Assignment cover sheet

Please complete this cover sheet for each assignment submission. For group assignments, please ensure that each group member completes and submits an individual cover sheet.

Module:	SOC10101 Honours Project
Assessment Title:	Encoding Sparse Robot Morphologies with Custom Losses and Variational Autoencoders for Diversity Measurement in an Evolutionary Framework
Module Leader:	Khristin Fabian
Student registration number:	40538519
Student name	Andrew Taison

Declaration

I declare, in accordance with Edinburgh Napier University's Academic Integrity Regulations that: except where explicit reference is made to the contribution of others*, this assignment is the result of my own work, and has not been submitted for any module, programme or degree at Edinburgh Napier University or any other institution.

*IMPORTANT: Contribution of includes use of generative Artificial Intelligence (AI) tools. Ensure you have read the University Guidelines for Students on AI & Writing Assistant Tools). Please declare here whether you have used such tools, and to what extent:

☑NO I have not used such tools

☐YES I have used such tools and I have provided details and included sample prompts and responses below <or: in an appendix>.

If you have used AI tools in completing this submission, please briefly describe in approximately 100 words in the box below how you have used these tools:

Encoding Sparse Robot Morphologies with Custom Losses and Variational Autoencoders for Diversity Measurement in an Evolutionary Framework

Andrew Taison
40538519

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc (Hons) Data Science

School of Computing, Engineering & The Build Environment

April 2025

Authorship Declaration

I, Andrew Taison confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

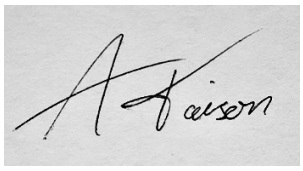
I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:



Date:

25th March 2025

Matriculation no:

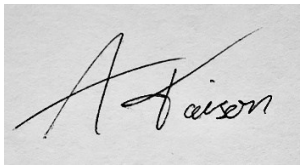
40538519

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.



28th March 2025

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Acknowledgements

Firstly, I would like to thank my supervisor, Simon Smith, for dedicating his time to guiding me through this project, providing honest feedback, and helping keep me on track.

Secondly, I would like to thank Leni Le Goff for providing the dataset and the initial visualisation code that helped shape the early stages of this work.

Finally, I would like to thank my wife, Helen Taison, for her patience and support throughout my time at university over the past four years.

My success in this project would not have been possible without the assistance of all these people, so thank you.

Abstract

This study explores whether Variational Autoencoders (VAEs) can be used to encode sparse robot morphologies and enable more meaningful diversity measurement in an evolutionary framework. This work begins by identifying the limitations of current diversity metrics, such as component counts and search space distances, which often fail to capture deeper structural relationships. To address these challenges, a custom VAE architecture was developed, inspired by PointNet and designed to process sparse, unordered voxel data. The study uses a CPPN-generated dataset of evolved robot designs and applied a sparse input representation format with padded voxels to ensure consistent input size. Custom loss functions were developed to promote spatial alignment, accurate reconstruction, and avoid duplicate voxels.

The main findings are that the trained VAE successfully learns a structured latent space in which similar morphologies are embedded closer together. PCA and UMAP dimensionality reduction techniques are used to generate two-dimensional projections of the latent space, revealing clear separation of predefined morphological groups. Overlaid visualisations show that samples near each other in the latent space often share key structural traits, indicating that the model captures meaningful relationships beyond simple spatial proximity or component quantity.

The project concludes by showing that VAEs can provide a strong foundation for measuring morphological diversity. This is supported by patterns observed in the latent space after encoding CPPN-generated robot morphologies held out for testing. These patterns suggest that meaningful structural relationships were learnt, offering a scalable, unsupervised alternative to traditional diversity metrics.

Promising future directions include incorporating more complete morphological data, such as skeletal structures, and leveraging the latent space to identify gaps and generate novel designs. These extensions could improve the model's generative ability and make its outputs more useful for analysing, exploring and evolving robot morphologies in an evolutionary framework.

