UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Kristian Tamm

SynthGuard: Scalable and Privacy-Preserving Synthetic Data Generation Workflow Framework

Bachelor's Thesis (9 ECTS)

Supervisor:
Eduardo Ribas Brito, MSc

Tartu 2025

SynthGuard: Scalable and Privacy-Preserving Synthetic Data Generation Workflow Framework

Abstract:

Synthetic data enables privacy-preserving data sharing, especially important in sensitive domains under strict regulations such as the GDPR. While demand for synthetic data generation (SDG) is growing, existing tools often lack scalability, integration, and usability. This thesis presents SynthGuard, a cloud-native workflow framework for scalable and secure synthetic data generation. Built using containerized microservices and Kubernetes orchestration, SynthGuard automates the deployment and management of complex data synthesis pipelines. Its modular design allows flexible integration of SDG tools, while abstracting infrastructure complexity. The framework was evaluated through use cases in EU-funded projects, showcasing reproducibility, scalability, and privacy compliance. SynthGuard simplifies the development and operation of synthetic data workflows, offering a practical foundation for responsible data ecosystems in both research and industry.

Keywords: synthetic data generation, synthetic data, data privacy, workflow automation, pipeline orchestration

CERCS: P170 Computer science, numerical analysis, systems, control

SynthGuard: Mastabeeritav ja privaatsust säilitav tehisandmte genereerimise töövooraamistik

Lühikokkuvõte:

Tehisandmed võimaldavad privaatsust säilitavat andmejagamist, mis on eriti oluline tundlikes valdkondades, kus kehtivad ranged regulatsioonid, nagu GDPR. Kuigi nõudlus tehisandmete genereerimise (SDG) järele kasvab, puuduvad olemasolevatel tööriistadel sageli mastaabitavus, integreeritavus ja kasutusmugavus. Käesolev bakalaureusetöö esitleb SynthGuardi: pilvepõhist töövookeskkonda mastaabitavaks ja turvaliseks tehisandmete genereerimiseks. SynthGuard kasutab konteinerdatud mikroteenuseid ja Kubernetese orkestreerimist, et automatiseerida keerukate tehisandmete töövoogude juurutamist ja haldamist. Selle modulaarne disain võimaldab SDG tööriistade paindlikku integreerimist, samal ajal abstraheerides infrastruktuuri keerukust. Raamistikku hinnati EL-i rahastatud projektide kasutusmallide kaudu, mis näitasid reprodutseeritavuse, mastaabitavuse ja privaatsuse paranemist. SynthGuard lihtsustab tehisandmete töövoogude arendamist ja toimimist, pakkudes väärtust tundlikele küberökosüsteemidele nii teaduses kui ka tööstuses.

Võtmesõnad: SynthGuard, tehisandmed, töövooraamisitk, privaatsuskaitse tehnoloogia

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

1. Introduction

In an era where data fuels innovation in fields from personalized medicine to financial modeling, access to high-quality datasets is essential. Yet in domains like healthcare, finance, and law enforcement, strict privacy regulations severely limit the availability of real-world data. This creates a critical tension: how can we encourage lawful research and development without compromising individual privacy?

One promising solution is *synthetic data generation* (SDG) - the creation of artificial data sets that mimic the statistical properties of real data but contain no identifiable personal information [1]. Synthetic data enables safe experimentation, testing, and analysis under privacy constraints, making it an increasingly attractive tool for privacy-preserving data science [2].

However, existing SDG tools face significant limitations. Many are difficult to scale, lack built-in privacy safeguards, and require advanced programming skills, posing an obstacle for domain experts who lack technical expertise, but still need high-quality synthetic data.

This thesis presents **SynthGuard**, a modular, cloud-native framework designed to make scalable, privacy-aware synthetic data generation accessible and practical. SynthGuard unifies components of the SDG lifecycle, namely generation, evaluation, and deployment, into an automated and reproducible workflow. Built on robust technologies including Kubernetes [3], Kubeflow Pipelines [4], and Nix [5], it provides a flexible foundation for developing and executing data synthesis pipelines at scale.

SynthGuard also includes a user-friendly Python library and automation tooling that significantly lower the entry barrier for non-experts. By enabling users to generate compliant, high-utility synthetic datasets with minimal friction, SynthGuard addresses the core challenges of existing SDG systems and contributes a practical, extensible platform for privacy-preserving data innovation.

This thesis is structured as follows:

1. **Background and Literature Review:** Reviews existing SDG methods, privacy regulations, and related projects, highlighting gaps in current solutions.
2. **Problem Statement and Requirements:** Defines the challenges SynthGuard addresses, such as scalability, privacy, and accessibility.

3. **Methodology:** Describes the system architecture, research approach, and workflow orchestration methods.
4. **Implementation:** Details the technical development of SynthGuard, including its modular components and workflow execution environment.
5. **Evaluation and Results:** Presents performance results, scalability testing, and feedback from real-world use cases.
6. **Discussion and Future Work:** Explores limitations, proposes enhancements, and outlines future directions for SynthGuard.
7. **Conclusion:** Summarizes the contributions of this thesis and its impact on synthetic data generation.

2. Background and Literature Review

2.1 Context and Motivation

The growing reliance on data-driven services across industries has increased the demand for accessible and high-quality data [6]. However, the need for data accessibility must be balanced with privacy concerns and compliance with global data protection regulations, which make it challenging for organizations to share and leverage data for innovation [7]. These regulations, while essential for safeguarding individual privacy, create obstacles for machine learning (ML) model training, system testing, and secure data sharing [8].

To address these challenges, synthetic data generation (SDG) has emerged as a privacy-preserving solution that allows organizations to utilize artificial datasets without exposing real personal information [1, 2, 9]. SDG maintains the statistical integrity of the original data while minimizing re-identification risks [10], making it particularly valuable in sectors such as healthcare, finance, and the public sector. Various types of synthetic data are designed to suit different application needs. (1) **realistic** data for training machine learning models [11], (2) **causal** data for policy simulations [12], (3) **artificial** data for synthetic prototyping without real data access [13], and (4) **hybrid** data that combines multiple methods to balance realism and flexibility [14].

In this paper, the terms (data) workflows and pipelines are used interchangeably. While SDG enables privacy-compliant data usage, its integration into workflow architectures remains an area requiring further exploration. Ensuring seamless adoption of synthetic data in real-world AI and analytics frameworks is critical for organizations aiming to harness its full potential. This section reviews developments in data protection laws, SDG methodologies, applications, and the need for effective integration within data ecosystems.

2.2 Impact of Global Data Protection Laws on Data Access

In recent years, several stringent data protection laws have been put in place worldwide to safeguard individuals privacy and data rights. The General Data Protection Regulation (GDPR) [15] implemented by the European Union in 2018 sets out strict guidelines on data use, processing, and sharing, emphasizing principles such as data minimization, purpose limitation, and the right to be forgotten. Similarly, the California Consumer Privacy Act (CCPA) [16] and the Health Insurance Portability and Accountability Act (HIPAA) [17] in the United States, along with the Personal Information Protection and Electronic Documents Act (PIPEDA) [18] in Canada, have all introduced robust frameworks for protecting personal data.

These laws have significantly impacted how organizations handle data access for innovation and development. For example:

- GDPR : This regulation imposes severe restrictions on the processing of personal data, making data sharing for AI model training and cross-border data transfers more complex [15].
- CCPA : Effective in California since 2020, the CCPA grants California residents certain rights regarding their personal data, including the right to opt-out of data sales and the right to delete [16].
- HIPAA : Enforced by the Department of Health and Human Services, HIPAA regulates how patient health information is handled, providing advanced protections for sensitive patient data [17].
- PIPEDA : In Canada, PIPEDA sets out the ground rules for how corporations and organizations manage personal information, helping to ensure that consumers provide the true consent [18].

Although these regulations improve data security and build user trust, they also create significant challenges for organizations seeking to innovate and develop new data applications.

2.3 Synthetic Data Generation Overview

To address the challenges posed by GDPR and other privacy regulations, organizations rely on Privacy-Enhancing Technologies (PETs) [19, 20] such as tokenization, homomorphic encryption, and Secure Multi-Party Computation (SMPC). Among these, SDG has emerged as a particularly effective solution, as it creates artificial datasets that preserve the statistical properties of real data while eliminating the risk of re-identification.

By generating privacy-preserving data, SDG allows organizations to share and analyze information without exposing sensitive details, ensuring compliance with global data protection laws. This makes it especially valuable for testing machine learning models, developing AI-driven applications, and enabling secure data sharing in regulated industries.

Unlike traditional anonymization methods, which remain vulnerable to re-identification attacks [21, 22], SDG does not rely on modifying real data. Instead, it produces entirely new data points that statistically resemble the original dataset while removing personally identifiable

information (PII). As a result, SDG is widely used in privacy-sensitive sectors such as healthcare and finance, where maintaining both data utility and privacy is crucial.

2.4 Types of Synthetic Data

There are four main types of synthetic data - *realistic synthetic data* (RSD), *causal synthetic data* (CSD), *artificial synthetic data* (ASD), and *hybrid synthetic data* (HSD).

- **Realistic Synthetic Data** (RSD) duplicates the statistical properties of the original data using advanced machine learning (ML) and deep learning (DL) models, such as Generative Adversarial Networks (GANs) [23].
- **Casual Synthetic Data** (CSD) preserves causal relationships between data points [24].
- **Artificial Synthetic Data** (ASD) relies on predefined rules or aggregated statistics, making it suitable when access to raw data is restricted [25].
- **Hybrid Synthetic Data** (HSD) combines these approaches, offering flexibility across all domains [14, 26].

This thesis mainly focuses on the design, implementation, and evaluation of RSD and ASD generation.

2.5 Use Cases and Applications

As mentioned earlier, Synthetic Data Generation (SDG) supports the use of data that comply with the privacy laws by removing personally identifiable information (PII). It is increasingly applied in regulated sectors such as healthcare, finance, and law enforcement. The following highlights known use cases across these domains.

2.5.1 Healthcare and Biomedical Research

Healthcare institutions must balance data-driven innovation with stringent privacy regulations like HIPAA. SDG enables:

- Medical research and AI training: Creating synthetic patient data for disease modeling and predictive analytics without exposing real patient records [27].
- Clinical trial simulation: Generating diverse datasets to test drug efficacy and model patient responses [28].

- Secure data sharing: Enabling cross-institutional research while ensuring compliance with data protection laws [27].

