MANOLO

CLOUD-EDGE EFFICIENT & TRUSTWORTHY AI

D2.1 Data Inspection and Generation v.1

NCSR-D

February 2025

Funded by
the European Union

Document Information

Issued by:	NCSR-D
Issue date:	February 2025
Due date:	31 December 2024
Work package leader:	NCSR-D
Dissemination level:	Public

Document History

Version	Date	Modifications made by
0.1	June 2024	Document structure by NCSR-D
0.2	December 2024	Sections 4, 5 & 6 finalized by NUIDUCD - CeADAR
0.3	December 2024	Section 2 finalized by ARX.NET
0.4	January 2025	Section 3 finalized by NCSR-D and FDI
0.5	January 2025	Subsection 3.5.1 finalized by BIT&BRAIN
0.9	February 2025	All partners addressed review comments
1.0	February 2025	D2.1 Delivered

Authors

Name	Beneficiary
T. Boura, N. Koliou, S. Konstantopoulos, C. Romesis	NCSR-D
E. Arazo, C. Bosch, H. Stoev, A. Suarez-Cetrulo, R. Simon Carbajo	NUIDUCD – CeADAR
P. I. Kaplanoglou, P. Doudoulakis	ARX.NET
P. Trakadas, M. Kelaidis	FDI
L. Montesano, M. Sierra	BIT&BRAIN

In case you want any additional information, or you want to consult with the authors of this document, please send your inquiries to Stasinos Konstantopoulos, konstant@iit.demokritos.gr

Quality Reviewers

Name	Beneficiary
Mounir Bensalem	TUBS
Guillaume Charpiat	INRIA

Disclaimer
Funded by the European Union under GA no. 101135782. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or CNECT. Neither the European Union nor the granting authority can be held responsible for them.

Contents

List of Figures

List of Tables

List of Terms and Definitions

Term	Definition
DD	Dataset Distillation
UMAP	Uniform Manifold Approximation and Projection
VAE	Variational Autoencoder
NN	Neural Network
KNN	K-Nearest Neighbor
INN	Invertible Neurla Network
CINN	Conditional Invertible Neural Network
GAN	Generative Adversarial Networks
CGAN	Conditional Generative Adversarial Networks
DCGAN	Deep Convolutional Generative Adversarial Networks
TSNE	T-Distributed Stochastic Neighbor Embedding
PCA	Pricnipal Component Analysis

Executive Summary

This report documents research & development work carried out in WP2 during Phase 2 *MANOLO Framework Implementation* of the project's workplan. This work comprises the development of the following MANOLO library components:

- The *Data Operations Manager*
- The *Data Quality Estimation Component*
- The *Data Distillation and Synthesis Component*

The *Data Operations Manager* provides functionality for the management of the project's digital assets (data, models, experimental results) ensuring that provenance and lineage metadata is automatically maintained. The Data Operations Manager is an asset management back-end and a Python API that is specifically designed for ease of use from the tools in the MANOLO library. The *Data Quality Estimation Component* is a comprehensive toolkit for automatically annotating data in terms of quality. The toolkit comprises both prior methods and the result of R&D work during MANOLO. The *Data Distillation and Synthesis Component* generates synthetic data using methods for distillation via data compression and hashing, feature extraction and synthesisation; as well as model inversion for synthesisation of data from labels. This enables the creation of useful dataset repositories to be utilized for posterior training processes, particularly for tasks where existing datasets are not adequate or they are not available due to ethical or regulatory considerations.

1 Introduction

1.1 Scope of Deliverable

This report, titled 'Data Inspection and Generation v.1', documents research & development work carried out in WP2 during Phase 2 *MANOLO Framework Implementation* of the project's workplan.

Besides the report itself, the scope of this deliverable also comprises the intermediate versions of the following software components:

- The *Data Operations Manager*
- The *Data Quality Estimation Component*
- The *Data Distillation and Synthesis Component*

Work in WP2 also includes preparing and ingesting the use case datasets in the data operations manager, but this is outside the scope of this version of the deliverable and will be reported in v.2 (M24).

1.2 Structure of Deliverable

Taking the above into consideration, this remainder of this document is structured as follows:

- Section 2: MANOLO presents to its cloud-edge operators a complex provenance and lineage environment where the different assets, i.e., datasets, algorithms, models, resources are multiply interlinked, providing explanation of the capacities, data, metadata and their relationships. For regulatory as well as pragmatic reasons, MANOLO will need to ensure that project assets are integrated with the MANOLO data (assets) management sub-system, so that provenance and lineage metadata is automatically maintained. The Data Operations Manager will manage this functionality and expose APIs which will be supporting the overall project and MANOLO research and toolset.

- Section 3: Mechanisms for data quality estimation will be developed, tested and integrated in this component by focussing on detecting and correcting anomalous data and automatically annotating data in terms of quality. Mechanisms for noise detection (including biased data due to gender, race, or other variables) as well as data maliciously manipulated will be given a special emphasis, employing adversarial machine learning while considering associated models.

- Sections 4 to 6: Techniques for data distillation will be explored here with the aim of providing a good foundation and richer datasets to support the research in the Hardware-aware Model Training and Optimisation component. The outcome of this work is the generation of new derived (synthetic) data using methods for distillation via data compression and hashing, feature extraction and synthesisation; as well as model inversion for synthesisation of data from labels. Data compression will lead to a reduction in storage requirements, as well as faster training pipelines and lighter models, all while preserving accuracy. Feature extraction and synthesisation will allow us to propose meta-data for meta learning tasks. The creation of synthetic data from labels is a technique that will help in the construction of reliable datasets from accurate pretrained architectures. This will enable the creation of useful dataset repositories to be utilized for posterior training processes, particularly for tasks where existing datasets are not of a high enough standard, whether this be due to poor quality, insufficient quantity or ethical reasons.

2 Data Management and Provenance Framework

2.1 Overview

MANOLO's Data Inspection & Generation component/framework will be implemented in this task as a generic cloud-based repository in which all Digital Artifacts of the project that are used across the edge-cloud continuum will be registered and stored. The artifacts could be any type of data sample, datasets, metadata, AI models, benchmark configuration and results, usage analytics and performance metrics which can be transferred between any requesting component of MANOLO through the use of an appropriate API. The API will provide all the necessary end points for user authentication implementing a secure identification scheme, for registering the artifacts and for manipulating them via the relevant operations execution.

All data transfers from the cloud repository to any other location of the edge-cloud continuum and vice versa, will be encrypted and concurrently a low latency application layer network protocol will be employed. This architecture provides a trustworthy solution for general use, with the greatest benefits expected to arise from the deployment on the Edge devices. A provenance and lineage system will be developed to relate both the data, models and metadata both inputted and derived from the application of different techniques of MANOLO which will support the overall activities of MANOLO in terms of algorithm design, testing, benchmarking and validation enhancing the explainability and reproducibility.

The repository will be enhanced with metadata about (a) data quality (see T2.2); and the auxiliary data and meta-data derived from distillation and synthetic functionality (see T2.3). As the repository will expose an API through which all data and model manipulation tools developed in MANOLO will be able to register the operations applied, metadata is automatically maintained through the usage of the tools without relying on explicit metadata-related actions by the users.

This task will also apply the methods developed in T2.2 and T2.3 to populate the repository with the data and models needed for the pilots.

2.2 Data tier main concepts

The MANOLO suite's data tier serves as a foundational component designed to meet the diverse requirements of the project. Its core is built around a flexible and scalable architecture that efficiently manages data storage, semantic relations and metadata. This design empowers seamless interaction with other components of the suite and ensures robust functionality for all user and application needs. At its essence, the data tier treats all entities as objects, adhering to an object persistent framework (OPF) that simplifies data management and interaction.

Objects within the data tier represent artifacts of the MANOLO project and their organization is facilitated through data structures. These data structures act as containers for multiple objects, enabling logical grouping and efficient manipulation of data. Every object in the system is assigned a globally unique identifier, known as the **Object ID (OID)**, which serves as its primary reference for access and management. Similarly, each data structure is uniquely identified by an incremental number called the **Data Structure Number (DSN)**, which distinguishes it from other structures in the system.

The API provided by the MANOLO suite leverages FastAPI, a modern framework for building RESTful APIs with Python, to deliver its services efficiently. This API facilitates seamless interaction with the MANOLO suite from any device in the cloud-edge continuum through the HTTP protocol. It enables the use of AI models and data as a service from various devices and is used by the MANOLO Python library, where it is integrated to execute specific tasks remotely, including benchmarking and monitoring processes. The Python implementation interacts with the APIs provided by the MANOLO suite, utilizing the requests library to communicate with the existing .NET APIs. The design of the API to offer generic capabilities of the API and its implementation with

a simple set of operations, together with comprehensive documentation, ensure that it is easy to integrate, adaptable and expandable.

The data tier is built on top of a relational database, but it can adapt and encapsulate versatile data stores, which can be relational, object-based, or vector-based; it also supports direct use of the file system as a data store for large files. Artifacts such as datasets and samples are stored as objects, with metadata like class annotations and metric values attached or semantically linked to their respective data objects. Additionally, the data tier supports user authentication, access control, logging of operations, and encryption for object data and metadata. The API hides low-level query language to the underlying data stores and file system operations done by the operating system, exposing a set of primitive methods that are consumed by higher-level Python framework wrapper objects. Artifacts are treated as item objects that belong to data structure objects, enabling the handling of arrays, lists, trees, graphs, and dictionaries. Each object includes its unique identifier, associated data, key-value metadata properties, and a list of relations with other objects. These relations are represented as semantic triplets subject (this), predicate, object (other), and offers the capability of defining custom predicates.

Data within the system is organized into containers, such as datasets containing samples, where each sample may include data points or vector values. Groups of samples, such as mini-batches, can be defined without duplicating data, using their IDs as part of semantic relations. The data tier dynamically selects the appropriate data store based on the kind of the data provided to the API methods, whether generic, vector, image tensor, or time-series signal.

The MANOLO data tier also supports model registries, where models and their associated artifacts are organized into collection data structures. These artifacts include hyperparameter configurations, model parameter values, training states, benchmarking metadata, and evaluation metrics. This design enables full reproduction of a model's training context, pausing and resuming training as needed, and facilitates loading and evaluating trained models. Additionally, relationships between models (e.g., model B derived from model A) and between models and datasets (e.g., models A and C trained on dataset D) are maintained using semantic relationships. The framework organizes multiple experiments under the same context, allowing for variations in training hyperparameters, model architectures, or datasets while maintaining consistent tracking and organization.

2.3 Data artifacts storage

The MANOLO data tier is designed to robustly host and organize diverse data artifacts in a hierarchical manner, enabling seamless management of datasets and their components. At its core, the data tier treats all data-related artifacts as objects that have a unique Object ID (OID). Datasets are at the root of the hierarchy, composed of lists of samples that can be categorized into subsets, such as training, validation, and testing sets.

Each subset is also represented as a list of samples, ensuring that the structure remains consistent and accessible. To optimize storage efficiency, the data tier employs a relational approach by leveraging the unique identifiers of DSN objects or their items. This allows shared samples across subsets to be referenced through their IDs rather than duplicating their data. By maintaining semantic relationships between objects in the form of subject-predicate-object triplets, the system ensures a clear, structured, and non-redundant organization of data.

Metadata plays a critical role in the MANOLO data tier, enriching the stored objects with context-related information. Each object is has an associative array for metadata, where its key-value pairs can include properties like class annotations, metrics, or provenance information. This feature enhances the traceability and usability of artifacts, making them more informative and adaptable to various applications required by the tasks of MANOLO. By attaching metadata directly to objects, the system facilitates efficient documentation of artifacts and retrieval of relevant information.

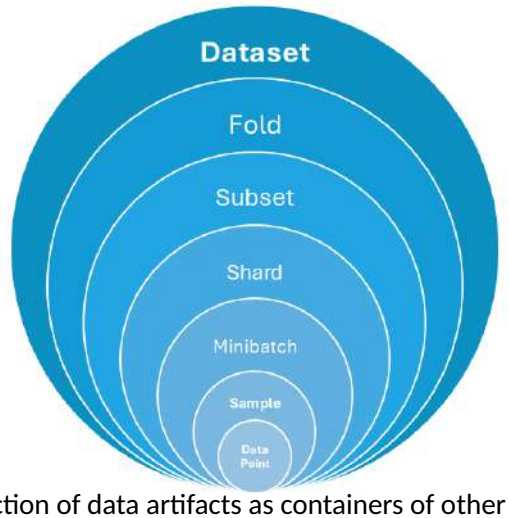


Figure 1: Depiction of data artifacts as containers of other data artifacts.

The "onion" depiction of data artifacts encapsulates the hierarchical nature of data management within the MANOLO suite. At the core, there are data points, which aggregate into samples. These samples, can belong to mini-batches or shards of multiple mini-batches. The container for those are subsets for training, validation, and testing, which collectively constitute a fold of the dataset for k-Fold cross validation experiments. This layered approach covers any organization of data artifacts, emphasizing the generic approach and scalability of the data tier.

The MANOLO data tier supports a wide range of artifact types, addressing the diverse needs of modern applications. Object/Record Sets represent collections of objects or records with fields that can vary in data type, such as numeric, string, date, or nested objects. Vector Sets consist of fixed-dimensional numeric vectors with uniform data types and ranges, suitable for structured numerical data. Image Sets manage tensors representing grayscale, RGB, or RGB-D images. Point Cloud Sets handle spatial data in three dimensions, where samples are sets of points in a 3D coordinate system. Video Sets accommodate sequences of images over time, capturing dynamic visual data. Time-Series Sets store sensor recordings over time, representing scalar or vector values at discrete timestamps. Graph Datasets encode relationships through vertices and edges, represented in adjacency lists or matrices, with associated values. Spatiotemporal Graph Datasets extend this by incorporating temporal dimensions, allowing each vertex and each edge to have a time series of its values.

2.3.1 Experiment artifacts and model registries

The MANOLO data tier extends its architecture to facilitate the hosting and management of machine learning experiments, encompassing models and associated artifacts. It offers the capability to seamlessly organize and connect various components of an ML experiment, beginning with the model itself and hyperparameters governing its architecture, the data handling and training processes. Each ML experiment is linked to a specific training set from a dataset, ensuring traceability and reproducibility of the training context. The data tier can store the final state of a trained model but also intermediate states during training, also known as checkpoints, including the model parameter state and the training process variable values. These artifacts enable pausing and resuming training sessions or evaluating different stages in a model's training process.

The relational structure of the MANOLO data tier ensures that all these components are interconnected. Models, training sets, and other associated artifacts are represented as objects with unique Object IDs (OIDs), while their relationships are captured using semantic triplets. This approach allows for defining the dependencies and associations between a model, its training data, and the parameters of its training process. Key-value pairs stored in metadata arrays further enrich these objects, providing additional context such as the provenance of the data, hyperparameter configurations, evaluation metrics, and any other relevant details.

The system accommodates various model-related artifacts, each with distinct roles. Model state objects (MS) represent the complete set of parameters and structural definitions for a pre-trained model, enabling efficient storage and retrieval of ready-to-use models. Training state objects (ST) capture the entire set of parameters necessary to resume a paused training session, which may include both the model and additional states related to the training algorithm. Model class objects (MC) define collections of related models, that are either generated during a hyperparameter search, or sharing common characteristics such as common architecture or dataset configurations. Finally, surrogate models (SM) are supported as explainable representations of non-explainable models, providing interpretability while maintaining the original model's predictive capabilities.

Through this architecture, the MANOLO data tier ensures that machine learning experiments are not only efficiently stored but also fully contextualized within a relational framework. By leveraging metadata, semantic relationships, and hierarchical object structures, the system supports complex ML workflows, enabling reproducibility, scalability, and adaptability without delving into the specific implementation details of individual methods.

2.4 General purpose artifacts

The data tier in MANOLO is designed to handle a wide variety of file types, accommodating diverse artifacts. These include plain text files such as unstructured text (.TXT) and tabular text formats (.CSV, .XLSX, .ODS), document files, and generic binary files like custom binary formats (.BIN) and compressed binary formats (.ZIP, .TAR). It also supports generic multimedia files, including original and compressed images, raw and encoded video formats. Additionally, the system manages raw and encoded signal data, as well as serialized objects in textual formats (.XML, .JSON) and binary serialized objects (e.g., Pickle, Protobuf). This generic approach allows MANOLO's data tier to handle both structured and unstructured data efficiently across various file types and formats.

2.4.1 Metadata and semantic relations between artifacts

Metadata can be attached to artifacts in a flexible way to support various project needs, such as recording metrics after evaluating experiments, or storing series of values from monitoring processes. Each object hosts an associative array to hold metadata in the form of key-value pairs, where the value may contain an objec with multiple fields (e.g., hyperparameter name-value pairs). Metadata can also be structured as a tree, allowing annotations to follow a hierarchical structure. Moreover, they can be organized in any kind of graph, utilizing an adjacency list or matrix for representing complex relationships.

Additionally, semantic triples in the form of subject-predicate-object, are used to define logical relations between artifacts, such as "dataset A is reduced into dataset B" or "model B is an alternative to model A," providing semantic meaning to the connections between different entities. For tasks requiring relational data, the metadata can be structured as relational rows, where each row corresponds to a set of related values. This flexibility in metadata structure allows MANOLO to support a wide range of tasks, from data reduction and model comparison to more complex monitoring and evaluation processes.

Furthermore, the data tier in MANOLO supports basic querying on these semantic relations, enabling users to easily retrieve useful information to facilitate project tasks. For example, users can query to "list all alternatives of model A" or "list all derived datasets of dataset A," making it easier to explore the relationships between artifacts and enabling more efficient project management and decision-making.

2.5 Manolo data tier web service

The MANOLO data tier is a robust and scalable web service built using .NET Core, designed to efficiently handle and store various types of artifacts and metadata. It operates as a **FastAPI** web service, enabling seamless communication with other systems through standard HTTP methods (GET, POST, PUT, DELETE). This service is deployed within a Docker container, ensuring ease of deployment, scalability, and isolation. It can be deployed

on the cloud for shared access or inside the local network edge for isolated use.

The core data management suite of the system is PostgreSQL, an SQL-compatible database that also supports NoSQL features through JSON and JSONB data types. PostgreSQL allows the creation of tables to store structured data, with each data structure (e.g., experiments, models, datasets) having its own dedicated table. For enhanced performance, PostgreSQL is configured with Generalized Inverted Indexes (GIN) to index JSON and JSONB columns, enabling fast and efficient querying of non-relational data. It also supports full-text search capabilities, crucial for querying metadata stored in JSON/JSONB format. For file storage, MANOLO utilizes the standard Linux file system (ext4) within the Docker container. Files are organized according to the Linux directory structure, ensuring compatibility and ease of access. This system is well-suited for storing large artifacts such as models, datasets, and experiment logs.

Each artifact or metadata entry in MANOLO is associated with a unique identifier, typically a **Unique Lexicographically Sortable Identifier (ULID)** ensuring global uniqueness across distributed systems. These ULIDs are used to reference and access individual records across the data tier. The system handles a wide range of data types, including structured data (e.g., tabular data, hyperparameters) stored in PostgreSQL tables and unstructured or semi-structured data (e.g., JSON/JSONB, raw files) stored in both PostgreSQL and NoSQL databases. Object of large volume, like model states or large-scale datasets, can also be stored in an auxiliary tensor store database to leverage high-performance operations and efficient storage schemes.

In the MANOLO system, each artifact type has a specific table in the database. For every data structure entry, a corresponding table named Items is created within the PostgreSQL database. Each record in the table contains a unique identifier implemented using ULIDs, a data field capable of storing variable-sized content in byte format, a field that specifies the type of data stored, and a timestamp that tracks the last modification of the entry. This design provides a generic approach to storing any artifact within the MANOLO framework.

The MANOLO data tier employs robust security measures to safeguard the confidentiality, integrity, and proper access control of artifacts. User authentication is managed through cookie-based authentication, which is mandatory for accessing or interacting with system endpoints. Each endpoint is governed by a designated authorization policy, with access determined by the cookie issued to the user during the sign-in process. This mechanism allows for the definition of access attribute flags tailored to each endpoint's requirements.

The system currently supports three authentication policies:

- **All**, which grants access to endpoints open to all users;
- **ModeratorOrHigher**, which permits access to most endpoints; and
- **AdminOnly**, which provides access to all endpoints.

These access logs allow for detailed tracking and auditing of user activity, helping detect unauthorized access and ensure compliance with security policies.

For data protection, MANOLO supports both encryption during transfer and storage. Data in transit is protected using transport layer encryption, such as TLS 1.3, to ensure confidentiality and integrity while being transmitted. For storage encryption, artifact objects that are not actively queried or are infrequently accessed could be encrypted when stored using symmetric end-to-end encryption, such as AES-256. Importantly, the symmetric encryption uses the end user to manage their encryption key, ensuring they maintain full control over access to sensitive data. This encryption process ensures that only authorized users with the correct decryption keys can access sensitive artifact content. Overall, the combination of authentication, access control, access logging, and encryption provides a strong security framework to safeguard data throughout the lifecycle of the project.

2.6 Data tier web API reference

The Data Tier Web API Reference provides an overview of the key endpoints used for managing data within the system. It covers operations for creating, retrieving, updating, deleting and restoring various entities such as data structures, items, predicates, relations, aliases and key-value pairs. Additionally, it includes authentication and management functionalities, such as user login and database backup and restore. Each section outlines available endpoints, their purpose, and the parameters required, offering a clear and concise framework for interacting with the API. This reference is designed to help efficiently utilize the system's capabilities while ensuring consistency and reliability in data management.

2.6.1 Data Structures

Method Name	Description
GetDataStructures	Summary: Handles the request to retrieve a list of data structures from the database. Returns: A Result object containing a list of data structure IDs if any exist, or a failure result with a domain error if no data structures are found.
CreateDataStructure	Summary: Handles the creation of a new data structure. Parameters: - Request: The CreateDataStructureQuery containing the details of the data structure to be created. Returns: A Task that represents the asynchronous operation. The task result contains a Result object. If successful, the Result contains the ID of the newly created data structure. If a data structure with the same name or DSN already exists, the Result indicates a failure with an appropriate error message.
DeleteDataStructure	Summary: Handles the deletion of a DataStructure based on the provided request. Parameters: - Request: The request containing the DataStructures ID, DSN, or Name to be deleted. Returns: A Result object indicating the success or failure of the deletion operation, along with any relevant error messages.
GetDataStructure	Summary: Summary: Handles the request to get a specific data structure by its ID. Parameters: - Request: The request containing the ID of the data structure to retrieve. Returns: A Result object containing the retrieved data structure if it exists, or a failure result with a domain error if the data structure does not exist.
RestoreDataStructure	Summary: Handles the request to restore a data structure. Parameters: - Request: The request containing the data structures ID, DSN, or name to be restored. Returns: A Result object indicating the success or failure of the restore operation, along with a message.
UpdateDataStructure	Summary: Handles the update of a data structure. Parameters: - Request: The request containing the data structures id and updated properties. Returns: A Result object indicating the success or failure of the operation, along with any relevant error messages.

