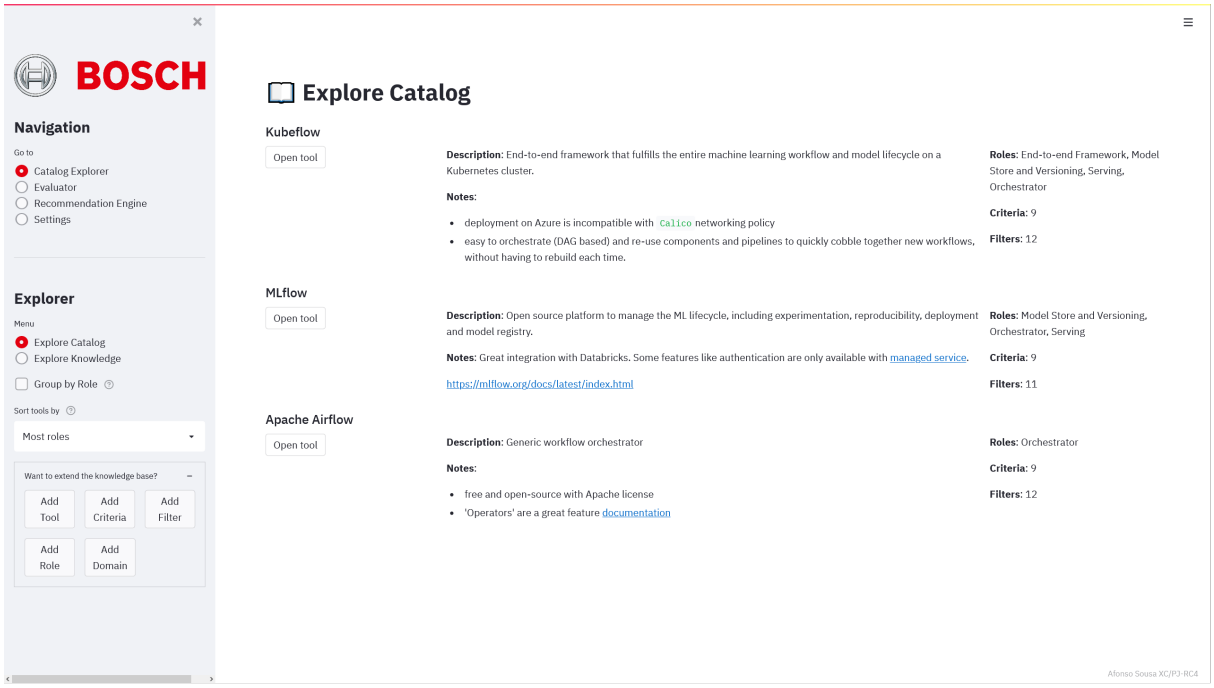


Figure 4.6: MetaTool UI: Explorer - Explore Catalog page.

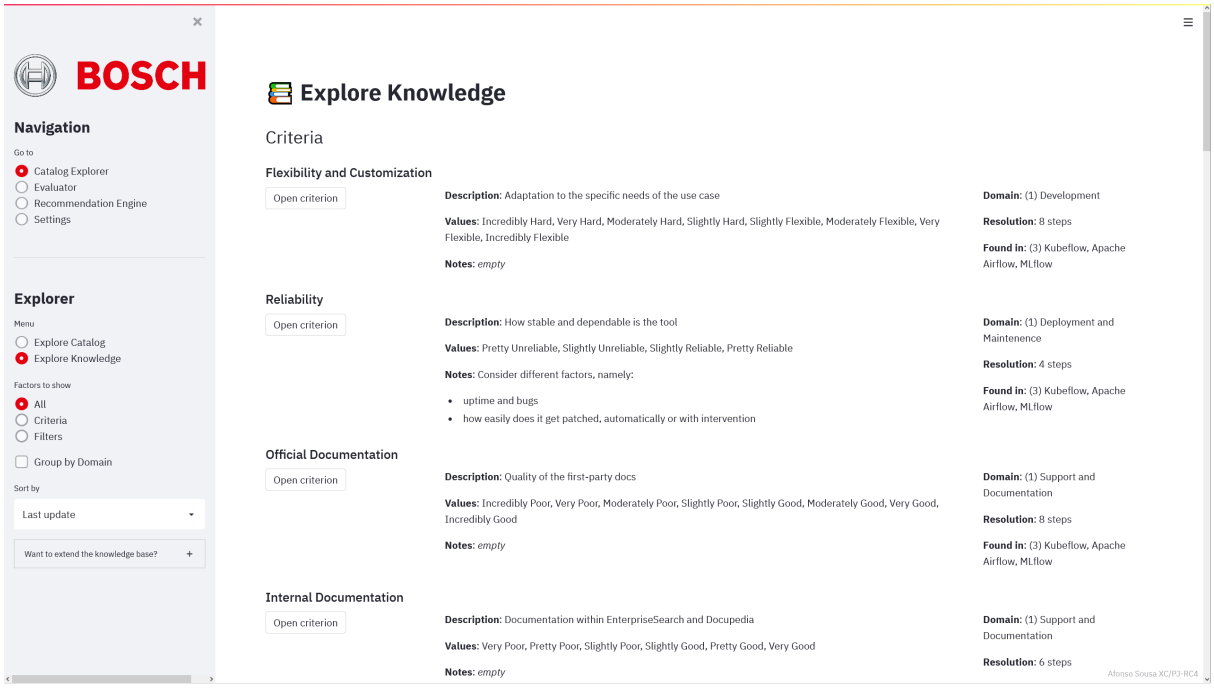


Figure 4.7: MetaTool UI: Explorer - Explore Knowledge page.

also provides a checkbox to collapse the list into a set of expandable boxes, one for each known role, with the list of tools that fulfill it. Tools that fulfill more than one role will be shown on all corresponding lists.

The other *Explorer* sub-application, *Explore Knowledge*, is analogous to the previous, adding some necessary adjustments to accommodate criteria and filters, including renaming some UI components, adjusting sorting methods (table 4.2) and providing a textual indicator of the type of factor being shown. Due to it dealing with two concepts, new radial buttons are made available, indicating which to show - just

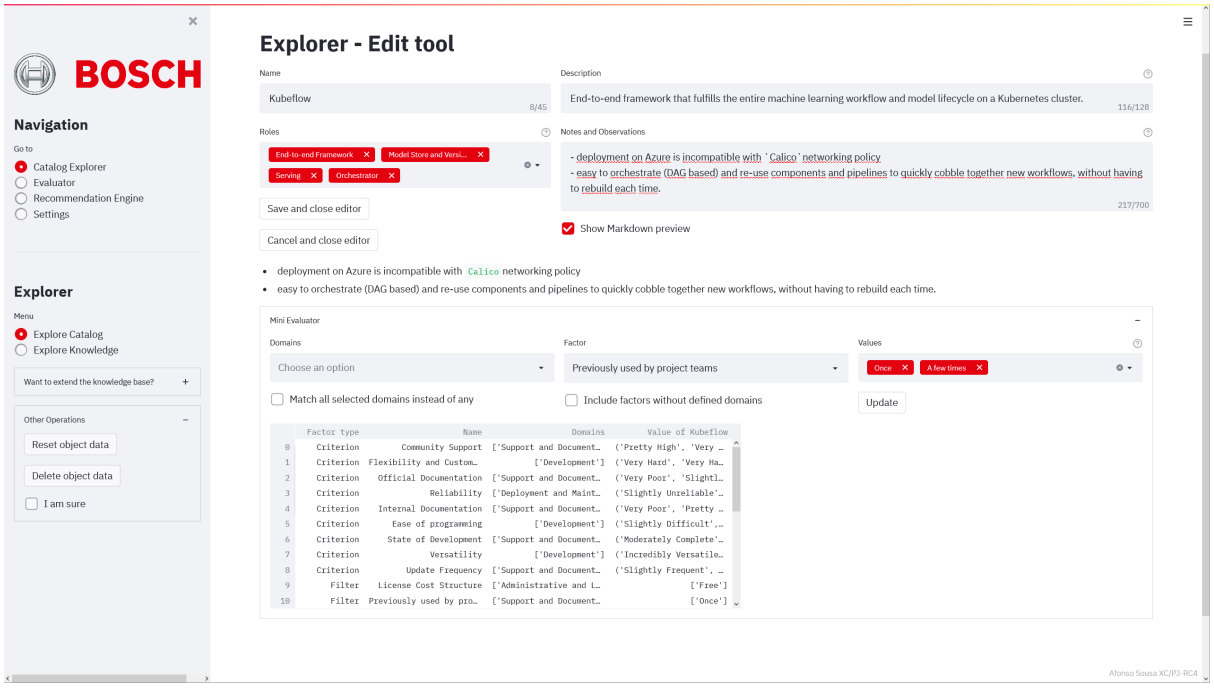


Figure 4.8: MetaTool UI: Explorer - Edit tool page.

criteria, just filters, or both. Sorting is done alphabetically by default. Both criteria and filters can also be grouped, now according to their domains. These groupings will still separate the aforementioned factor types as selected.

On either sub-application, when grouping content by their collections, the currently selected sorting method is mapped into each of the resulting lists.

MetaTool has pages dedicated to displaying each type of knowledge object. If the user presses the button to open the object in more detail, the application takes the corresponding view and sets the widget values to those currently in the object (figure 4.8). Conversely, if the user presses the button to add a new object, these are displayed with default values i.e. blank input boxes. In the latter, if the object is of type *Option* or *Factor*, the page will not have a mini-evaluator at the bottom of the page, as adding evaluations relative to an object before it is created could lead to inconsistencies. Refer to figures III.5, III.7 and III.6for the detailed page of domains and roles, respectively.

For evaluations, i.e. how an option's values relate with those of a factor, users have two places where they can provide that input. The first is the *Evaluator* application (figure 4.10), which is completely dedicated to this activity. It features a set of widgets for quick pin-pointing of what *Option* is to be evaluated, as well as the target factor. This could be either a *Criterion* or a *Filter* and, depending on the type, the application will display different widgets suited for its inputs.

If a user visits the details page of an *Option* or factor, he is in effect selecting one of the two entities of an evaluation. As such, the details page for these feature a mini-evaluator, allowing the user to quickly update multiple evaluations. It also provides a tabular view of the object in question is evaluated with other objects.

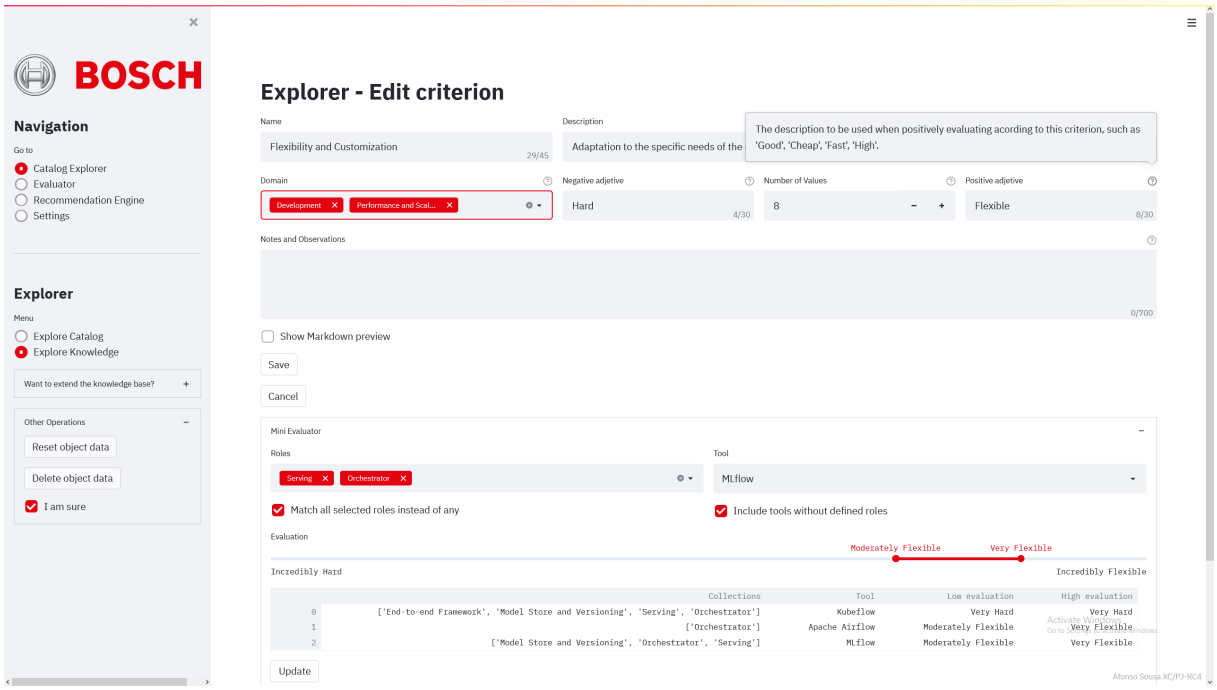


Figure 4.9: MetaTool UI: Explorer - Edit criterion page.