2.5.2 Financial Services and Banking

Financial institutions handle sensitive user data, making privacy protection a priority. SDG is used for:

- Fraud detection and risk assessment: Training AI models on synthetic transaction data to detect suspicious activities [29].
- Regulatory compliance and stress testing: Simulating financial market conditions while avoiding exposure of real customer data [30].
- Secure data monetization: Allowing financial firms to share insights with third parties without disclosing real customer information [31].

2.6 Influence of EU-Funded Data Projects

During my work at Cybernetica, the project was significantly influenced by two EU-funded initiatives: the LAGO and TEADAL projects.

2.6.1 LAGO Use Case

The LAGO (Lessen Data Access and Governance Obstacles) project aims to establish a trusted European Research Data Ecosystem (RDE) to address challenges in fighting crime and terrorism (FCT). By focusing on decentralization, data sovereignty, security, and trust, LAGO seeks to overcome barriers to data sharing in the FCT domain [32].

In sensitive domains like law enforcement, organizations (referred to here as LEAs – Law Enforcement Agencies) often cannot share real-world datasets due to confidentiality, GDPR, Law Enforcement Directive (LED) [33], or internal regulations. This restriction hampers research and the development of machine learning models.

Instead of transferring sensitive data to external cloud services (even trusted ones), the LEA can download and deploy the SDG pipeline within their own secure environment as demonstrated in Figure 8 that can be found in the Appendices. Those generated synthetic datasets can then be safely shared or advertised.

2.6.2 TEADAL Use Case

Conversely, the TEADAL (Trustworthy, Energy-Aware federated DAta Lakes along the computing continuum) project focuses on developing technologies to create stretched data lakes across the cloud-edge continuum and multi-cloud environments. TEADAL emphasizes building trustworthy, mediatorless federations of data lakes to facilitate effective data exchange among organizations while preserving privacy and confidentiality constraints [34]. TEADAL encompasses several pilot use cases across diverse sectors, including healthcare, finance, mobility, agriculture, industry, and energy. Among these, the energy pilot focuses on regional planning for environmental sustainability in Tuscany, Italy.

The energy pilot use case involves generating synthetic data to demonstrate the feasibility of a data pipeline, rather than fully replicating original datasets. The data includes public (e.g., temperature, air quality) and private (e.g., energy performance certificates) datasets from the Tuscany region. The structure of the pipeline can be found in Figure 9, in the Appendices section.

Synthetic records were generated based on predefined schemas, with no privacy concerns for private data. The data pipeline will run daily to generate partial records and randomized values within specified ranges. The goal is to create a uniform representation of territorial identifiers across datasets, enabling joint analysis of environmental and energy data within the TEADAL federated system.

3. Problem Statement and Objective

This section outlines the context and motivation for integrating SDG into modern data infrastructures, identifies challenges and technological gaps in current solutions, and presents the goals of SynthGuard, a prototype framework designed to bridge these shortcomings. Through this work, the aim is to facilitate a more accessible, scalable, and regulation-aware approach to SDG.

3.1 Problem Context and Motivation

Modern data processing and analytics heavily rely on data pipelines and workflow engines to handle large-scale operations efficiently. Frameworks such as Apache Airflow [35], AWS Glue [36], and Kubeflow [4] orchestrate complex data workflows by decomposing them into modular, reusable tasks. These technologies have proven highly effective in scaling big data analytics, machine learning (ML) training pipelines, and ETL (Extract, Transform, Load) processes.

The expansion of cloud computing platforms such as AWS [37], Google Cloud [38], and Azure [39] further enhances the capabilities of workflow automation. Cloud-native technologies like Kubernetes [3] and Docker [40] allow organizations to dynamically scale workloads, ensuring efficient and cost-effective utilization of computational resources.

However, despite these advancements, modern workflow engines and data pipelines are not designed with Synthetic Data Generation (SDG) in mind. Current SDG tools such as SDV [41], Synthcity [42], and Synthea [43] excel at producing high-quality synthetic datasets but lack structured, scalable execution pipelines. This often demands significant programming expertise for setup and integration, creating barriers for widespread adoption. Moreover, most SDG workflows remain rigid, isolated, and difficult to scale across hybrid cloud and on-premise environments, limiting their applicability in industrial settings where synthetic data is critical for privacy-preserving analytics.

The work aims to address these limitations by developing a scalable, modular, and cloud-native framework for orchestrating SDG within modern data pipelines, lowering the entry barrier and enabling seamless adoption across diverse operational contexts.

3.2 Core Challenges and Gaps

First, there is a fundamental disconnect between SDG tools and modern orchestration frameworks. Although existing open-source libraries can generate high-quality synthetic data, their rigid and isolated workflows complicate deployment within containerized or cloud-based systems. This lack of integration raises the technical level for adoption and prevents scalable production-grade SDG workflows.

Second, the inherent trade-off between data utility and privacy remains an open problem. While SDG techniques strive to replicate the statistical properties of real-world data, maintaining high analytical utility while adhering to increasingly limiting global privacy regulations is complex. Fine-tuning generative models such as GANs to achieve this balance often demands extensive domain expertise and customized workflows, which currently lack standardization across industries.

Third, the evolving landscape of data protection regulations imposes additional operational tasks. Organizations must frequently update and validate their synthetic data workflows to stay compliant, which can significantly slow innovation and increase operational costs.

The work seeks to address these critical gaps by proposing an integrated framework that enables scalable, regulation-aware SDG within modern data pipelines. The focus is on bridging the technical disconnect between SDG tools and orchestration engines, embedding privacy-aware best practices into SDG pipelines, and promoting standardized, modular workflows that adapt flexibly to regulatory changes.

3.3 Objectives of SynthGuard

SynthGuard is a SDG framework prototype that I developed during my internship and work as a junior programmer at Cybernetica. During the early stages of exploration and brainstorming, these were the main objectives and goals of SynthGuard:

- **Ensure Scalability And Complexity**: A solution that uses a workflow orchestration engine to be able to handle complex workflow cases and large volumes of data.
- **Enable Widely Compatible Deployment**: Structure that allows deployment in multiple operating systems, both locally and in the cloud.

- **Develop a Library to Support Framework Operations**: Creating a comprehensive library of tools and utilities to support and extend the functionality of the SynthGuard framework.
- **Prioritize Privacy and Utility**: Ensure that the SDG methods are privacy-preserving while still offering value with utility by using appropriate evaluation techniques.
- **Promote Accessibility and Ease of Use**: Design a framework that does not require deep technical knowledge or programming skills, enabling broader adoption among researchers and domain-specific professionals.

4. Methodology

This section covers the technological choices, system architecture, data management strategies, and workflow orchestration methods used in SynthGuard.

4.1 Research and Development Approach

The objective of this project is to develop a scalable, containerized workflow framework for SDG that can effectively manage data workflows while ensuring reproducibility and security. The chosen approach leverages containerized environments for modularity, workflow orchestration for efficient task management, and secure package management to maintain consistency across development stages.

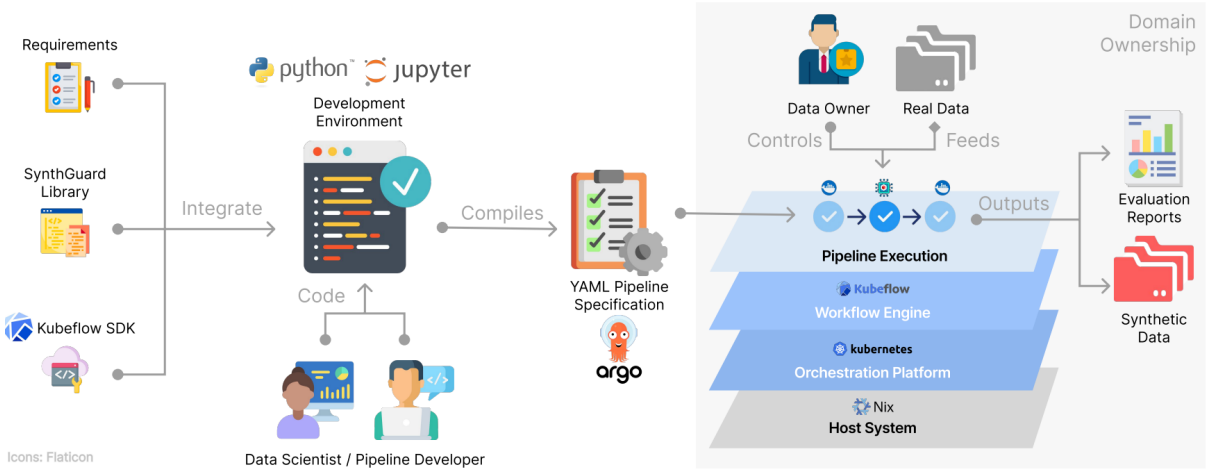


Figure 1. SynthGuard Workflow Framework Overview

The technological stack used for the initial prototype was selected with the guidance of my supervisor at Cybernetica, whose expertise in the field helped shape the project's direction. By combining tools such as Kubeflow [4] and Kubernetes [3] with Nix [5] for dependency management, the approach ensures a scalable and reproducible environment. The focus on modularity also allows the system to adapt to future extensions or integrations as the project evolves. The right side of Figure 1 showcases the combination of technologies used in the pipeline execution system.

Beyond building an SDG pipeline, this research focuses on lowering the technical barrier for domain experts by developing a Python package with reusable pipeline components for data input, preprocessing, generation, and evaluation. To further simplify development, a prototype automation tool was created to convert JSON specifications into preconfigured

Jupyter Notebooks, requiring few adjustments before compiling it into an executable pipeline specification file. Also before compiling, the user has the possibility of executing the code of the SDG pipeline component locally inside the Jupyter Notebook cells. By abstracting complex pipeline construction and automating steps, this approach makes SDG workflows more accessible, scalable, and reproducible, supporting broader adoption in privacy-sensitive fields. An overview of the development process that is automated for the SDG pipelines within the SynthGuard framework is shown on the left side of Figure 1.