All source code developed as part of this project is available at: https://github.com/Andy-Taison/dissertation_code

Table of Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

2D: Two-Dimensional

3D: Three-Dimensional

AE: Autoencoder

BCE: Binary Cross-Entropy

CNN: Convolutional Neural Network

CPPN: Compositional Pattern Producing Network

CPU: Central Processing Unit

CSV: Comma-separated values

DBSCAN: Density-Based Spatial Clustering of Algorithms with Noise

DE: Differential Evolution

EC: Evolutionary Computation

EP: Evolutionary Programming

ES: Evolutionary Strategies

FC: Fully Connected

FN: False Negatives

FP: False Positives

GA: Genetic Algorithms

GAN: Generative Adversarial Network

GMM: Gaussian Mixture Model

GP: Genetic Programming

GPU: Graphics Processing Unit

KL: Kullbak-Leibler Divergence

LeakyReLU: Leaky Rectified Linear Unit

LON: Local Optima Network

MLP: Multi-Layer Perceptron

MSE: Mean Squared Error

PCA: Principal Component Analysis

ReLU: Rectified Linear Units

ResNet: Residual Network

SOTA: State-of-the-Art

T-Net: Transformation Network

TP: True Positives

t-SNE: t-distributed Stochastic Neighbour Embedding

UMAP: Uniform Manifold Approximation and Projection

VAE: Variational Autoencoder

1 Introduction

Measuring morphological diversity in evolved robot designs remains a significant challenge in evolutionary robotics. Diversity plays a key role in avoiding premature convergence, improving exploration, and producing robust and varied solutions (Eiben & Smith, 2015; Le Goff & Smith, 2024; Medvet et al., 2021). However, existing diversity metrics, including component counts, search space distances, or manually defined categories, often fail to capture spatial arrangements or subtle structural changes. This limits their effectiveness as robot morphologies become complex.

This project is motivated by the need for more meaningful ways to measure morphological diversity – approaches that depend on representations which go beyond what a robot has, and instead reflect what a robot is, in terms of structure, composition, and spatial relationships. In contrast, existing metrics, although straightforward to compute, often rely on domain-specific assumptions about which features are relevant.

The aim of this project is to explore whether meaningful representations of robot morphologies can be captured through unsupervised learning. The goal is to enable diversity to be measured in a more structurally aware and data-driven way.

Two central research questions guide this study:

- **RQ1** – Can a Variational Autoencoder (VAE) effectively encode sparse robot morphologies in an evolutionary framework?
- **RQ2** – Can the learnt latent space be used to meaningfully measure diversity between robot morphologies?

To explore these questions, this project implements a custom VAE architecture designed to encode sparse robot morphologies. It then investigates whether the resulting latent space captures meaningful structural differences that can support diversity measurement. The key idea is that a trained VAE embeds robot designs into a structured latent space, extracting important underlying patterns. In this space, distances between points are expected to reflect morphological similarity. If successful, this approach would enable diversity to be measured not by counting

components, but by analysing how different designs are arranged in a continuous latent space that captures their underlying characteristics.

A standard autoencoder is not well suited to this task, as its latent space is unstructured and lacks a defined distribution, making it difficult to sample from (Foster, 2023; Shende, 2023). Without regularisation, the model may overfit to the training data, resulting in poor generalisation and limited control over how morphologies are represented. In contrast, a VAE introduces a structured, continuous latent space by encouraging the latent distributions to follow a predefined prior. This latent organisation creates the potential for analysing morphological diversity in terms of structure and similarity within the latent space.

The dataset used in this project originates from a generative evolutionary process developed by Le Goff & Smith (2024) which combines morpho-evolution with intrinsic motivation to efficiently generate functional and varied robot designs. Whilst the designs produced through their method are diverse, the metrics used to assess diversity focus on raw spatial descriptors and may overlook deeper relationships between components and structural patterns. This limitation may be addressed by learning a latent representation that captures information not explicitly found in raw descriptors.

The original data represents each robot within an $11x11x11$ voxel grid, where each voxel is a three-dimensional (3D) pixel. Each voxel is assigned a descriptor, indicating either a component type or empty space. In this format, input ordering is spatially meaningful. To reduce sparsity and improve learning, the data is converted into a sparse representation in which only active voxels are retained, each represented by their 3D coordinates and a one-hot component descriptor.

This transformation introduces unordered input, since the sequence of voxels no longer corresponds to fixed spatial positions. Additionally, to ensure no positional bias, the voxels are shuffled. To handle this, a custom network architecture inspired by PointNet is developed (Qi et al., 2016).

However, unlike standard PointNet architectures, which are designed for classification and segmentation, typically focusing only on coordinate encodings, this model must learn a joint embedding of both spatial and component information. It

must also support reconstruction from the latent space to allow the quality of learning to be assessed during training. These requirements present unique architectural challenges, particularly given the limited number of voxels per robot and the loss of any inherent ordering.