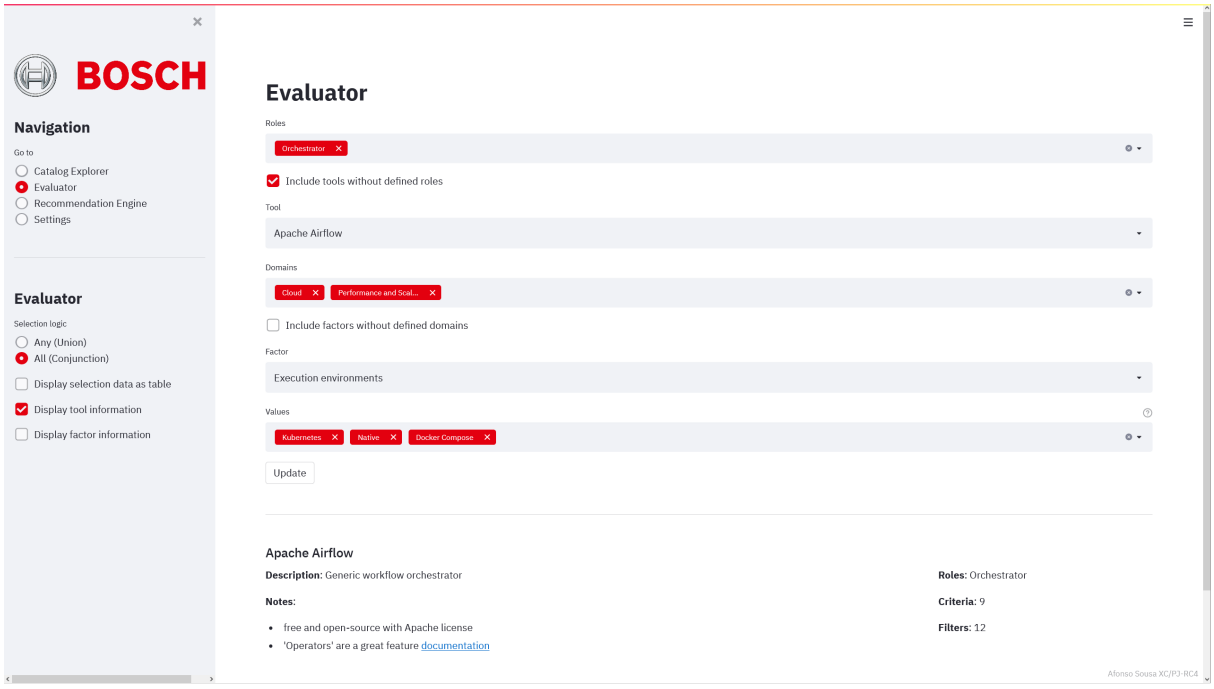


Figure 4.10: MetaTool UI: Evaluator page.

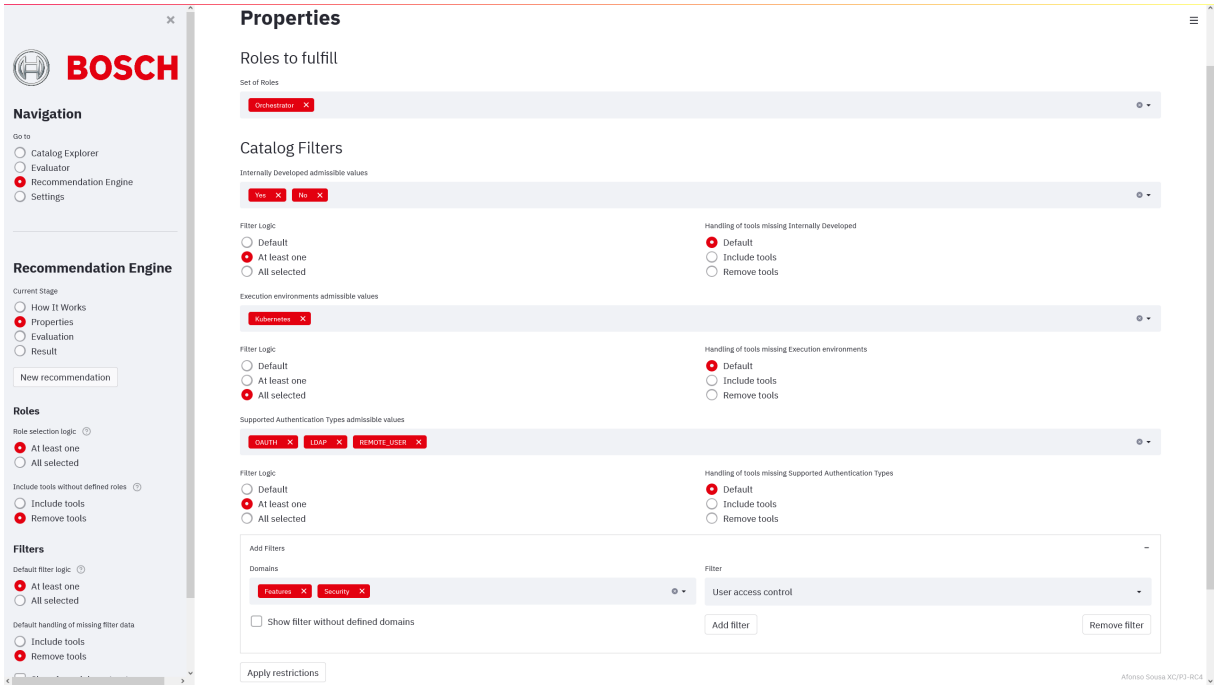


Figure 4.11: MetaTool UI: Recommendation filter selection page.

4.3.2 Obtain a recommendation

The recommendation process begins with the selection of filters. The user uses a selection box to indicate what to add, or remove, from his form. To facilitate navigation, the UI allows filtering these according to the domains they are associated with.

Added filters are displayed with a multi-selection box, allowing the user to indicate the set of values he desires. These values can be interpreted differently, depending on how the user wishes to interpret and apply them. MetaTool allows the user to indicate whether to use an existential ($\exists$) or universal ($\forall$) quantifier. For situations when some tools may not have data in relation to the filter in question, these can be assumed to be either compliant or not compliant. Selection settings can be applied globally through the sidebar selectors, or individually. After the filters are applied, the user is presented with a page for the introduction of criteria. In the same fashion as with filters in the previous page, the user indicates which criteria to add. Those are displayed with an interval slider, used to indicate the weight of the criterion. Being an interval is important to account for situations where the user is not absolutely sure about his needs. In those cases, the computation will be performed with an average of the two bounds.

As before, the application lets the user decide whether to include or exclude options without an evaluation for some of the criteria i.e. how to handle undefined knowledge. Proceeding to the next page submits the form for computation by the *Recommendation* component, producing the MCDM matrix and dumping its results into a DataFrame, as seen on the lower half of the page. The results page then transforms that data into interactive plots, allowing easy visualization of the outcome. Here, the user can alter the settings used for plotting e.g. the plot type as shown on the sidebar. These results can then be exported from the applications, if desired. The data used and produced with this recommendation can be sorted within the

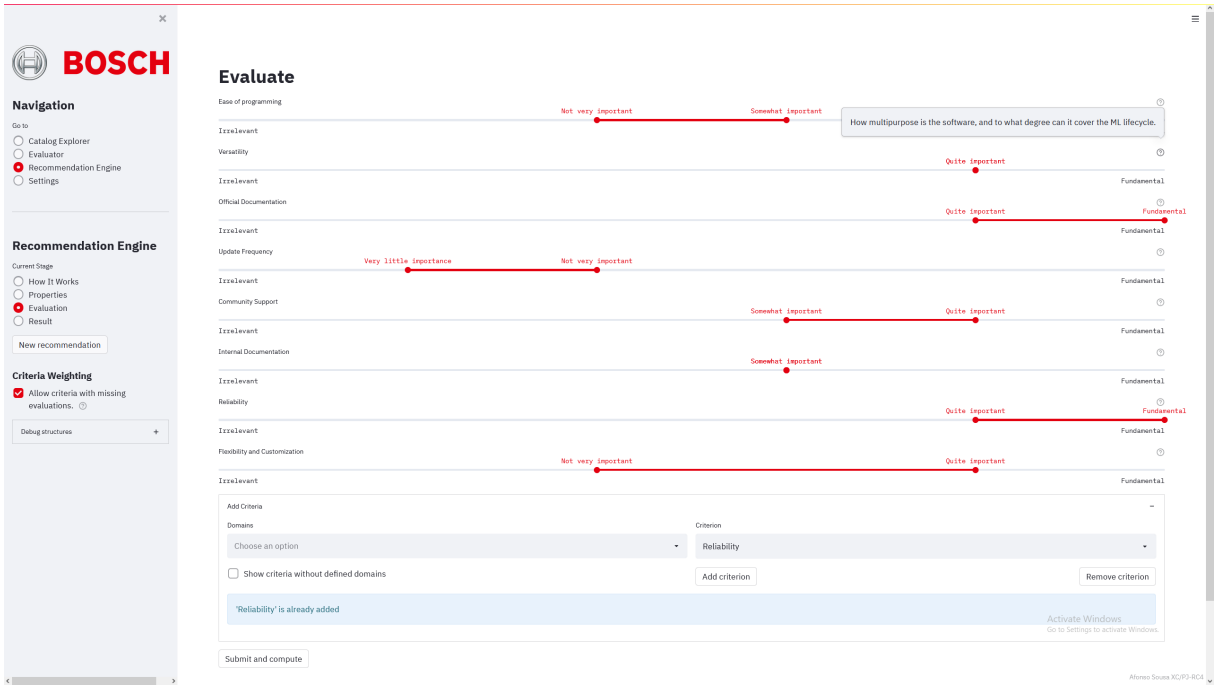


Figure 4.12: MetaTool UI: Recommendation criteria selection page.