4.2 Pipeline Execution System Architecture and Design Choices

Kubeflow is an open-source framework that simplifies the deployment of machine learning (ML) workflows on Kubernetes [4]. Kubernetes, also known as K8s, is an open source system for automating deployment, scaling, and management of containerized applications [3]. Originally developed by Google, it has become the industry standard for managing cloud-native infrastructure. Kubernetes abstracts away the complexity of infrastructure management, enabling systems to run reliably and scalably across distributed environments. Because SDG often relies on machine learning algorithms, the framework adopts Kubeflow to orchestrate and manage its containerized pipelines. It is deployed on Minikube [44], a specific Kubernetes implementation suitable for development and prototyping. This setup simulates a production environment while remaining accessible for local testing and development. Using Minikube ensures that the pipeline orchestration closely resembles a real-world scenario, supporting future scalability to larger Kubernetes clusters or cloud environments.

For package management and dependency isolation, Nix [5] is used to encapsulate the development environment. Nix provides reproducibility, allowing identical environments to be created across different machines. This reduces compatibility issues and ensures that all dependencies are consistently managed throughout the project lifecycle.

The system is designed to maintain modularity, with each component representing a distinct stage in the SDG workflow. The use of containerized components allows independent development, testing, and deployment of each module.

4.3 Data Management and Workflow Orchestration

The data generation and processing tasks are divided into modular components using Python. Each module is designed as a standalone function, which is then containerized and compiled into a Kubeflow pipeline component using the Kubeflow Pipelines (KFP) SDK [45]. The SDK

streamlines the creation of reusable components, promoting a flexible design and facilitating the integration of additional data processing steps when needed.

To ensure consistent data transfer between components, input and output artifacts are stored in MinIO object storage system [46]. This approach guarantees that the output from one module can be reliably accessed and used as input by subsequent modules, maintaining efficient data flow throughout the pipeline.

Once the components are connected to form a complete pipeline, the KFP SDK is used to compile the configuration into a YAML file. It is specifically designed for the Argo Workflow Engine [47], which serves as the back-end orchestrator for Kubeflow. This integration ensures compatibility with Kubernetes environments and provides flexibility for incorporating other Argo-compatible technologies. An example of extensibility was a set of additional experiments running Trusted Execution Environments (TEE) orchestrated by Argo at Cybernetica [48].

4.4 Simplifying SDG Pipeline Development

Developing SDG pipelines requires expertise in machine learning, data engineering, and workflow orchestration, making it difficult for domain experts and researchers without a strong programming background to implement their own pipelines. To bridge this gap, this project introduces a Python package that abstracts complex pipeline development into a more accessible framework.

The package provides a structured library of reusable components, categorized into four main classes:

- **Input Handling**: Functions for loading and preprocessing raw datasets from various sources.
- **Preprocessing**: Methods for metadata extraction, data transformation, and cleaning.
- **Generation**: A set of SDG models for generating synthetic data, including integrations with existing libraries like SDV.
- **Evaluation**: Tools for assessing the quality, privacy, and utility of synthetic data using statistical comparison metrics.

In addition to core pipeline components, the package includes helper functions for common operations such as data conversions, zipping/unzipping files, and format standardization to better streamline the development process.

To further lower the barrier for non-technical users, the SDG pipeline development automation tool is introduced to programmatically arrange and assemble SynthGuard library methods based on a user-defined pipeline structure. This structure can be inputted through a configuration file, allowing users to describe their desired pipeline layout at a high level without writing code.

Once the user defines the pipeline structure—specifying steps such as preprocessing, model training, and evaluation—the tool dynamically maps these stages to the appropriate SynthGuard methods. For example, if the user includes an ”Evaluate Privacy” step, the tool inserts and configures the `PrivacyRiskEvaluator` class with default or user-defined settings.

By combining a modular component library with an intelligent orchestration layer, the tool enables rapid prototyping, testing, and deployment of synthetic data pipelines. Researchers and practitioners can experiment with different models and configurations, compare evaluation results, and iterate on synthetic data workflows—all without needing to understand the underlying orchestration or data handling logic.

5. Implementation

This section provides a detailed overview of the implementation process for SynthGuard, outlining the steps taken to develop its core components and integrate them into a scalable and modular workflow framework. It highlights design decisions, technical choices, and the challenges encountered during development, along with the solutions implemented to overcome them. Additionally, this section examines how various tools and frameworks were leveraged to ensure efficiency, reproducibility, and ease of use. The full source code of the SynthGuard library and framework is available on GitHub[1].

5.1 Data Synthesis Pipeline Development and Execution

The implementation of SynthGuard is centered around building modular, reproducible, and scalable pipelines for SDG. This subsection describes the setup of the pipeline execution environment using Kubernetes and Kubeflow Pipelines, the process of developing modular pipeline components, and the integration of those components into full SDG workflows. It also outlines optimizations made to streamline execution, such as the use of a shared custom Docker image for all pipeline tasks.

5.1.1 Kubernetes and Kubeflow Pipelines Setup

During the early development phase, the full Kubeflow Central Dashboard was deployed to support the execution of demo pipelines. This deployment includes a broad suite of services for machine learning (ML) workflows, such as TensorBoard, Katib for hyperparameter tuning, and KFServing for model deployment. However, it soon became clear that these components were unnecessary for SynthGuard's goals, which are focused solely on SDG workflows rather than complete ML lifecycle management.

To simplify the environment and reduce overhead, the setup was reconfigured to deploy only the essential components using a standalone Kubeflow Pipelines (KFP) deployment [49]. This leaner configuration retains all the functionality required to define, execute, and monitor containerized pipelines, while avoiding the complexity of unused ML tools.

The entire pipeline execution environment runs on a local Kubernetes cluster managed with Minikube. This local setup simulates a realistic orchestration platform while remaining

[1] https://github.com/SynthGuard

lightweight and reproducible for development. Deployment is automated using a script (shown in Figure 11 in the Appendix) executed within a Nix [5] interactive shell using `nix-shell`.

Once the script completes, the KFP dashboard becomes accessible via http://localhost:8080. A trimmed-down version of the Kubeflow dashboard appears, as shown in Figure 2. This environment includes the essential features for pipeline development and management: uploading pipelines, tracking experiments, monitoring runs, and viewing generated artifacts. By focusing only on what is necessary, this setup enhances system performance and streamlines the workflow for SDG use cases.

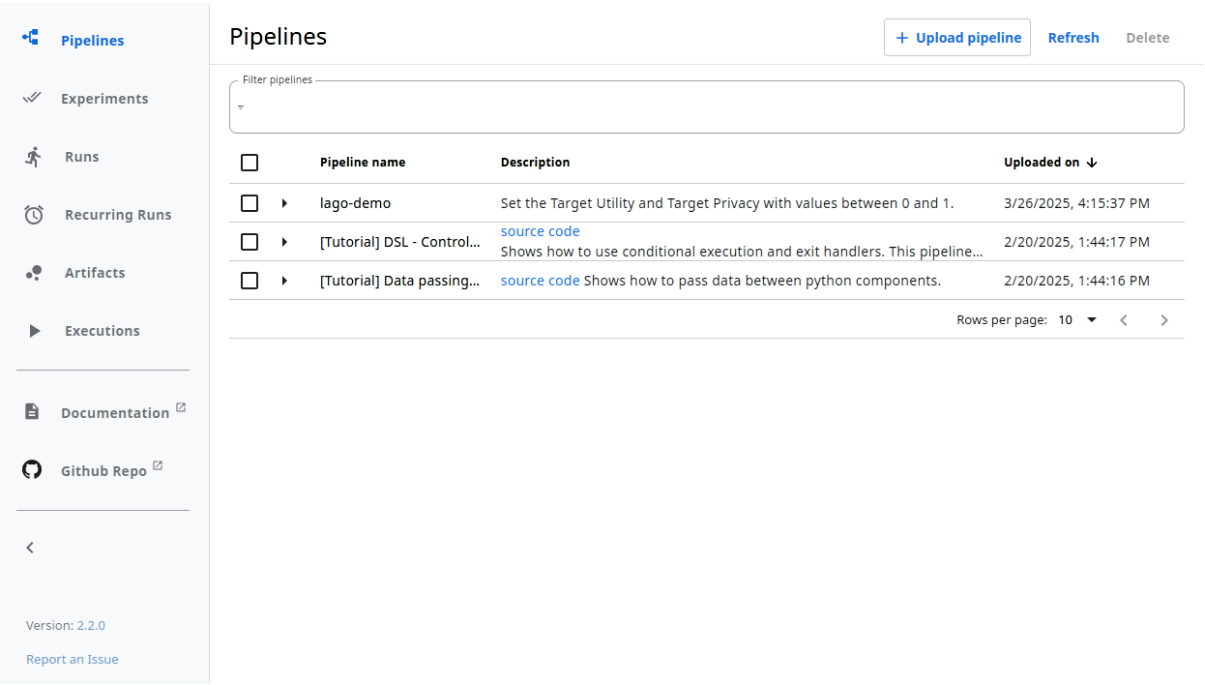


Figure 2. Kubeflow Pipelines dashboard

5.1.2 Data Synthesis Pipeline Development

The development of pipeline components is facilitated by the `kfp` Python SDK (version 1.8.22) [50], which allows Python functions to be converted into containerized pipeline components. The following features were central to the development process:

- **Component Creation:** Each module of the SDG process is written as a Python function and converted into a Kubeflow component using `kfp.components.create_component_from_func`. The resulting YAML file contains the function's execution logic, inputs, outputs, and dependencies. Figure 3 shows the Python code for an example pipeline component.