2.6.2 Items

Method Name	Description
CreateItem	Summary: Handles the creation of a new item based on the provided request. Parameters: - Request: The create item query containing the necessary data for item creation. Returns: A task that represents the asynchronous operation. The result of the task is a Result object indicating success or failure, along with the new items ID.
DeleteItem	Summary: Handles the deletion of an item based on the provided request. Parameters: - Request: The delete item query containing the DSN and item ID. Returns: A Result indicating the success or failure of the deletion operation.
DownloadItemData	Summary: Handles the download of item data based on the provided DSN and item ID. Parameters: - Request: The download item data query containing the DSN and item ID. Returns: An IActionResult representing the downloaded file or a NotFoundResult if the item or data structures are not found.
GetItem	Summary: Handles the GetItemQuery request by retrieving an item from the database based on the provided DSN and item ID. Parameters: - Request: The GetItemQuery request containing the DSN and item ID. Returns: A Result object containing the retrieved item if successful, or an error message if the item does not exist or an error occurs.
GetItemData	Summary: Handles the GetItemDataQuery request to retrieve item data based on the provided DSN and item ID. Parameters: - Request: The GetItemDataQuery request containing the DSN and item ID. Returns: A Result object indicating the success or failure of the operation. If successful, the Result contains the decoded item data as a string. If unsuccessful, the Result contains a DomainError indicating the specific error.
GetItems	Summary: Handles the GetItemsQuery request to retrieve a list of items based on the provided DSN. Parameters: - Request: The GetItemsQuery request containing the DSN. Returns: A Result object indicating the success or failure of the operation. If successful, the Result contains a list of item IDs. If unsuccessful, the Result contains a DomainError indicating the reason for the failure.
RestoreItem	Summary: Handles the restore operation for a specific item in the database. Parameters: - Request: The restore item query containing the DSN and item ID. Returns: A Result indicating the success or failure of the restore operation.
UpdateItem	Summary: Handles the update of an item in the database. Parameters: - Request: The update item query containing the necessary parameters. Returns: A Result indicating the success or failure of the operation.

2.6.3 Predicates

Method Name	Description
CreatePredicate	Summary: Handles the creation of a new predicate. Parameters: - Request: The request containing the description of the predicate to be created. Returns: A Result representing the asynchronous operation. The task result contains the unique identifier of the newly created predicate if successful, or an error message if the predicate already exists.
DeletePredicate	Summary: Handles the deletion of a predicate based on the provided description. Parameters: - Request: The delete predicate query contains the description of the predicate to be deleted. Returns: A task that represents the asynchronous operation. The task result is a Result indicating the success or failure of the operation. If the predicate with the given description does not exist, the result will be a failure with a corresponding error.
GetObjectsOfPredicate	Summary: Handles the request to get objects related to a specific predicate. Parameters: - Request: The request containing the predicate description. Returns: A Result object indicating success or failure. If successful, the Result contains a list of objects related to the predicate. If unsuccessful, the Result contains a DomainError indicating the reason for failure.
GetPredicates	Summary: Handles the GetPredicatesQuery request to retrieve a list of predicates from the database. Parameters: - Request: The GetPredicatesQuery request object containing any necessary parameters. Returns: A Result object indicating the success or failure of the operation. If successful, the Result contains a list of existing predicates. If unsuccessful, the Result contains a DomainError indicating the reason for the failure.
GetSubjectsOfPredicate	Summary: Handles the request to retrieve subjects related to a specific predicate. Parameters: - Request: The request containing the predicate description. Returns: A Result representing the asynchronous operation. The result contains either a list of subjects related to the predicate or an error message.
GetSubjectsOf	Summary: Handles the GetSubjectsOfQuery request to retrieve the subjects related to a specific object. Parameters: - Request: The GetSubjectsOfQuery request containing the object and predicate description. Returns: A Result object indicating the success or failure of the operation. If successful, the Result contains a list of related subjects. If unsuccessful, the Result contains a DomainError indicating the reason for the failure.
GetObjectsOf	Summary: Handles the GetObjectsOfQuery request to retrieve objects related to a given subject. Parameters: - Request: The GetObjectsOfQuery request containing the subject and description. Returns: A Result object indicating success or failure. If successful, the Result contains a list of related objects. If failure, the Result contains a DomainError indicating the reason for the failure.

2.6.4 Relations

Method Name	Description
CreateRelation	Summary: Handles the creation of a new relation between two entities (subject and object) based on the given predicate and DSN. Parameters: - Request: The create relation query containing the necessary parameters. Returns: A task that represents the asynchronous operation. The result is an indication of success or failure. If successful, the result is Result.Success; otherwise, it is Result.Failure with an appropriate error.
DeleteRelation	Summary: Handles the deletion of a relation between two entities in the Manolo-DataTier system. Parameters: - Request: The delete relation query containing the details of the relation to be deleted. Returns: A Result indicating the success or failure of the deletion operation.
GetRelations	Summary: Handles the GetRelationsQuery request to retrieve a list of relations from the database. Parameters: - Request: The GetRelationsQuery request containing any necessary parameters. Returns: A Result object containing either a list of relations or an error message. If the list of relations is empty, it returns a failure Result with a NoRelations domain error. Otherwise, it returns a success Result with the list of relations.
AddChild	Adds a child item object under a parent item object in a tree data structure. Parameters: - DSN: data structure number. - Parent: parent object ID - Child: child object ID
GetChildren	Return a list with the object IDs for the children of the given parent item in a tree data structure. Parameters: - DSN: data structure number. - Parent: parent object ID
AddEdge	Add an edge between two node item objects in a graph data structure. Parameters: - DSN: data structure number. - node1: first (source) node object ID - node2: second (destination) node object ID - value (optional): the value for the edge - is_directed (optional): — 0 (default) undirected edge — 1: directed edge.
GetAdjacencyList	Return a list with the object IDs for the adjacent nodes to the given node item in a graph data structure. Parameters: - DSN: data structure number. - node: node object ID.

2.6.5 Global Aliases

Method Name	Description
CreateUpdateAlias	Summary: Handles the creation or update of an alias. Parameters: - Request: The CreateUpdateAliasQuery containing the alias details to be created or updated. Returns: A Task that represents the asynchronous operation, containing a Result object. The Result is successful if the alias was created or updated, or a failure if an alias with the same name already exists.
DeleteAlias	Summary: Handles the deletion of an alias based on the provided query. Parameters: - Request: The DeleteAliasQuery containing the alias name to be deleted. Returns: A Result object indicating the outcome of the deletion operation. It would be a Success result if the alias was successfully deleted, or a Failure result with an AliasDoesNotExist error if no alias was found.
GetAlias	Summary: Handles the GetAliasQuery request by retrieving an alias from the database. Parameters: - Request: The GetAliasQuery containing the ID of the alias to retrieve. Returns: A Result object containing: - Success with the alias name if the alias is found. - Failure with a DomainError if the alias is not found.
GetId	Summary: Handles the GetIdQuery request by retrieving the ID of an alias from the database. Parameters: - Request: The GetIdQuery object containing the alias name to search for. Returns: A Result object containing either: - A successful result with the ID of the found alias, or - A failure results in an AliasDoesNotExist error if the alias is not found.

2.6.6 Key-Value Pairs

Method Name	Description
CreateUpdateKeyValue	Summary: Handles the creation or update of a Key-Value pair in the database. Parameters: - Request: The request containing the Key-Value data. Returns: A Result indicating the success or failure of the operation.
DeleteKeyValue	Summary: Handles the deletion of a key-value pair from the database. Parameters: - Request: The delete key-value query containing the key to be deleted. Returns: A Result representing the asynchronous operation. The result is an indication of success or failure. If the key does not exist in the database, the result will be a failure with a corresponding error.
GetKeys	Summary: Handles the GetKeysQuery request to retrieve all keys associated with a specific object. Parameters: - Request: The GetKeysQuery request containing the object for which keys are requested. Returns: A Result object indicating the success or failure of the operation. If successful, the Result contains a list of keys associated with the object. If unsuccessful, the Result contains a DomainError indicating the reason for the failure.
GetValue	Summary: Handles the logic for retrieving a value associated with a given key from the database. Parameters: - Request: The request containing the key to retrieve. Returns: A Result representing the asynchronous operation. The result is a Result object containing the retrieved value if the key exists, or an error message if the key does not exist.

2.6.7 Authentication

Method Name	Description
AddUser	Summary: Handles the addition of a new user to the database. Parameters: - Request: The AddUserQuery containing the new user's information. Returns: A Task that represents the asynchronous operation, containing a Result object. The Result is successful if the user was added, or a failure with an error message if the username already exists.
Login	Summary: Handles the login process for a user. Parameters: - Request: The login query containing the username and password. Returns: A Result object indicating the outcome of the login attempt. Returns a success result if the login is successful, or a failure result if the user does not exist or the password is incorrect.
Logout	Summary: Handles the logout process for the current user. Returns: A Result object indicating the success of the logout operation.

2.6.8 Management

Method Name	Description
Backup	Summary: Handles the backup request by retrieving data from the database and storing it in a JSON file. Returns: A Result indicating the success or failure of the backup operation.
Restore	Summary: Handles the restore operation by processing the backup data and restoring it to the database. Parameters: - Request: The restore query containing the backup data. Returns: A task that represents the asynchronous operation. The result is an indication of success or failure.

3 Data Quality Estimation

This section addresses the challenge of estimating the quality of time-series data and introduces a task-agnostic framework for anomaly detection. First, we define the problem and point out the limitations of current methods. Next, we explain our proposed approach, which includes both machine learning and statistical techniques for estimating noise. Finally, we evaluate our framework on the Bitbrain dataset, comparing our results against Bitbrain's own noise detection method, as well as MNE Filtering.

3.1 Problem Statement

The quality of time-series data plays a crucial role in the performance of machine learning models, yet existing methods often fail to effectively identify and handle noise in a task-agnostic manner. Traditional data quality estimation techniques are typically tailored to specific datasets, limiting their applicability to a wider range of applications.

To address these challenges, we propose a task-agnostic framework for time-series data quality estimation. Towards this goal, we propose methods for anomaly and noise detection, which allow us to identify biased, noisy, inconsistent, or otherwise low-quality data, as well as data that may have been maliciously manipulated to contaminate the model during training. Our framework is task-agnostic, meaning it can be applied across different time-series datasets, regardless of the specific task or domain.

As a case study to validate our methods, we use the Bitbrain dataset, which consists of EEG signals collected from a headband across 128 recordings. This headband includes two EEG channels synchronized with medical-grade EEG devices. The signals are annotated by three experts, with each 30-second segment classified into one of five sleep stages: Wake, N1, N2, N3, and REM. These stages correspond to specific brain activity patterns, such as slow eye movements, sleep spindles, and other characteristic waveforms.

3.2 Proposed Methodology

We propose a task-agnostic framework for time-series data quality estimation, using a combination of machine learning-based and statistical approaches to detect noise and anomalies in data. The techniques used in this framework estimate noise via:

(a) **Reconstruction error:** unsupervised machine learning using autoencoders, which compress data into a lower-dimensional representation and then reconstruct the original input.
(b) **Prediction error:** unsupervised machine learning with transformers, which predict the next sequence of data points.
(c) **Attention matrix:** supervised and unsupervised machine learning with transformers and autoencoders, which perform tasks such as classification, reconstruction and prediction.
(d) **Statistical function:** statistical methods that detect anomalies by analyzing changes in data characteristics over time.

To apply these techniques, we implement the following 10 methods: LSTM Autoencoder, Convolutional LSTM Autoencoder, Attention-based Autoencoder, Transformer Classifier, Transformer Predictor, Cumulative Sum Control, Page-Hinkley Test, Kullback-Leibler Divergence, Principal Component Analysis and Adaptive Windowing. The last 5 statistical methods are implemented using the Frouros framework. Table 9 summarizes the categorization of each method in the framework across three dimensions: (1) whether the approach is supervised or unsupervised, (2) whether it falls under machine learning or statistical techniques, and (3) how noise is estimated (i.e., reconstruction error, prediction error, attention matrix, or statistical function).

In our analysis, we chose to estimate noise across 30-second segments of the EEG data, which, in the Bitbrain dataset, corresponds to 7,680 samples. Since each method outputs noise values in different scales, we trans-

Table 9: Categorization of methods based on noise estimation approach, learning type, and technique.

Approach	Machine Learning		Statistical	
	Unsupervised	Supervised	Unsupervised	Supervised
Reconstruction Error	LSTM-AE Conv-LSTM-AE Attn-AE	—	—	—
Prediction Error	Predictor	—	—	—
Attention Matrix	Attn-AE Predictor	Classifier	—	—
Statistical Function	—	—	CUSUM PH-Test KL-Divergence PCA ADWIN	—

form these values into binary based on predefined thresholds per channel to establish a common ground for comparison. To define these thresholds, we need to have a sense of how much noise is typically present in the dataset. For example, in the Bitbrain dataset, which contains brain signals during sleep, we don't expect much noise—typically around 99% of the data is clean.

To evaluate our noise estimations, we use 2 ground truth methods designed for task-specific noise estimation in EEG signals: MNE Filtering and the original Bitbrain estimation method. MNE Filtering is implemented using the MNE library, which provides tools for filtering and analyzing neurophysiological data. We treat Bitbrain's method as a black-box approach, to ensure that our task-agnostic estimations remain unbiased by the specifics of its implementation. Similarly, when using the MNE library, we treat it strictly as a tool, integrating it into our code without really accessing its internal methodology.

3.3 Machine Learning Approach

The general concept of using machine learning (ML) for noise estimation revolves around training models to solve specific tasks and then using their outputs during inference to estimate noise. These tasks may vary in nature — ranging from signal reconstruction and prediction to classification — but the core idea is that the model's performance on these tasks can be used to infer the presence of noise.

To train the models, we split the dataset into training (43 recordings), validation (3 recordings), and testing subsets (10 recordings). Noise estimation is performed using the test dataset. For the training configuration, we set the batch size to 512 and train the models for a maximum of 1000 epochs. To prevent overfitting, we employ early stopping with a patience of 30 epochs, meaning that if the validation loss does not improve for 30 consecutive epochs, training will stop. We set the learning rate to $1e^{-4}$ and use the Adam optimizer to adjust the model weights. Additionally, we implement a *ReduceLROnPlateau* scheduler to dynamically adjust the learning rate based on the validation loss. If the validation loss plateaus, the scheduler reduces the learning rate to help the model continue improving.

To ensure robust model training and reliable outcomes, we rely on data preprocessing techniques. Proper preprocessing standardizes the input data, mitigates inconsistencies, and enhances the model's ability to learn

meaningful representations. In our case, preprocessing involves handling missing or ambiguous data by removing rows containing NaN values, as well as any samples where expert annotators could not agree on the class label (i.e., samples with a label value of 8). Additionally, we apply normalization to assist model training by making the data more uniform. Due to the low variability and the presence of extreme outliers in the EEG signals, we adopt a robust normalization approach. Instead of using the mean and standard deviation, which are sensitive to outliers, we use the median and interquartile range (IQR). We scale the data based on statistics computed across the entire dataset, rather than on a per-batch basis, since the statistics remain consistent across the full dataset.

$$x_{\text{norm}} = \frac{x - \text{median}(x)}{\text{IQR}(x)} \tag{1}$$

where x represents the values of a particular feature, median(x) is the median value of the feature, and IQR(x) is the interquartile range, which measures the spread of the middle 50% of the data. We use the median as a measure of central tendency because it is robust against extreme values, making it a more reliable indicator for our dataset, which exhibits low variability and where even small differences matter. Additionally, we use the interquartile range IQR $= Q_3 - Q_1$ to reduce the influence of outliers by focusing on the range within which the central portion of the data lies. This approach ensures that our normalization process allows the model to learn from the true patterns in the dataset.

To process the input data, we split each 30-second segment of EEG data (7,680 samples) into 32 smaller chunks, resulting in 240 sequences per segment. Each sequence has a length of 240 time steps, with 2 features corresponding to the EEG channels. This structure allows the model to focus on both the short- and long-term temporal dependencies within the data. Additionally, we notice that our dataset tends to overfit with more complex models due to its low variability and extreme outliers. To mitigate this, we use simple architectures with only a few layers, e.g. a single attention layer in the attention-based models. This approach avoids unnecessary complexity, ensuring more stable training and better generalization on the data.

For all machine learning methods, we need to establish a common standard to determine what is considered noise and what is not. As the models output different value ranges, we need a policy to classify these outputs into binary values that indicate noise presence or absence. One such approach involves defining thresholds that act as a boundary between noise and clean data. For these methods, we define thresholds based on the 99th percentile of noise values for each channel. Values above the thresholds are set to 1 (indicating noise), while those below are set to 0 (indicating no noise).

Noise estimation within this machine learning approach can be divided into two categories: label-based and label-free:

- The label-based approach involves training classifier models on labeled data to learn how to assign class labels to signals. The *Transformer Classifier*, trained to classify signals into sleep stages, belongs to this category. The idea is to use the model's attention matrix, which highlights the importance of different time steps in data, to infer the presence of noise. Higher attention values correspond to segments of data that are likely noisy, whereas lower attention values indicate cleaner segments of data.
- The label-free approach does not rely on labeled data for training. Instead, it focuses on unsupervised techniques to estimate noise based on the characteristics of the signal itself. In this category, we use models like autoencoders and transformers that learn to reconstruct and predict signals respectively. The three *Autoencoders* focus on reconstructing the input data, and noise is identified through the reconstruction error — the larger the error, the more likely the segment is noisy. On the other hand, the *Transformer Predictor* learns to predict the next sequence of data points, and noise is identified through the prediction error. Additionally, both the *Attention-based Autoencoder* and *Transformer Predictor* can

estimate noise using their attention matrices. In these models, higher attention values indicate segments of data that are more likely to be noisy, while lower attention values correspond to cleaner segments.

3.3.1 LSTM Autoencoder

We implement an *LSTM-based Autoencoder* that uses Long Short-Term Memory (LSTM) layers to reconstruct EEG signals from a lower-dimensional latent representation. The key idea behind this autoencoding approach is to compress the input signal while preserving the essential features, then reconstruct it from the compressed representation. The LSTM layers in the encoder are designed to capture the temporal patterns and dependencies inherent in the EEG signals. These temporal dependencies are crucial for accurately modeling EEG data, which often exhibits long-range correlations over time. The decoder, in turn, reconstructs the original signal from this latent representation, hoping that only non-noisy information is preserved during compression.

The encoder processes each sequence in the data and compresses it into a latent representation of size (1, 8), where the first dimension represents a single compressed time step, and the second dimension represents 8 learned features. The decoder then takes this compressed representation and attempts to reconstruct the original signal, capturing both trends and amplitudes.

To balance the models' focus on both the amplitude and trends of the EEG signals, we define a custom loss function, called *BlendedLoss*. This function combines the median and mean of the powered absolute differences between the predicted ($\hat{x}$) and target values (x):

$$\text{Loss} = (1 - \text{blend}) \cdot \text{median}(|\hat{x} - x|^p) + \text{blend} \cdot \text{mean}(|\hat{x} - x|^p) \tag{2}$$

where p is the power parameter that controls the sensitivity of the loss to the differences, x is the original signal, and $\hat{x}$ is the reconstructed signal. The blend factor controls the trade-off between learning the overall trends (via the median) and capturing the amplitude (via the mean). We experiment with different blend values (0.1 and 0.8) to observe how this affects the model's performance in reconstructing the signals.

The *LSTM Autoencoder* estimates noise by measuring the reconstruction error between the original and the reconstructed signal. During training, the model learns to compress and then reconstruct the input signal with minimal error. The goal is for the model to capture the common patterns and trends in the data, such as periodic fluctuations or typical signal behavior. When the model encounters unusual spikes, often caused by noise, it struggles to reconstruct them because they don't fit the typical patterns it has learned. As a result, the reconstruction error for these segments will be larger, indicating noise.

3.3.2 Convolutional LSTM Autoencoder

We implement a *ConvLSTM-based Autoencoder* that combines convolutional and LSTM layers to reconstruct EEG signals from a lower-dimensional latent representation. The convolutional layers capture local features in the data, while the LSTM layers capture long-range temporal dependencies. This hybrid architecture is designed to effectively process spatiotemporal data like EEG signals, which contain both spatial (local) and temporal (long-range) patterns.

We use the same *BlendedLoss* function, as defined in Equation 2, to balance the reconstruction of amplitude and trends. Similar to the LSTM approach, noise is estimated through the reconstruction error.

3.3.3 Attention-based Autoencoder

We implement an *Attention-based Autoencoder* that combines convolutional layers with a multi-head attention mechanism to reconstruct EEG signals from a lower-dimensional latent representation. The convolutional layers help extract spatial features from the input data, while the attention mechanism is particularly effective in modeling temporal dependencies. This allows the model to focus on important time steps and capture long-term patterns, without the need for recurrent layers like LSTM. The key innovation of this hybrid model lies

in the use of multi-head attention, which enables the model to focus on different parts of the input sequence simultaneously. The attention mechanism computes a weighted sum of the input features, with the weights learned during training. This allows the model to prioritize certain features and temporal patterns over others, making it particularly effective for detecting noise or anomalies that deviate from typical signal behavior.

We use the same *BlendedLoss* function, as defined in Equation 2, to balance the reconstruction of amplitude and trends. Regarding noise estimation, we adopt two strategies:

- The first strategy defines noise as the reconstruction error, similar to the other two autoencoders.
- The second strategy defines noise based on the attention weights in the attention matrix. Attention mechanisms allow the model to assign varying levels of importance (i.e., attention weights) to different parts of the input sequence. The key idea is that the attention mechanism can distinguish between relevant and irrelevant time steps by analyzing how much focus the model places on them during the encoding and decoding process. When a time step receives a low attention weight, it suggests that the information at that moment is likely noisy and, therefore, unimportant for the reconstruction task.