	Option	Criterion	Linguistic	Resolution	Option evaluation	Usecase importance	Weight	Factor score
0	Kubeflow	Ease of programming	['Difficult', 'Easy']	6	[2, 2]	('Not very important', 'Somewhat important')	0.8333	1.7361
1	Kubeflow	Versatility	['Narrow', 'Versatile']	8	[7, 7]	('Quite important', 'Quite important')	3.5000	7.2917
2	Kubeflow	Official Documentation	['Poor', 'Good']	8	[1, 4]	('Quite important', 'Fundamental')	1.5000	3.1250
3	Kubeflow	Update Frequency	['Infrequent', 'Frequent']	6	[3, 3]	('Very little importance', 'Not very important')	0.7500	1.5625
4	Kubeflow	Community Support	['Low', 'High']	6	[4, 5]	('Somewhat important', 'Quite important')	2.6667	5.5556
5	Kubeflow	Internal Documentation	['Poor', 'Good']	6	[0, 1]	('Somewhat important', 'Somewhat important')	0.2500	0.5208
6	Kubeflow	Reliability	['Unreliable', 'Reliable']	4	[1, 3]	('Quite important', 'Fundamental')	2.3750	4.9479
7	Kubeflow	Flexibility and Customization	['Hard', 'Flexible']	8	[1, 1]	('Not very important', 'Quite important')	0.3750	0.7813
8	Apache Airflow	Ease of programming	['Difficult', 'Easy']	6	[4, 5]	('Not very important', 'Somewhat important')	1.9167	3.9931
9	Apache Airflow	Versatility	['Narrow', 'Versatile']	8	[0, 0]	('Quite important', 'Quite important')	0	0
10	Apache Airflow	Official Documentation	['Poor', 'Good']	8	[6, 7]	('Quite important', 'Fundamental')	3.6875	7.6823
11	Apache Airflow	Update Frequency	['Infrequent', 'Frequent']	6	[2, 3]	('Very little importance', 'Not very important')	0.6667	1.3889
12	Apache Airflow	Community Support	['Low', 'High']	6	[4, 5]	('Somewhat important', 'Quite important')	2.6667	5.5556
13	Apache Airflow	Internal Documentation	['Poor', 'Good']	6	[0, 0]	('Somewhat important', 'Somewhat important')	0	0

Figure 4.13: MetaTool UI: Recommendation results page.

interactive DataFrame, increasing the ease of use and explainability of the result.

4.3.3 Knowledge IO operations

The settings page is where the user can interface with the entire session knowledge base. The sidebar provides an upload widget for the user to send knowledge JSONs into his session. This content is put through validation mechanisms to ensure its consistency. Errors discovered during this process are presented here.

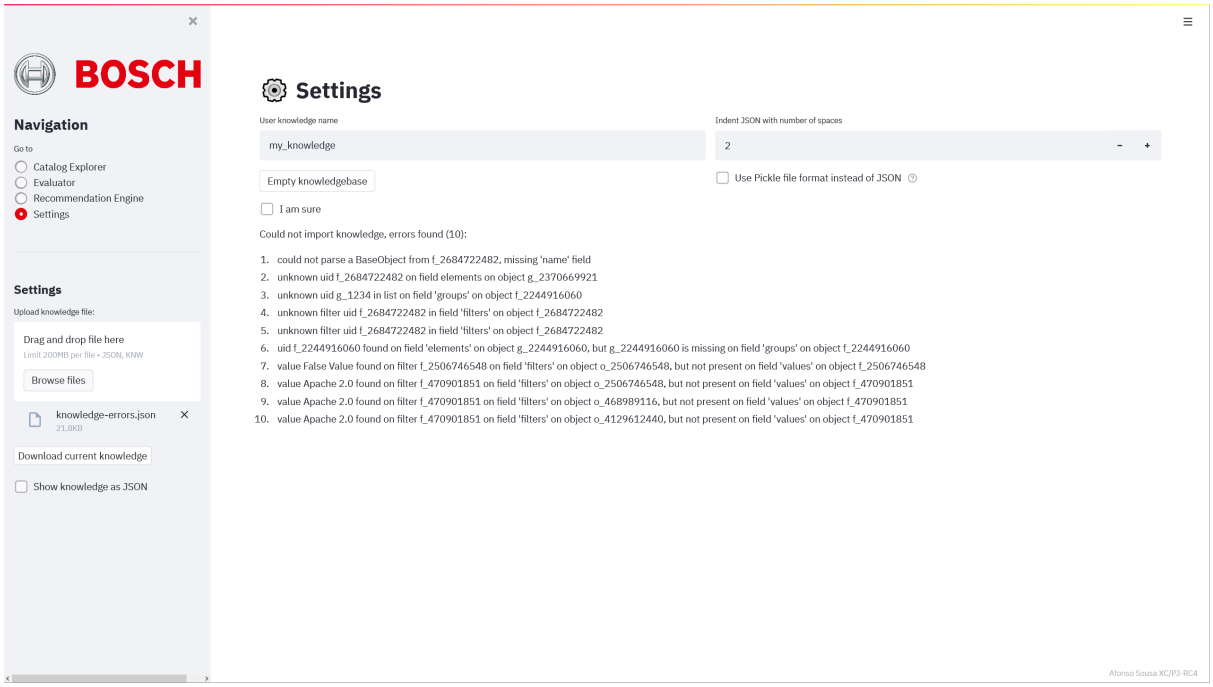


Figure 4.14: MetaTool UI: Knowledge validation detecting errors in uploaded file.

Figure 4.14 shows a situation with four manually inserted mistakes: delete a *name* field, remove one value from the list of values for a Filter, make an Option have an evaluation with a value that does not exist, and have a non-symmetric reference between a Collection and one of its elements i.e. one not mentioning the other.

This page also allows the user to download his session knowledge, either as a JSON or serialized into a Pickle file format, and visualize what the exported knowledge base would look like. For convenience in case the user wants to start anew, this page allows emptying the entire session knowledge, if the confirmation checkbox is selected. Figure III.8shows this page in more detail.

Chapter 5

Results and discussion

5.1 Gen2 and Kubeflow

The selection process of an adequate orchestrator was performed according to the internal process of the company in order to ensure it would be well adjusted to its operations.

However, due to changes in the direction of the project, it was not possible to take this process further, meaning it is not possible to determine how well adjusted Kubeflow would be when used for proper model experimentation, not just a proof of concept. It would be interesting to see how it would affect the perceptions about it, and what kind of criteria would be discovered.

Still, its deployment was performed to corporate standards, and there are some findings worthy of analysis:

- During consultation with other teams from different locations worldwide, when questioned about their experience with ML orchestrators, there was a dominant factor that lead them to not adopt any of the ones being considered: they are not offered as a managed service. Teams wanted the ability to simply rent an instance with the orchestrator deployed and maintained by the provider, akin to how it is currently done with container orchestration and Kubernetes.

  Their motivation was centered around reducing costs, which meant reducing time spend managing infrastructure. If we take into account the quick release cycles of the most popular ML orchestrators, and that some, including Kubeflow, are community projects, we find most tools will be very time consuming if adopted at their current state, as can be seen by the problems found during Kubeflow's deployment (section 3.4). As such, instead of adopting orchestrators, they resorted to simpler automation tools such as Jenkins, Azure DevOps, or simple shell scripts.

- During the course of the project, it was decided that Kubeflow would not be adopted and the AKS instance would have to be decommissioned because other parts of the project were using AWS

and its ML platform SageMaker, and taking the same approach would allow for better synergies, collaboration and knowledge sharing.

Naturally, this move comes at the cost of the effort spent on the previous approach and has some downsides i.e. vendor lock-in, but it reveals just how important it can be to adopt solutions previously used within the organization, even with increased costs and sacrifice of desired attributes, since their integration process will be more streamlined.

This leads to the conclusion that solutions such as Kubeflow could experience increased adoption if they too became offered as managed service, and remained easy to integrate with major cloud providers. This seems unlikely, precisely due to how immature this field still is, turning this whole situation into a dilemma: practitioners do not use them because they require lots of effort, leading to the desire of manages services; but if they are offered as such, they would probably be mature enough to be managed internally, as occurs with Jenkins. Thus, it becomes a matter of waiting for orchestrators to become mature.

From this we conclude the objectives 1, 2 and 3 of this thesis were reached, but 4 was not due to aforementioned changes in the direction of the project.

5.2 MetaTool

In general, the MetaTool web application meets the objectives outlined for it. It speeds up the tool discovery process because it keeps them available on a single location searchable with different factors, thanks to grouping and the recommendation process, since users can use it without any criteria to obtain the set of admissible options. The recommendation process also facilitates the comparison of admissible options with its automation and easy to interpret results.

In terms of the last two objectives i.e. ease of use and promote knowledge retention/build organizational knowledge, the current approach has some downsides. First, although easy to maintain technologically - Python application with a web server deployed as a simple Docker container, it does require human intervention in order to maintain a high quality knowledge base. Then, in relation to knowledge retention, the current architecture will suffer from the problems usually associated with version control, possibly hindering its adoption - what if a user got his knowledge base from an old default, and has since made many modifications in one direction, but someone else has done the same in another direction, not merging nor rebasing with newer defaults? Will the maintainer be able to handle such big diversions? What if users become too attached to their knowledge and refuse to rebase? Would a simple version control system e.g. GIT allow maintainers to sustain the application in the long term, with thousands of different tools and factors? Also, it is absolutely possible for different perspectives to be valid at the same time, but each knowledge base can only hold one evaluation between pairs. The current implementation would imply disregarding all but one.

Besides these points, there are other considerations that should be taken into account:

- As the catalog grows and becomes ever more rich, users might become too dependant on it, and not even consider looking for alternatives yet unknown, leading to biased selection.
- Although the recommendation form allows users to indicate logic quantifiers for the application of inputs, these might not be enough for more complex situations, such as "either it satisfies A, or B and C".
- The application treats omission as a lack of an evaluation. Thus, users must create some null value to differentiate between this and situations where a tool does not meet any of the values associated with the target factor.
- A platform with user-generated content needs serious commitment before achieving critical mass. Without it, it will be forgotten.
- As users become more experienced in one particular field, they will be less interested in using this tool, as they will be aware of what exists and what is best for the use case. The negative impact is doubled by the loss of potential highly quality knowledge.

5.3 Lessons learned

In order to build the initial knowledge base, there were a series of interviews planned for knowledge collection, which would take place after MetaTool's development, so that collected knowledge could immediately be added into an initial knowledge base. By doing so, it was expected that the initial version would be quick to create, and rich in substance.

Unexpectedly, the interviewed practitioners did not provide much feedback besides the reason why their team does not use ML orchestrators in general. This was not a total loss as it provides data for excluding non-admissible options, but does not give any information about criteria related to usability, performance, etc. This did not occur due to lack of contribution, as many people, mostly from other Bosch locations, were willing to be interviewed and provided their feedback. This left actual experimentation as the only viable approach, as consulting with people outside the company would risk making the knowledge impure and poorly adjusted to internal needs. Thus, there was a small reference pipeline developed, which was then used with all of the three orchestrators in order to obtain first-hand experience and accurately populate the knowledge base.

5.4 Future work

Taking into account the current state of the application, there are multiple areas where improvement can be made if the application gets further development:

- As MetaTool's user base grows, it will be interesting to see how its users are interacting with the tool, particularly what type of tools are most often demanded, and what criteria they include for

their recommendations. Such information can provide important insights, both on a per-team level, as well as to the company in general. It could, for example, indicate areas of expertise where the company may need to hire more developers, or invest in formations.

- So far, the only users of MetaTool are the members of the team hosting this masters dissertation, and the application has been designed with their input. It would be good to get feedback from other teams as they are likely to have constructive ideas for how to improve the application.
- The dependency on maintainers is a tangible operational cost to its operation. Implementing a way for automatic knowledge base merging, albeit technically challenging, could tremendously improve the company's willingness to adopt MetaTool on a large scale.

5.5 Closing note

In order to produce MetaTool and jump-start the knowledge base, its development went through a series of time-consuming stages, culminating with the presented results. Still, MetaTool's success is dependant on its level of adoption by the teams at Bosch. Hopefully, it will prove itself valuable over time.

Chapter 6

Conclusion

6.1 Lessons learned

In order to build the initial knowledge base, there were a series of interviews planned for knowledge collection, which would take place after MetaTool's development, so that collected knowledge could immediately be added into an initial knowledge base. By doing so, it was expected that the initial version would be quick to create, and rich in substance.