- **Data Passing Mechanisms:** Components communicate using file-based data exchange. `InputPath` and `OutputPath` decorators allow for clean and consistent passing of artifacts between isolated containers.
- **Persistent Volume Claims (PVCs):** For handling large files or maintaining state between pipeline stages, PVCs are configured and mounted to components via the `PipelineVolume` class.
- **Control Flow Features:** Pipelines support dynamic execution patterns including conditional branching with `dsl.Condition` and parallel task execution with `dsl.ParallelFor`.

```python
from kfp import components as comp

def generation(input_csv: comp.InputPath('csv'),
               input_json:comp.InputPath('json'),
               output_csv: comp.OutputPath('csv')):
    # Import
    import synthguard.helper_functions as sd
    from synthguard.synthetic_data_generator import SyntheticDataGenerator

    # Parameters
    n_rows = 10000
    epochs = 64
    locales = 'ee_ET'
    synthetic_data_type = 'realistic'

    #Loading metadata and real data
    metadata = sd.load_metadata(input_json)
    real_data = sd.load_data_csv(input_csv)

    #Defining SyntheticDataGenerator object
    syntheticDataGenerator = SyntheticDataGenerator(output_csv = output_csv,
                                                    n_rows = n_rows,
                                                    method= synthetic_data_type,
                                                    locales=locales)
    #Generating synthetic data
    generated_data = syntheticDataGenerator.generate_synthetic_data(metadata = metadata,
                                                                    processed_data = real_data,
                                                                    Nepochs=epochs)

# Compiling funtion into a KFP component using a custom Docker image
generation_component = comp.create_component_from_func(func = generation,
                                                       base_image = BASE_IMAGE)
```

Figure 3. Python code for a Kubeflow pipeline component

5.1.3 Data Pipeline Integration with Kubeflow

Once individual components are built, they are assembled into full pipelines using the KFP SDK. The workflow is compiled into a YAML specification file that can be uploaded through the KFP dashboard. The pipeline construction and execution workflow consists of the following stages:

- **Pipeline Assembly and Compilation:** Components are connected using standard Python syntax to define the execution order and dependencies (see Figure 4). The workflow is then compiled into a YAML file using `kfp.compiler.Compiler().compile()`.
- **Uploading and Execution:** The compiled YAML is uploaded via the KFP dashboard's "Pipelines" interface. The uploaded workflow is displayed as an interactive DAG (directed acyclic graph), as shown in Figure 5, enabling easy inspection of component structure and status.
- **Developed SDG Pipelines:** Several demo pipelines were implemented to support various use cases and test available functionalities:
  - Estonian citizen artificial data generator
  - GAN-based synthetic image generation using the MNIST dataset
  - Causal synthetic data generator using Bayesian learning
  - Synthetic transaction data pipeline for Fintech testing
  - Residence permit application data generator for the LAGO project
  - Synthetic energy dataset generator for the TEADAL project

```python
from kfp import dsl
from kfp.compiler import Compiler

# Defining pipeline component structure
@dsl.pipeline(name='Demo Pipeline', description='')
def pipeline():
    input = input_component()
    preprocess = preprocess_component(input.output)
    generation = generation_component(input.output, preprocess.output)
    dianostic = diagnostic_component(input.output, generation.output, preprocess.output)
    utility = utility_component(input.output, generation.output, preprocess.output)
    privacy = privacy_component(input.output, generation.output, preprocess.output)

# Compile pipeline to a runnable YAML file
Compiler().compile(pipeline, 'pipeline.yaml')
```

Figure 4. Python code defining an SDG pipeline

5.1.4 Using a Custom Docker Image for Pipelines

To optimize execution and reduce redundant builds, a shared Docker image was created for all pipeline components. This image is based on `python:3.10-slim` and includes the `synthguard` package, resulting in a minimal, fast-loading container.

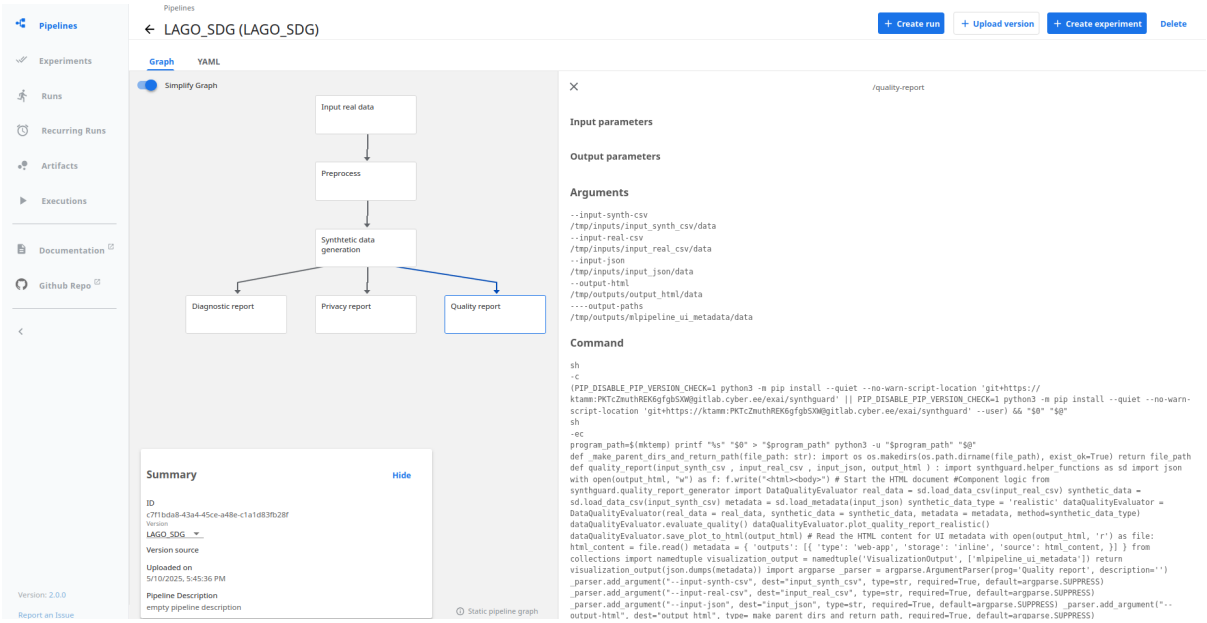


Figure 5. Interactive DAG view of an uploaded pipeline in the KFP dashboard

Using a prebuilt image significantly reduces start-up time for each component, as Kubernetes nodes can cache the image between pipeline runs. The image is stored in a private GitLab Container Registry, although it could also be hosted on Docker Hub. Since private registry access requires authentication, a Kubernetes secret is generated at startup using the `kubectl create secret docker-registry` command. This secret (`regcred`) is then referenced during pipeline execution.

To enable the pipeline to access the image securely, two methods were used:

- **Pipeline SDK configuration:** A `dsl.PipelineConf` object was created and linked to the image pull secret using the `set_image_pull_secrets()` method.
- **Manual YAML editing:** After pipeline compilation, the YAML file was modified using the `PyYAML` package to include the `imagePullSecrets` field under the pipeline `spec`.

This setup improves pipeline performance, simplifies image management, and ensures consistent environments across all pipeline components.

5.2 SynthGuard Library

The SynthGuard library is a Python-based framework designed for generating synthetic data while ensuring privacy and utility. It provides a modular and extensible architecture for handling various SDG methods, privacy evaluations, and quality assessments. The library is mostly

built on top of the The Synthetic Data Vault ecosystem[41] and integrates custom tools for domain-specific data generation, processing, and evaluation.

5.2.1 Input Handler and Data Preprocessor Classes

SynthGuard supports multiple data input formats, including `CSV`, `TXT`, `Parquet`, and `JSON` files, as well as data from URLs or `ZIP` archives. The `InputHandler` class and helper functions such as `load_data_csv` and `load_data_from_txt` streamline the process of loading data efficiently. The main components are as follows:

- **Automatic Delimiter Detection**: The `load_data_csv` function automatically detects the delimiter (e.g., `,`, `;`, or `\t`) by analyzing the structure of the file, ensuring compatibility with diverse `CSV` formats.
- **Support for Nested Data**: The `handle_nested_data_json` function in `helper_functions.py` flattens nested `JSON` structures into a tabular format, making it easier to work with complex data.
- **Flexible Data Sources**: Data can be loaded from local files, remote URLs, or compressed `ZIP` archives, providing flexibility in accessing datasets.

The preprocessing pipeline, implemented in the `DataPreprocessor` class, provides a modular approach to cleaning and transforming data. It integrates seamlessly with helper functions to handle various preprocessing tasks. The following features are implemented into the class:

- **Handling Missing Values**: Missing values are addressed using strategies such as mean, median, or mode imputation, or by dropping rows/columns. This ensures data completeness without introducing bias.
- **Metadata Extraction**: The `extract_metadata` method leverages the SDV library to extract metadata, including column types and datetime formats, which can guide further preprocessing.
- **Encoding Categorical Variables**: Categorical data is encoded using `LabelEncoder`, converting string labels into numerical representations.
- **Scaling Numerical Data**: Numerical columns are scaled using `StandardScaler` to normalize the data for machine learning models.

- **Timestamp Conversion**: Timestamp columns are converted to a standard datetime format, ensuring consistency across time-related data.

The `helper_functions.py` module provides additional utility functions to enhance the preprocessing workflow:

- **Progress Tracking**: The `show_progress` function displays a progress indicator for long-running tasks, improving user experience.
- **Error Handling**: The `handle_error` function logs errors and optionally terminates the process, ensuring robust error management.
- **Timer Utilities**: The `start_timer` and `end_timer` functions measure the duration of preprocessing tasks, aiding in performance optimization.
- **Data Validation**: Functions like `validate_data_format` and `validate_not_empty` ensure that input data meets expected criteria before processing.

5.2.2 Synthetic Data Generator Classes

This section describes the core components of the Synthguard library responsible for SDG and personal data simulation. These functionalities are implemented in the `SyntheticDataGenerator` and `PersonalFaker` classes, respectively. Both classes provide robust and customizable methods for generating high-quality synthetic datasets tailored to specific use cases.

The `SyntheticDataGenerator` class is designed to generate synthetic datasets while preserving the statistical properties of the original data. It supports multiple generation methods, allowing users to choose the most suitable approach for their needs. The class integrates with the SDV library to ensure high-quality SDG. The available methods include:

- *Hybrid Method:* Combines multiple techniques to generate synthetic data. *(We provide a mock interface to offer flexibility in choosing the generation strategy based on the specific application context.)*
- *Causal Method:* Generates data based on causal relationships.*(We provide a mock interface to offer flexibility in choosing the generation strategy based on the specific application context.)*

- *Knowledge-Based Method:* Leverages domain knowledge for data generation. *(We provide a mock interface to offer flexibility in choosing the generation strategy based on the specific application context.)*
- *Realistic Method:* Uses advanced techniques such as Gaussian Copula or CopulaGAN to generate realistic synthetic data.
- *Realistic Multi-Table Method:* Supports multi-table data generation using the HMASynthesizer.

The `PersonalFaker` class is a specialized utility for generating synthetic personal data, with a focus on Estonian-specific data. It leverages the `Faker` library [51] to create realistic and diverse datasets for testing, simulation, and other use cases. The class offers the following features:

- **Estonian-Specific Data:** The class generates Estonian-specific data such as personal identity codes using the `ssn` method and VAT IDs using the `vat_id` method. It also supports Estonian names through methods like `first_name_est`, `last_name_est`, and `name`. Additionally, the `generate_data_addresses` method creates synthetic Estonian addresses, including municipality codes and postal codes.
- **Customizable Data Generation:** Users can generate data for various domains, including addresses and license plates. The `generate_local_addresses` method generates synthetic addresses, while the `license_plate` method creates Estonian license plates. The class also supports generating random dates using the `generate_random_date` and `generate_random_dates` methods. For structured records, methods like `generate_base_record` and `generate_data_for_address` provide tailored data for specific use cases.