3.3.4 Transformer Predictor

We implement a *Transformer Predictor* that uses multi-head attention for time-series forecasting. This model consists of a sequence-to-sequence architecture, with multi-head attention layers in both the encoder and decoder components. The encoder captures temporal dependencies in the input sequence, while the decoder predicts the next sequence based on these encoded features. The attention mechanism enables the model to focus on relevant time steps, improving its ability to capture long-term dependencies and trends.

To train the model, we use the *BlendedLoss* function, as defined in Equation 2, to balance the reconstruction of amplitude and trends.

The *Transformer Predictor* estimates noise based on two distinct strategies: prediction error and attention weights:

- The first strategy, prediction error, focuses on the difference between the model's predicted output and the actual observed values. When the model predicts the next time step in the sequence, a large difference between the predicted and actual values suggests that the input data may be noisy. This is because the Transformer model is designed to learn patterns and trends in the data, and noise, being irregular and unpredictable, disrupts this learning process. As a result, larger prediction errors typically correspond to noisy data, which the model struggles to predict accurately.
- The second strategy involves the attention weights, derived from the model's attention mechanism. Similar to the *Attention-based Autoencoder*, the *Transformer Predictor* uses attention to focus on different parts of the input sequence. Noisy segments make it harder for the model to detect consistent patterns and, as a result, tend to receive lower attention weights. On the other hand, time steps that align well with the learned patterns and trends in the sequence typically receive higher attention values.

3.3.5 Transformer Classifier

We implement a *Transformer Classifier* that uses a multi-head attention to classify sleep stages from EEG signals. The model architecture includes both an encoder and a decoder, with multi-head attention layers in each component to capture temporal dependencies and refine learned features across the input sequence. The encoder captures complex relationships in the input data, while the decoder further processes these features for classification. The attention mechanism enables the model to focus on key time steps in the sequence, helping it identify the most relevant temporal information for predicting sleep stages. Finally, a feedforward classifier layer produces the predicted sleep stages based on the features processed by the encoder and decoder.

To train the model, we use Cross-Entropy loss, which measures the dissimilarity between the predicted probability distribution and the true distribution. This loss function is commonly used for classification problems to penalize the model based on how confident and accurate its predictions are. Given that the dataset is quite imbalanced, with class frequencies reflected by the weights as shown in Table 10, we employ a weighted version of the Cross-Entropy loss (see: Equation 3). This approach adjusts the influence of each class in the loss calculation, helping the model learn across all class distributions, regardless of their prevalence in the dataset.

$$\text{Loss} = -\sum_{i=1}^{N} w_{y_i} \cdot \log(p_{y_i}) \tag{3}$$

where N is the total number of samples, y_i represents the true class label for the i-th sample, p_{y_i} is the predicted probability for the true class, and w_{y_i} is the weight assigned to the class y_i.

Table 10: Class frequencies and weights used in weighted cross-entropy loss.

Class	Freqs (%)	Weights
0	14.11	0.12
1	4.45	0.39
2	62.16	0.03
3	4.96	0.35
4	14.36	0.12

The *Transformer Classifier* estimates noise based on the attention weights in the attention matrix, similar to the approach used in the attention-based autoencoder. The attention mechanism in the Transformer model assigns varying levels of importance to different time steps in the input sequence. By analyzing the attention weights, we can identify time steps that the model considers noisy, as these time steps receive lower attention weights. The rationale behind this is that the classifier makes strong, confident predictions when the data is clean and well-defined. Noise, by nature, does not fit well into any particular class, and the model finds it harder to assign a clear class label to noisy samples.

3.4 Statistical Approach

In addition to machine learning-based methods, statistical techniques offer a robust alternative for detecting noise in time-series data and are widely used. These methods focus on analyzing changes in the statistical properties of the data over time, which can help identify deviations from expected patterns.

Unlike machine learning methods, no normalization is applied to the data, as statistical methods require the raw data to function properly. This ensures that the characteristics of the original signal are preserved, which is crucial for accurate anomaly detection.

In terms of noise thresholding, the Frouros-based methods inherently provide binary outputs based on internal thresholds. These thresholds are designed to classify data as noisy or clean. To handle cases where sample aggregation results in float values, we apply the same noise thresholding method—using the 99th percentile, as in the machine learning-based methods—to convert these results into binary form.

To ensure consistency across all methods and enable meaningful comparisons, the same test recordings used for machine learning-based noise estimation are also applied to the statistical methods.

3.4.1 Cumulative Sum Control Chart

Cumulative Sum Control Chart (CUSUM) is a sequential analysis technique commonly used in statistical quality control to monitor change detection [1] by identifying moments when the probability distribution of a stochas-

tic process or time series shifts. Unlike approaches that independently analyze measurements at specified times, CUSUM accumulates information from both current and past samples [2]. Thanks to this cumulative property, CUSUM proves to be more efficient [3] compared to simpler methods such as Shewhart charts [4]. CUSUM is represented through the following equation:

$$S_m = \sum_{i=1}^{m} (\tilde{x}_i - \hat{\mu}_0) \quad \text{or} \quad S'_m = \frac{1}{\sigma_{\tilde{x}}} \sum_{i=1}^{m} (\tilde{x}_i - \hat{\mu}_0), \tag{4}$$

where: m is the sample number, $\hat{\mu}_0$ is the estimated in-control mean, and $\sigma_{\tilde{x}}$ is the known or estimated standard deviation of the sample means.

In the context of MANOLO, CUSUM is applied due to its capability to detect abrupt changes. By continuously monitoring cumulative deviations from the baseline, CUSUM facilitates rapid detection of this kind of changes. These properties make CUSUM a strong candidate for noise detection tasks, where, according to the literature, it has shown highly promising results [5], [6], [7], [8].

3.4.2 Page-Hinkley Test

The Page-Hinkley Test is a sequential analysis technique used to detect abrupt changes [9] in the mean value of a signal or data stream over time, which may indicate noise or data drift. Additionally, the Page-Hinkley Test is used to monitor the performance of industrial processes [10] or assess the efficiency of learning models [11].

When applying the algorithm, an initial threshold value must be defined, which serves as a reference point. If a change in the mean exceeds this threshold, the change is considered significant. Furthermore, a decision function must be defined, which applies the Page-Hinkley technique by evaluating the data and returning 1 if a change is detected, and 0 if no change is detected.

The Page-Hinkley Test is beneficial for anomalies related to gradual shifts since it accumulates evidence over time, making it robust for detecting slower, cumulative shifts rather than just abrupt changes.

3.4.3 Kullback-Leibler Divergence

Kullback-Leibler Divergence (KLD), also known as relative entropy or I-divergence, quantifies the proximity of two probability distributions [12]. It measures, in bits, how close a probability distribution p is to a model distribution q [13], and is defined as:

$$D_{\mathrm{KL}}(p \parallel q) = \sum_i p_i \log_2 \left(\frac{p_i}{q_i} \right) \tag{5}$$

where p_i and q_i represent the probabilities of each outcome in the distributions p and q, respectively. The KLD is not symmetric, meaning $D_{\mathrm{KL}}(p \parallel q) \neq D_{\mathrm{KL}}(q \parallel p)$. The divergence $D_{\mathrm{KL}}(p \parallel q)$ is zero only when the distributions p and q are identical. In other words, this equation represents the average number of extra bits needed to encode samples from distribution p using an optimal code for distribution q.

To implement KLD, the data gets divided into consecutive windows. The probability distribution of each window is then estimated, and the KLD between successive windows is computed. If the resulting divergence exceeds a predefined threshold, drift is detected.

In general, KLD is widely used in statistics and pattern recognition, but it also finds applications in speech and image recognition [14].

3.4.4 Principal Component Analysis

Principal Component Analysis (PCA) is a linear dimensionality reduction technique. It analyzes a data table where observations are described by quantitative dependent inter-correlated variables. With PCA, the important information is extracted from the table and represented in a new set of orthogonal variables, called principal components [15]. The similarity patterns of the observations and the variables can then be depicted as points on maps [16].

PCA is frequently used when a large number of variables have a high degree of correlation with one another, and it is preferable to reduce them to an independent set. Specifically, the variable created as a linear combination of the original variables that accounts for the greatest amount of variance is the first principal component of a set of p variables. After the effect of the first component is eliminated, the second principal component explains the greatest amount of the variance in what remains, and the process can continue through p iterations until all of the variance is explained [17], [18].

3.4.5 Adaptive Windowing

Adaptive Windowing (ADWIN) is a technique that allows for handling concept drift and distribution changes when learning from data sequences that may change over time [19], [20].

ADWIN maintains a variable-length window of recent data points, ensuring that the data distribution remains stable. To detect any changes, this window is subdivided into two sub-windows (W_0 and W_1). To determine whether a change has occurred, ADWIN compares the averages of W_0 and W_1. If the equality of the distributions is no longer maintained, concept drift is detected, and W_0 is replaced by W_1, with a new W_1 initialized.

ADWIN uses a significance value $\delta \in (0, 1)$ to assess whether the two sub-windows correspond to the same distribution. If the absolute difference between the means of W_0 and W_1 exceeds a predefined threshold, an alarm is triggered.

ADWIN is particularly effective for detecting both sudden and gradual changes in data streams due to its dynamic adjustment of the window size. Additionally, ADWIN is well-suited for cases that often exhibit unpredictable shifts. Its ability to adaptively resize the window makes it ideal for handling such changes without relying on predefined parameters.

3.5 Evaluation and Results

To evaluate the effectiveness of the task-agnostic framework, we use two ground-truth methods, both designed for task-specific noise estimation in EEG signals: A) The original estimation method used by Bitbrain, the headband manufacturer B) MNE Filtering.

3.5.1 Bitbrain Method

One of the main concerns when dealing with electroencephalographic signals (EEG) is assuring that clean data with a high signal to noise ratio is recorded. The EEG signal amplitude is in the microvolts range, and it is easily contaminated with noise, known as artifacts, which need to be filtered from the neural processes to keep the valuable information needed for different applications.

In this domain, an artifact is denoted as any component of the EEG signal that is not directly produced by human brain activity, making the system register noise that contaminates the neural EEG data. The ability to recognize these artifacts is the first step in removing them. EEG artifacts can be classified depending on their origin, which can be physiological or external to the human body (technical/non-physiological).

The Bitbrain team develops task-specific algorithms to automatically estimate noise and evaluate the overall

quality of EEG signals, identifying the artifacts present in recordings made using their wearable textile headband.

These algorithms detect several artifacts of different origin, including:

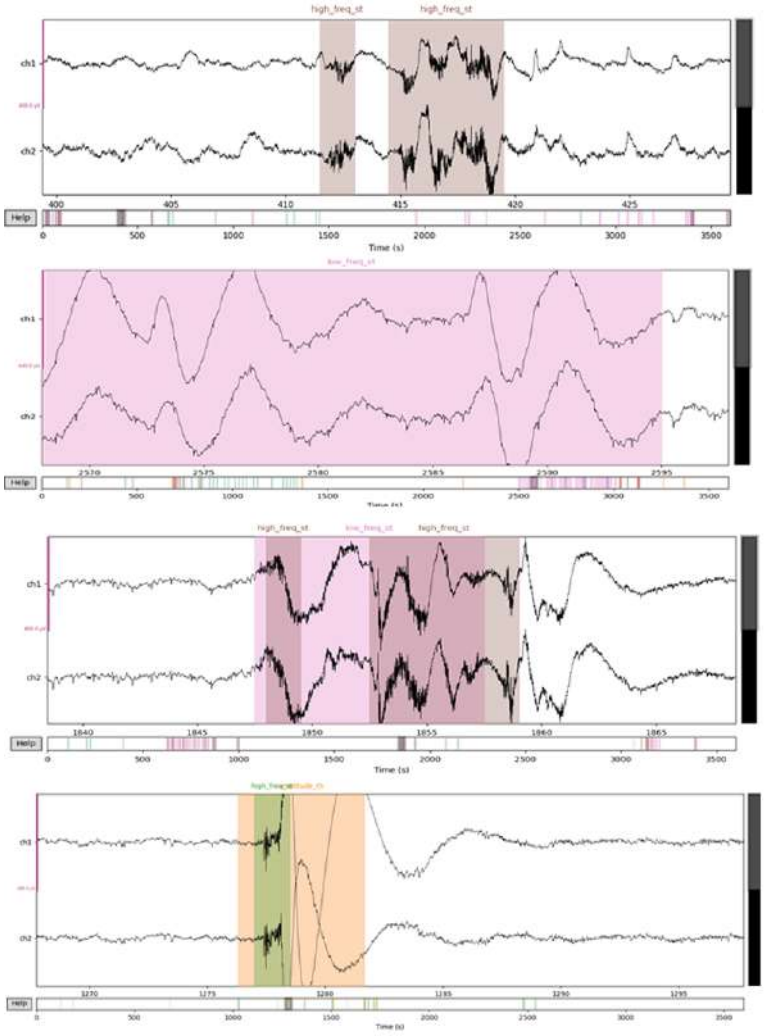
- High amplitude artifacts that may be due to:
  - Temporary failures in contact between the EEG sensor and the scalp produced by touching the sensor or by spontaneous changes in electrode-skin contact.
  - Movement of the cables connecting the electrodes and the amplification system.
- High frequency noise (30-45 Hz) that may originate from:
  - Electrical activity produced by the muscles when they are contracted. This activity can be measured, and the resulting signal is called electromyography (EMG). Typical effects are muscle tension in the jaw or forehead, that can take place when clenching or frowning, respectively.
  - AC electrical and electromagnetic interferences that may arise due to insufficient lack of wire shielding.
  - Poor electrode-skin contact.
- Low frequency noise (0.2-4 Hz) mainly due to:
  - Perspiration originated from small drops of sweat produced by the skin glands, which cause changes in the electrical baseline of the electrodes. In case of intense perspiration, it could even create shorts between electrodes.
- Empty segments because of Bluetooth connection loss.
- Flat line segments mainly caused by instrumental error where contact between electrodes and skin is lost.

Prior to applying these algorithms, a band-pass filter is applied to the EEG in the range 0.2-45 Hz. Below, the development of the artifact detectors is discussed in more detail:

- **High amplitude artifacts:** This method is based on a static amplitude threshold, specified in microvolts. A window is then selected around any point that exceeds the threshold.
- **High frequency noise:** The power in the 30-45Hz power range is estimated using signal windows. Basically, to label a zone as artifactual, the power in the mentioned range must exceed a threshold, which is fixed in a logarithmic scale.
- **Low frequency noise:** The power is estimated in the 0.2-4Hz power range using signal windows. The method applied to recognize an area as an artifact is the same as for high frequency noise.
- **Flat line segments:** This detector identifies time windows where the amplitude does not exceed a very small threshold, close to zero.

All default thresholds were selected based on a statistical analysis of more than 150 EEG sleep recordings.

Figure 2: Examples of the temporal visualization of EEG signals together with the labelling of noisy segments (shading). It can be seen which areas of the signal have been identified as artifacts, as well as by which detectors. (Image 1-3: Pink - Low frequency noise (sweat); Brown: High frequency noise (muscle tension). Image 4: Green – High frequency noise; Orange: High amplitude noise).



The output of the above algorithms provides binary masks that locate and identify noise-contaminated signal segments and their possible source. This automatically generated information can be very valuable for the experimenter, since it is resource-intensive to manually inspect a sleep recording that can last approximately eight hours.

Within Manolo's Use Case 3, Bitbrain is designing a closed-loop auditory stimulation system during sleep, which will integrate and validate the technology developed throughout the project. The first task performed by the system is the real-time characterization of sleep stages. This is accomplished by an automatic sleep scoring algorithm, consisting of a deep neural network (DNN) that is being trained with the sleep datasets recorded using the wearable device.

Therefore, the artifact masks generated for this purpose contain an estimate every 30 seconds. In this case, the thresholds of the different detectors were estimated empirically. They were established with the aim of optimizing the artifact removal to maximize data analysis while minimizing the amount of discarded data. By this means, only those artifacts that negatively impact the performance of the model are detected and eliminated.

Table 11 presents the number of artifacts identified by Bitbrain's method across the ten recordings used for

evaluating the task-agnostic framework. It provides the total count of 30-second segments in which at least one artifact was detected for each recording, with results reported separately for each EEG channel (designated as HB1 and HB2).

Table 11: Number of segments containing artifacts in the test recordings identified by Bitbrain's method

Recording	1	2	3	4	5	6	7	8	9	10
HB1	9	57	0	1	27	0	0	1	1	0
HB2	9	56	0	0	26	151	0	1	1	0

The table shows that in three out of the ten recordings (3, 7, and 10), neither channel detects any artifacts. In three other recordings (4, 8, and 9), only one segment is found to be contaminated with noise by both channels, except in recording 4, where noise is detected solely by the HB1 channel. In recordings 1 and 2, noisy segments are almost unanimously identified by both channels. Additionally, in recording 6, only the HB2 channel detects noise in several segments, suggesting a potential issue with that particular sensor.

Table 12 presents the Precision and Recall metrics, for the evaluated methods across the seven test recordings (1, 2, 4, 5, 6, 8, and 9) where an artifact was detected by the Bitbrain's method. All values are expressed as percentages.

The Precision metric denotes the percentage of actual segments identified as having artifacts relative to the total segments predicted to contain artifacts. However, this metric fails to convey information regarding missed artifacts—specifically, those not identified by the anomaly detection methodologies. This aspect is quantified through the Recall metric, which indicates the ratio of segments predicted to have artifacts against the total number of actual segments containing artifacts. For instance, the PCA method exhibits a precision of 100% from the data obtained during the HB1 test recording 1, yet it only achieves a recall of 11.11%. This discrepancy arises because the PCA method successfully detected a singular artifact characterized by noise, deemed a noisy artifact; thus, the precision remains at 100%. However, it failed to identify the additional eight noisy segments (refer to Table 11), resulting in a significantly reduced recall of approximately 11%. Instances where there are empty cells signify scenarios where the respective method did not identify any artifacts. Consequently, precision cannot be ascertained, and recall is recorded as 0%.

In this application, our primary objective is to identify as many segments as possible that contain artifacts while allowing for some tolerance regarding detecting noisy segments. The principal aim should be to reduce the number of instances requiring the application of reliable yet non-automated anomaly detection methods. Consequently, the key performance metric to prioritize is Recall.

From the results presented in Table 12, we conclude that all considered methods exhibited suboptimal performance. However, the PCA demonstrated relatively high recall values in two recordings. One plausible explanation for this observation could be the selected time-window of 30 seconds that defines the examined segment. The efficacy of anomaly detection methods in time series analysis may be significantly influenced by the length of the time series, as supported by existing literature [21].

Due to this reasoning, we applied the selected methodologies to the same dataset while extending the segment duration from 30 seconds to 2, 5, and 10 minutes. Table 13, Table 14, and Table 15 illustrate the resulting recall values of the selected methods, respectively.

One initial conclusion drawn is that across all methods analyzed, the detection of anomalies is significantly enhanced when the duration of the time-window assigned to each segment is increased. However, this improvement comes at the expense of broader time-windows during which the detected anomalies may actually occur—a consideration that necessitates careful evaluation.

Table 12: Precision/Recall of the task-agnostic methods with respect to the Bitbrain's Method

	HB1							HB2						
Test Recording	1	2	4	5	6	8	9	1	2	4	5	6	8	9
LSTM-AE		5/2							7/4		3/4	6/1		
Conv-LSTM-AE				11/4										
Attn-AE [e]														
Predictor [e]														
Classifier														
Attn-AE [a]		4/4										20/1		
Predictor [a]		5/2						1/11	5/14		1/8	13/15		
PCA	–/11			100/85		100/100		67/22	1/4		24/96	41/9	100/100	
ADWIN								15/22			11/8	65/58		
CUSUM														
Page-Hinkley									1/4					
KL									1/4					

Table 13: Recall (2min) of the task-agnostic methods with respect to the Bitbrain's Method

	HB1							HB2						
Test Recording	1	2	4	5	6	8	9	1	2	4	5	6	8	9
LSTM-AE		7.02	100						14.29		15.39	5.96	100	
Conv-LSTM-AE				22.22			100							
Attn-AE [e]		3.51										2.65		
Predictor [e]														
Attn-AE [a]		7.02		11.11			100					2.65		
Predictor [a]		7.02						33.33	58.93		46.15	60.93	100	100
Classifier														
PCA	22.22			85.19		100	100	22.22	5.36		96.15	8.61	100	100
ADWIN								22			8	74	100	
CUSUM														
Page-Hinkley														
KL														

Table 14: Recall (5min) of the task-agnostic methods with respect to the Bitbrain's Method

	HB1							HB2						
Test Recording	1	2	4	5	6	8	9	1	2	4	5	6	8	9
LSTM-AE		17.54							37.50		34.62	21.85		
Conv-LSTM-AE				29.63			100	11.11				1.33		
Attn-AE [e]		14.04										3.31		
Predictor [e]														
Attn-AE [a]		17.54	100	22.22			100					5.96		
Predictor [a]		19.30						100	100		100	84.11	100	100
Classifier														
PCA	88.89			85.19		100	100	88.89	26.79		96.15	20.53	100	100
ADWIN								89			12	76	100	
CUSUM														
Page-Hinkley														
KL														

Table 15: Recall (10min) of the task-agnostic methods with respect to the Bitbrain's Method

	HB1							HB2						
Test Recording	1	2	4	5	6	8	9	1	2	4	5	6	8	9
LSTM-AE		36.84	100	11.11					64.29		42.31	41.72	100	
Conv-LSTM-AE				66.67			100	100				1.33		
Attn-AE [e]		31.58										5.30		
Predictor [e]														
Attn-AE [a]		35.09	100	88.89			100					8.61		
Predictor [a]		38.60		3.70				100	100		100	90.07	100	100
Classifier														
PCA	100			85.19		100	100	100	62.50		96.15	40.40	100	100
ADWIN								100			12	81	100	
CUSUM														
Page-Hinkley														
KL														

The primary conclusions are drawn through a comparative analysis of the performance of the evaluated methodologies. The introduced concept of using attention mechanisms as a critical metric for anomaly detection appears to be promising. Across all segment durations (30 seconds, 2 minutes, 5 minutes, and 10 minutes), the Recall metrics for the Attn-AE [a] and Predictor [a] methods, which identify anomalies based on their attention matrix values, surpass those of their counterparts, Attn-AE [e] and Predictor [e], which detect anomalies via reconstruction or prediction errors.

The performance of attention-based methods is generally comparable to, if not superior to, that of other state-of-the-art techniques, such as LSTM-AE and Conv-LSTM-AE. A similar conclusion can be derived from comparing the Recall values of the examined attention-based methods with the baseline method, PCA. However, it is evident that the Predictor [a] method significantly outperforms the Attn-AE [a] method.