To support learning from unordered and sparse inputs, the model uses several custom loss functions. Each loss targets a specific challenge: promoting spatial alignment, supporting accurate reconstruction of component types, and discouraging normalised outputs from collapsing to duplicate coordinate positions when scaled.

The main contributions of this project are:

- A custom VAE architecture for encoding voxel-based robot morphologies.
- A sparse input representation introduced to address data sparsity.
- A set of custom loss functions developed to support learning from unordered and sparse inputs.
- An evaluation of the resulting latent space using both visual and statistical techniques, with comparisons across model configurations to assess its effectiveness for capturing morphological diversity.

The trained model was evaluated from multiple perspectives. Predefined datasets showed clear separation in the latent space, suggesting that the model captured meaningful structural and compositional differences. Further analysis of individual robot placements showed that samples positioned close together in the reduced latent space often shared structural features, even when oriented differently. This observed clustering suggests that the model had learnt to capture deeper morphological similarities and complex relationships beyond direct spatial alignment. These results indicate the model's potential for measuring diversity, whilst also suggesting that further work is needed to better interpret and understand the latent space.

All source code developed as part of this project is available at: https://github.com/Andy-Taison/dissertation_code

The remainder of this dissertation is structured as follows. Chapter 2 presents the literature review. Chapter 3 outlines the methodology. Chapter 4 details the results,

followed by a discussion of key findings and their implications in Chapter 5. Finally, Chapter 6 concludes the study.

2 Literature Review

2.1 Evolutionary Computation

2.1.1 Brief History and Context

Using evolutionary principles for computational problem solving is an idea that has existed for decades. The foundation of evolutionary computation (EC) dates back to the mid 20th century when scientists began exploring nature inspired computation models to solve complex problems. Early efforts were influenced by the work of Charles Darwin, whereby models mimic the basic features of natural selection (Jong, 2009). The idea of “Genetical or evolutionary search” was suggested ahead of its time by Alan Turing in 1948, according to Eiben & Smith (2015). However, major breakthroughs occurred in the 1960s. Independently, groups of researchers pioneered some of the core methods in EC; Fogel, Owens and Walsh introduced evolutionary programming, Holland developed genetic algorithms, whilst Rechenberg and Schwefel invented evolutionary strategies (as summarised by Eiben & Smith, 2015; Jin, 2023; Jong, 2009). Later in the 1990s, Koza established genetic programming and around the same time these four separate branches were merged into what is now known as evolutionary computation (as summarised by Bäck et al., 2023; Eiben & Smith, 2015).

The field has since expanded significantly, with many more algorithms being added, and has evolved into a crucial branch of artificial intelligence and computational sciences (Bäck et al., 2023; Cicirello, 2024). Today, EC’s applications are found in diverse areas, such as those highlighted by Cicirello (2024), ranging from data science and bioinformatics to hardware validation and neural architecture search. The increase of specialised algorithms, and expansion of application domains, allow researchers to tackle larger and more complex problems emphasising the importance of EC today.

2.1.2 Introduction to Evolutionary Computation

EC, inspired by biological evolution, solves complex optimisation problems through algorithms simulating selection, crossover, mutation, and evaluation. At their core,

they iterate the operations (generations) of selection (survival of the fittest), recombination/crossover (reproduction), mutation, and fitness evaluation, repeating until some criterion is met (Jin, 2023). This allows populations of candidate solutions to adapt and improve (evolve) towards an optimum over time.

One of the central features of EC is the use of genotypes and phenotypes. The phenotype refers to the actual (candidate) solution or expression being evaluated in the problem domain. It is an “individual” or object that could, for example, take the form of a set of integers or a vector of real values. These values may represent different aspects, such as defining the architecture of a neural network or corresponding to the geometry of a real-world object (Eiben & Smith, 2015; Jin, 2023). The genotype is equivalent to DNA in biology and is the encoded *representation* of the phenotype. It is the genotype that can be manipulated by genetic operations like crossover and mutation and where the evolutionary search takes place (Eiben & Smith, 2015).

The mapping between the genotype and phenotype is crucial, as it influences the algorithms effectiveness. Jin (2023) suggests that any neighbourhood in the genotype space should be preserved in the phenotype space. EC algorithms can be excellent at solving complex problems, provided they maintain a diverse population of solutions, which enables efficient exploration of the solution space (Sudholt, 2020). Because EAs are fitness-oriented and variation-driven, they excel at tackling challenging problems, particularly computationally hard tasks where traditional methods may fall short (Yu & Gen, 2010).