The `PersonalFaker` class is ideal for creating test datasets, simulating user data, and generating domain-specific synthetic data for various applications. Its integration with the Faker library ensures high-quality, diverse data generation.

5.2.3 Evaluation Classes

The SynthGuard library provides three core evaluation classes for assessing synthetic data utility and privacy. Each class supports a realistic generation method and includes placeholders for additional methods like causal, knowledge-based, and hybrid. These classes compute various

metrics and generate visualizations, which are saved as HTML files as illustrated in Figure 10 (located in the Appendices).

The `PrivacyRiskEvaluator` class, implemented in `privacy_report_generator.py`, focuses on assessing privacy risks associated with synthetic data by evaluating how well synthetic data protects sensitive information from potential attackers. The privacy metrics are the following:

- **CategoricalCAP Score**: Evaluates the risk of disclosing sensitive information through inference attacks.
- **NewRowSynthesis Score**: Measures the risk of an attacker inferring sensitive information about new individuals using synthetic data.
- **Inference Attack Score**: Measures the accuracy of predicting sensitive attributes using machine learning models trained on synthetic data.
- **TCAP Score**: Evaluates the correctness of key-target pairings in synthetic data when compared to real data.
- **Min Nearest Neighbour Distance**: Examines the overlap and similarity between real and synthetic datasets.
- **Sample Overlap Score**: Measures overlap between real and synthetic data samples.

The `DiagnosticEvaluator` class, implemented in `diagnostic_report_generator.py`, evaluates the diagnostic quality of synthetic data by assessing its structural and validity aspects using the SDV library's diagnostic tools. Set of diagnostic metrics consist of:

- **Data Validation**: Ensures that continuous and discrete values in synthetic data adhere to the range and categories in the real data, visualized using a radar chart.
- **Data Structure**: Checks that the real and synthetic data have the same column names and structural consistency, visualized using a horizontal bar chart.
- **Overall Diagnostic Score**: Summarizes the quality of synthetic data for both validity and structure.

The `DataQualityEvaluator` class, implemented in `quality_report_generator.py`, focuses on assessing the utility and statistical quality of synthetic data. The class provides the folowing list of metrics:

- **Observed pMSE**: Measures the average squared difference between predicted probabilities from a classifier distinguishing real and synthetic data.
- **Standardized pMSE**: Facilitates comparison across different datasets and models.
- **Observed-null pMSE Ratio**: Indicates the degree to which synthetic data deviates from the original data.
- **Kolmogorov-Smirnov (KS) Distance**: Using the SPECKS Metric to assess distributional similarity.

5.3 SynthGuard Pipeline Development Automation Tool

To increase user-friendliness and enhance the workflow efficiency, a prototype automation tool was developed. This tool relies on a JSON configuration file to specify the SynthGuard (SDG) pipeline specifications. The choice of JSON format was deliberate, as it offers both readability for users and flexibility for future enhancements (see Section 7).

```json
{
    "pipeline_name": "police_call_pipeline",
    "pipeline_description": "Police & Border Guard (PPA) police call dataset synthetic data pipeline",
    "components":{
        "kfp_functions": true,
        "local_functions": true,
        "input":{
            "type": "csv",
            "n_rows": 10000,
            "source": "https://opendata.smit.ee/ppa/csv/valjakutsed.csv"
        },
        "preprocess":{},
        "generation":{
            "type": "realistic",
            "n_rows": 1000
        },
        "privacy_report":{},
        "utility_report":{},
        "diagnostic_report":{}
    }
}
```

Figure 6. Pipeline specification JSON file for the development automation tool

The prototype leverages the `nbformat` package [52] to populate Jupyter Notebook cells with essential code. Much of this code consists of methods from the SynthGuard library. By adopting

this approach, researchers are able to make minimal manual changes before executing their pipeline in a local environment. Alternatively, the system can compile the pipeline into a YAML configuration file using the Kubeflow Pipelines (KFP) SDK. This automation significantly reduces the effort needed to develop and deploy SDG workflows, making advanced SDG techniques more accessible to a broader range of users.

While the current prototype primarily supports the *realistic method* due to the predominance of use cases in this domain, the framework is designed with expandability in mind. Future versions will integrate additional SDG methods to cater to a variety of research and application scenarios. This inclusive design philosophy aims to support a diverse set of use cases and facilitate the seamless inclusion of novel techniques within the SynthGuard pipeline ecosystem.

The example JSON file, illustrated in Figure 6, includes basic fields such as pipeline name, source data type and location, and generated synthetic data parameters. This configuration flexibility allows users to customize pipelines to fit their specific needs.

6. Evaluation and Results

6.1 Performance and Scalability Testing

The goal of this evaluation is to assess how well the SynthGuard pipeline scales with increasing data sizes and to identify potential performance bottlenecks in real-world use cases.

6.1.1 Experiment Setup

The performance and scalability tests are conducted using the execution stack described in Subsection 4.2, deployed on a virtual machine (VM) hosted in a cloud environment. This setup simulates a realistic deployment scenario and ensures consistent performance benchmarking across different test cases. The VM specifications are as follows:

- **CPU:** 8 VCPU
- **RAM:** 12 GB
- **Storage:** 40 GB

To evaluate the pipeline's performance under varying data loads, three synthetic datasets of increasing size were used: 1,000, 10,000, and 100,000 rows.

Each dataset is used as input to a complete SDG pipeline, where the number of generated synthetic rows is matched to the size of the input data. During each pipeline run, the execution time for each individual pipeline component is measured in the code itself and outputted in the component logs. Also the whole pipeline execution time data is gathered from the Kubeflow Pipelines dashboard. This allows us to visualize execution time relative to the size of the dataset, which will provide information on scalability and efficiency.

The pipeline employed in this evaluation is adapted from the existing LAGO use case. It follows a structured sequence consisting of four main stages: loading the input data, preprocessing and metadata extraction, SDG, and parallel evaluation report generation. The evaluation phase includes the creation of quality reports, diagnostic reports, and privacy assessment reports, each executed concurrently to optimize performance. During the initial runs, a bottleneck was discovered in the quality report generation. The time required to calculate propensity metrics for the 10,000 row dataset was excessive, so I modified the SynthGuard library method to use a smaller sample when the input dataset was too large. For the experiments, the pipeline uses

open real-world data published by the Estonian Police and Border Guard Board—specifically, the dataset on residence permit applications[2].

6.1.2 Results

After executing the pipelines and gathering the data, several components show stable performance across dataset sizes as shown in Figure 7. Specifically, *Input Data Loading*, *Preprocessing & Metadata Extraction*, and *Diagnostic Evaluation Report* exhibit near-constant runtimes. For example, the data loading stage takes approximately 0.10–0.11 minutes across all configurations, suggesting that these steps involve fixed-cost operations that do not scale with dataset size.

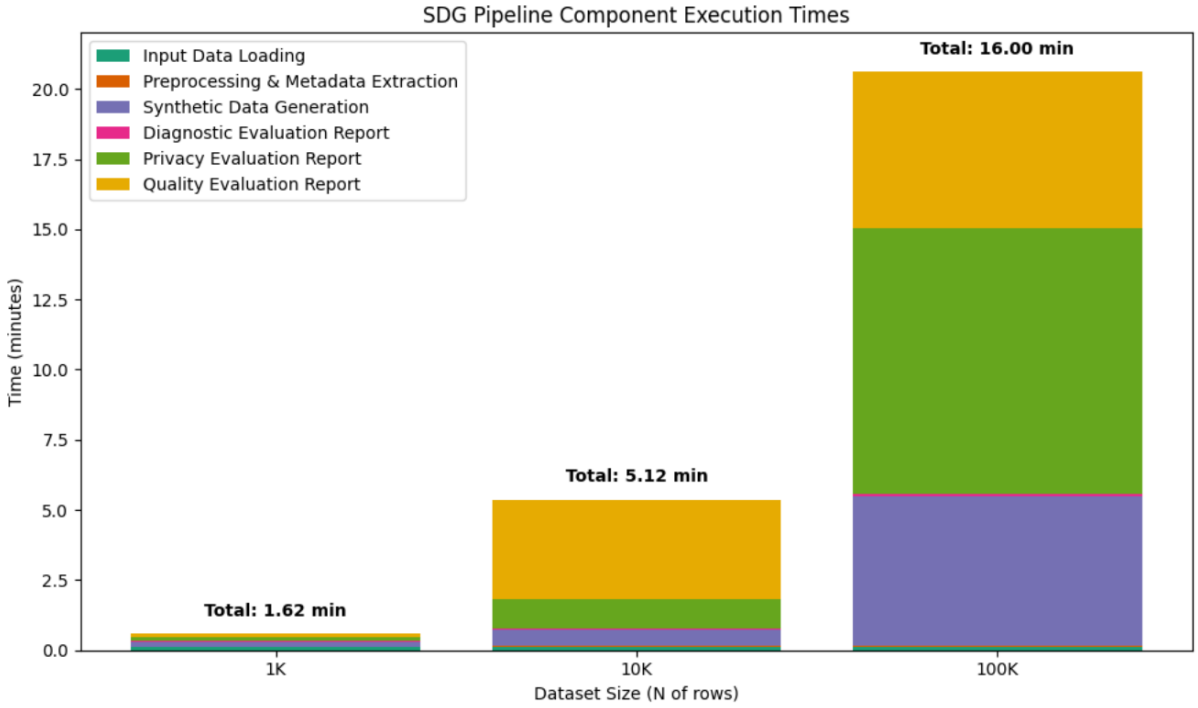


Figure 7. Execution time per SDG pipeline component across dataset sizes.

By contrast, the *Synthetic Data Generation*, *Privacy Evaluation Report*, and *Quality Evaluation Report* components demonstrate significant scaling behavior. As dataset size increases from 1K to 100K rows:

- Synthetic Data Generation time increases from 0.14 to 5.35 minutes ($\sim 38\times$),
- Privacy Evaluation time rises from 0.14 to 9.46 minutes ($\sim 68\times$),

[2] https://www.politsei.ee/et/juhend/politseitoeoega-seotud-avaandmed/elamislubade-taotlused

- Quality Evaluation grows from 0.10 to 5.59 minutes ($\sim 56\times$).