Key Takeaways

- Attention-based methods (Attn-AE [a] and Predictor [a]) outperformed state-of-the-art methods across all time windows, highlighting the effectiveness of attention mechanisms in capturing anomalies.
- Predictor [a] had better recall than Attn-AE [a], suggesting that prediction-based approaches might be more reliable.
- PCA performed better than Attn-AE [a] for several recordings and was comparable to Predictor [a], with slightly better recall in HB1 and slightly worse in HB2.
- Longer time windows (10 minutes) improved recall for most methods, but this may have also introduced trade-offs in precision.

3.5.2 MNE Filtering

MNE filtering is a method used to remove unwanted frequencies from time-series data, typically EEG signals. It applies a bandpass filter to the data, allowing signals within a specific frequency range (e.g., 0.5 to 40 Hz) to pass through while reducing frequencies outside this range. This is important for eliminating noise, such as power line interference or muscle activity, that falls outside the range of interest. MNE acts as a smoothing or cleaning tool, refining the data by focusing on the frequencies that are relevant for analysis.

To estimate noise, MNE filtering works by first applying the bandpass filter to the data. The filtered signal, which retains the relevant neural activity, is compared to the original signal. The difference between the two signals is then considered as noise, representing unwanted components of the data that lie outside the targeted frequency range.

For MNE filtering, the noise thresholding approach is consistent with the machine learning-based methods, using the 99th percentile to separate noise (1) from no noise (0).

Table 16 presents the Precision and Recall metrics,for the evaluated methods across the same seven test recordings (1, 2, 4, 5, 6, 8, and 9) used in Bitbrain's analysis. All values are expressed as percentages.

In Table 16, we observe that all methods performed suboptimally overall. The attention-based models slightly outperformed the baseline methods (LSTM, Conv-LSTM, Attn-AE [e], Predictor [e]), although their performance is still considered weak. As for the statistical methods, only ADWIN achieved a recall of 100, and even that occurred only in one test recording.

Similary to the evaluations with respect to the Bitbrain's method, we applied the selected methodologies while

Table 16: Precision/Recall of the task-agnostic methods with respect to MNE Filtering

	HB1							HB2						
Test Recording	1	2	4	5	6	8	9	1	2	4	5	6	8	9
LSTM-AE			100/1.16							100/2.84		3.03/20		
Conv-LSTM-AE			100/0.53							100/0.95				
Attn-AE [e]										100/0.11				
Predictor [e]										100/0.11				
Classifier														
Attn-AE [a]			100/3.57							100/0.74				
Predictor [a]			100/1.37							100/17.02		0.57/20		
PCA										100/0.11		11.77/40		
ADWIN										100/0.84		3.73/100		
CUSUM														
Page-Hinkley														
KL														

extending the segment duration from 30 seconds to 2, 5, and 10 minutes. Table 17, Table 18, and Table 19 illustrate the resulting recall values of the selected methods, respectively.

From those results, we observe that Predictor [a] demonstrates the most consistent and superior performance, achieving 100% recall in multiple recordings across both channels (HB1 and HB2). This method significantly outperforms the baseline PCA method, particularly in HB2 test recordings 4 and 6, where PCA recall values are much lower. Attn-AE [a] also performs reasonably well, with higher recall in HB1 test recording 4 and HB2 test recording 6, though it still falls short compared to Predictor [a].

Regarding the state-of-the-art methods, Predictor [e] and Attn-AE [e] show poor performance, with recall values close to zero across most recordings. LSTM-AE and Conv-LSTM-AE perform slightly better, but their results remain moderate compared to the attention-based models.

3.6 Conclusion

In this study, we introduced a task-agnostic framework for time-series data quality estimation, addressing the limitations of existing task-specific methods that fail to generalize across different datasets. The framework includes both machine learning (ML)-based and statistical methods, designed to identify noisy and inconsistent data. We evaluated our methods using the Bitbrain EEG dataset, comparing our results against two ground truth methods: Bitbrain's own approach, treated as a black-box, and MNE Filtering, which focuses on filtering EEG signals by isolating noise through bandpass filters. To assess the robustness of our framework, we experimented with different time windows for data segmentation, ranging from 30 seconds to 10 minutes. This variation in time windows allowed us to explore how segment duration influences the performance of different methods in detecting noise.

Our analysis revealed that attention-based methods, particularly the Predictor [a] and Attn-AE [a] models, consistently outperformed state-of-the-art methods like LSTM-based autoencoders. These attention-based models proved to be highly effective in detecting noise, making attention mechanisms a promising tool for noise detection within machine learning models. Interestingly, among the statistical methods, Principal Component Analysis (PCA) demonstrated the best performance, achieving results comparable to the Predictor [a] method.

Regarding data segmentation, we observed that extending the time window generally improved the perfor-

Table 17: Recall (2min) of the task-agnostic methods with respect to MNE Filtering

	HB1							HB2						
Test Recording	1	2	4	5	6	8	9	1	2	4	5	6	8	9
LSTM-AE			4.2							10.92		40		
Conv-LSTM-AE			2.11							3.8				
Attn-AE [e]										0.42				
Predictor [e]										0.42				
Classifier														
Attn-AE [a]			13.45							2.94				
Predictor [a]			5.04					100		51.68		80		
PCA	57.14									0.42		60		
ADWIN										2.84		100		
CUSUM														
Page-Hinkley														
KL														

Table 18: Recall (5min) of the task-agnostic methods with respect to MNE Filtering

	HB1							HB2						
Test Recording	1	2	4	5	6	8	9	1	2	4	5	6	8	9
LSTM-AE			11.56							27.31		40		
Conv-LSTM-AE			5.25					100		9.35				
Attn-AE [e]										0.42				
Predictor [e]										1.05				
Classifier														
Attn-AE [a]			30.46							7.35		20		
Predictor [a]			12.61					100		87.4		100		
PCA	90.91									1.05		60		
ADWIN										6.62		100		
CUSUM														
Page-Hinkley														
KL														

Table 19: Recall (10min) of the task-agnostic methods with respect to MNE Filtering

	HB1							HB2						
Test Recording	1	2	4	5	6	8	9	1	2	4	5	6	8	9
LSTM-AE			21.01							48.32		80		
Conv-LSTM-AE			10.50					100		17.75				
Attn-AE [e]										2.10				
Predictor [e]										2.10				
Classifier														
Attn-AE [a]			46.22							14.71		20.00		
Predictor [a]			25.21					100		97.90		100		
PCA	100		0.95					100		3.05		80.00		
ADWIN								100			12	81	100	
CUSUM														
Page-Hinkley														
KL														

mance metrics for most methods, especially in terms of recall. However, this improvement came with an increased risk of false positives. The trade-off suggests that while larger time windows may help in detecting anomalies more effectively, it is important to carefully consider the duration of the segments to balance performance and minimizing false detections.

4 Data Distillation

4.1 Introduction

Considering the AI efficiency dimension of the MANOLO project, tackling this task from the data perspective will involve the development of, among other techniques, Dataset Distillation (DD) methods. These would aim to create compact, high-fidelity data summaries from large datasets while retaining essential data patterns and dynamics. This DD functionality is critical for achieving MANOLO's project goals of improved efficiency, as well as the mitigation of ethical issues. This is in response to a clear trend in AI of model and dataset size growth. Compare the 3.2 million image size of the Imagenet dataset [22] from 2009, with the 5.85 billion images of 2022's LAION-5B dataset [23], integral to the training of many popular image generation models, such as Stable Diffusion. Similarly, the Imagen generative model is trained on 15 billion images [24], while ChatGPT's GPT4 was trained on 499 billion tokens of text data [25]. While this massive growth has enabled new applications and capabilities for AI, it has also brought upon a number of problematic implications in the dimensions of efficiency and ethics. Current training processes used for AI language models such as ChatGPT have significant energy requirements, rivaling the annual consumption of hundreds of American households [26], contributing to greenhouse gas emissions. Additionally, large datasets are hard to accommodate on hardware with limitations on power consumption, storage capacity, and network bandwidth, such as edge devices. With respect to the ethical implications, the costs associated with large models and datasets may present a barrier to entry for smaller organizations and teams, which could in turn contribute to the monopolization of AI technologies. DD represents one technique that could be used by AI researchers to alleviate these challenges.

The most widely studied approaches to DD can be categorized into Data Matching and Meta-Learning [27]. Data Matching aims to match the statistical properties of a model that is trained on a distilled dataset with those of a model that is trained on the original, non-distilled dataset. Different techniques in the Data Matching field include Gradient Matching, Trajectory Matching, and Distribution Matching. While this approach has achieved state-of-the-art performance in many DD tasks [28], it involves the use of computationally expensive optimization due to its iterative nature, particularly across long training periods or large parameter spaces. In addition to this drawback, Data Matching has limitations with respect to different computational architectures. For instance, Gradient Matching requires the computation of the gradient of the loss function of the model trained on the distilled data in order to compare it with that of the model trained on the original data. This is incompatible with neuromorphic computing paradigms, where the incoming data is in the form of discrete spike trains, and as such, their gradient cannot be precisely calculated [29]. Other works make use of meta-learning approaches to DD, particularly back-propagation through time (BPTT) and Kernel Ridge Regression (KRR). However, both of these approaches also suffer from scalability issues, in the case of BPTT due to the high computational overload of unrolling the recursive computation graph across long trajectories, and in the case of KRR due to inefficiencies in kernel computation [27].

In the following sections, we outline our efforts towards the development of a novel, clustering-based DD approach. This method differs from the traditional DD approaches described above by performing the distillation task not on the original data, but rather on the data's encodings in the latent space of a variational autoencoder (VAE) trained on the original dataset. We have initially evaluated the technique on two image datasets due to time constraints, and future research efforts will focus on testing its validity for other modalities, as well as on improving the distillation performance.

4.2 Clustering-Based DD

One of the key goals of the MANOLO project is to enable efficient training and deployment of AI models across the Cloud-Edge Continuum (CEC), where computational resources are often constrained. This is where DD can align with MANOLO's efficient AI goals by creating reduced datasets which preserve essential information, thus ensuring faster, lower-memory, and resource-efficient model training. Furthermore, since MANOLO's techniques are intended for use with diverse data types, DD with agnostic techniques (e.g., clustering) will

help standardise the data preparation activity across various use cases. As such, particular attention was given to developing a DD technique that would be modality-agnostic.

4.2.1 Methodology

Our research into DD focused on distilling computer vision datasets, namely MNIST and CIFAR10. The MNIST dataset is a collection of 70,000 grayscale images of handwritten digits (0–9), while the CIFAR-10 dataset consists of 60,000 color images across 10 distinct classes (e.g., animals, vehicles). These datasets are commonly used for training and testing in image classification tasks.

Our proposed DD methodology uses a combination of Variational Autoencoders (VAEs) and K-means clustering to reduce the size of computer vision datasets while maintaining high accuracy metrics when used for classification tasks. Variational Autoencoders are neural networks that encode input data into a compressed latent space, which captures the data's essential features. A peripheral benefit of training VAEs for use in DD is their potential to be used for data synthesis. This is done by sampling values from the latent space, then subsequently generating new datapoints from the conditional distribution of the data given the latent variable [30]. This further aligns with the MANOLO project's efficiency goals, as it allows for the solution of two different tasks, DD and dataset generation, with the same technique.

In our approach, we train a Variational Autoencoder on the dataset in order to generate a latent space. The latent dimension used for the MNIST dataset is 15, and the one used for the CIFAR-10 dataset is 100 due to its additional complexity. After this, the latent space is clustered into N clusters, where N is the number of target classes in the dataset being distilled. Since the number of clusters are pre-determined, the clustering algorithm used to achieve this is K-means [31]. Using this algorithm has a number of advantages in the context of MANOLO's project goals, specifically computational efficiency superior to that of more complex DD methods, low memory overhead on edge devices, and ease of implementation due to its availability in standard ML libraries. Figure 3 shows a visualization of the K-means clustered latent space for the MNIST image dataset, generated using the UMAP dimensionality reduction technique [32]. Once the clustering process is complete, the centroid of each cluster is calculated, and the distances from each dataset sample encoded in the latent space to its corresponding cluster centroid are found. The measure used to calculate this is Euclidean distance.

In order to distill the clustered dataset, the Euclidean distances from each encoded sample to its corresponding cluster centroid was found. The samples with the shortest distance to their centroid were then removed from the dataset. The logic behind this is that these samples are the easiest to classify and contribute the least amount of information to the training process, so their removal from the dataset will not adversely affect model performance. In addition to this, we also removed samples that were furthest from their cluster centroids, as well as randomly selected samples. This was done in order to gain more insight into how this DD method affected underlying data structures.

To examine the effects of our clustering-based DD method on the accuracy of classification tasks, a ResNet-50 classifier was trained on the MNIST and CIFAR-10 image dataset for a total of 50 and 200 epochs respectively. The classifier was trained on a number of different distillation levels, from 10% to 90%, and the test set accuracy was evaluated for each different level. This was done to investigate the trade-off between DD and classification accuracy. A base classifier was also trained in each dataset's case with a distillation of 0% for comparison purposes.

4.2.2 Results

In this section, we present the results of our experiments on how Clustering-Based DD affects the performance of ResNet-50 classifiers trained on the MNIST and CIFAR-10. Table 20 presents the accuracy of each ResNet-50 classifier trained on the MNIST dataset after 50 epochs. When removing the nearest samples to each cluster centroid, the drop in accuracy as the distillation increases is small, with the classifier trained at 90% distillation

Figure 3: UMAP visualization of the K-Means clustered latent space of the MNIST dataset

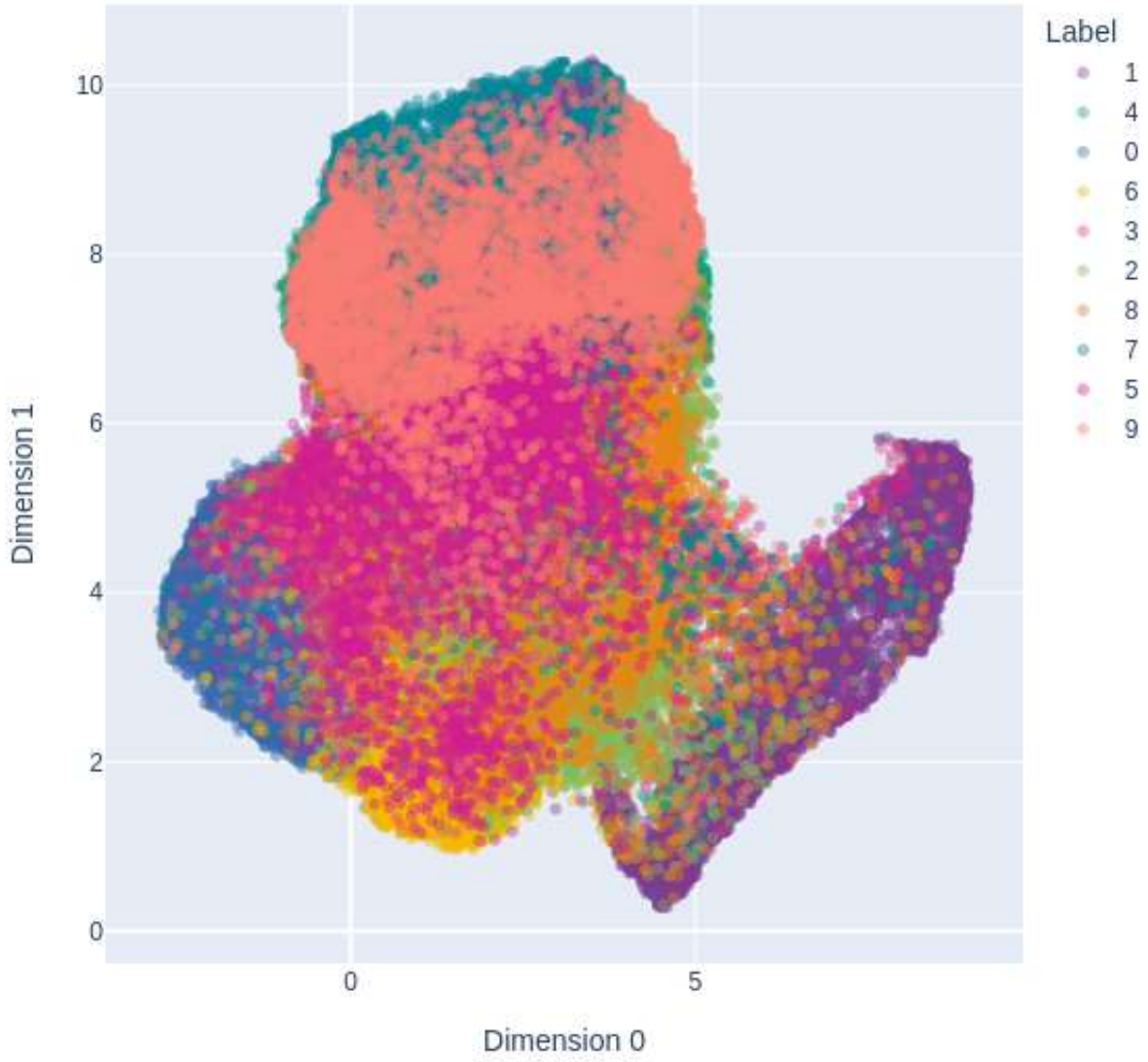


Table 20: Comparison of ResNet-50 classifier accuracies for the MNIST dataset at distillation proportions between 10% and 90%. Baseline classifier accuracy is shown at a distillation of 0%.

	Accuracy (%)					
	MNIST			CIFAR-10		
Distillation (%)	Random	Nearest	Furthest	Random	Nearest	Furthest
0	99.08	98.90	98.97	90.60	90.09	90.09
10	98.81	98.78	98.95	89.42	89.40	89.91
20	98.78	98.77	98.82	88.59	88.70	88.79
30	98.71	98.79	98.78	88.29	87.77	88.06
40	98.83	98.71	98.89	87.65	87.63	87.04
50	98.41	98.51	98.57	86.02	85.90	85.55
60	98.54	98.30	98.31	84.67	84.47	84.78
70	98.08	97.93	98.09	82.59	81.39	80.70
80	97.18	97.53	97.48	71.52	78.19	75.59
90	96.31	96.45	96.25	52.88	56.36	61.53

only performing 2.45% worse than the baseline model in terms of accuracy on the test set. Similar metrics can also be observed for the Random and Furthest removal criteria. However, it is worth noting that ResNet-50 is a deep model with 50 layers, and the MNIST dataset is simple, with low variability, so even with a very small amount of data it is expected that the classifier will perform well.

Figure 4 shows the accuracy of the ResNet-50 classifier on the MNIST dataset at different distillation levels. In order to preserve the visual clarity of the line chart, only a subset of tested distillation levels are presented. We noticed an interesting pattern in our experimental results where performance tended to group into distinct clusters. One can see in Table 20 that the accuracy at a distillation of 0% and that at 80% is fairly similar, while the accuracy at a distillation of 90% is significantly lower. Table 20 shows experimental results for our DD method on the CIFAR-10 image dataset after 200 epochs of training. The overall performance here is lower than on the MNIST image dataset due to CIFAR-10's multi-channel nature, higher complexity, and increased variability. Similar to the MNIST results presented in Table 20 and Figure 4, we can observe two distinct performance clusters with the CIFAR-10 dataset. As distillation levels increase from 0% to 80%, there is an accuracy drop of less than 12%, but when the distillation level is 90%, the accuracy drops by 33.73% when compared to the baseline. This sudden drop in accuracy can also be observed in Figures 5, 6, and 7, which show the test accuracy over epochs of training for the CIFAR-10 classifiers using the 3 different DD removal criteria - Nearest, Furthest, and Random. This pattern is worth investigating in future research as it could potentially indicate the existence of a consistent elbow point for our Clustering-Based DD method across multiple datasets. It is also worth noting that the distance criterion used to distill the data had an effect on the accuracy of the classifier. From Table 20 we can see that at a distillation levels of 80% and 90%, the performance of the CIFAR-10 classifier is lowest when points are removed randomly, as opposed to removing samples nearest or furthest from their respective cluster centroid.

The effect of our Clustering-Based DD method on the data's global structure was also investigated. This was done by using UMAP [32] to perform dimension reduction on the dataset in order to be able to represent certain classes as 2D heatmap plots. This process was repeated at different levels of DD, and using Random, Nearest and Furthest distance criteria for data sample removal. Figure 8 shows the UMAP heatmap visualizations of CIFAR-10 class 0 (airplane), and Figure 9 shows them for CIFAR-10 class 1 (automobile).

Interpreting Figures 8 and 9, we can see that while the Clustering-Based DD method is able to significantly reduce the size of image datasets, it also leads to a substantial shift in the underlying structure of the data. For both classes 0 and 1, we can observe that 90% DD fundamentally changes the shape of the data structure with all three distance criteria. However, when we look at the UMAP visualizations for the 50% DD, there are certain

Figure 4: Accuracy over Epochs of training for ResNet-50 Classifiers trained on the MNIST dataset with different levels of Clustering-Based DD. The removal criterion used was Nearest distance to cluster centroid.