Unexpectedly, the interviewed practitioners did not provide much feedback besides why it is their team does not use ML orchestrators in general. This was not a total loss as it provides data for excluding non-admissible options, but does not five any information about criteria related to usability, performance, etc. This did not occur due to lack of contribution, as many people, mostly from other Bosch locations, were willing to be interviewed and provided their feedback. This left actual experimentation as the only viable approach, as consulting with people outside the company would risk making the knowledge impure and poorly adjusted to internal needs. Thus, there was a small reference pipeline developed, which was then used with all of the three orchestrators in order to obtain first-hand experience and accurately populate the knowledge base.

6.2 Future work

Taking into account the current state of the application, there are multiple areas where improvement can be made if the application gets further development:

- As MetaTool's user base grows, it will be interesting to see how its users are interacting with the tool, particularly what type of tools are most often demanded, and what criteria they include for their recommendations. Such information can provide important insights, both on a per-team level,

as well as to the company in general. It could, for example, indicate areas of expertise where the company may need to hire more developers, or invest in formations.

- So far, the only users of MetaTool are the members of the team hosting this masters dissertation, and the application has been designed with their input. It would be good to get feedback from other teams as they are likely to have constructive ideas for how to improve the application.
- The dependency on maintainers is a tangible operational cost to its operation. Implementing a way for automatic knowledge base merging, albeit technically challenging, could tremendously improve the company's willingness to adopt MetaTool on a large scale.

6.3 Closing note

In order to produce MetaTool and jump-start the knowledge base, its development went through a series of time-consuming stages, culminating with the presented results. Still, MetaTool's success is dependant on its level of adoption by the teams at Bosch. Hopefully, it will prove itself valuable over time.

Bibliography

Algorithmia. (2020). 2020 state of enterprise machine learning. Retrieved December 1, 2021, from https://info.algorithmia.com/hubfs/2019/Whitepapers/The-State-of-Enterprise-ML-2020/Algorithmia_2020_State_of_Enterprise_ML.pdf

Alla, S., & Adari, S. K. (2021). What is mlops? In *Beginning mlops with mlflow: Deploy models in aws sagemaker, google cloud, and microsoft azure* (pp. 79–124). DOI 10.1007/978-1-4842-6549-9_3. Berkeley, CA: Apress.

Amodei, D., & Hernandez, D. (2018). Ai and compute. Retrieved December 8, 2021, from https://openai.com/blog/ai-and-compute/

Ashmore, R., Calinescu, R., & Paterson, C. (2021). Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys (CSUR)*, *54*(5), 1–39.

Atamel, M. (2020, January 1). *Better service orchestration with workflows*. Retrieved November 4, 2021, from https://cloud.google.com/blog/topics/developers-practitioners/better-service-orchestration-workflows

Authors, T. K. (2020). Overview of Kubeflow Pipelines. Retrieved February 5, 2021, from /docs/pipelines/overview/pipelines-overview/

Bisong, E. (2019). Kubeflow and Kubeflow Pipelines. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform* (pp. 671–685). DOI 10.1007/978-1-4842-4470-8_46. Apress, Berkeley, CA.

Brys, M. (2019). Kubeflow — a machine learning toolkit for Kubernetes. Retrieved February 5, 2021, from https://medium.com/@michal.brys/kubeflow-a-machine-learning-toolkit-for-kubernetes-d8686f6c91b6

Chou, S.-Y., Chang, Y.-H., & Shen, C.-Y. (2008). A fuzzy simple additive weighting system under group decision-making for facility location selection with objective/subjective attributes. *European Journal of Operational Research*, *189*(1), 132–145. DOI 10.1016/j.ejor.2007.05.006.

Dean, J., Patterson, D., & Young, C. (2018). A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro*, *38*(02), 21–29. DOI 10.1109/MM.2018.112130030.

Google. (2020, January 7). *Mlops: Continuous delivery and automation pipelines in machine learning*. Retrieved September 3, 2021, from https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Gubala, T., & Hoheisel, A. (2007). Highly dynamic workflow orchestration for scientific applications. In *Coregrid integration workshop 2006 (ciw06)* (pp. 309–320).

Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Pearson Education.

IBM. (2020). What is Workflow? Retrieved February 4, 2021, from https://www.ibm.com/cloud/learn/workflow

Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is devops? a systematic mapping study on definitions and practices. In *Proceedings of the scientific workshop proceedings of xp2016*. DOI 10.1145/2962695.2962707, Edinburgh, Scotland, UK: Association for Computing Machinery.

Kanchanahalli, V., McKittrick, M., & Hughes, L. (2021, July 9). Nc-series. Retrieved December 15, 2021, from https://docs.microsoft.com/en-us/azure/virtual-machines/nc-series

Kanwarpal, M., Buck, A., Ghodratigohar, M., Zwiefel, E., Weaver, A., Eikelenboom, D., ... WendyRing. (2021). Machine learning devops guide. Retrieved from https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/ai-machine-learning-mlops

Lohr, H., Boher, S., Ross, E., Schmid, M., Wilhite, C., & Schonning, N. (2021, December 11). Windows container orchestration overview. Retrieved December 12, 2021, from https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/overview-container-orchestrators

Mäkinen, S. et al. (2021). Designing an open-source cloud-native mlops pipeline.

Makridakis, S. (2017). The forthcoming artificial intelligence (ai) revolution: Its impact on society and firms. *Futures*, *90*, 46–60.

Marcondes, F. S., Durães, D., Gonçalves, F., Fonseca, J., Machado, J., & Novais, P. (2021). In-vehicle violence detection in carpooling: A brief survey towards a general surveillance system. In Y. Dong, E. Herrera-Viedma, K. Matsui, S. Omatsu, A. González Briones, & S. Rodríguez González (Eds.), *Distributed computing and artificial intelligence, 17th international conference* (pp. 211–220). DOI 10.1007/978-3-030-53036-5_23. Cham: Springer International Publishing.

Microsoft. (2021). Machine learning operations maturity model. Retrieved from https://docs.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-maturity-model

Moore, B. A., Rougier, E., O'Malley, D., Srinivasan, G., Hunter, A., & Viswanathan, H. (2018). Predictive modeling of dynamic fracture growth in brittle materials with machine learning. *Computational Materials Science*, *148*, 46–53. DOI 10.1016/j.commatsci.2018.01.056.

Nguyen, T. T., & Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, *10*(4), 56–76. DOI 10.1109/SURV.2008.080406.

O'Leary, K., & Uchida, M. (2020). Common problems with Creating Machine Learning Pipelines from Existing Code. Retrieved October 29, 2020, from https://research.google/pubs/pub48984/

Ozkaya, I. (2019). The golden age of software engineering [from the editor]. *IEEE Software*, *36*(01), 4–10. DOI 10.1109/MS.2018.2877032.

Parkhe, M., Ann Hong, S., Damji, J., & Mewald, C. (2020). How to Share and Control ML Model Access with MLflow Model Registry. Retrieved February 5, 2021, from https://databricks.com/blog/2020/04/15/databricks-extends-mlflow-model-registry-with-enterprise-features.html

Perrault, R., Shoham, Y., Brynjolfsson, E., Clark, J., Etchemendy, J., Grosz, B., ... et al. (2019). The ai index 2019 annual report. AI Index Steering Committee, Human-Centered AI Institute, Stanford University, Stanford. Retrieved from https://hai.stanford.edu/sites/default/files/ai_index_2019_report.pdf

Quick Start — Airflow Documentation. (n.d.). Retrieved February 5, 2021, from https://airflow.apache.org/docs/apache-airflow/stable/start.html#basic-airflow-architecture

Raj, E. (2021). *Engineering mlops*. Birmingham, England: Packt Publishing.

Schmitt, M. (2020). Airflow vs Luigi vs Argo vs Kubeflow vs MLFlow. Retrieved February 5, 2021, from https://www.datarevenue.com/en-blog/airflow-vs-luigi-vs-argo-vs-mlflow-vs-kubeflow

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... Young, M. (2014). Machine learning: The high interest credit card of technical debt. In *Se4ml: Software engineering for machine learning (nips 2014 workshop)*.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... Dennison, D. (2015). Hidden Technical Debt in Machine Learning Systems. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28* (pp. 2503–2511). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

Singh, P. (2019). Airflow. In *Learn PySpark* (pp. 67–84). DOI 10.1007/978-1-4842-4961-1_4. Apress, Berkeley, CA.

TA, N. B. D., & NGUYEN, Q. S. (2018). Distributed machine learning on IAAS clouds. *Proceedings of the 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), Nanjing, China, 2018 November 23-25*. DOI 10.1109/CCIS.2018.8691150.

The Kubeflow Authors. (2021a, November 5). Authentication using oidc in azure. Retrieved December 15, 2021, from https://www.kubeflow.org/docs/distributions/azure/authentication-oidc/

The Kubeflow Authors. (2021b, November 5). Install kubeflow - instructions for deploying kubeflow. Retrieved December 15, 2021, from https://www.kubeflow.org/docs/distributions/azure/deploy/install-kubeflow/

The Kubeflow Authors. (2021c, November 24). Introduction - an introduction to the goals and main concepts of kubeflow pipelines. Retrieved December 17, 2021, from https://www.kubeflow.org/docs/components/pipelines/introduction/

The Kubeflow Authors. (2021d, November 19). Kubeflow notebooks. Retrieved December 17, 2021, from https://www.kubeflow.org/docs/components/notebooks/

Treveil, M., Omont, N., Stenac, C., Lefevre, K., Phan, D., Zentici, J., ... Heidmann, L. (2020). *Introducing mlops*. O'Reilly Media, Inc.

Triantaphyllou, E. (2000). Multi-criteria decision making methods. In *Multi-criteria decision making methods: A comparative study* (pp. 5–21). DOI 10.1007/978-1-4757-3157-6_2. Boston, MA: Springer US.

Virmani, M. (2015). Understanding devops bridging the gap from continuous integration to continuous delivery. In *Fifth international conference on the innovative computing technology (intech 2015)* (pp. 78–82). DOI 10.1109/INTECH.2015.7173368.

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Ann Hong, S., Konwinski, A., ... Zumar, C. (2018). Accelerating the Machine Learning Lifecycle with MLflow. Databricks Inc.

Zhong, Z., Xu, M., Rodriguez, M. A., Xu, C., & Buyya, R. (2021). Machine learning-based orchestration of containers: A taxonomy and future directions. arXiv: 2106.12739 [cs.LG]

Annex I

Kubeflow deployment

I.1 Kfctl configuration used for Kubeflow deployment, with correction.

```
apiVersion: kfdef.apps.kubeflow.org/v1
kind: KfDef
metadata:
  creationTimestamp: null
  namespace: kubeflow
spec:
  applications:
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: namespaces/base
    name: namespaces
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: application/v3
    name: application
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: stacks/azure/application/istio-1-3-1-stack
    name: istio-stack
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: stacks/kubernetes/application/cluster-local-gateway-1-3-1
    name: cluster-local-gateway
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: stacks/azure/application/istio
    name: istio
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: stacks/azure/application/cert-manager-crds
    name: cert-manager-crds
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: stacks/azure/application/cert-manager-kube-system-resources
    name: cert-manager-kube-system-resources
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: stacks/azure/application/cert-manager
    name: cert-manager
  - kustomizeConfig:
```

```
      repoRef:
- kustomizeConfig:
      repoRef:
        name: manifests
        path: knative/installs/generic
    name: knative
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: kfserving/installs/generic
    name: kfserving
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: metacontroller/base
    name: metacontroller
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: stacks/azure/application/oidc-authservice
    name: oidc-authservice
  - kustomizeConfig:
      repoRef:
        name: manifests
        path: stacks/azure
    name: kubeflow-apps
  repos:
  - name: manifests
    uri:
    ↪  https://github.com/kubeflow/manifests/archive/ebeb4c285a0cf49750c4f07015b58cc27ca4c3a1.tar.gz
  version: v1.2-branch
# These lines were appended, missing on the official documentation, found the solution on a
↪  github issue.
status:
  reposCache:
    - localPath: '".cache/manifests/manifests-ebeb4c285a0cf49750c4f07015b58cc27ca4c3a1/"'
      name: manifests
```