Despite these increases, the total pipeline runtime scales sublinearly:

- 1K rows: 1 minute 37 seconds,
- 10K rows: 5 minutes 7 seconds,
- 100K rows: 16 minutes.

This sublinear growth, despite a $100\times$ increase in data size, implies that certain computational costs are amortized or benefit from internal efficiencies such as batch processing.

Another contributing factor to this sublinear scaling is the parallel execution of the evaluation components. Utility, diagnostic, and privacy evaluation reports are triggered as independent tasks at the end of the pipeline and executed concurrently by Kubeflow through Argo's workflow engine. This design ensures that increased evaluation complexity does not linearly impact total runtime, and showcases the architectural advantage of modular, parallelizable components within SynthGuard.

At the largest scale (100K rows), the *Privacy Evaluation Report* and *Quality Evaluation Report* together account for over 90% of the total runtime. These stages are likely performing intensive, row-wise or distribution-level statistical comparisons. As such, they represent performance bottlenecks and opportunities for optimization in future iterations of the pipeline.

6.2 Feedback Analysis from LAGO Project Pilot Run

As part of the LAGO project, we conducted a pilot run of SynthGuard, during which project participants tested the tool in a practical setting. Users accessed the deployed Kubeflow Pipelines dashboard, executed a sample pipeline using the provided YAML configuration, and followed the accompanying user manual.

Feedback received was largely positive, validating both design decisions and the practical utility of the tool in real-world scenarios. The participants appreciated the potential of the tool and provided constructive suggestions for improvement.

This feedback serves as a valuable input to shape our next steps, helping prioritize enhancements and refine SynthGuard's usability, documentation, and feature set in future development cycles.

7. Discussion and Future Work

Several opportunities exist to extend the SynthGuard framework further, both in terms of functionality and usability. This section outlines directions for future work based on identified limitations and new potential use cases.

7.1 Enhancing the SDG Pipeline Development Automation Tool

One of the main goals of SynthGuard is to reduce the technical barrier for domain experts working with synthetic data. The current JSON-to-Notebook prototype validates this concept, but further development is needed to realize its full potential. A logical next step is to develop a user-friendly graphical interface (GUI) that would allow users to visually assemble SDG pipelines, configure components, and generate executable pipeline files without requiring knowledge of Python or Kubernetes.

In addition, integrating a large language model (LLM) could enable conversational pipeline construction, where users describe their objectives in natural language, and the system interprets and converts the request into a pipeline specification. This approach would significantly improve accessibility and usability for non-technical users.

7.2 Standardizing Artificial and Causal Synthetic Data Generation

While SynthGuard currently supports realistic SDG methods, solid support for *artificial synthetic data* (ASD) and *causal synthetic data* (CSD) remains limited. ASD is based on predefined rules or statistical heuristics, and is especially useful in environments where real data is inaccessible. CSD aims to preserve meaningful causal relationships between data points, which is important for simulations and scenario analysis.

To integrate ASD and CSD more effectively into the framework, future work should focus on standardizing how generation rules and causal structures are defined. This includes developing clear configuration formats and ensuring the generated data is validated against expected constraints and relationships. Expanding support for these methods would significantly increase SynthGuard's flexibility and usefulness across a broader range of applications.

7.3 Extending Support for Multi-Table and Relational Data Generation

Most real-world datasets are not flat but relational, involving multiple interdependent tables. SynthGuard currently supports only single-table data generation. Extending the framework to handle multi-table or relational SDG would significantly broaden its practical utility.

This extension would require the ability to ingest database schemas, model foreign relationships, and ensure consistency between linked tables in the synthetic output. Integrating or developing generation models that can learn and preserve these dependencies is a challenging but worthwhile area for future work. Additionally, relational data evaluation metrics would need to be incorporated to assess the structural and statistical integrity of the generated datasets.

8. Conclusion

In this thesis, I developed SynthGuard, a scalable and privacy-preserving SDG workflow framework. SynthGuard integrates the essential components of SDG into a unified, modular, and cloud-native system, addressing the limitations of existing SDG tools, such as lack of scalability, limited privacy-preserving features, and high technical barriers for non-experts. The framework leverages technologies like Kubernetes, Kubeflow, and NixOS to ensure secure, efficient, and reproducible workflows. Additionally, a Python-based library and an automation tool were created to simplify pipeline development for domain experts.

The results demonstrate that SynthGuard achieves its objectives of scalability, privacy preservation, and accessibility. The evaluation showed that the framework scales sublinearly with increasing data sizes, due to its modular and parallelized architecture. Privacy and utility evaluations confirmed that SynthGuard-generated synthetic data maintains statistical integrity while minimizing re-identification risks. Feedback from the LAGO and TEADAL use cases highlighted the framework's practical applicability in privacy-sensitive domains such as law enforcement and energy data analysis.

Future work includes extending SynthGuard's capabilities to support multi-table and relational data generation, enhancing the automation tool with a graphical user interface (GUI), and integrating conversational AI for natural language pipeline configuration. Additionally, further standardization of artificial and causal SDG methods and the inclusion of advanced evaluation metrics will broaden the framework's applicability across diverse industries. These advancements will ensure that SynthGuard continues to evolve as a robust solution for SDG in real-world scenarios.

References

[1] Taub J., Elliot M., Pampaka M., and Smith D. Differential Correct Attribution Probability for Synthetic Data: An Exploration. *Privacy in Statistical Databases: UNESCO Chair in Data Privacy, International Conference, PSD 2018, Valencia, Spain, September 26–28, 2018, Proceedings*. Springer, 2018, pp. 122–137. DOI: 10.1007/978-3-319-99771-1_9. https://link.springer.com/chapter/10.1007/978-3-319-99771-1_9.

[2] Park N., Mohammadi M., Gorde K., Jajodia S., Park H., and Kim Y. Data Synthesis Based on Generative Adversarial Networks. *Proceedings of the VLDB Endowment* 11.10 (2018), pp. 1071–1083. DOI: 10.14778/3231751.3231757. https://www.vldb.org/pvldb/vol11/p1071-park.pdf.

[3] Authors K. Kubernetes. https://kubernetes.io/. Accessed: 2025-03-18.

[4] Aronchick D., Lewi J., and Kannan V. Kubeflow: Machine Learning Toolkit for Kubernetes. https://www.kubeflow.org/. Accessed: 2025-03-28.

[5] Dolstra E. Nix: The Purely Functional Package Manager. https://nixos.org/nix/. Accessed: 2025-03-28.

[6] Mittal S., Thakral K., Singh R., Vatsa M., Glaser T., Ferrer C. C., and Hassner T. On Responsible Machine Learning Datasets with Fairness, Privacy, and Regulatory Norms. *arXiv preprint arXiv:2310.15848* (2023).

[7] Strobel M. and Shokri R. Data Privacy and Trustworthy Machine Learning. *arXiv preprint arXiv:2209.06529* (2022).

[8] El Mestari S. Z. and Lenzini G. Preserving data privacy in machine learning systems. *Computers & Security* 137 (2024), p. 103605.

[9] Osorio-Marulanda P. A., Epelde G., Hernandez M., Isasa I., Reyes N. M., and Iraola A. B. Privacy Mechanisms and Evaluation Metrics for Synthetic Data Generation: A Systematic Review. *IEEE Access* 12 (2024), pp. 88048–88074. DOI: 10.1109/ACCESS.2024.3281234. https://www.researchgate.net/publication/381621195_Privacy_mechanisms_and_evaluation_metrics_for_Synthetic_Data_Generation_A_systematic_review.

[10] Ping H., Stoyanovich J., and Howe B. DataSynthesizer: Privacy-Preserving Synthetic Datasets. *Proceedings of the 29th International Conference on Scientific and Statistical Database Management (SSDBM '17)*. New York, NY, USA: Association for Computing Machinery, 2017, 42:1–42:5. DOI: 10.1145/3085504.3091117. https://doi.org/10.1145/3085504.3091117.

[11] Simonite T. Some Startups Use Fake Data to Train AI. *WIRED* (2018). https://www.wired.com/story/some-startups-use-fake-data-to-train-ai.

[12] Wen B., Colon L. O., Subbalakshmi K. P., and Chandramouli R. Causal-TGAN: Generating Tabular Data Using Causal Generative Adversarial Networks. *arXiv preprint arXiv:2104.10680* (2021). https://arxiv.org/abs/2104.10680.

[13] Wang K., Shi F., Wang W., Nan Y., and Lian S. Synthetic Data Generation and Adaption for Object Detection in Smart Vending Machines. *arXiv preprint arXiv:1904.12294* (2019). https://arxiv.org/abs/1904.12294.

[14] Prenestino F. B., Barbierato E., and Gatti A. Robust Synthetic Data Generation for Sequential Financial Models Using Hybrid Variational Autoencoder–Markov Chain Monte Carlo Architectures. *Future Internet* 17.2 (2025), p. 95. DOI: 10.3390/fi17020095. https://www.mdpi.com/1999-5903/17/2/95.

[15] European Parliament and Council. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC. 2016. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679.

[16] California State Legislature. California Consumer Privacy Act of 2018. 2018. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=2017AB375.

[17] U.S. Department of Health and Human Services. HIPAA Privacy Rule. 2013. https://www.hhs.gov/hipaa/for-individuals/guidance-materials-for-consumers/index.html.

[18] Office of the Privacy Commissioner of Canada. Personal Information Protection and Electronic Documents Act (PIPEDA). 2000. https://laws-lois.justice.gc.ca/eng/acts/P-8.6/.

[19] Dilmegani C. Top 10 Privacy Enhancing Technologies in 2025. Tech. rep. AIMultiple, 2025. https://research.aimultiple.com/privacy-enhancing-technologies/.

[20] Kamm L., Bogdanov D., Brito E., and Ostrak A. Blueprints for Deploying Privacy Enhancing Technologies in E-Government. *Privacy and Identity Management. Sharing in a Digital World*. Ed. by Bieker F., Conca S. de, Gruschka N., Jensen M., and Schiering I. Vol. 695. IFIP Advances in Information and Communication Technology. Springer, 2024, pp. 3–19. DOI: 10.1007/978-3-031-57978-3_1. https://link.springer.com/chapter/10.1007/978-3-031-57978-3_1.