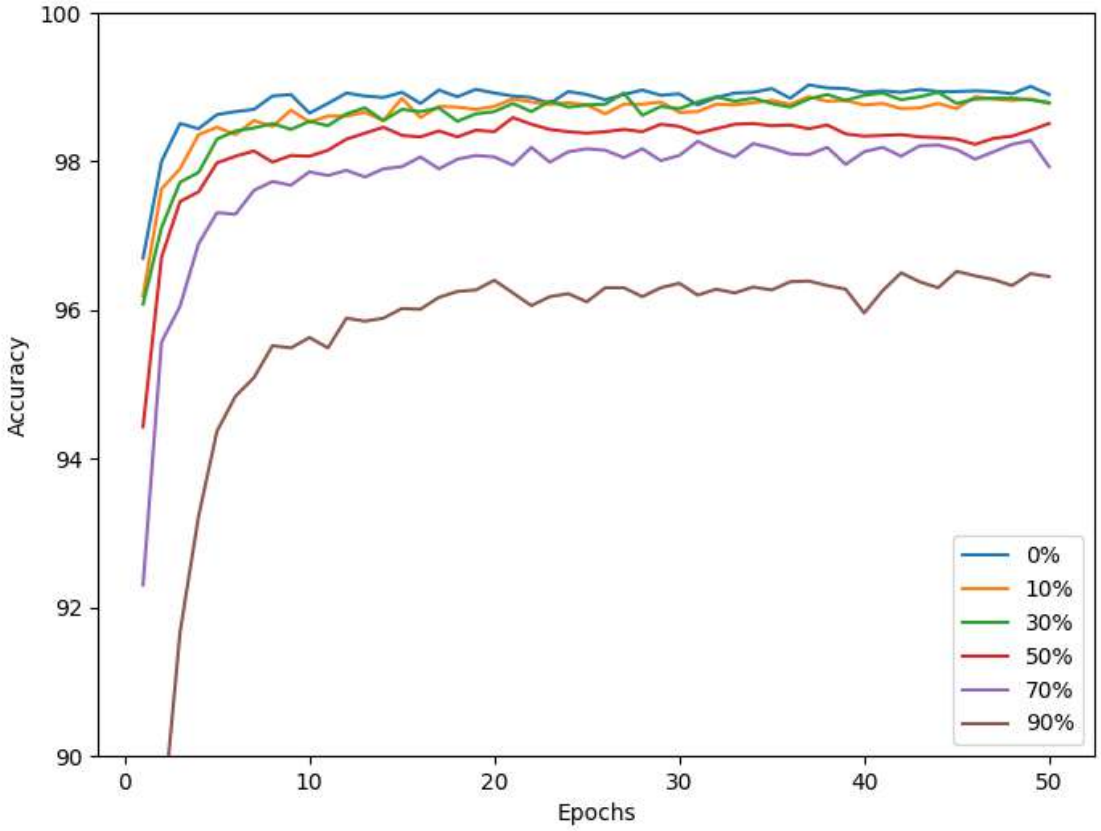


Figure 5: Accuracy over Epochs of training for ResNet-50 Classifiers trained on the CIFAR-10 dataset with different levels of Clustering-Based DD. The removal criterion used was Nearest distance to cluster centroid.

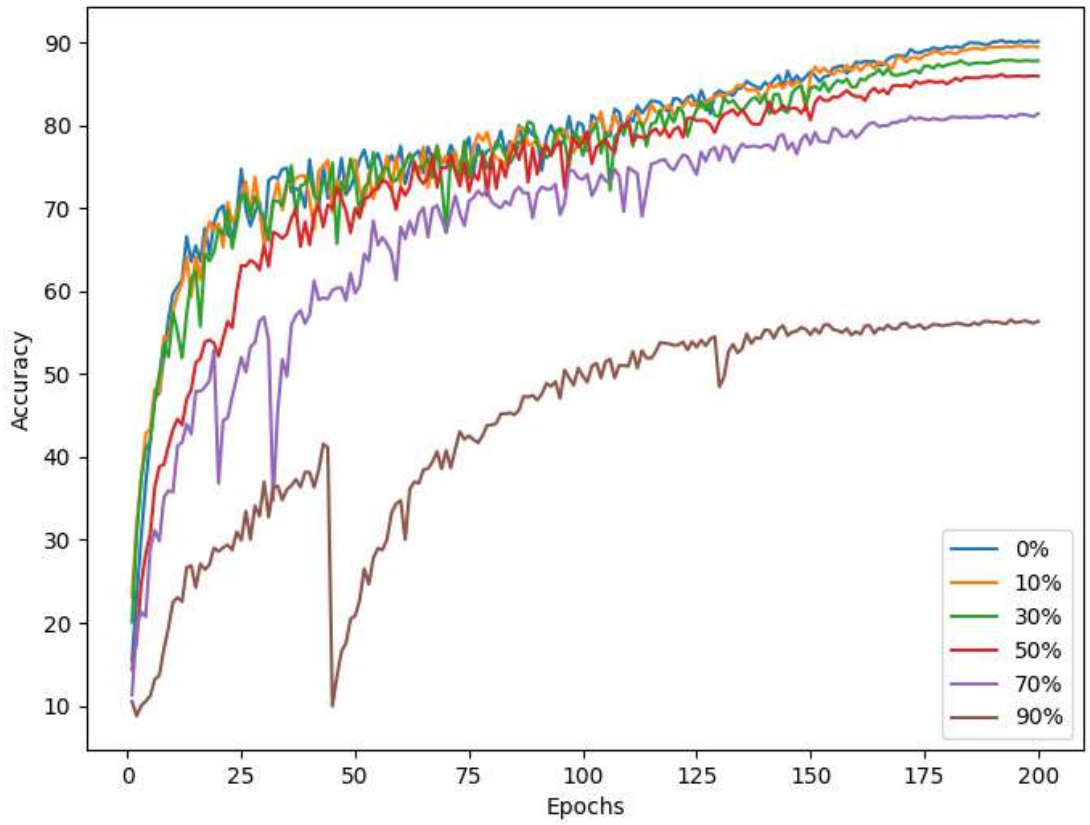


Figure 6: Accuracy over Epochs of training for ResNet-50 Classifiers trained on the CIFAR-10 dataset with different levels of Clustering-Based DD. The removal criterion used was Furthest distance to cluster centroid.

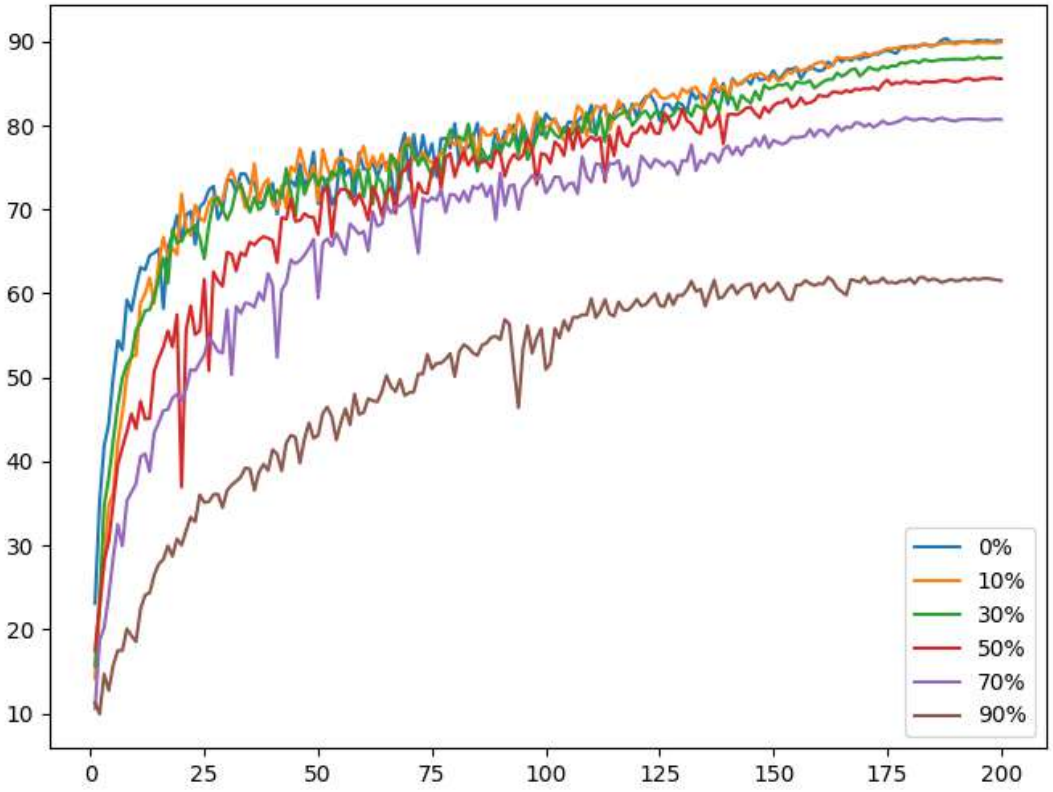
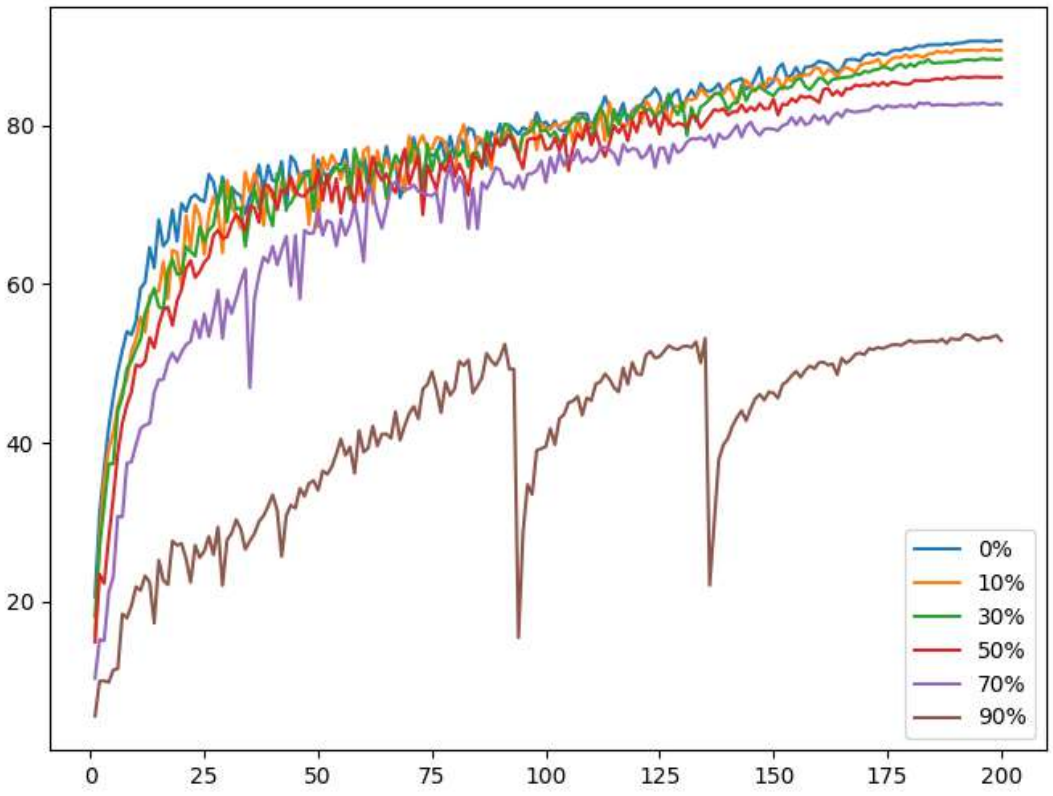


Figure 7: Accuracy over Epochs of training for ResNet-50 Classifiers trained on the CIFAR-10 dataset with different levels of Clustering-Based DD. The removal criterion used was Random sample removal



distance criteria affect the data structure more than others. In the case of Random Sample Removal at 50% DD (Figures 8(b) and 9(b)), the heatmaps do not change a lot from their respective baselines (Figures 8(a) and 9(a)), while for Nearest (Figures 8(e) and 9(e)) and Furthest Sample Removal at 50% DD (Figures 8(h) and 9(5)), the heatmap shapes deviate significantly from those of the baseline. This indicates that the Clustering-Based DD method also changes the distribution of the data after distillation is performed.

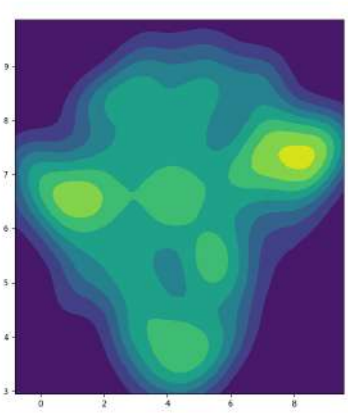
4.2.3 Conclusion and Future Work

The experimental results achieved for the MNIST and CIFAR-10 datasets indicate that our Clustering-Based DD method can significantly reduce the size of image datasets while causing a relatively small loss in classification accuracy. In the case of the MNIST dataset, we achieved a 50% decrease in dataset size while losing less than 0.4% model accuracy compared to the baseline, and for the CIFAR-10 dataset, the accuracy loss at 50% distillation was just 4.19% compared to the baseline (Table 20). These are promising initial results, especially with our current basic clustering method.

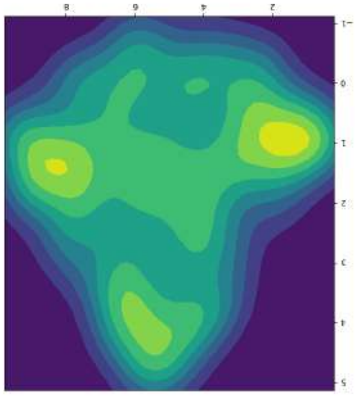
In addition to our method's adherence to MANOLO's project goal of improved efficiency for AI techniques, the use of Clustering-Based DD also meets MANOLO's criteria for ethicism and modality agnosticism. Due to the fact that we only require the use of the dataset's latent space for the distillation process, there is no need for the transfer of sensitive or private user data - the clustering can be completed on the user or edge device, and only the encoded latent space needs to be transferred to MANOLO. Also, as mentioned in Section 4.1, our DD methodology is compatible with computing paradigms that are not compatible with more traditional DD techniques, specifically Neuromorphic Computing.

Further work in this area will explore the use of more complex clustering approaches. While K-means clustering has a number of benefits that are in line with MANOLO's project goals, it is important that the shape of the data distribution is kept the same before and after the distillation process. Examining the UMAP heatmaps (Figures 8 and 9) generated of the CIFAR-10 dataset before and after the application of the Clustering-Based DD method, it's clear that in its current iteration, the technique does not sufficiently preserve the underlying data distribution. A more sophisticated, hybrid approach will be employed in the future to attempt to alleviate this problem, rather than just distilling based on the points' distance to cluster centroids.

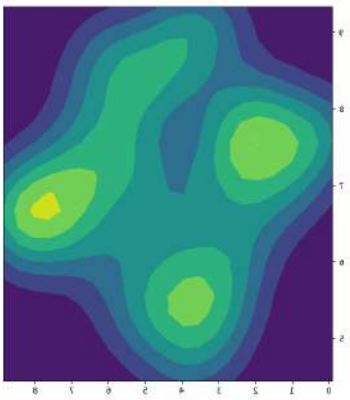
Figure 8: CIFAR-10 Class 0 (airplane) represented as a 2D heatmap using UMAP, at different levels of DD using Random, Nearest, and Furthest cluster distance criteria



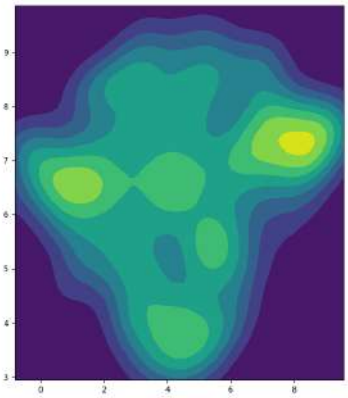
(a) Random Sample Removal - 0% DD



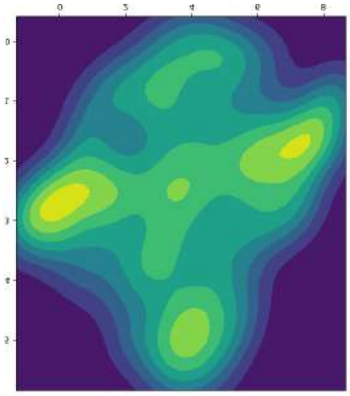
(b) Random Sample Removal - 50% DD



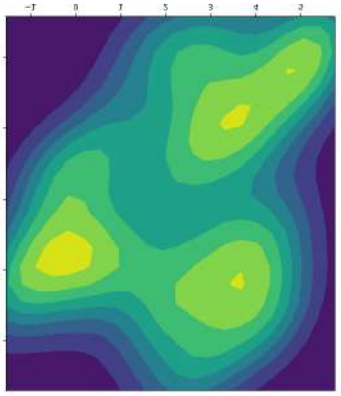
(c) Random Sample Removal - 90% DD



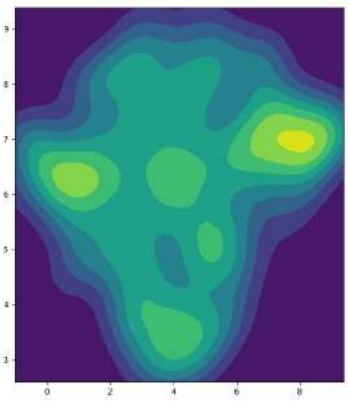
(d) Nearest Sample Removal - 0% DD



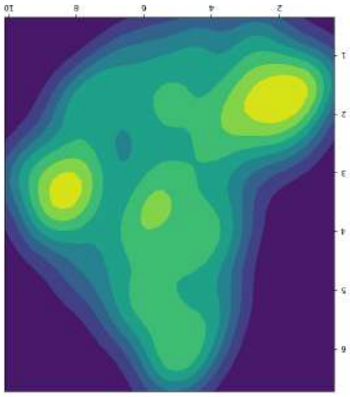
(e) Nearest Sample Removal - 50% DD



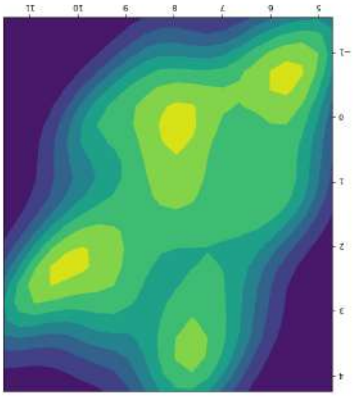
(f) Nearest Sample Removal - 90% DD



(g) Furthest Sample Removal - 0% DD

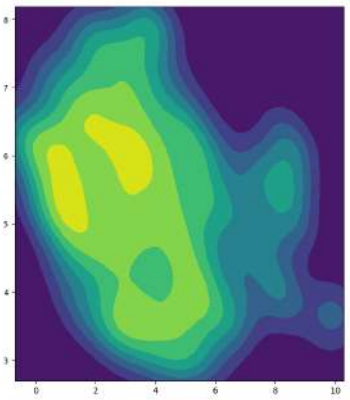


(h) Furthest Sample Removal - 50% DD

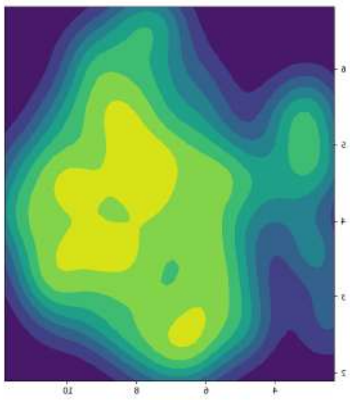


(i) Furthest Sample Removal - 90% DD

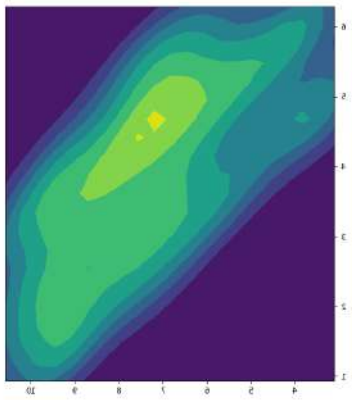
Figure 9: CIFAR-10 Class 1 (automobile) represented as a 2D heatmap using UMAP, at different levels of DD using Random, Nearest, and Furthest cluster distance criteria



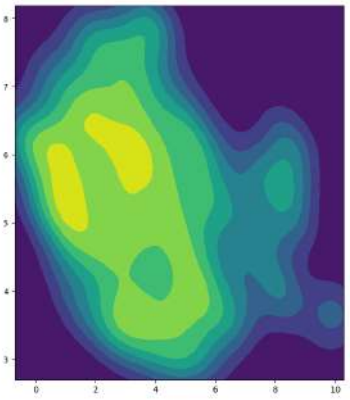
(a) Random Sample Removal - 0% DD



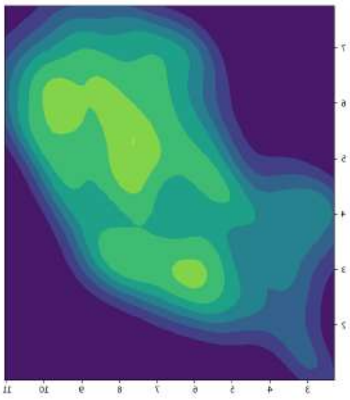
(b) Random Sample Removal - 50% DD



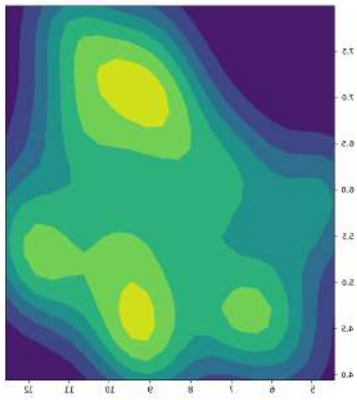
(c) Random Sample Removal - 90% DD



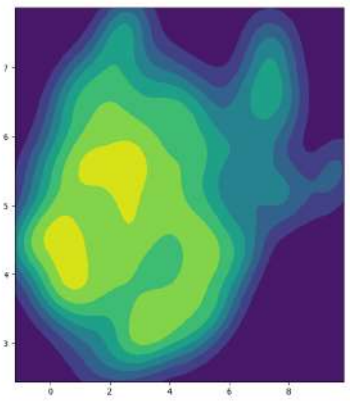
(d) Nearest Sample Removal - 0% DD



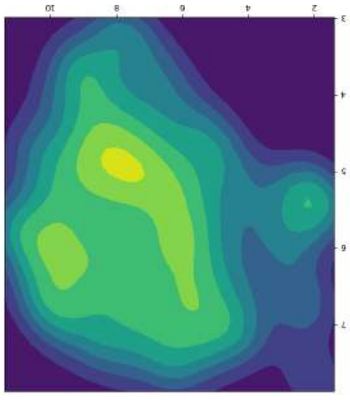
(e) Nearest Sample Removal - 50% DD



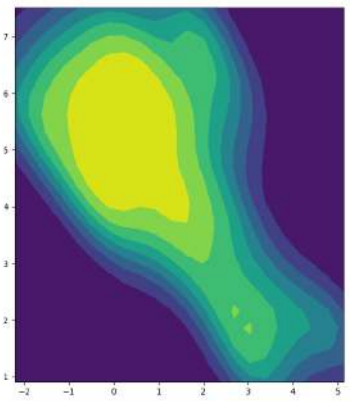
(f) Nearest Sample Removal - 90% DD



(g) Furthest Sample Removal - 0% DD



(h) Furthest Sample Removal - 50% DD



(i) Furthest Sample Removal - 90% DD

5 Feature Extraction

5.1 Introduction

Feature Extraction is another element of the MANOLO functionality directed towards the enhancement of AI efficiency. This submodule will provide the user with a set of tools and functions to extract relevant information from the data and represent it in compact features. This results in a memory and computationally efficient representation of the dataset that contains the necessary information to replicate results obtained with the original data.