I.2 Images from Kubeflow deployment

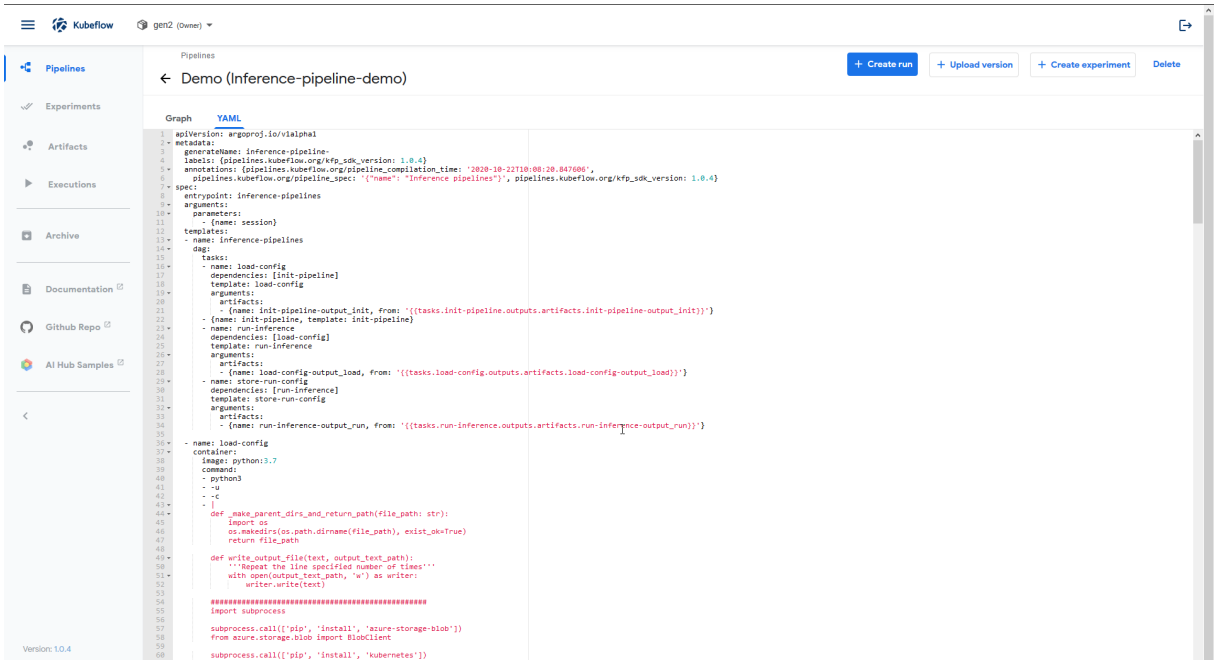


Figure I.1: Dashboard view of demo pipeline's YAML configuration.