[21] Antoniou A., Dossena G., MacMillan J., Hamblin S., Clifton D., and Petrone P. Assessing the risk of re-identification arising from an attack on anonymised data. 2022. arXiv: 2203.16921 [cs.LG]. https://arxiv.org/abs/2203.16921.

[22] Sarmin F. and Kenthapadi K. Synthetic Data: Revisiting the Privacy-Utility Trade-off. *arXiv* (2024). https://ui.adsabs.harvard.edu/abs/2024arXiv240707926J.

[23] Goodfellow I., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., and Bengio Y. Generative Adversarial Nets. *Advances in Neural Information Processing Systems*. Vol. 27. 2014.

[24] Louizos C., Shalit U., Mooij J. M., Sontag D., Zemel R., and Welling M. Causal Effect Inference with Deep Latent-Variable Models. *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[25] Baraka K., Melo F. S., and Veloso M. Data-Driven Generation of Synthetic Behavioral Feature Vectors Modeling Children with Autism Spectrum Disorders. *Proceedings of the 8th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACII)*. IEEE. 2017.

[26] Patki N., Wedge R., and Veeramachaneni K. The Synthetic Data Vault. *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2016, pp. 399–410.

[27] Giuffrè M. and Shung D. L. Harnessing the power of synthetic data in healthcare: innovation, application, and privacy. *npj Digital Medicine* 6.1 (2023), p. 186. DOI: 10.1038/s41746-023-00927-3. https://doi.org/10.1038/s41746-023-00927-3.

[28] Beigi M., Shafquat A., Chen J., Aptekar J., and Itzkovich Y. Accelerating Breakthroughs with Synthetic Clinical Trial Data. *Applied Clinical Trials* 33 (2024). https://www.appliedclinicaltrialsonline.com/view/accelerating-breakthroughs-with-synthetic-clinical-trial-data.

[29] Potluru V. K., Borrajo D., Coletta A., Dalmasso N., El-Laham Y., Fons E., Ghassemi M., Gopalakrishnan S., Gosai V., Kreačić E., et al. Synthetic Data Applications in Finance. *arXiv preprint arXiv:2401.00081* (2023).

[30] Sivathapandi P., Paul D., and Selvaraj A. AI-Generated Synthetic Data for Stress Testing Financial Systems: A Machine Learning Approach to Scenario Analysis and Risk Management. *Journal of Artificial Intelligence Research* 2.1 (2022), pp. 246–287. https://thesciencebrigade.com/JAIR/article/view/366.

[31] Research J. M. A. Synthetic Data for Real Insights. *J.P. Morgan Technology Blog* (2023). https://www.jpmorgan.com/technology/technology-blog/synthetic-data-for-real-insights.

[32] Consortium L. LAGO: Trusted European Research Data Ecosystem for Fighting Crime and Terrorism. https://lago-europe.eu/about. Accessed: 2025-04-01. 2025.

[33] Directive (EU) 2016/680 on the protection of personal data for law enforcement purposes. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016L0680. OJ L 119, 4.5.2016, p. 89–131. 2016.

[34] Consortium T. TEADAL: Trusted, Verifiable, and Energy-Efficient Data Lakes. https://teadal.eu/about-teadal. Accessed: 2025-04-01. 2025.

[35] Foundation A. S. Apache Airflow. https://airflow.apache.org/. Accessed: 2025-03-18.

[36] Services A. W. AWS Glue. https://aws.amazon.com/glue/. Accessed: 2025-03-18.

[37] Services A. W. Amazon Web Services (AWS). https://aws.amazon.com/. Accessed: 2025-03-18.

[38] Cloud G. Google Cloud Platform. https://cloud.google.com/. Accessed: 2025-03-18.

[39] Microsoft. Microsoft Azure. https://azure.microsoft.com/. Accessed: 2025-03-18.

[40] Docker I. Docker. https://www.docker.com/. Accessed: 2025-03-18.

[41] DataSynthesizer. Synthetic Data Vault (SDV). https://sdv.dev/. Accessed: 2025-03-18.

[42] Vanderschueren S., Martinc M., and De Turck F. Synthcity: Facilitating Innovative Use Cases of Synthetic Data in Machine Learning. https://github.com/vanderschuea/synthcity. Accessed: 2025-03-18.

[43] Walonoski J., Kramer M., Nichols J., Quina A., Moesel C., Hall D., Duffield T., Bharadwaj R., Cambell T., and McLachlan S. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. *Journal of the American Medical Informatics Association* 25.3 (2018), pp. 230–238. DOI: 10.1093/jamia/ocx079. https://academic.oup.com/jamia/article/25/3/230/4098271.

[44] Authors K. Minikube: Run Kubernetes Locally. https://minikube.sigs.k8s.io/. Accessed: 2025-03-28.

[45] Community K. Kubeflow Pipelines SDK. https://www.kubeflow.org/docs/components/pipelines/sdk/sdk-overview/. Accessed: 2025-03-28.

[46] MinIO I. MinIO: High Performance Object Storage. https://min.io/. Accessed: 2025-03-28.

[47] Contributors A. P. Argo Workflows: Get Stuff Done with Kubernetes. https://argoproj.github.io/argo-workflows/. Accessed: 2025-03-28.

[48] Brito E., Castillo F., Pullonen-Raudvere P., and Werner S. TrustOps: Continuously Building Trustworthy Software. *Enterprise Design, Operations, and Computing. EDOC 2024 Workshops*. Springer Nature Switzerland, 2025, pp. 53–67.

[49] Kubeflow Contributors. Deploying Kubeflow Pipelines Standalone. https://www.kubeflow.org/docs/components/pipelines/legacy-v1/installation/standalone-deployment/. 2023.

[50] Authors K. Kubeflow Pipelines SDK (kfp) v1.8.22. https://pypi.org/project/kfp/1.8.22/. 2021. https://pypi.org/project/kfp/1.8.22/.

[51] Faraglia D. and Contributors. Faker: Python Package for Generating Fake Data. Version 24.3.0. 2025. https://faker.readthedocs.io/.

[52] Team J. D. nbformat: Jupyter Notebook Format. https://github.com/jupyter/nbformat. Version 5.9.2. 2024.

Appendices

Figure 8: LAGO Use Case Process Diagram

This figure illustrates the data flow and privacy-preserving deployment model used in the LAGO project. It highlights how sensitive data remains within the secure environment of law enforcement agencies, while synthetic data is generated for safe external use.

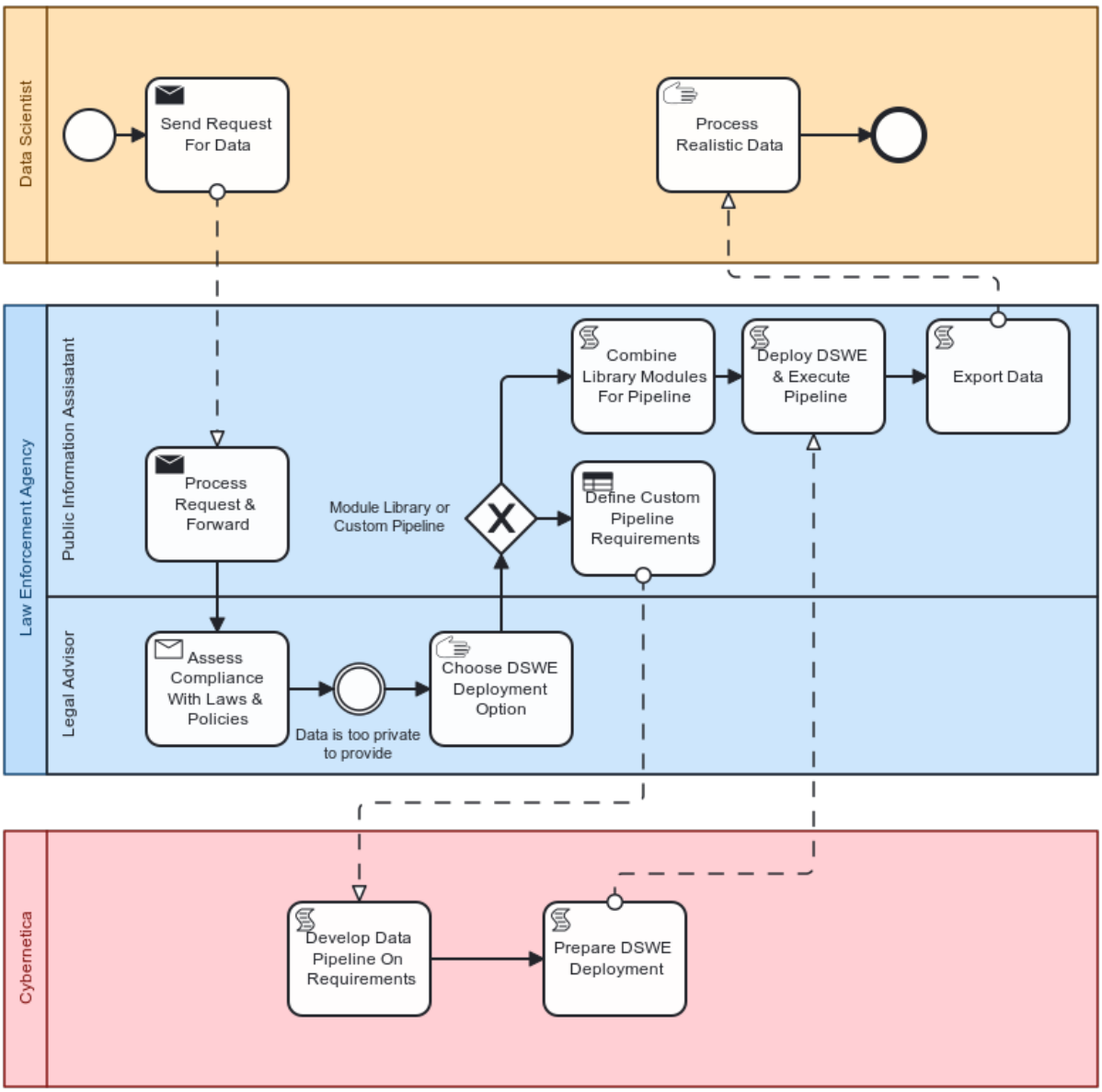


Figure 8. Process diagram showcasing the LAGO use case.