Early approaches to feature extraction leverage domain specific hand crafted features like color, texture or shapes for images [33]. These approaches presented poor generalization across datasets and required time-consuming human supervision. Hence, current state-of-the-art has moved towards the exploitation of pre-trained models to extract features from the data [23, 34, 35]. Pre-trained models are exposed to a high variety of samples from different domains during training and learn to extract the most relevant features. As a widely adopted practice in the Computer Vision community, practitioners use ImageNet [22] pre-trained models as an initial data pre-processing step to obtain informative and memory efficient features as representations of the samples. The adaptation of these features to the target domain depends on the similarity between the target and the pre-training, or source, domains [36]. The direct solution to feature extraction when working with specialized datasets is to train models directly on the target dataset. The main caveat of this approach, apart from increased computational costs, is the reliance on dataset quality; concretely, on annotations quality. An effective approach in this case, is to leverage models that learn the distribution of the data independently from the labels. Variational Auto-Encoders (VAE) are a widely adopted example of these models [37, 38, 39]. Finally, recent advances focus on larger models trained on vasts amounts of data that provide more descriptive and discriminative features at the cost of more computational requirements during training and inference [40, 41].

At the time of this deliverable, this section presents two basic techniques included in the Feature Extraction MANOLO submodule: a pretrained Neural Network (NN), specifically a Convolutional Neural Network (CNN), and a complex Auto-Encoder (AE) architecture, specifically a Vector Quantized Variational AE (VQ-VAE). These two techniques have the advantage of being easily scalable to different datasets and types of data. The CNN-based feature extractor is a very generic approach that leverages pre-trained weights and is able to accommodate different types of models to address different types of data and modalities. The VQ-VAE-based approach learns the distribution of the data through a training stage that we then use to extract representative features.

The implementation we are presenting in this version of the deliverable is designed for time-series and image data but simple adaptations in the pre-processing stage allow the models to generalize to other types of data, which will be included in future versions of the deliverable. These two approaches are well suited to exploit synergies with explainability metrics [42, 43], which we will also address in the next version of this deliverable. The following subsections will delve into these approaches separately, explaining the associated development methodology, and the results obtained up to this stage.

5.2 Pre-trained Neural Network as a Feature Extractor

The initial iteration of the Feature Extraction MANOLO submodule leverages pre-trained NNs to extract features from data, their discriminative capabilities and adaptability across data types and modalities make these models perfect candidates for this task. There are numerous models trained in diverse and large datasets that can be easily downloaded and used in particular tasks. The models included in this version of the report are Convolutional Neural Networks (CNNs) pre-trained in Computer Vision classification tasks with the ImageNet dataset. However, the range of datasets used for pre-training CNNs is wide, spanning from classification to semantic segmentation. It is worth noting that these features, as it happens with raw data, can be used to train models intended to perform tasks unrelated to that used to train the feature extractor.

The feature extraction procedure presented in this version of the deliverable intends to obtain a reduced representation of a sample as the byproduct of being processed by a CNN. In this manner, we use CNNs as functions that map samples into lower dimensional vectors. The performance achieved when replacing the original dataset with the extracted features heavily depends on the relationship between the target task and the task used to pre-train the feature extractor [36]. Nonetheless, these extracted features provide a valuable resource-efficient alternative to the full dataset and a competitive starting point for a wide variety of applications and tasks.

5.2.1 Methodology

We establish a baseline approach to feature extraction by the use of CNNs trained on the 1,000 classes ImageNet classification task, predicting a class probability distribution after the last layer, linear, has its logits activated through a softmax function. In particular and for this report, we focus on two types of architectures: VGG [44] and ResNet [45]. To obtain more generic features we discard the last linear layer and its activation function and directly use the output of the previous residual block for ResNet architectures, or linear layer for VGG architectures.

Feature evaluation

In order to evaluate the quality of the features extracted, we have followed a method widely adopted in self-supervised tasks, consisting in training two classifier networks, a linear classifier [46] and a KNN classifier [47], with the extracted features. The linear classifier proves the features' ability to linearly separate classes, and the KNN the quality of the feature space in terms of associating sample similarity to semantic class.

Furthermore, as we aim to create compact versions of the dataset by extracting the features from the data, we have also included a non-linear evaluation test for feature fitness for complex classifiers, in terms of model generalization and adaptability. The architecture of the third evaluator is that of a non-linear classifier composed by two linear layers with a ReLu activation, creating non-linearity between them. Additionally, we include a qualitative evaluation showing the ability of the feature extractors to allocate similar samples close in the feature space. This is a retrieval-based evaluation that, as illustrated later in Section 5.2.2 in Figure 10, shows the closest samples to a few randomly selected query samples. This evaluation is done by measuring distance between samples with the cosine similarity metric.

Experimental setup

The experiments in this subsection are carried out on the CIFAR-10 dataset for image classification. This dataset consist of 60,000 natural images of 32×32 RGB pixels distributed across ten classes and split into a set of 50,000 samples for training and a set of 10,000 samples for testing. Each of the experiments is done for three different NN architectures: VGG [44], VGG with batch normalization, and ResNet [45]. In each case, we evaluate two different model sizes: for the VGG architecture we chose VGG-11 and VGG-19, and for the ResNet architecture we chose ResNet-18 and ResNet-50. Table 21 summarizes the difference in number of parameters between these models. Additionally, given that the selected models are initially trainedo on 256×256 resolution images, we evaluate the performance of featres extracted from an upsampled version of CIFAR-10 images.

5.2.2 Results

The results reported in this section evaluate the fitness of certain CNNs to be part of the MANOLO Feature Extraction submodule. Concretely, the experiments carried out provide a comparison of the feature quality when using different pre-trained and randomly initialized models as feature extractors. The results reported

Table 21: Model specifications: number of parameters in millions and feature size.

Architecture	Number of parameters (M)	Feature size
VGG-11	128.77	4,096
VGG-11 + BN	128.77	4,096
VGG-19	139.57	4,096
VGG-19 + BN	139.58	4,096
ResNet-18	11.18	512
ResNet-50	22.51	2,048

in Table 22 show the reliability of this approach across architectures and the consistency of the results when changing the number of parameters of a model. In particular, Table 22 provides a baseline accuracy of the models directly trained on the CIFAR-10 dataset, and three different accuracy evaluations of the performance of features extracted from the original 32×32 images and of the 256×256 upsampled images. Additionally, Figure 10 provide a qualitative evaluation that demonstrates the ability of the extracted features to preserve visual similarity between samples. The figure illustrates this by depicting a query image in the first row and its ten closest samples according to the cosine similarity of the extracted features. Note that the top right pane in Figure 10 shows the closest images when the features are extracted with randomly initialized weights and the top left pane when the features are extracted with a pre-trained model.

Table 22: Accuracy metrics (%) of features extracted with pre-trained weights evaluated in the CIFAR-10 test set for the original image size and for upsampled images.

		32×32 images			256×256 images		
Architecture	Baseline	KNN	Linear	Non-linear	KNN	Linear	Non-linear
VGG-11	91.90	60.12	61.61	66.02	75.79	85.16	84.86
VGG-11 BN	91.61	61.56	66.12	68.96	75.89	85.87	86.60
VGG-19	94.50	58.16	61.96	64.69	77.15	84.88	84.99
VGG-19 BN	94.40	60.56	64.37	67.43	80.31	86.97	87.32
ResNet-18	94.94	56.97	66.90	69.88	80.97	86.36	87.87
ResNet-50	95.41	59.98	70.51	71.44	83.03	89.48	90.44

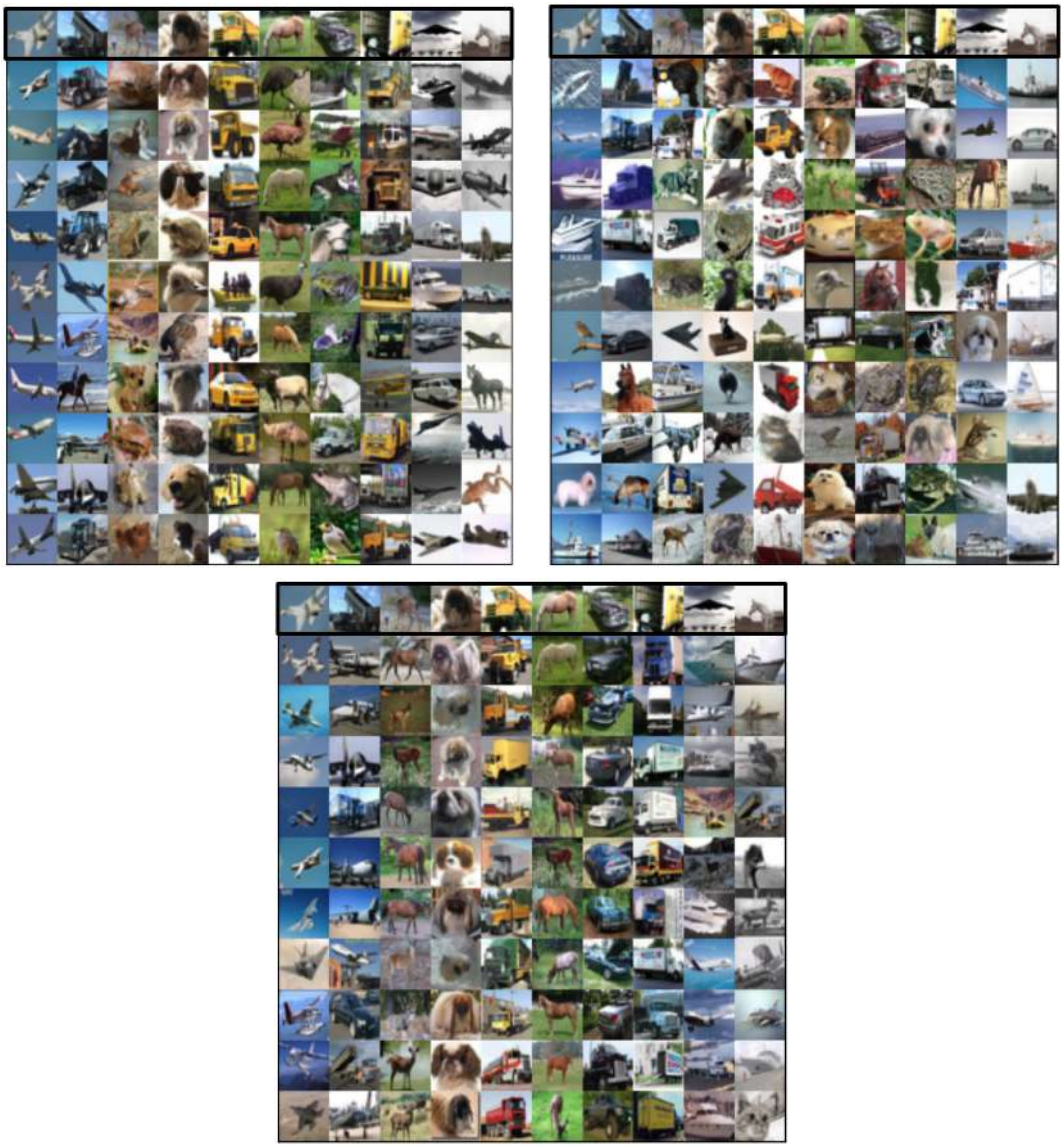
The first evaluation of the extracted features, by means of the performance of a linear classifier trained on the features extracted from the different pre-trained models, presented in Table 22, range from 61.96% accuracy to 70.51% in VGG-19 and ResNet-50 when 32×32 images are used and from 84.88% to 89.48% for 256×256 images. Similar results are obtained from the KNN and nn-linear evaluation. This is far from the state-of-the-art results reported for CIFAR-10 when training the same models from scratch, especially when using 32×32 resolution images. The results of the models trained from scratch in CIFAR-10 are reported under Baseline in Table 22 and typically surpass 90% of accuracy. The computational cost, however, is drastically lower when using pre-trained weights. It is worth noting that re-scaling CIFAR-10 images to 256×256 pixels provides better results at the cost of higher computational demands: a ResNet-18 on a MacBook Pro with an Apple M3 Max chip and a batch size of 256 takes 95.45 seconds to extract the features of 256×256 rescaled CIFAR-10 images and achieves 87.18% of accuracy in the linear evaluation, while the same model on the original 32×32 images takes 5.72 seconds and achieves 66.9%. The bottom pane in Figure 10 provides the qualitative evaluation of the features from the rescaled image. Note that the size of one of the color resized images used is 256×256×3 resulting in roughly 196,000 values while the largest of the features used in this version of the deliverable is 4,096.

Finally, we compare the results from pre-trained models with the results of randomly initialized models in Table 23. The results obtained from randomly initialized models, are considerably lower; while these are still higher than random chance (10% accuracy), the difference clearly motivates the adoption of pre-trained models and further research in the adaptation of feature extraction approaches across domain and tasks.

Table 23: Accuracy of features extracted with randomly initialized weights in the CIFAR-10 dataset.

Architecture	KNN acc. (%)	Linear acc. (%)	Non-linear acc. (%)
VGG-11	41.10	39.48	45.24
VGG-11 BN	41.08	39.48	45.27
VGG-19	37.79	27.39	33.07
VGG-19 BN	37.78	27.39	33.05
ResNet-18	34.98	41.16	43.70
ResNet-50	33.89	35.96	10.00

Figure 10: Qualitative evaluation of features: The top row of each set of samples correspond to a query image and the ten rows beneath to the ten closest samples. The features were extracted with a ResNet-18 with pre-trained weights (top left images), randomly initialized weights (top right images), and pre-trained weights on rescaled images (bottom images).



5.3 TimeVQVAE for feature extraction of time-series data.

This section describes the approach used to expand the feature extraction capabilities of MANOLO to the time-series domain: variational auto-encoders (VAEs). These are a valuable alternative to extract features in

applications where labeled data are scarce, pre-trained models are not available, or the target task is very specific and knowledge transfer is not effective. Concretely, we use TimeVQVAE, the vector-quantized variational auto encoder (VQ-VAE) proposed by Lee et al. [37].

Subsection 5.3.1 briefly introduces the VQ-VAE model as a VAE, explaining its value as a feature extractor to be included in the MANOLO Data module, and provides details on the experimental setup used and the evaluation process followed to obtain the results reported in Subsection 5.3.2.

5.3.1 Methodology

VQ-VAEs were originally designed for image generation as an improvement of VAE that simplified training and improved the quality of the generated samples [39]. These generative models are trained in an unsupervised manner to reconstruct the input samples, making them independent from possible perturbations in the labeling process. TimeVQVAE as proposed in [37], and adopted in this submodule, is an adaptation to time-series data that outperforms contemporary alternatives. Its latent space generates discriminative and representative features that provide a reduced memory-efficient version of the original dataset.

VQ-VAEs are a quantized version of traditional VAEs with a discrete latent space that allows for more realistic sample generation. The particular implementation adopted in MANOLO, TimeVQVAE, consists of two training stages. In the first stage an encoder block maps samples to a latent space, a quantizer discretizes the latent space into learned tokens, and a decoder block maps vectors from the latent space back to the sample space. This stage is trained with a reconstruction loss that assures the generation of realistic samples and an MSE minimization loss that optimizes the tokens learned by the quantizer. Then, the second stage uses a bi-directional transformer to learn the prior distribution of the latent space. TimeVQVAE, as VAEs do, produces an output that resembles the input sample while passing through a latent space of reduced dimensionality. This compression of the representations, and quantization in the case of VQ-based models, guides VAEs to learn a mapping function, the encoder, that preserves the most relevant features to reconstruct the input sample.

Feature evaluation setup

To repurpose TimeVQVAE for the MANOLO feature extraction functionality, we leverage the latent space learned by the encoder and use the latent vectors as extracted features. Hence, the trained encoder of the TimeVQVAEs becomes the feature extractor model. Additionally, to homogenize the features and further compress the data, we use a PCA and keep the 50 most relevant components of the extracted features. The evaluation of these features is carried out following the procedure described in Section 5.2.1, i.e. we report accuracy with a linear classifier, with a non-linear classifier, and with a KNN-classifier. In addition, due to the class imbalance in the datasets studied in this section, we report the balanced accuracy metric, as implemented in the scikit-learn library [48], for these three classifiers [1]. This metric reports a more accurate measure of the performance of a model when the number of samples in each class is imbalanced.

Experimental setup

The TimeVQVAE implementation integrated in MANOLO closely follows the architecture and adopts most of the hyperparameters suggested in the original paper [37]. To increase the data compression capabilities of these technique, we reduced the size of the model by decreasing the size of the hidden dimension and increasing downsampling of the samples. The features extracted are the flattened vectors produced by the encoder. Given the convolutional nature of the encoder, the dimensionality of the resulting features depend on the size of the input samples. To obtain same size features across datasets we post-process the features with a PCA dimensionality reduction and obtain 50 component features, this achieves an additional compression of the data.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html

We use four time-series datasets easily accessible from the tslearn library [49]: ECG5000, ElectricDevices, FordA, and FordB. For all the datasets we used the same pre-processing of the data: Min-Max normalized all the samples and rounded all the values to two decimals. We followed the train-validation split defined in the tslearn library and used the train set to train the VRAE and the classifiers for the evaluation, then computed the reported metrics on the validation set.

The ECG5000 dataset [50] is constituted by 5,000 randomly selected heart bits from a patient with severe congestive heart failure automatically annotated as one of five classes. The distribution of samples is heavily biased towards the first and second class, which have over 4,000 samples between them while the third, fourth, and fifth classes have less than 500 samples altogether. This dataset is often used in medical time-series research for anomaly detection and classification tasks [51, 52, 53], where the 5,000 samples are usually split into 4,500 samples for testing and 500 for training. The imbalance of the data labels makes it a challenging dataset well-suited to one of the purposes of MANOLO: address possible biases. Each sample has a length of 140 measurements of a single variable.

The ElectricDevices dataset, available in the UCR time-series archive [54], is a collection of electricity consumption readings from 251 households sampled in two-minute intervals over a month, resulting in 16,637 samples split into 8,926 samples for the train set and 7,711 for the test set. The samples are annotated into one of seven classes. In this case, five of the classes have around 1,000 samples each while the other two around 2,000 each. This illustrates a less severe type of class imbalance when compared to the ECG5000 dataset. ElectricDevices is a univariate dataset and the length of each sample is 96 measurements. It is commonly used in time-series classification, forecasting, and explainability among others [55, 56, 57, 58].

Finally, the FordA and FordB datasets, also from the UCR archive [54] are a collection of samples from an automobile subsystem labeled to identify the presence of a particular symptom resulting in a binary classification task. The first dataset is split into a set of 3,601 samples for training and 1,320 for testing, and the second dataset into 3,636 and and 810 for training and testing respectively. All the samples consist of 500 measurements from a single sensor (univariate time series). Additionally, FordB dataset presents a more challenging task than FordA due to the inclusion of noise in the measurements in the test set. These datasets are used in time-series research as benchmarks for binary tasks as well as robustness against noisy measurements [59, 60, 61].

5.3.2 Results

We have evaluated the latent space learned by the TimeVQVAE both qualitatively and quantitatively. The former consist of the standard evaluation used for self-supervised representation learning, described in Section 5.2.1. In this case we use the encoder of the trained TimeVQVAE as a feature extractor and represent each input sample as a vector from the latent space. The latter consist of a PCA and t-SNE visualization of the latent space that illustrates the ability of the encoder to allocate samples from a given class in a particular region of the space, which showcases its potential applications to other tasks such as retrieval or metric-based learning.

Table 24 shows the results of a linear classifier, a KNN classifier and a non-linear classifier trained on the features from the training set extracted with the encoder of the TimeVQVAE, and evaluated on the test set with the available labels. Table 25 reports the performance of the same evaluation procedures under the balanced accuracy metric. This provides an accurate indication of the ability of the TimeVQVAE to extract class-discriminative features. An initial comparison baseline for these results is the performance of random features, for five, seven, and two classes, this would correspond to 20%, 14%, and 50% , reported in Table 25 under Random. Additionally, under Supervised, Table 25 includes the state-of-the-art results when training a model for classification with the available labels as reported in [59, 55]. Finally, the significant decrease in performance when comparing Table 25 and Table 24 illustrates the impact that imbalanced classes have in the training of feature extrac-

tors, which will be addressed by approaches introduced in the MANOLO Data Synthesis module described in Section 6.1 of this report. Note that, given that the class balanced accuracy is not commonly reported in the literature, this table does not include the supervised baseline.

Table 24: Accuracy (%) of features extracted with the TimeVQVAE encoder.

	Random	Supervised	TimeVQVAE		
Dataset			KNN	Linear	Non-linear
ECG5000	20.00	94.62	95.00	93.90	94.40
ElectricDevices	14.00	64.00	68.93	65.35	70.58
FordA	50.00	96.44	77.06	82.13	86.29
FordB	50.00	92.86	59.66	54.38	53.37

Table 25: Class-balanced accuracy (%) of features extracted with the TimeVQVAE encoder.

	Random	TimeVQVAE		
Dataset		KNN	Linear	Non-linear
ECG5000	20.00	54.93	46.59	47.72
ElectricDevices	14.00	59.50	57.35	62.88
FordA	50.00	76.70	81.81	86.10
FordB	50.00	59.55	54.01	53.26

Figure 11 provides the two dimensional t-SNE and PCA visualizations of the extracted features. The image shows that features belonging to the same class align in the feature space. In particular, for ECG500, the visualization show that the two classes with the larger number of samples are separated. The classes with fewer samples also seem to fall in close regions of the space but overlapping the bigger clusters. It is worth noting that this is only a visualization method and not a reliable manner to evaluate the performance of a classifier. The techniques used (PCA and t-SNE) project the features in two-dimensional plots and this remove nuances and details present in the original features in higher dimensional spaces.