A wide range of evolutionary algorithms have been developed, including genetic algorithms (GA), genetic programming (GP), differential evolution (DE), evolutionary strategies (ES), and evolutionary programming (EP). These approaches share core principles such as maintaining a population of solutions and applying selection, mutation, and recombination, although they differ in their specific mechanisms and applications (Alhijawi & Awajan, 2024; Emmerich et al., 2018; A. Gandomi et al., 2015; Slowik & Kwasnicka, 2020). Their lasting relevance and adaptability across fields such as optimisation, control, and automated design have been widely demonstrated (Alhijawi & Awajan, 2024; Liu et al., 2023; Ma et al., 2023; Slowik & Kwasnicka, 2020). Whilst this project does not directly implement these algorithms, it

draws on the broader context of EC to examine morphological diversity in evolved robot structures.

2.1.3 Current Work in Evolutionary Computation

Yang et al. (2024) enhance DE adaptability by using reinforcement learning, enabling subpopulations to dynamically select mutation strategies based on population diversity. Diversity is measured via average *distances* to the population's mean *solution*, using threshold values to maintain effective convergence and avoid local optima. This allows strategy choices to be rewarded or punished, reducing computational costs and improving optimisation.

Thomson et al.'s (2024) work in using Local Optima Network (LON) analysis to study fitness landscapes directly contributes to the field of EC by offering a novel way to visualise and understand the structure of search spaces. Although their paper focuses more on morpho-evolution and different encodings (discussed in Section 2.2.2), they provide a framework for understanding how the arrangement of local optima in landscapes can affect the performance of evolutionary algorithms and objective functions.

Whilst EC offers powerful tools for optimisation, its effectiveness depends on how solutions are encoded and evaluated. In morphological evolution, this challenge becomes more evident when evolving robotic designs, as traditional diversity metrics often fail to capture complex differences in structure and function.

2.2 Morphological Evolution

2.2.1 Introduction to Morphological Evolution

Morphological evolution, inspired by the biological study of morphology – defined as "the structure and form of organisms" – refers to the evolution of physical designs in computational systems (Cambridge University Press & Assessment, n.d.; Floreano, 2008, p. 419). In robotics, evolutionary algorithms are used to evolve and optimise control systems, which tends to be for a fixed morphology (body). This however comes at the cost of limiting the "brain" (controller) to a specific body and its sensors – not likened to nature which coevolves both body and brain to changing environments and tasks (Floreano, 2008, p. 499).

Le Goff et al. (2023) note that many research projects in the area focus on evolving either the body or the controller independently due to the challenges of incompatibility between inherited controllers and evolved morphologies. Pigozzi et al. (2023) further highlight this challenge, emphasising this is due to entanglement of the brain and body as described by the embodied cognition paradigm, which suggests that both internal and external factors influence the evolutionary process. Despite these challenges, the ultimate goal of morphological evolution is to develop robots - initially in simulation - that can adapt and mould to their environment or task, allowing for more natural and efficient interactions than traditional, manually designed robots.

Although coevolution of body and brain is the ideal (termed morpho-evolution by Thomson et al. (2024)), this section focuses on the evolution of morphology, providing context on how robot bodies are encoded and their applications, which this project uses as its foundation. The following section goes on to discuss the importance of diversity in evolutionary processes, setting the stage for the project's focus.

2.2.2 Encoding Robot Bodies

Encoding robot bodies involves defining how phenotypes (realised robotic designs) are mapped to genotypes (encoded representations). Decoding refers to the inverse process, where genotypes are transformed back into phenotypes (Eiben & Smith, 2015). The genotype, represented by encoding values, provides the framework for evolutionary algorithms to operate on and manipulate during the optimisation process. Whilst the phenotype, which contains descriptor values and typically obtained by decoding the genotype after evolution, is what can be used for visualisation, construction, or fitness evaluation.

The genotype to phenotype mapping determines how encoding values translate into descriptor values to produce the phenotype. Two primary encoding methods are used to define how the genotype encodes the phenotype:

1. **Direct Encoding:**
   In direct encoding, a one-to-one mapping exists between the genotype and the phenotype, where each decision variable corresponds to a component in the phenotype (Jin, 2023; Thomson et al., 2024). Jin (2023) notes that this

encoding grows proportionally with the quantity of decision variables, making it hard to scale for large scale problems. Additionally, direct encoding is stationary, meaning it does not adapt dynamically through processes like growth (as seen in biology). However, Pigozzi et al. (2023) find that as the encoding preserves locality of individuals, it can lead to enhanced learning ability.