Figure 9: TEADAL Energy Pilot Workflow

This figure shows the pipeline designed for the TEADAL energy pilot, which integrates various public and private datasets from the Tuscany region. The pipeline demonstrates how synthetic data is used to unify and analyze environmental and energy indicators across domains.

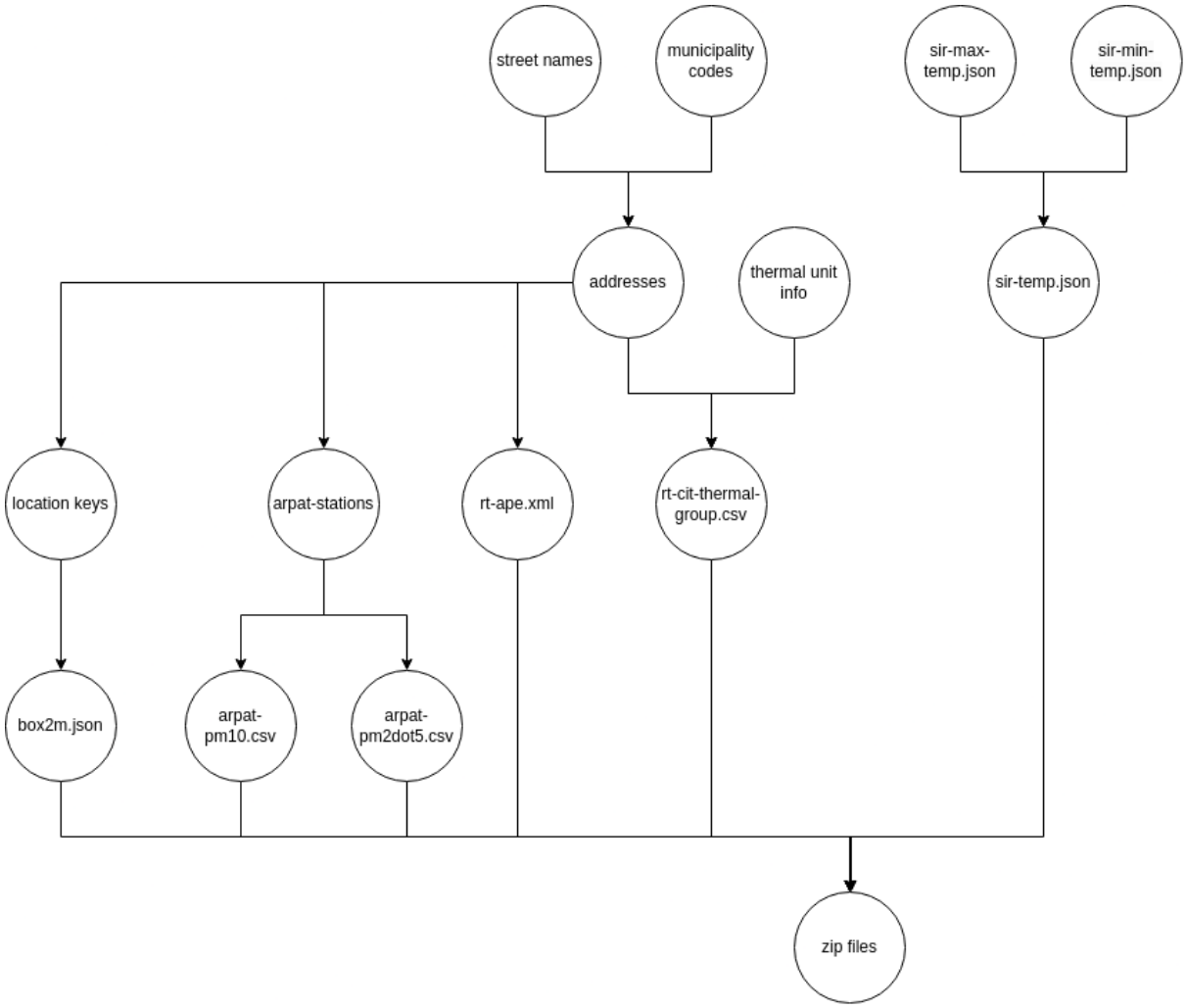
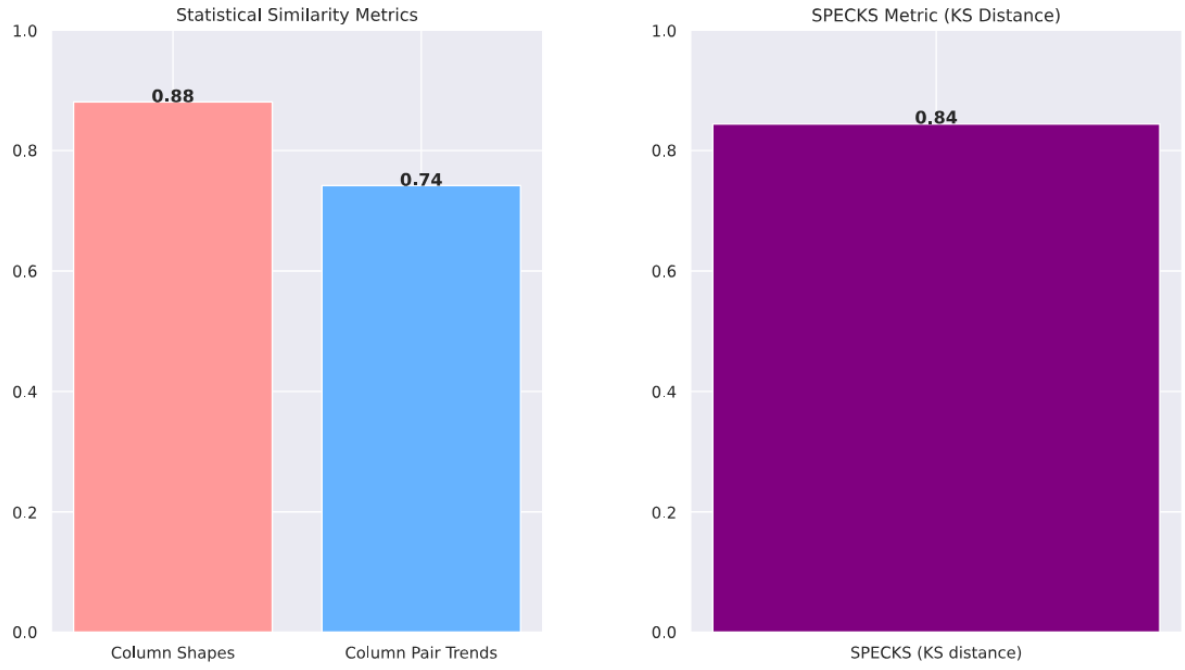


Figure 9. TEADAL energy pilot workflow. **RT-CIT**: thermal and AC systems check reports; **RT-APE**: building energy performance certificates; **SIR-temp**: daily temperature extremes (2009–2023); **ARPAT**: air quality data (PM10, PM2.5); **BOX2M**: energy monitoring device readings.

Figure 10: SynthGuard Utility Report Output

Displayed here is the HTML-based utility report generated by the SynthGuard Library. This output is rendered in the Kubeflow Pipelines dashboard and provides visual summaries of synthetic data quality and utility metrics.



Propensity Metrics:
Observed pMSE: 0.25 - Measures the difference between real and synthetic data.
Standardized pMSE: 60.72 - Normalized version of observed pMSE.
Observed-null pMSE ratio: 2.33 - Ratio of observed pMSE to null pMSE.

Figure 10. Utility report HTML generated by the SynthGuard Library.

Figure 11: `kfp-start.nix` Deployment Script

This figure presents the `kfp-start.nix` script used to automate the deployment of the SynthGuard execution stack. The script is run inside a Nix [5] interactive shell and initializes the Kubeflow Pipelines environment on Minikube.

```
echo "Starting Minikube..."
alias kubectl='minikube kubectl --'

minikube start --memory=8096 --cpus=8 --disk-size=30g

echo "Installing Portainer ${portainerVersion}"
kubectl apply -n portainer -f ${portainerNodePort}
until kubectl get pod -n portainer -o jsonpath='{.items[0].status.phase}' 2>/dev/null | grep -q
"Running"; do
  sleep 1
  echo "Waiting for Portainer to start..."
done

echo "Portainer is running and can be accessed via:"
minikube service -n portainer --all --url --https

export PIPELINE_VERSION=2.0.0
kubectl apply --kustomize="github.com/kubeflow/pipelines/manifests/kustomize/cluster-scoped-
resources?ref=$PIPELINE_VERSION&timeout=90s"
kubectl wait crd/applications.app.k8s.io --for=condition=established --timeout=60s
kubectl apply --kustomize="github.com/kubeflow/pipelines/manifests/kustomize/env/platform-
agnostic?ref=$PIPELINE_VERSION&timeout=90s"

echo "Waiting for Kubeflow components to settle..."

while true; do
  total_pods=$(kubectl get pods -A --no-headers | wc -l)
  running_pods=$(( $(kubectl get pods -A --field-selector=status.phase==Running --no-headers |
wc -l) + $(kubectl get pods -A --field-selector=status.phase==Succeeded --no-headers | wc -l)
))
  non_running_pods=$(kubectl get pods -A --field-selector=status.phase!=Running --no-headers)
  echo "Pods running: $running_pods/$total_pods"
  if [ "$non_running_pods" ]; then
    echo "Non-running pods:"
    echo "$non_running_pods"
  fi
  # Break if all pods are running successfully
  if [ "$running_pods" -eq "$total_pods" ]; then
    break
  fi
  sleep 30
done

echo "Creating secret for Docker credentials..."
kubectl create secret docker-registry regcred -n kubeflow --docker-server= --docker-username=
--docker-password= --docker-email=

echo "Setting up port forwarding for Kubeflow dashboard..."
kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80 &
```

Figure 11. `kfp-start.nix` script for deploying the SynthGuard execution environment

License

Non-exclusive License to Reproduce the Thesis and Make the Thesis Public

I, **Kristian Tamm**,

1. Grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation (including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright), my thesis entitled:

SynthGuard: Scalable and Privacy-Preserving Synthetic Data Generation Workflow Framework

Supervised by **Eduardo Ribas Brito**.

2. Grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence **CC BY-NC-ND 4.0**, which allows, by giving appropriate credit to the author, to reproduce and distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;
3. Acknowledge that the author retains the rights specified in points 1 and 2;
4. Confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from personal data protection legislation.

Kristian Tamm
15/05/2025