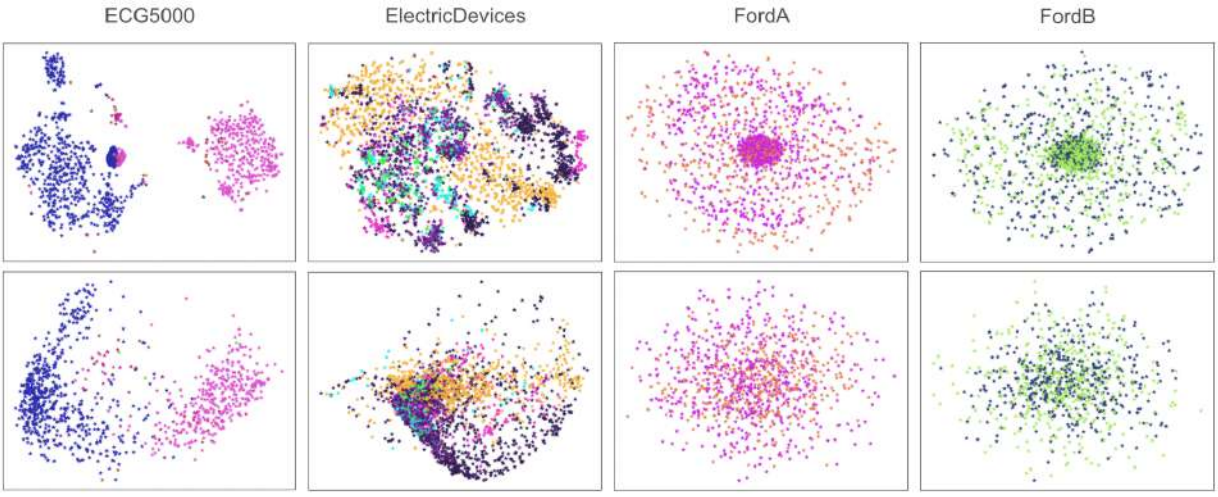


Figure 11: T-SNE (top) and PCA (bottom) visualization of the features extracted with the TimeVQVAE encoder from the four datasets used in this report.

6 Data and Feature Synthesis

6.1 Introduction

AI practitioners training models for classification, forecasting, or more complex tasks usually find that their datasets lack balance in the representation that certain attributes or classes have, which can cause biases during inference time, or affect the robustness of the models. Thus, MANOLO will provide tools to generate synthetic data, or synthetic features (following the philosophy of the previous section), to make sure that the model training and optimization phase increases its trustworthiness in another additional manner.

Data synthesis approaches can be grouped in two main categories: knowledge based and model based approaches. Knowledge based approaches utilize traditional engineering approaches and require the expert knowledge of humans to design strategies to generate new samples [62, 63, 64]. These approaches are time consuming, require human intervention and are usually limited to certain domains and tasks. Model based approaches use trained models that are wether trained in the target dataset for the specific domain of the target application [65, 66, 67], or trained on large datasets that often combine multiple modalities [68, 69, 70]. The model based approaches outperform the knowledge based in data generation: provides more variety, higher sample quality, and less human intervention required. Note that the computational costs of this approaches is overall higher and increases as model size increases.

At the time of this deliverable, this section presents two initial frameworks for conditional sample generation that will be include in the Data Generation MANOLO submodule: invertible neural networks and generative adversarial networks. For a better evaluation of these frameworks, this section currently focuses on image datasets. The first framework introduced in this deliverable, invertible neural networks, is an efficient solution to data generation that provides a considerable degree of freedom in the conditioning stage allowing the user to alter the class and style of the selected samples. The second framework, generative adversarial networks, provides additional versatility in the selection of model architecture, which facilitates the scalability of the models to larger datasets and different modalities.

6.2 Conditional INNs

Ideally, it would be very convenient to train a simple classification NN model with a given dataset and then input a label (previously the target) so as to generate a new data sample (previously the input). Unfortunately, NNs are not invertible by definition, and need to be designed for this specific purpose. This is where Invertible Neural Networks enter. These NNs, bijective by definition, have a constrained set of possible layers, which exclude batch normalization and pooling, for instance, and allow the generation of samples from a desired label. We take a step forward with the use of Conditional Invertible Neural Networks (CINNs), a variation of invertible neural networks that provides the possibility of selecting the class of the generated sample. Additionally we include a methodology to furhter condition the sample generation stage and allow the user to tweak the style of the generated samples. In this subsection we address data generation with the conditional invertible neural networks proposed by Ardizonne et al. [67].

When conditional generation is introduced, INNs are trained to map an input sample to a latent space while being conditioned by the class of the sample. Then, at inference time, the inverted model can be used as a unit that processes random vectors from the latent space and produces unseen samples in the input space. This inversion procedure is conditioned by the sample class to constrain the generation to a particular class. In the case of computer vision, for instance, INNs map an image from the image space (RGB representation) to a latent vector. This deliverable will show an initial exploration of class and style conditioning alternatives. These approaches allow the model to generate class conditioned images with a particular style living in the latent space.

6.2.1 Methodology

This subsection describes the technical aspects of the approach introduced in this version of the deliverable: the CINN proposed by [67]. This approach uses linear neural networks and conditions the sample generation with class labels. This subsection also includes the adaptation we proposed to condition the style of the generated samples, and a description of the experimental setup used to obtain the results reported in the following subsection.

Class-conditioned sample generation.

Sample generation with INNs, as with normalising flows [71], is based on the principle behind the change of variable statistics formula

$$p_X(X) = p_Z(f(X))|\det Df(X)|^{-1}, \tag{6}$$

where $p_X(X)$ and $p_Z(f(X))$ are the probability distributions from the input and the latent space respectively, $\det Df(X)$ is the determinant of the Jacobian of $f(X)$, and $f(X)$ is the model or mapping function that is going to be learned. We train this model, as is most commonly done, via maximum likelihood. Following the mathematical developments by Ardizonne et al. [67], which include the prior conditioning of the model with the class of the input sample, this results in the minimization of three terms: the squared module of the model prediction, the logarithm of the Jacobian of the model weights, and a regularization to enforce a Gaussian space being learned.

Class-conditioned sample generation relies on a prior introduced during training and inference that alters the mapping and constricts it to a particular region of the latent space, the region that belongs to the samples under the given condition. The prior used for class-conditional sample generation consist of a one-hot-encoded vector representing the semantic class of a sample. This is shown in Figure 12, where the input z_i is a random Gaussian noise vector, the condition c_i is a vector indicating the class "3" in this particular example, and x_i, in this case the output of the inverted model, is a newly generated image of a digit "3". By using different noise vectors as seeds for the generation of samples, the model outputs different new samples all corresponding to the category indicated by the condition vector.

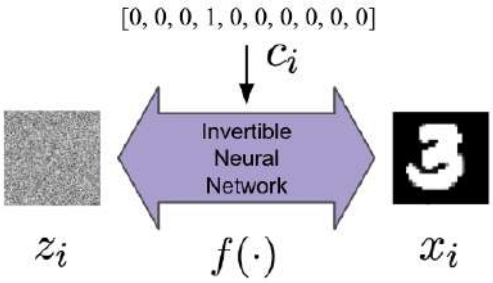


Figure 12: The invertible neural network $f(\cdot)$ generates an unseen sample x_i from a Gaussian noise vector z_i given a class condition c_i and vice-versa.

Style-and-class-conditioned sample generation

Style-and-class-conditioned sample generation goes a step farther, assumes that the class condition is fixed, and conditions the sample generation to adhere to a particular style. We define the style by exploring the latent space learned by the CINN and select a set of seed noise vectors near the area of interest, i.e. the noise vectors corresponding to a set of samples that exhibit the style in which we are interested. Since different latent vectors result in different images being generated, by moving a random noise vector towards a region in the latent space where a particular set of samples belong, we condition the generation of samples. This is depicted Figure 13.

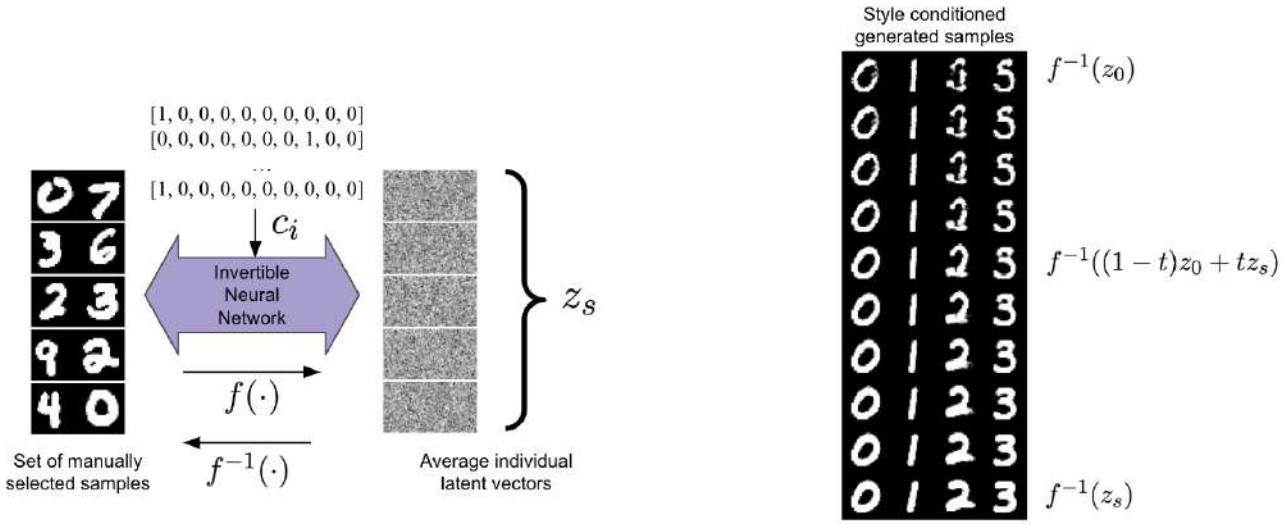


Figure 13: The latent vectors of a manually selected set of samples are averaged to create the latent vector z_S that defines the style (left). Through a weighted average of the style vector and the seed random vector, the CINN regulates the strengths of the style-conditioning (right).

To define the region of the latent space that contains the style we want, we define a style vector z_s that represents the style we are interested in. We do this by manually selecting K samples that share the particular style we want (bold, italic, faint writing in the MNIST example) and mapping them to the latent space, resulting in a latent style vector per sample $\{z_{s1}, z_{s2}, ..., z_{sK}\}$. Then we define the style vector as the centroid of the individual style vectors

$$z_s = 1/K \sum_i z_{si}. \tag{7}$$

Finally, we generate samples using the interpolation of z_s with a random Gaussian noise vector z_i as a conditioned seed vector

$$z_c = (1-t)z_i + tz_s \tag{8}$$

that falls close to a region in the latent space that encodes a particular style. Figure 13 illustrates this process with the "bold" style for the MNIST dataset. The strength of the interpolation modulates how close the z_c vector is to the style vector z_s and, consequently, how strong will the presence of the particular style be in the generated image.

Experimental setup

The sample generation functionality for the Data Inspection and Generation MANOLO submodule is currently developed and tested on the MNIST [72] and Fashion-MNIST [73]. These are two small datasets, widely used in the research community that allow for faster experiemtnation. MNIST contains 60,000 black and white 28×28 images of hand-written digits group into ten different classes corresponding to the digits 0-9. Fashion-MNIST is a more challenging dataset with very similar characteristics: black and white 28×28 images, 60,000 for training and 10,000 for testing, distributed across 10 classes that belong to the retail domain each containing one clothing item.

The experiments presented in this section use the CINN model proposed in [67], composed of 20 linear layers as implemented in the FrEIA library [74]. Hence, the images are flattened into a 784 dimensional vector in the input of the model and then unflattened back at the output. The INN is trained on 256 samples batches with Adam optimizer and a learning rate of 10^{-5} that we reduced by a factor of 10 at epoch 20 and epoch 40 of a 60 epochs training.

6.2.2 Results

Class-conditioned sample generation

The experiments on class-conditional sample generation illustrate the ability of a CINN to generate samples from a given class. Figure 14 and 15 provide examples of images generated by a CINN for each class in the MNIST and Fashion-MNIST dataset respectively.

Figure 14: Samples generated by the CINN for each of the ten classes in the MNIST dataset.

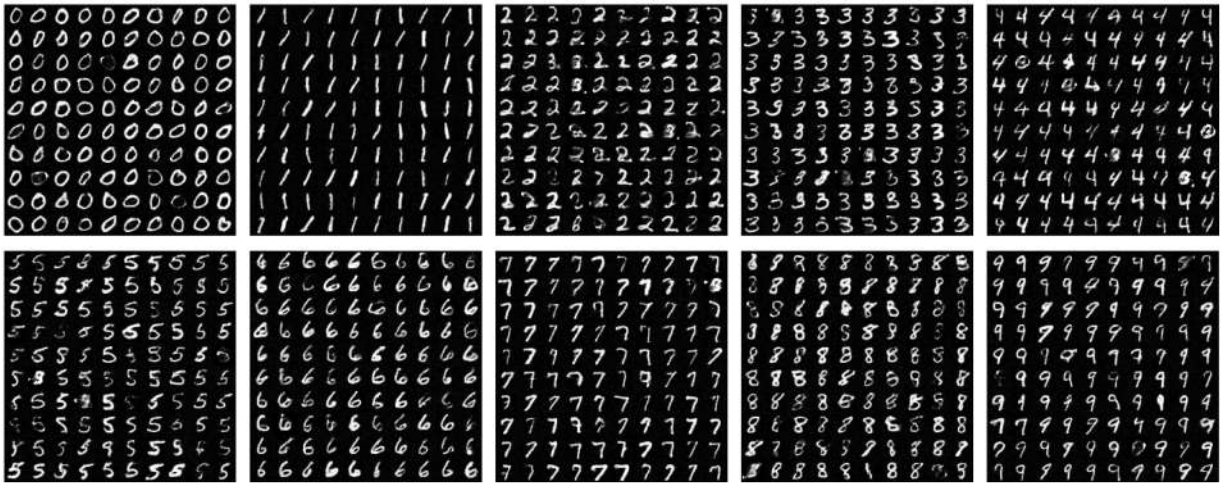


Figure 15: Samples generated by the CINN for each of the ten classes in the Fashion-MNIST dataset.

Style-and-class-conditioned generation.

The results in this section demonstrate how we further condition the sample generation process to enforce a particular style in the generated samples. Figures 16 and 17, provide examples of style-and-class conditioned generated samples and the corresponding initial seeds used for the MNIST dataset. Similarly, Figure 18 provide the generated samples for Fashion-MNIST, in this case we only use a single image to define the style due to the large variety in styles present in the dataset. These figures help illustrate the generation process: the left side of the figures contain samples manually selected that belong to a particular style, bold and italic for MNIST and striped for Fashion-MNIST. Then, the images to the right of the figures correspond to the samples generated for the ten different class conditions (digits from zero to nine in MNIST and the different retail categories in Fashion-MNIST), and each row from top to bottom, correspond to an increasing level of style conditioning strength. The top row samples are generated from a completely random seed vector, the bottom row samples from the style vector obtained from the manually selected samples, and the rows in between interpolations of these vectors. The strength of the interpolation,

and of the conditioning consequently, is defined by a parameter t and in these results is linearly and increased between zero and one.

Figure 16: Style-and-class-conditioned samples for the bold style in the MNIST dataset.

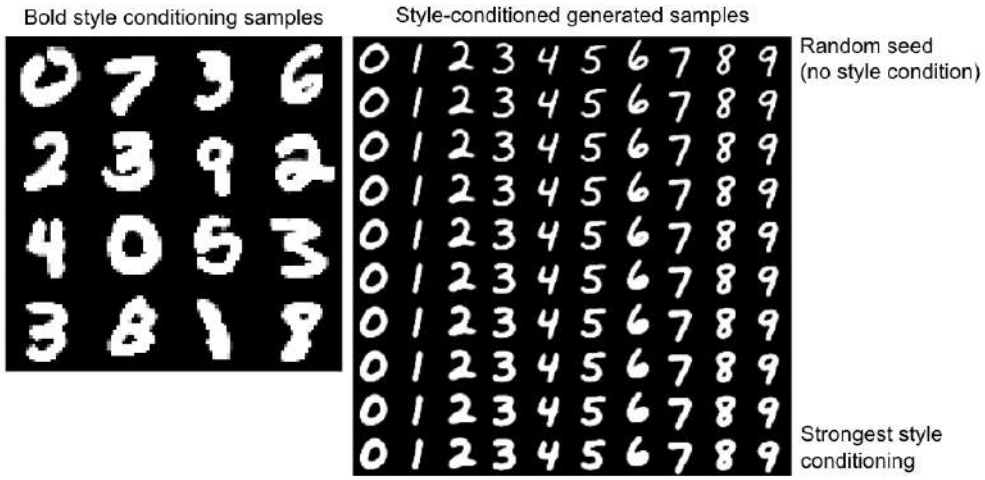


Figure 17: Style-and-class-conditioned samples for the italic style in the MNIST dataset.

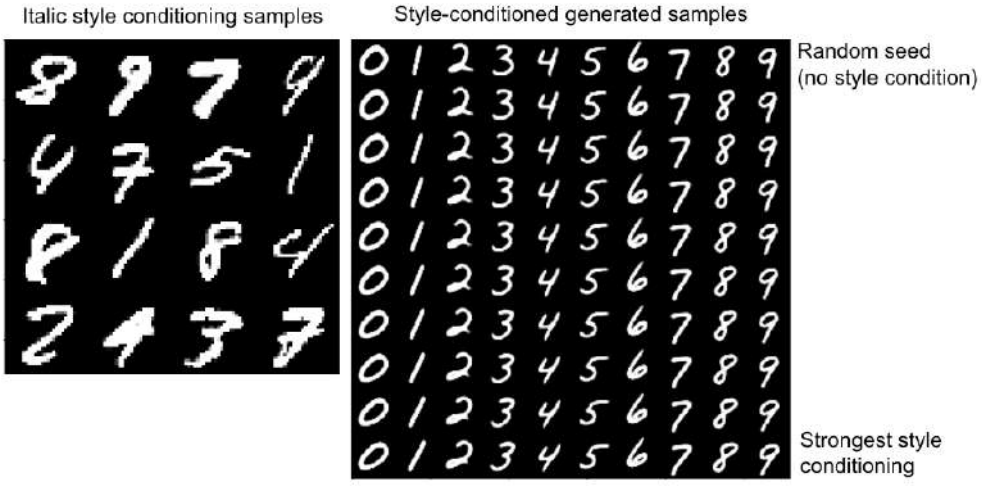
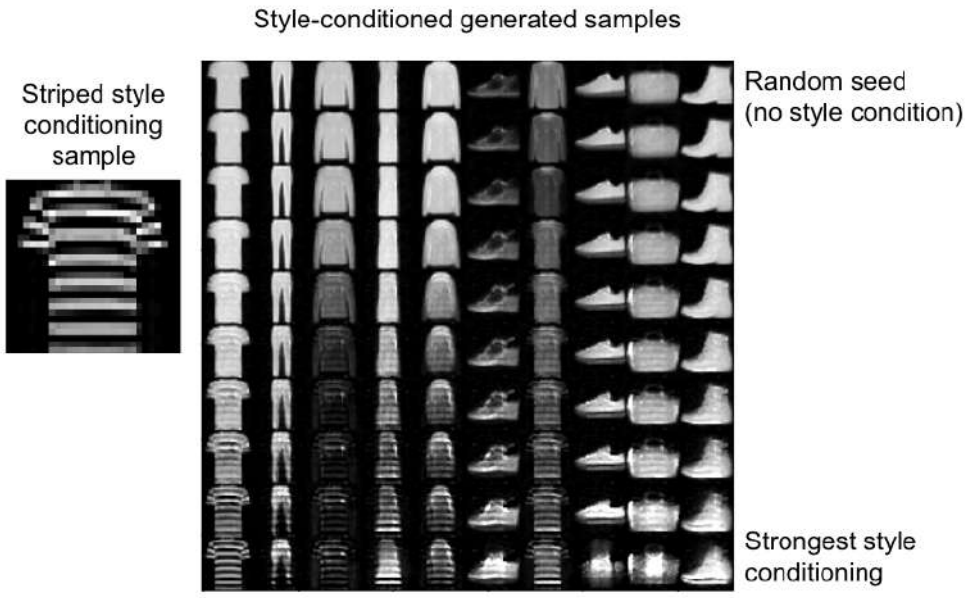


Figure 18: Style-and-class-conditioned samples for the striped style in the Fashion-MNIST dataset.

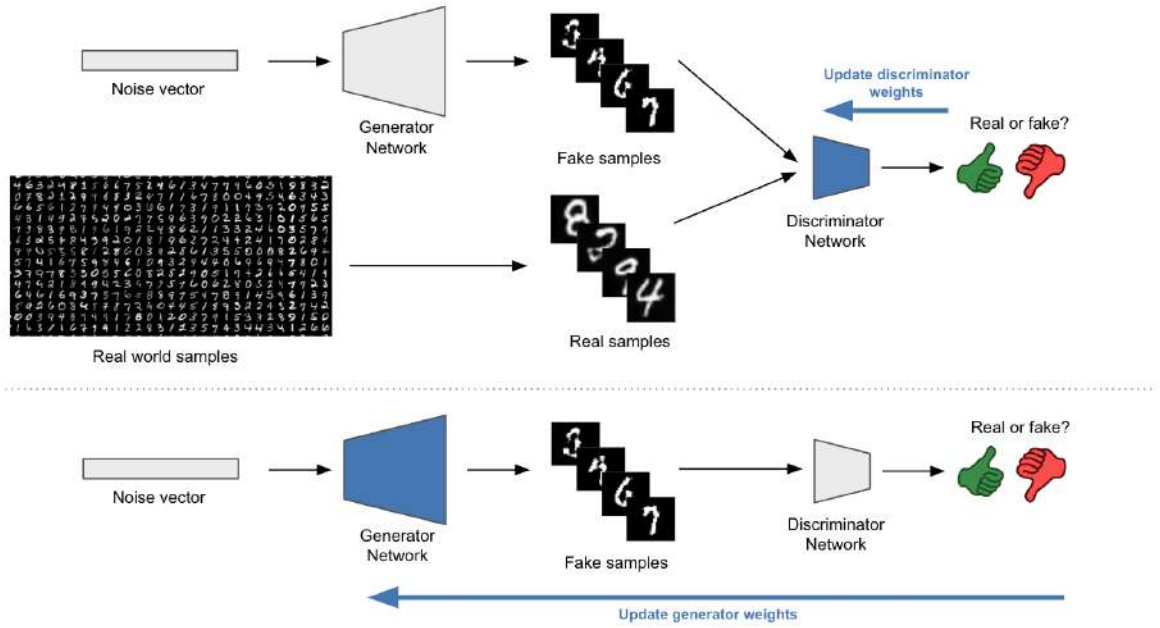


6.3 Generative Adversarial Networks

Data generation requires custom models to address different datasets and domains, which is particularly challenging when models are restricted to a limited number of architectures, as is the case in invertible neural networks 6.2. As a solution to this challenge and given their high quality results, Generative Adversarial Networks (GANs) [65] have become widely adopted by the machine learning community. These models provide a versatile framework that accommodates a wide range of architectures allowing for a broader applicability. Additionally, the strategy used when training GANs require relatively low computational resources. Note that, while GANs are successful in generating images, they do not inherently present the style conditioning capabilities that other models do (see Section 6.2).