Annex II

Exported knowledge base sample


```
{
  "Option": {
    "o_2506746548": {
      "uid": "o_2506746548",
      "name": "Kubeflow",
      "description": "End-to-end framework that fulfills the entire machine learning workflow and model lifecycle on a Kubernetes cluster.",
      "notes": "- deployment on Azure is incompatible with `Calico`networking policy\n- easy to orchestrate (DAG based) and re-use components and pipelines to quickly cobble together new workflows, without having to rebuild each time.",
      "createdate": "2021-08-11 11:22:27.940505",
      "lastupdate": "2021-10-20 14:52:58.878752",
      "categories": [
        "t_52105614",
        "t_2444859668",
        "t_468989116",
        "t_2244916060"
      ],
      "criteria": {
        "c_52105614": [
          4,
          5
        ],
        "c_2244916060": [
          1,
          1
        ],
        "c_468989116": [
          1,
          4
        ],
        "c_2506746548": [
          1,
          3
        ],
        "c_2444859668": [
          0,
          1
        ],
        "c_3142990153": [
          2,
          2
        ],
        "c_4129612440": [
          5,
          6
        ],
```

```
        "c_2370669921": [
          7,
          7
        ],
        "c_470901851": [
          3,
          3
        ]
      },
      "filters": {
        "f_2506746548": [
          "Free"
        ],
        "f_52105614": [
          "Once"
        ],
        "f_2444859668": [
          "No"
        ],
        "f_2244916060": [
          "No"
        ],
        "f_470901851": [
          "Apache 2.0"
        ],
        "f_4129612440": [
          "Yes"
        ],
        "f_2414592160": [
          "Yes"
        ],
        "f_3142990153": [
          "S3"
        ],
        "f_2684722482": [
          "LDAP",
          "Open ID",
          "REMOTE_USER",
          "OAUTH"
        ],
        "f_2472200801": [
          "Role Based Access Control (RBAC)"
        ],
        "f_2370669921": [
          "Kubernetes"
        ],
        "f_468989116": [
          "Third-party"
        ]
      }
    },
    "o_468989116": {
      "uid": "o_468989116",
      "name": "Apache Airflow",
      "description": "Generic workflow orchestrator",
      "notes": "- free and open-source with Apache license\n- 'Operators' are a great feature
      ↪  [documentation](https://airflow.apache.org/docs/apache-airflow/stable/concepts/operators.html)",
      "createdate": "2021-08-18 09:36:30.374236",
      "lastupdate": "2021-10-20 14:54:54.161618",
      "categories": [
        "t_2244916060"
      ],
      "criteria": {
        "c_52105614": [
          4,
          5
        ],
        "c_2244916060": [
          5,
          6
        ],
        "c_468989116": [
          6,
          7
```

```
        ],
        "c_2506746548": [
          3,
          3
        ],
        "c_2444859668": [
          0,
          0
        ],
        "c_3142990153": [
          4,
          5
        ],
        "c_4129612440": [
          6,
          7
        ],
        "c_2370669921": [
          0,
          0
        ],
        "c_470901851": [
          2,
          3
        ]
      },
      "filters": {
        "f_2506746548": [
          "Free"
        ],
        "f_52105614": [
          "Once"
        ],
        "f_468989116": [
          "Third-party"
        ],
        "f_2444859668": [
          "Yes"
        ],
        "f_2244916060": [
          "No"
        ],
        "f_470901851": [
          "Apache 2.0"
        ],
        "f_4129612440": [
          "Yes"
        ],
        "f_2414592160": [
          "Yes"
        ],
        "f_3142990153": [
          "DBFS",
          "S3"
        ],
        "f_2370669921": [
          "Kubernetes",
          "Native",
          "Docker Compose"
        ],
        "f_2472200801": [
          "Role Based Access Control (RBAC)",
          " Resource Based Access Control(Permissions)"
        ],
        "f_2684722482": [
          "Database",
          "Open ID",
          "LDAP",
          "REMOTE_USER",
          "OAUTH"
        ]
      }
    },
    "o_4129612440": {
```

```
      "uid": "o_4129612440",
      "name": "MLflow",
      "description": "Open source platform to manage the ML lifecycle, including
      ↪  experimentation, reproducibility, deployment and model registry.",
      "notes": "Great integration with Databricks. Some features like authentication are only
      ↪  available with [managed
      ↪  service](https://databricks.com/product/managed-mlflow).\n\nhttps://mlflow.org/docs/latest/index.ht
      "createdate": "2021-10-18 16:03:01.530348",
      "lastupdate": "2021-10-20 14:55:32.092895",
      "categories": [
        "t_2444859668",
        "t_2244916060",
        "t_468989116"
      ],
      "criteria": {
        "c_52105614": [
          3,
          3
        ],
        "c_2506746548": [
          2,
          3
        ],
        "c_3142990153": [
          3,
          4
        ],
        "c_2444859668": [
          0,
          0
        ],
        "c_2244916060": [
          5,
          6
        ],
        "c_468989116": [
          6,
          6
        ],
        "c_4129612440": [
          6,
          7
        ],
        "c_2370669921": [
          4,
          4
        ],
        "c_470901851": [
          3,
          4
        ]
      },
      "filters": {
        "f_2506746548": [
          "Free",
          "Subscription",
          "Free trial"
        ],
        "f_470901851": [
          "Apache 2.0"
        ],
        "f_4129612440": [
          "Yes"
        ],
        "f_2414592160": [
          "Yes"
        ],
        "f_3142990153": [
          "DBFS",
          "S3"
        ],
        "f_2244916060": [
          "No"
        ],
```

```
        "f_2444859668": [
          "Yes"
        ],
        "f_468989116": [
          "First-party",
          "Third-party"
        ],
        "f_2370669921": [
          "Kubernetes",
          "Docker Compose",
          "Native",
          "Databricks"
        ],
        "f_52105614": [
          "Never"
        ],
        "f_2472200801": [
          "Role Based Access Control (RBAC)"
        ]
      }
    }
  },
  "Factor": {
    "f_2244916060": {
      "uid": "f_2244916060",
      "name": "Internally Developed",
      "description": "Tool is/was developed within Bosch",
      "notes": "",
      "createdate": "2021-08-10 08:57:44.559243",
      "lastupdate": "2021-10-18 23:24:54.017777",
      "values": [
        "Yes",
        "No"
      ],
      "groups": [
        "g_2244916060"
      ]
    },
    "f_2506746548": {
      "uid": "f_2506746548",
      "name": "License Cost Structure",
      "description": "Type of cost for the software license",
      "notes": "",
      "createdate": "2021-08-11 14:43:39.504885",
      "lastupdate": "2021-10-18 17:49:49.605889",
      "values": [
        "Free",
        "Free trial",
        "One-time license",
        "Subscription"
      ],
      "groups": [
        "g_2244916060"
      ]
    },
    "f_468989116": {
      "uid": "f_468989116",
      "name": "Team Support (incl. Consultants)",
      "description": "Available sources of help with using the tool",
      "notes": "",
      "createdate": "2021-08-11 16:20:24.500056",
      "lastupdate": "2021-10-20 14:04:27.737316",
      "values": [
        "First-party",
        "Third-party",
        "Internal"
      ],
      "groups": [
        "g_2506746548"
      ]
    },
    "f_2444859668": {
      "uid": "f_2444859668",
      "name": "Offered as managed service",
```

```
      "description": "Possibility to obtain the tool as a managed service instead of having
      ↪  the team deal with its maintenence",
      "notes": "",
      "createdate": "2021-08-18 09:37:58.193159",
      "lastupdate": "2021-10-18 16:10:07.578036",
      "values": [
        "Yes",
        "No"
      ],
      "groups": [
        "g_2244916060",
        "g_2506746548",
        "g_468989116",
        "g_4129612440"
      ]
    },
    "f_52105614": {
      "uid": "f_52105614",
      "name": "Previously used by project teams",
      "description": "Wether or not there is previous experience within the project teams",
      "notes": "",
      "createdate": "2021-08-18 09:39:36.809722",
      "lastupdate": "2021-08-18 09:39:36.809722",
      "values": [
        "Never",
        "Once",
        "A few times"
      ],
      "groups": [
        "g_2506746548",
        "g_52105614"
      ]
    },
    "f_3142990153": {
      "uid": "f_3142990153",
      "name": "IO with distributed storage",
      "description": "Take input from, and write output to, distributed storage systems.",
      "notes": "",
      "createdate": "2021-10-18 16:04:35.049917",
      "lastupdate": "2021-10-18 23:16:45.694368",
      "values": [
        "DBFS",
        "S3"
      ],
      "groups": [
        "g_4129612440",
        "g_3142990153"
      ]
    },
    "f_4129612440": {
      "uid": "f_4129612440",
      "name": "Parallel job execution",
      "description": "Run mutiple jobs concurrently",
      "notes": "",
      "createdate": "2021-10-18 16:12:29.609906",
      "lastupdate": "2021-10-18 18:10:08.078912",
      "values": [
        "Yes",
        "No"
      ],
      "groups": [
        "g_4129612440"
      ]
    },
    "f_2370669921": {
      "uid": "f_2370669921",
      "name": "Execution environments",
      "description": "",
      "notes": "",
      "createdate": "2021-10-18 17:46:43.037561",
      "lastupdate": "2021-10-19 15:45:01.659657",
      "values": [
        "Native",
        "Docker Compose",
```

```
        "Kubernetes",
        "Hadoop",
        "Databricks"
      ],
      "groups": [
        "g_2444859668",
        "g_468989116",
        "g_4129612440"
      ]
    },
    "f_470901851": {
      "uid": "f_470901851",
      "name": "OSS License type",
      "description": "Open Source Software Licenses",
      "notes": "- Permissive: Apache 2.0, BSD 3-Clause, MIT\n\n- Copyleft:\n  - Strong: GPL v2, GLP v3, AGPL\n  - Weak: LGPL, Mozilla Public License 2.0\n\n\n",
      "createdate": "2021-10-18 17:59:25.711274",
      "lastupdate": "2021-10-18 18:09:28.533397",
      "values": [
        "Apache 2.0",
        "MIT",
        "GPL v2",
        "GPL v3",
        "BSD 3-Clause",
        "AGPL",
        "LGPL",
        "Mozilla Public License 2.0"
      ],
      "groups": [
        "g_2244916060"
      ]
    },
    "f_2414592160": {
      "uid": "f_2414592160",
      "name": "REST API",
      "description": "Use the application in a programatic way through a REST API.",
      "notes": "",
      "createdate": "2021-10-18 18:19:41.399399",
      "lastupdate": "2021-10-18 18:20:31.519990",
      "values": [
        "Yes",
        "No"
      ],
      "groups": [
        "g_52105614",
        "g_3142990153"
      ]
    },
    "f_2472200801": {
      "uid": "f_2472200801",
      "name": "User access control",
      "description": "",
      "notes": "",
      "createdate": "2021-10-18 23:29:09.737840",
      "lastupdate": "2021-10-18 23:32:45.883891",
      "values": [
        "Role Based Access Control (RBAC)",
        " Resource Based Access Control(Permissions)"
      ],
      "groups": [
        "g_2370669921"
      ]
    },
    "f_2684722482": {
      "uid": "f_2684722482",
      "name": "Supported Authentication Types",
      "description": "Supported Authentication Types",
      "notes": "Bosch authentication is integrated with Microsoft Active Directory (Azure), an LDAP server.",
      "createdate": "2021-10-18 23:32:28.774499",
      "lastupdate": "2021-10-20 13:53:25.520648",
      "values": [
        "Database",
        "Open ID",
```

```
        "LDAP",
        "REMOTE_USER",
        "OAUTH"
      ],
      "groups": [
        "g_2370669921"
      ]
    },
    "c_2244916060": {
      "uid": "c_2244916060",
      "name": "Flexibility and Customization",
      "description": "Adaptation to the specific needs of the use case",
      "notes": "",
      "createdate": "2021-08-11 14:39:45.371149",
      "lastupdate": "2021-10-19 16:09:28.239088",
      "values": [
        "Hard",
        "Flexible"
      ],
      "groups": [
        "g_52105614"
      ],
      "resolution": 8
    },
    "c_2506746548": {
      "uid": "c_2506746548",
      "name": "Reliability",
      "description": "How stable and dependable is the tool",
      "notes": "Consider different factors, namely:\n- uptime and bugs\n- how easily does it
      ↪  get patched, automatically or with intervention",
      "createdate": "2021-08-11 15:14:00.550003",
      "lastupdate": "2021-10-19 16:09:34.144685",
      "values": [
        "Unreliable",
        "Reliable"
      ],
      "groups": [
        "g_468989116"
      ],
      "resolution": 4
    },
    "c_468989116": {
      "uid": "c_468989116",
      "name": "Official Documentation",
      "description": "Quality of the first-party docs",
      "notes": "",
      "createdate": "2021-08-11 16:15:16.087546",
      "lastupdate": "2021-08-11 16:15:16.087546",
      "values": [
        "Poor",
        "Good"
      ],
      "groups": [
        "g_2506746548"
      ],
      "resolution": 8
    },
    "c_2444859668": {
      "uid": "c_2444859668",
      "name": "Internal Documentation",
      "description": "Documentation within EnterpriseSearch and Docupedia",
      "notes": "",
      "createdate": "2021-08-11 16:16:48.106476",
      "lastupdate": "2021-08-11 16:16:48.106476",
      "values": [
        "Poor",
        "Good"
      ],
      "groups": [
        "g_2506746548"
      ],
      "resolution": 6
    },
    "c_52105614": {
```

```
      "uid": "c_52105614",
      "name": "Community Support",
      "description": "How much support can be found in online forums",
      "notes": "",
      "createdate": "2021-08-11 17:07:03.553758",
      "lastupdate": "2021-08-11 17:07:03.553758",
      "values": [
        "Low",
        "High"
      ],
      "groups": [
        "g_2506746548"
      ],
      "resolution": 6
    },
    "c_3142990153": {
      "uid": "c_3142990153",
      "name": "Ease of programming",
      "description": "Degree of added difficulty when coding for the tool. Does it require
      ↪  lots of refactoring?",
      "notes": "Describes code development overhead to use the tool in question, how
      ↪  accessible and intuitive it is - just some API calls or is it more complex and with
      ↪  lots of small details?",
      "createdate": "2021-10-18 18:27:30.281460",
      "lastupdate": "2021-10-20 13:51:23.012160",
      "values": [
        "Difficult",
        "Easy"
      ],
      "groups": [
        "g_52105614"
      ],
      "resolution": 6
    },
    "c_4129612440": {
      "uid": "c_4129612440",
      "name": "State of Development",
      "description": "How complete is the tool according to its scope.",
      "notes": "",
      "createdate": "2021-10-19 16:13:52.990168",
      "lastupdate": "2021-10-19 16:15:33.601358",
      "values": [
        "Incomplete",
        "Complete"
      ],
      "groups": [
        "g_2506746548",
        "g_52105614"
      ],
      "resolution": 8
    },
    "c_2370669921": {
      "uid": "c_2370669921",
      "name": "Versatility",
      "description": "How multipurpose is the software, and to what degree can it cover the ML
      ↪  lifecycle.",
      "notes": "",
      "createdate": "2021-10-19 16:19:09.891676",
      "lastupdate": "2021-10-20 13:50:02.876766",
      "values": [
        "Narrow",
        "Versatile"
      ],
      "groups": [
        "g_52105614"
      ],
      "resolution": 8
    },
    "c_470901851": {
      "uid": "c_470901851",
      "name": "Update Frequency",
      "description": "How regularly does the tool get updated (release cycle)",
```

```
      "notes": "| More than a year |  Year | Semester |  Trimester | Month | Less than a month|\n|--------------|\n|Incredibly infrequent | Pretty infrequent | Somewhat infrequent | Somewhat frequent | Very Frequent | Incredibly frequent|",
      "createdate": "2021-10-20 14:45:47.493845",
      "lastupdate": "2021-10-20 14:55:36.021324",
      "values": [
        "Infrequent",
        "Frequent"
      ],
      "groups": [
        "g_2506746548",
        "g_2244916060"
      ],
      "resolution": 6
    }
  },
  "Collection": {
    "g_2244916060": {
      "uid": "g_2244916060",
      "name": "Administrative and Legal",
      "description": "Cost, licencing, compliance, etc.",
      "notes": "",
      "createdate": "2021-08-09 22:54:07.692393",
      "lastupdate": "2021-10-20 14:45:47.493874",
      "elements": [
        "f_2506746548",
        "f_2244916060",
        "f_2444859668",
        "f_470901851",
        "c_470901851"
      ]
    },
    "g_2506746548": {
      "uid": "g_2506746548",
      "name": "Support and Documentation",
      "description": "Help with the software during the project lifecycle.",
      "notes": "Prioritize sources and resources _within_ Bosch, namely `EnterpriseSearch` and `Docupedia`.",
      "createdate": "2021-08-09 22:58:27.340108",
      "lastupdate": "2021-10-20 14:45:47.493868",
      "elements": [
        "c_468989116",
        "c_2444859668",
        "f_468989116",
        "c_52105614",
        "f_2444859668",
        "f_52105614",
        "c_4129612440",
        "c_470901851"
      ]
    },
    "g_468989116": {
      "uid": "g_468989116",
      "name": "Deployment and Maintenence",
      "description": "Move the software system to production and maintain it during its lifecycle",
      "notes": "",
      "createdate": "2021-08-09 23:22:05.416900",
      "lastupdate": "2021-10-18 17:46:43.037627",
      "elements": [
        "c_2506746548",
        "f_2444859668",
        "f_2370669921"
      ]
    },
    "g_2444859668": {
      "uid": "g_2444859668",
      "name": "Cloud",
      "description": "",
      "notes": "",
      "createdate": "2021-08-09 23:22:42.308331",
      "lastupdate": "2021-10-18 17:46:43.037609",
      "elements": [
        "f_2370669921"
```

```
      ]
    },
    "g_52105614": {
      "uid": "g_52105614",
      "name": "Development",
      "description": "Tool attributes that affect the project development",
      "notes": "",
      "createdate": "2021-08-09 23:25:37.684513",
      "lastupdate": "2021-10-19 16:19:09.891692",
      "elements": [
        "c_2244916060",
        "f_52105614",
        "f_2414592160",
        "c_3142990153",
        "c_4129612440",
        "c_2370669921"
      ]
    },
    "g_3142990153": {
      "uid": "g_3142990153",
      "name": "Features",
      "description": "Integration with other solutions, plus some extra *nice to have*s",
      "notes": "",
      "createdate": "2021-08-09 23:32:02.983417",
      "lastupdate": "2021-10-18 23:16:45.694364",
      "elements": [
        "f_2414592160",
        "f_3142990153"
      ]
    },
    "g_4129612440": {
      "uid": "g_4129612440",
      "name": "Performance and Scalability",
      "description": "Handling of increasing volumes of data.",
      "notes": "",
      "createdate": "2021-10-18 16:05:21.172396",
      "lastupdate": "2021-10-18 23:16:45.694360",
      "elements": [
        "f_2444859668",
        "f_4129612440",
        "f_2370669921",
        "f_3142990153"
      ]
    },
    "g_2370669921": {
      "uid": "g_2370669921",
      "name": "Security",
      "description": "",
      "notes": "",
      "createdate": "2021-10-18 16:06:08.366821",
      "lastupdate": "2021-10-18 23:32:28.774533",
      "elements": [
        "f_2472200801",
        "f_2684722482"
      ]
    },
    "t_2244916060": {
      "uid": "t_2244916060",
      "name": "Orchestrator",
      "description": "Coordinate the components of a workflow",
      "notes": "",
      "createdate": "2021-08-10 16:35:38.036014",
      "lastupdate": "2021-10-19 15:36:24.608079",
      "elements": [
        "o_2506746548",
        "o_468989116",
        "o_4129612440"
      ]
    },
    "t_2506746548": {
      "uid": "t_2506746548",
      "name": "Data Store",
      "description": "Keep processed data for future use",
      "notes": "",
```

```
      "createdate": "2021-08-10 16:35:55.067677",
      "lastupdate": "2021-10-19 15:36:04.989029",
      "elements": []
    },
    "t_468989116": {
      "uid": "t_468989116",
      "name": "Serving",
      "description": "Expose trained models for end-user",
      "notes": "",
      "createdate": "2021-08-10 16:36:11.312708",
      "lastupdate": "2021-08-10 16:36:11.312708",
      "elements": [
        "o_2506746548",
        "o_4129612440"
      ]
    },
    "t_2444859668": {
      "uid": "t_2444859668",
      "name": "Model Store and Versioning",
      "description": "Maintain catalog of developed models",
      "notes": "",
      "createdate": "2021-08-10 16:36:29.440321",
      "lastupdate": "2021-08-10 16:36:29.440321",
      "elements": [
        "o_2506746548",
        "o_4129612440"
      ]
    },
    "t_52105614": {
      "uid": "t_52105614",
      "name": "End-to-end Framework",
      "description": "Framework capable of fulfilling entire workflows and model lifecycles",
      "notes": "",
      "createdate": "2021-08-10 16:37:00.777762",
      "lastupdate": "2021-08-10 16:37:00.777762",
      "elements": [
        "o_2506746548"
      ]
    },
    "t_3142990153": {
      "uid": "t_3142990153",
      "name": "Inference",
      "description": "Take imput data and trained model to generate the prediction",
      "notes": "",
      "createdate": "2021-08-11 12:27:15.714723",
      "lastupdate": "2021-08-11 12:27:15.714723",
      "elements": []
    },
    "t_4129612440": {
      "uid": "t_4129612440",
      "name": "Role ABC",
      "description": "",
      "notes": "",
      "createdate": "2021-08-11 15:20:17.104446",
      "lastupdate": "2021-08-18 09:35:34.422814",
      "elements": []
    }
  }
}
```