The GAN framework presented in this section is constituted of two neural networks, a generator and a discriminator, that are trained in an adversarial fashion: the generator is optimized to generate realistic samples and the discriminator to distinguish between real and generated samples. This creates a competition between models that forces them to improve without external supervision. As illustrated in Figure 19, the training process iterates between two stages: in the first stage the discriminator's parameters are optimized to better identify the fake or generated images and in the second stage the generator's parameters are optimized to outsmart the discriminator by making samples that look as real as possible.

Figure 19: GAN training framework. In stage one (top) the discriminator is trained and in stage two (bottom) the generator is trained. Blue color is used to represent the blocks being trained in each stage.



The initial GAN-based models introduced in the MANOLO data synthesis module for class-conditional sample generation are the Conditional GAN (CGAN) [75], and the Deep Convolutional GAN (DCGAN) [76]. The first one is composed of linear components and is used for smaller datasets where the model easily captures the features of the data while the second one includes convolutions to address more complex data.

6.3.1 Methodology

This subsection introduces the methodology followed in this version of the deliverable to generate samples with GAN-based models conditioned to a particular class. The generator network in GANs learns to map an input noise vector in a fixed latent space to the sample space resulting in a newly generated sample. By concatenating a representation of the class label to the input noise vector, the generator network is conditioned to generate a sample belonging to a particular class. This label representation could be a one-hot-encoding

of the label as done by Mirza and Osindero [75] to condition the CGAN model, or more complex alternatives including linear or embedding layers, as we do in this version of the deliverable to condition the DCGAN model.

Experimental setup.

Following the methodology in Section 6.2 we develop and test the CGAN model in MNIST and Fashion-MNIST. Additionally, to test the scalability capabilities of the GAN-framework we use CIFAR-10 [77] for the DCGAN model. CIFAR-10 is an RGB natural-images dataset with 60,000 32×32 samples divided into 10 classes and split into a set for training with 50,000 samples and a set for testing with 10,000.

The CGAN model used in this section is constituted by a generator network with four linear layers followed by LeakyReLU non-linearity and a discriminator network with four linear layers followed by LeakyReLU non-linearity and dropout layers with a probability of 0.3. We use a 100-dimensions noise vector and condition the model by concatenating the noise vector to the one-hot-encoding of the labels. The DCGAN model, on the other hand, is constituted by a generator network with three transpose convolution layers followed by batch normalization and ReLU non-linearity layers and a last transpose convolution layer followed by a tanh non-linearity layer. Then the class conditioning is done by projecting the noise vector to a $7 \times 7 \times 128$ dimension space and concatenating it to a $7 \times 7 \times 1$ projection of a learned 50-dimensions embedding of the labels. This results in a $7 \times 7 \times 129$ input vector for the generator. The discriminator, then, consist of four convolutional layers followed by batch normalization and LeakyReLU non-linearity layers, followed by a final linear layer and a sigmoid non-linearity layer. The conditioning of the discriminator is done in a similar fashion, where the labels are projected to a $32 \times 32 \times 1$ space and concatenated to the input samples, both real and newly generated. Both models are trained with the Adam optimizer with learning rate of 10^{-4} on 128 samples batches for 200 epochs.

6.3.2 Results

The results showed in this section illustrate the effectiveness of GAN to generate images. Figure 20 and 21 provide images generated with a CGAN for MNIST and Fashion-MNIST respectively. This visualization show that the model successfully generate images according to the specified class. We also notice that the quality of the images is still lower than the quality of the real images found in the dataset, this could be improved with further exploration of the hyperparamter space (e.g. larger training time or deeper models or different optimization strategies). Then, Figure 22 provides images generated with the DCGAN model for the CIFAR-10 dataset. These images also show how the generator of the DCGAN captures the general distribution of the data and is able to reproduce CIFAR-10-like images. Note that, when examined in more detail, these images depict parts of objects and realistic shapes that resemble visual features present in the original dataset.

Figure 20: Samples generated by the CGAN for each of the ten classes in the MNIST dataset.

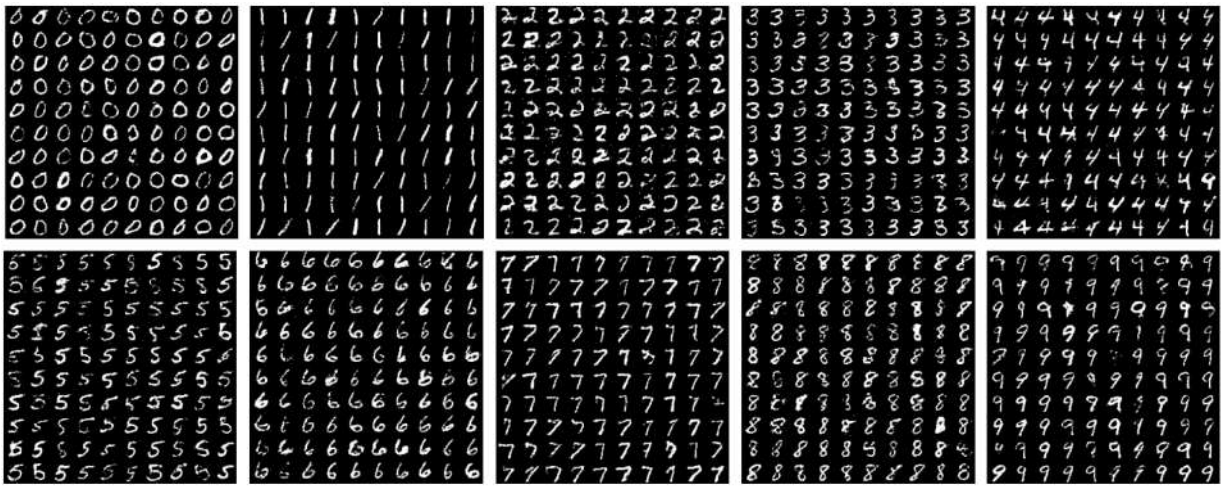


Figure 21: Samples generated by the CGAN for each of the ten classes in the Fashion-MNIST dataset.

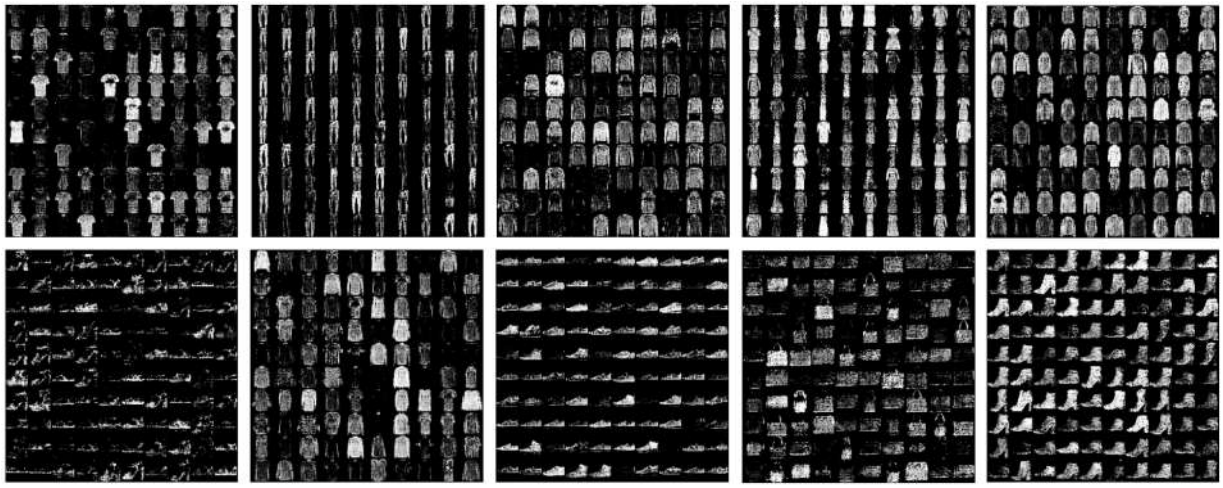
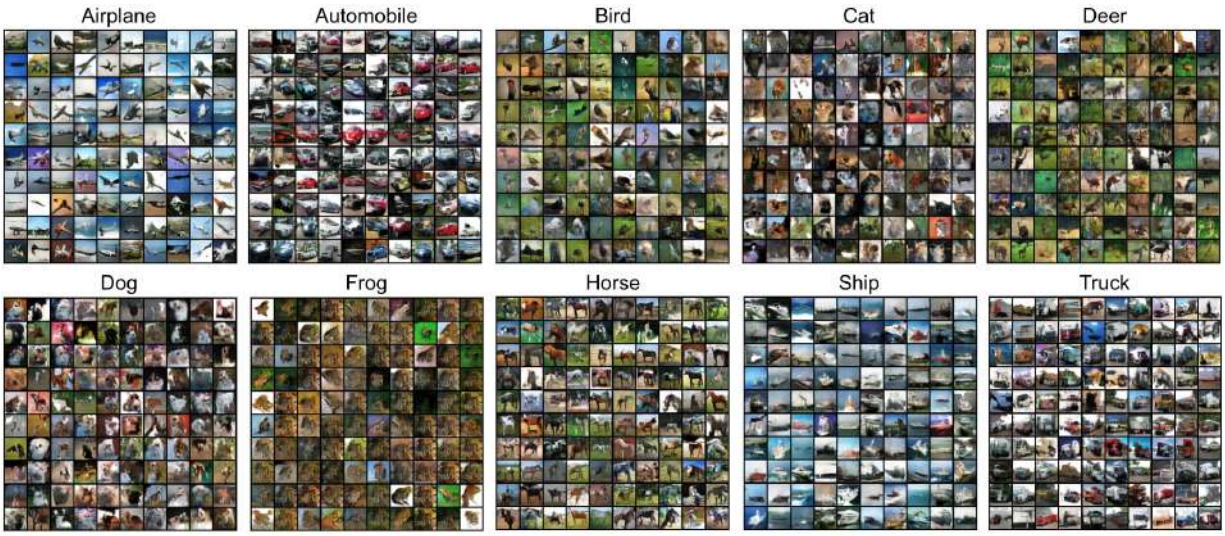


Figure 22: Samples generated by the CDGAN for each of the ten classes in the CIFAR-10 dataset.



7 Conclusion

This report, titled 'Data Inspection and Generation v.1', documents research & development work carried out in WP2 during Phase 2 *MANOLO Framework Implementation* of the project's workplan.

This work aims at (a) developing a comprehensive assets repository, and (b) enriching it with methods for processing the data that will be used for machine learning.

Regarding the assets repository, we are developing an asset management system where data, method implementations, trained models, and benchmarking results are not only reposited, but also multiply interlinked. Metadata extraction is automated by offering APIs through which training and testing can be performed in a way that allows the automated tracking of the interactions between the different objects of the repository.

Data pre-processing is organized on two major strands of work: Identifying subsets within the data that should be excluded from training due to low quality, as well as deriving synthetic data using methods for distillation via data compression and hashing, feature extraction and synthesisation, and model inversion. Regarding data quality estimation, we developed a task-agnostic framework that combines a battery of ML-based and statistical methods for identifying noisy and inconsistent data. These include both refinements and calibrations of prior methods as well as a novel attention-based method which was validated to be highly effective in detecting noise. Regarding data distillation, experimental results indicate that our method significantly reduces the size of image datasets at a relatively small loss in classification accuracy. As this method only requires the dataset's latent space, this comes with the added advantage of not requiring the transfer of sensitive or private user data for the purposes of training machine learning models.

References

[1] Michèle Basseville and Igor V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] NCSS Statistical Software. Cusum charts. Online source, `https://www.ncss.com/wp-content/themes/ncss/pdf/Procedures/NCSS/CUSUM_Charts.pdf`. Last accessed: November 2024.

[3] Lloyd S. Nelson. The shewhart control chart—tests for special causes. *Journal of Quality Technology*, 16(4):237–239, 1984.

[4] Vijay Koshti. Cumulative sum control chart. *International Journal of Physics and Mathematical Sciences*, 2011.

[5] V. M. Artyushenko and V. I. Volovach. Modeling the algorithm of cumulative sums in the applied problems of detecting the signals with random time of occurrence in non-gaussian noise. In *2021 Systems of Signals Generating and Processing in the Field of on Board Communications*, pages 1–5, Moscow, Russia, 2021.

[6] V. I. Volovach and V. M. Artyushenko. Detection of signals with a random moment of occurrence using the cumulative sum algorithm. In *2021 Systems of Signals Generating and Processing in the Field of on Board Communications*, pages 1–6, Moscow, Russia, 2021.

[7] D. Tam. A theoretical analysis of cumulative sum slope (cusum-slope) statistic for detecting signal onset (begin) and offset (end) trends from background noise level. *The Open Mathematics, Statistics and Probability Journal*, June 2009.

[8] Fan Yi and Peihua Qiu. An adaptive cusum chart for drift detection. *Quality and Reliability Engineering International*, 38, 2021.

[9] R. Sebastião and J. M. Fernandes. Supporting the page-hinkley test with empirical mode decomposition for change detection. In *International Symposium on Methodologies for Intelligent Systems*, pages 492–498. Springer, Cham, 2017.

[10] Hayet Mouss, M.Djamel Mouss, Kinza Mouss, and Linda Sefouhi. Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. *Proceedings of the 5th International Symposium on Automation and Control*, 2:815–818, 2004.

[11] Moez Ali. Understanding data drift and model drift: Drift detection in python. Online source, `https://www.datacamp.com/tutorial/understanding-data-drift-model-drift`, January 2023. Last accessed: November 2024.

[12] James Joyce. Kullback-Leibler divergence. In *Encyclopedia of Complexity and Systems Science*, pages 3373–3381. Springer, 2011.

[13] T. van Erven and P. Harremos. Rényi divergence and Kullback-Leibler divergence. *IEEE Transactions on Information Theory*, 60(7):3797–3820, 2014.

[14] J. R. Hershey and P. A. Olsen. Approximating the Kullback Leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, pages IV–317–IV–320, 2007.

[15] R. Bro and A. K. Smilde. Principal component analysis. *Analytical Methods*, 6(9):2812–2831, 2014.

[16] Hervé Abdi and Lynne Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Compu-*

tational Statistics, 2:433–459, 2010.

[17] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065), 2016.

[18] Junming Shao, Zahra Ahmadi, and Stefan Kramer. Prototype-based learning on concept-drifting data streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[19] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the 7th SIAM International Conference on Data Mining*, page 42, 2007.

[20] Y. Sun, Z. Wang, H. Liu, C. Du, and J. Yuan. Online ensemble using adaptive windowing for data streams with concept drift. *International Journal of Distributed Sensor Networks*, 12(5), 2016.

[21] Ming-Chang Lee, Jia-Chun Lin, and Ernst Gunnar Gran. How far should we look back to achieve effective real-time time-series anomaly detection? In *Advanced Information Networking and Applications*. Springer, 2021.

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[23] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. Laion-5b: An open large-scale dataset for training next generation image-text models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 25278–25294. Curran Associates, Inc., 2022.

[24] Daniel Koehler. More than anything: Advocating for synthetic architectures within large-scale language-image models. *International Journal of Architectural Computing*, 21(2):242–255, 2023.

[25] Nuno Guimarães, Ricardo Campos, and Alípio Jorge. Pre-trained language models: What do they know? *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 14(1):e1518, 2024.

[26] Qiang Wang, Yuanfan Li, and Rongrong Li. Ecological footprints, carbon emissions, and energy transitions: the impact of artificial intelligence (ai). *Humanities and Social Sciences Communications*, 11, 08 2024.

[27] Shiye Lei and Dacheng Tao. A comprehensive survey of dataset distillation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(1):17–32, 2024.

[28] Ahmad Sajedi, Samir Khaki, Ehsan Amjadian, Lucy Z. Liu, Yuri A. Lawryshyn, and Konstantinos N. Plataniotis. DataDAM: Efficient Dataset Distillation with Attention Matching . In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17051–17061, Los Alamitos, CA, USA, October 2023. IEEE Computer Society.

[29] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[30] Zhiqiang Wan, Yazhou Zhang, and Haibo He. Variational autoencoder based synthetic data generation for imbalanced learning. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7, 2017.

[31] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[32] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[33] Ayodeji Olalekan Salau and Shruti Jain. Feature extraction: A survey of the types, techniques, applications. In *2019 International Conference on Signal Processing and Communication (ICSC)*, pages 158–164, 2019.

[34] Simon Kornblith, Jonathon Shlens, and Quoc V. Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[35] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[36] Amir R. Zamir, Alexander Sax, William Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[37] Daesoo Lee, Sara Malacarne, and Erlend Aune. Vector quantized time series generation with a bidirectional prior model. In *The 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 2023.

[38] Fabius, Otto and Van Amersfoort, Joost R. Variational recurrent auto-encoders. In *Workshop contriution at International Conference on Learning Representations (ICLR 2015)*, 2015.

[39] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[40] Devon Myers, Rami Mohawesh, Venkata Ishwarya Chellaboina, Anantha Lakshmi Sathvik, Praveen Venkatesh, Yi-Hui Ho, Hanna Henshaw, Muna Alhawawreh, David Berdik, and Yaser Jararweh. Foundation and large language models: fundamentals, challenges, opportunities, and social impacts. *Cluster Computing*, 27(1):1–26, 2024.

[41] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[42] Denis Baskan and Patrick K Erdelt. Neighborhood-based loss functions for explainability of autoencoders. *Available at SSRN 4212995*, 2022.

[43] Marek Pawlicki, Aleksandra Pawlicka, Federica Uccello, Sebastian Szelest, Salvatore D'Antonio, Rafa l Kozik, and Micha l Choraś. Evaluating the necessity of the multiple metrics for assessing explainable ai: A critical examination. *Neurocomputing*, 602:128282, 2024.

[44] Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR 2015)*, pages 1–14, 2015.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[46] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for con-

trastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[47] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3733–3742, 2018.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, and Eli Woods. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1–6, 2020.

[50] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

[51] Yanping Chen, Yuan Hao, Thanawin Rakthanmanon, Jesin Zakaria, Bing Hu, and Eamonn Keogh. A general framework for never-ending learning from time series streams. *Data mining and knowledge discovery*, 29:1622–1664, 2015.

[52] Honghao Gao, Binyang Qiu, Ramon J Duran Barroso, Walayat Hussain, Yueshen Xu, and Xinheng Wang. Tsmae: a novel anomaly detection approach for internet of things time series data using memory-augmented autoencoder. *IEEE Transactions on network science and engineering*, 10(5):2978–2990, 2022.

[53] Matteo Risso, Alessio Burrello, Francesco Conti, Lorenzo Lamberti, Yukai Chen, Luca Benini, Enrico Macii, Massimo Poncino, and Daniele Jahier Pagliari. Lightweight neural architecture search for temporal convolutional networks at the edge. *IEEE Transactions on Computers*, 72(3):744–758, 2022.

[54] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

[55] Udo Schlegel and Daniel A Keim. A deep dive into perturbations as evaluation technique for time series xai. In *World Conference on Explainable Artificial Intelligence*, pages 165–180. Springer, 2023.

[56] Siying Zhou, Chaohui Wang, and Fei Wang. Gmtpm: A general multitask pretrained model for electricity data in various scenarios. *IEEE Transactions on Industrial Informatics*, 2024.

[57] Chang Wei Tan, François Petitjean, and Geoffrey I Webb. Fastee: fast ensembles of elastic distances for time series classification. *Data Mining and Knowledge Discovery*, 34(1):231–272, 2020.

[58] Daoyuan Li, Tegawendé François D Assise Bissyande, Jacques Klein, and Yves Le Traon. Time series classification with discrete wavelet transformed data: Insights from an empirical study. In *The 28th international conference on software engineering and knowledge engineering (SEKE 2016)*, 2016.

[59] Chao-Han Huck Yang, Yun-Yun Tsai, and Pin-Yu Chen. Voice2series: Reprogramming acoustic models for time series classification. In *International conference on machine learning*, pages 11808–11819. PMLR, 2021.

[60] Elizabeth Fons, Alejandro Sztrajman, Yousef El-Laham, Alexandros Iosifidis, and Svitlana Vyetrenko. Hy-

pertime: Implicit neural representation for time series. *arXiv preprint arXiv:2208.05836*, 2022.

[61] Udo Schlegel and Daniel A Keim. A deep dive into perturbations as evaluation technique for time series xai. In *World Conference on Explainable Artificial Intelligence*, pages 165–180. Springer, 2023.

[62] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[63] Akbar Karimi, Leonardo Rossi, and Andrea Prati. Aeda: an easier data augmentation technique for text classification. *arXiv preprint arXiv:2108.13230*, 2021.

[64] Shuguang Chen, Gustavo Aguilar, Leonardo Neves, and Thamar Solorio. Data augmentation for cross-domain named entity recognition. *arXiv preprint arXiv:2109.01758*, 2021.

[65] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[66] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany, August 2016. Association for Computational Linguistics.

[67] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Guided image generation with conditional invertible neural networks. *arXiv preprint arXiv:1907.02392*, 2019.

[68] Perla Doubinsky, Nicolas Audebert, Michel Crucianu, and Hervé Le Borgne. Semantic generative augmentations for few-shot counting. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 5443–5452, 2024.

[69] Qi Chen, Xiaoxi Chen, Haorui Song, Zhiwei Xiong, Alan Yuille, Chen Wei, and Zongwei Zhou. Towards generalizable tumor synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11147–11158, 2024.

[70] Wangyu Wu, Tianhong Dai, Xiaowei Huang, Fei Ma, and Jimin Xiao. Image augmentation with controlled diffusion for weakly-supervised semantic segmentation. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6175–6179. IEEE, 2024.

[71] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

[72] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[73] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[74] Lynton Ardizzone, Till Bungert, Felix Draxler, Ullrich Köthe, Jakob Kruse, Robert Schmier, and Peter Sorrenson. Framework for Easily Invertible Architectures (FrEIA), 2018-2022.

[75] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[76] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convo-

lutional generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[77] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.