Annex III

MetaTool user interface

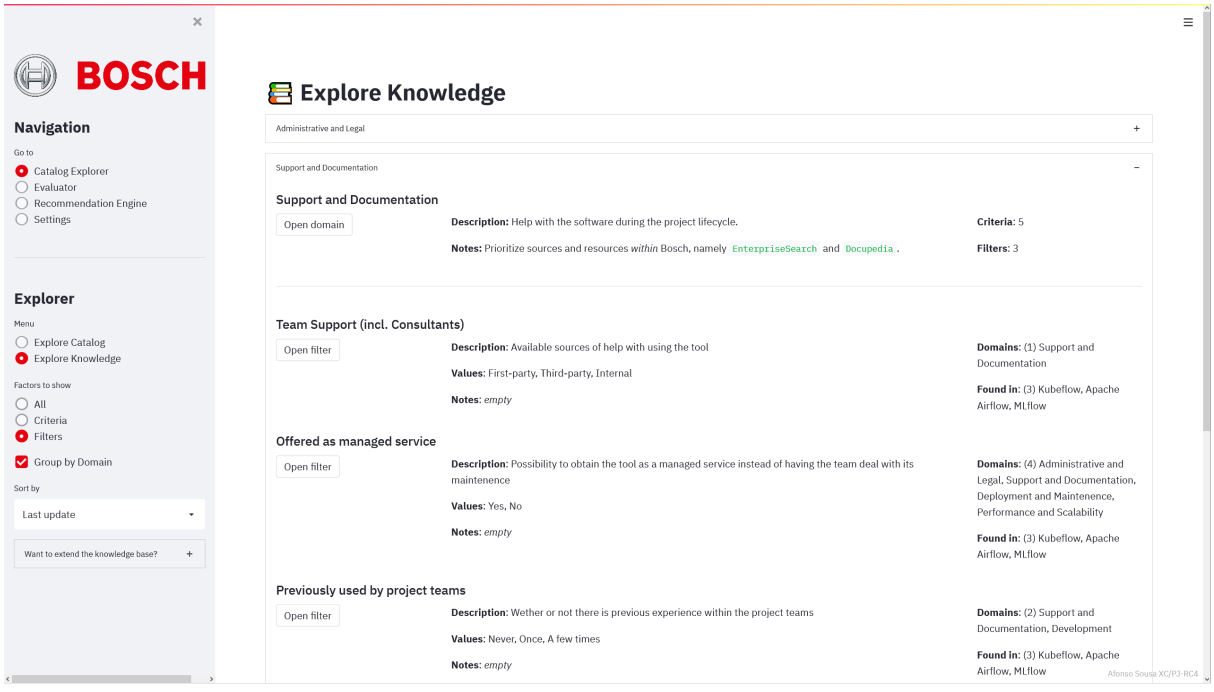


Figure III.1: MetaTool UI: Explorer - Explore Knowledge page, grouped by domain, opened.

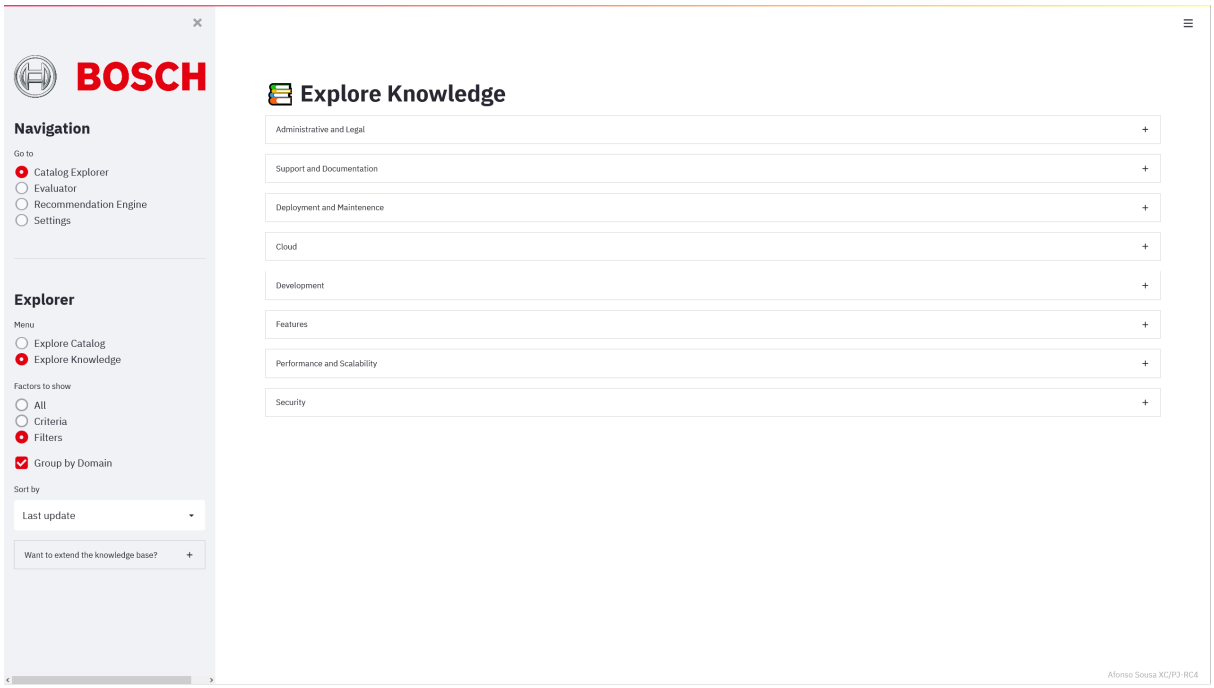


Figure III.2: MetaTool UI: Explorer - Explore Knowledge page, grouped by domain, collapsed.

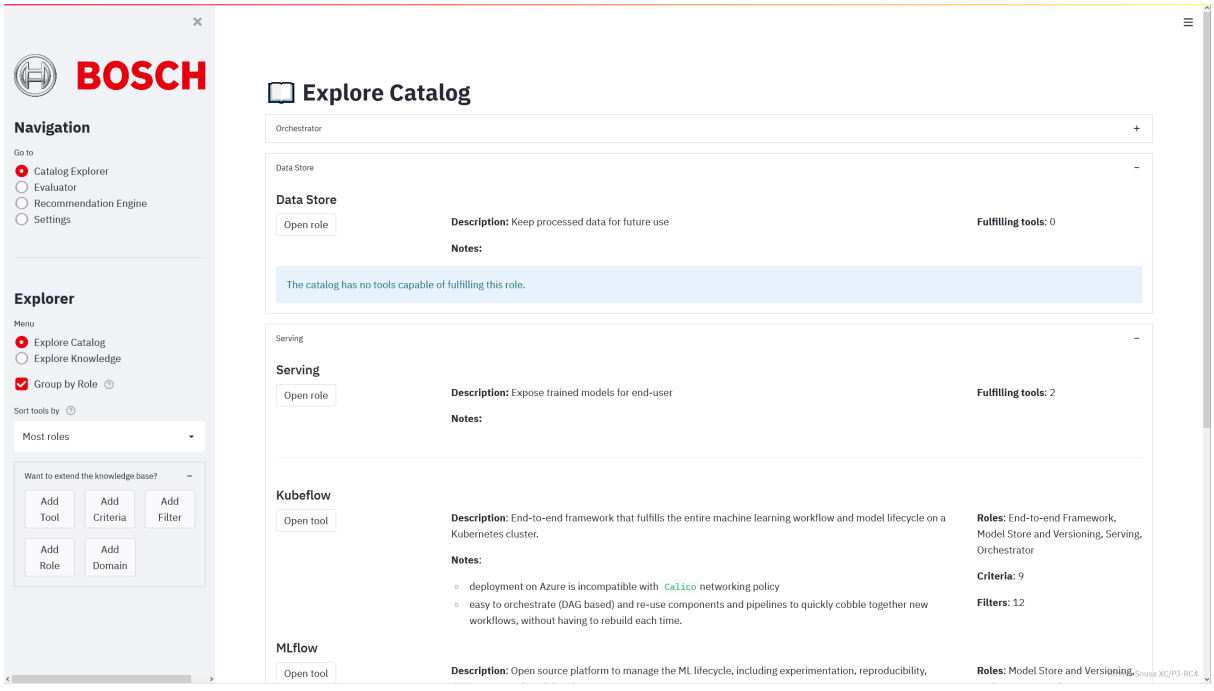


Figure III.3: MetaTool UI: Explorer - Explore Catalog page, grouped by role, opened.

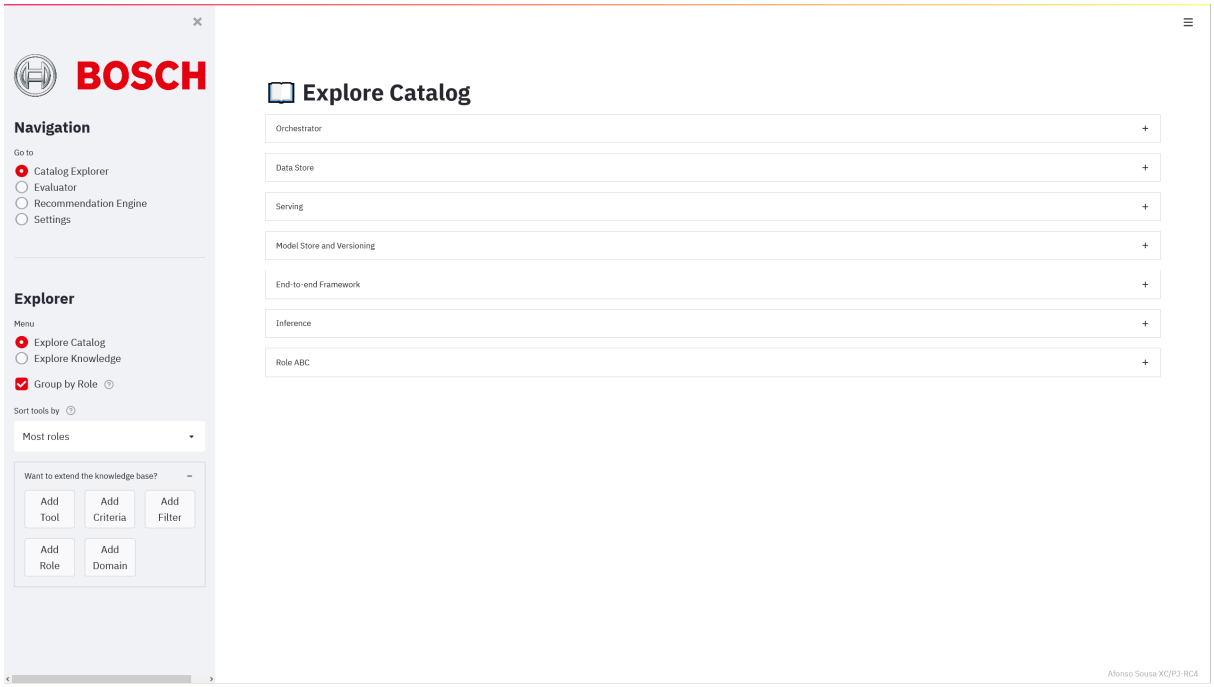


Figure III.4: MetaTool UI: Explorer - Explore Catalog page, grouped by role, collapsed.

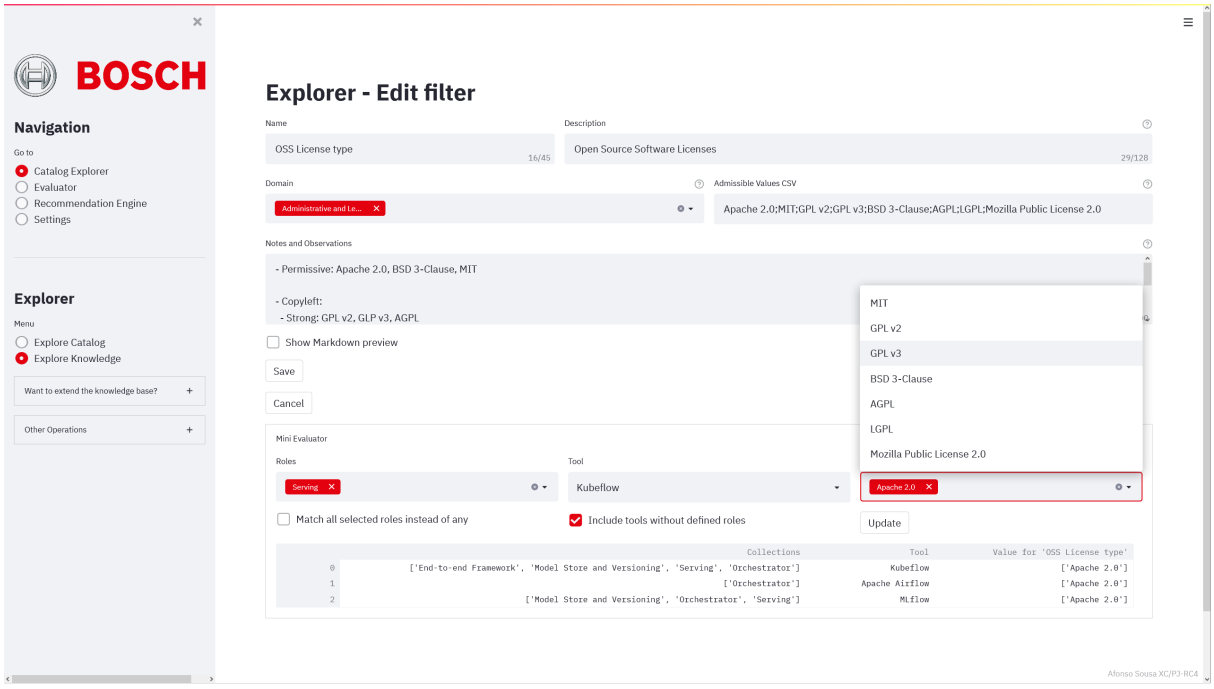


Figure III.5: MetaTool UI: Explorer - Edit filter page.

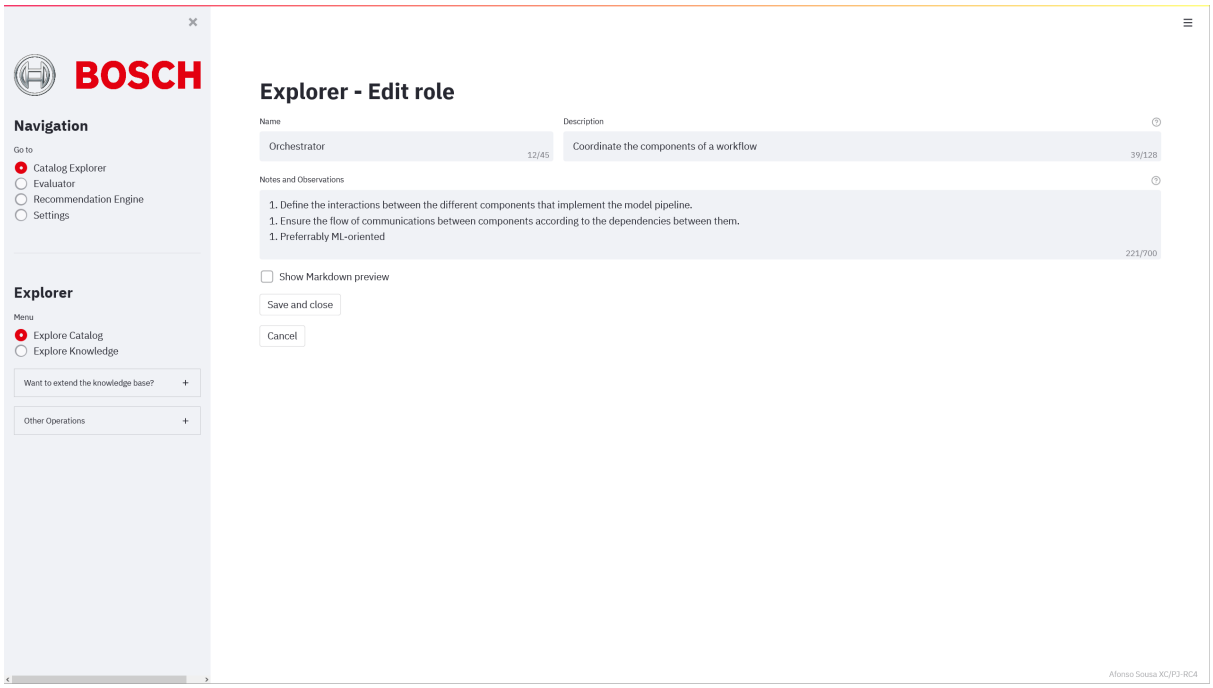


Figure III.6: MetaTool UI: Explorer - Edit role page.

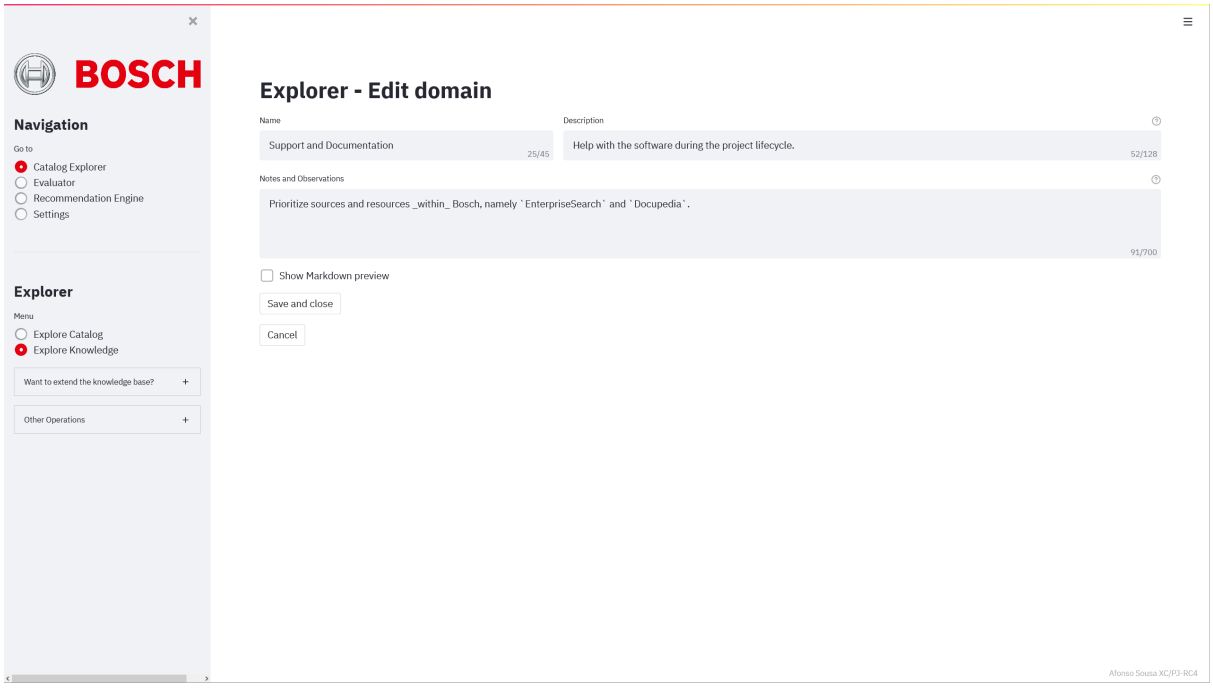


Figure III.7: MetaTool UI: Explorer - Edit domain page.

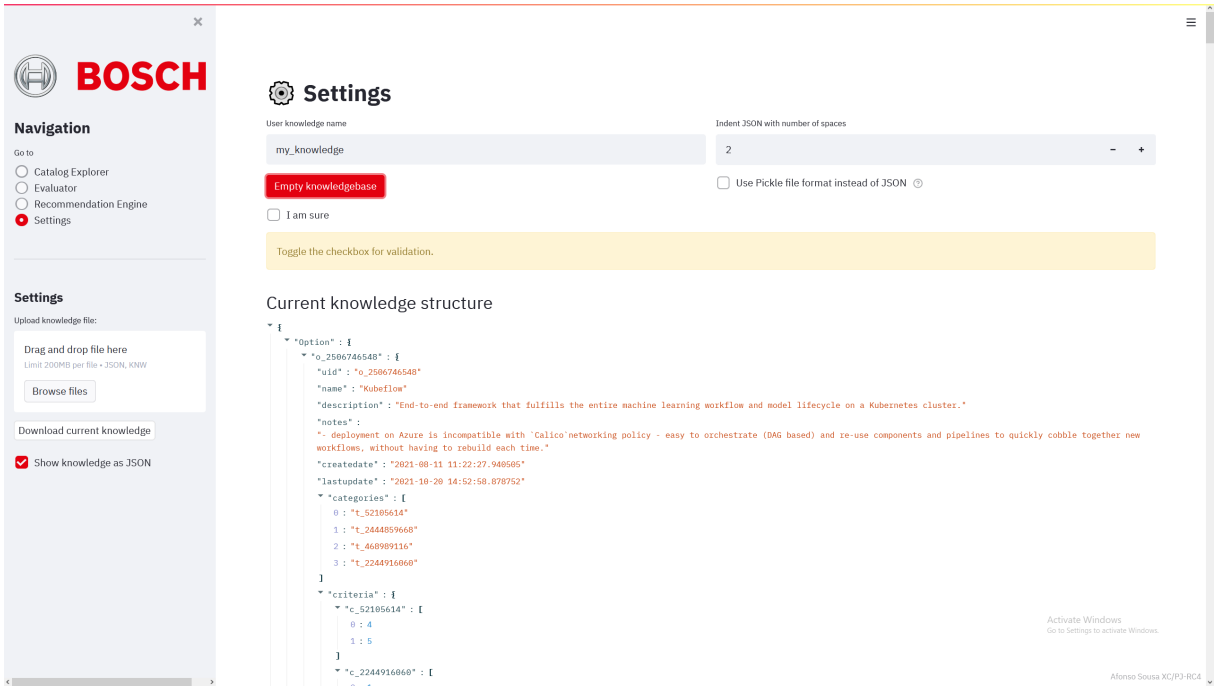


Figure III.8: MetaTool UI: Settings page, attempt to empty knowledge base, with JSON structure being shown.