2. **Indirect Encoding:**
   Typically a rule-based method which reduces the dependency of chromosome length on the number of decision variables, improving scalability for large scale problems (Jin, 2023). Indirect encoding uses a more complex, many-to-one mapping between genotype and phenotype, where multiple genotype rules refer to an aspect of the phenotype (Thomson et al., 2024). Example methods include Compositional Pattern Producing Networks (CPPNs) and L-Systems, which can allow complex and beneficial repeating patterns to develop, however indirect encoding has been observed to slow the evolutionary process compared to direct (Pigozzi et al., 2023; Thomson et al., 2024).

Encoding methods, whether direct or indirect, allow for morphological evolution to take place and form the foundation for robotic development. These approaches allow for robotic designs that can adapt to their environment or specific tasks, demonstrating practical functionality. However, maintaining diversity within the evolving population is essential to fully explore the solution space and ensure efficient convergence towards optimal designs.

2.3 The Role of Diversity in Morphology

In the work of Medvet et al. (2021), employing a diversity mechanism based on human determined species is shown to enhance both optimisation effectiveness and biodiversity. However, other factors, such as phenotype representation and the chosen evolutionary algorithm (EA), also play a significant role. This emphasis on diversity reflects the broader understanding that variation in morphology and behaviour is key to resilience and adaptability, both in nature and artificial systems. Inspired by biodiversity being robust to disruptive conditions, Medvet et al. (2021) highlight how diverse robotic ecosystems can adapt to changing environments

without humans overseeing. Although they go on to note that achieving and defining what constitutes good biodiversity and its effects is an ongoing challenge.

A recent study by Le Goff and Smith (2024) introduces a new approach to generating efficient and diverse robot designs by combining morpho-evolution with intrinsic motivation. Their method replaces traditional, computationally intensive learning phases with a homeokinetic controller that produces exploratory behaviour without the need to specify goals. This strategy not only reduces computational time, but also enhances diversity and prevents premature convergence. The authors demonstrate that the designs generated perform better across various tasks in their comparison, attributing it to this diversity.

Although this is a limited study, diversity in morphology refers to the variation among candidate solutions, which can lead to several key benefits:

- Preventing premature convergence and avoiding local optima (Eiben & Smith, 2015; Le Goff & Smith, 2024).
- Enhancing adaptability due to a broader range of unique solutions (Eiben & Smith, 2015; Medvet et al., 2021).
- Increased resilience to change by mirroring the robustness found in nature (Medvet et al., 2021).
- Expanding the exploration capacity to the solution space, allowing for the discovery of novel designs (Le Goff & Smith, 2024; Medvet et al., 2021).

As the value of diversity becomes increasingly apparent, so does the need to effectively measure and quantify it. This enables comparability across studies and provides a benchmark for assessing diversity and its impacts.

2.4 Measuring Diversity in Robotics

When diversity is high, individuals are more dispersed which facilitates global search space exploration. Various methods are currently implemented in research to measure diversity, each having their benefits and pitfalls. The methods chosen typically are task dependant and align with what is being investigated.

2.4.1 Existing Methods

Yang et al. (2024) measure diversity by averaging distances between normalised individuals in the search space. However, their approach unconventionally sums raw differences across dimensions before applying a square root, without squaring or taking absolute values. They use defined thresholds to classify diversity states – low (exploitation), balanced, or high (exploration) – guiding their mutation strategy dynamically.

In Le Goff and Smith (2024)'s paper, diversity is measured using two key methods:

1. **Counting the number of each type of component** (wheels, sensors, etc) in each design, which are then analysed against their exploration scores (how many unique cells were visited by the robot).
2. **Calculating a sparsity score** by taking the average Euclidean distance of each design to its 15 nearest neighbours in the descriptor space. Each morphological descriptor is represented as a 3D matrix that encodes the robot's design and describes the space it occupies, where each voxel (a three-dimensional pixel) stores a phenotypic descriptor – an integer value indicating component type (i.e. zero for empty space, values one to four indicate component types).

Medvet et al. (2021) use a slightly different method to measure diversity. Inspired by nature, they define species by first randomly selecting individuals and using these to identify and set species labels through manual inspection:

- **3 shape classes:** "Blob", "Legged" and "Other".
- **4 behaviour classes:** "Galloping", "Crawling", "Vibrating" and "Idle".

These labels are then used to train machine learning models to automatically classify the shape and behaviour of several million robots. Biodiversity is then measured using the Simpson index, whereby the following formula is used to determine the probability that two randomly and independently chosen individuals belong to the same species:

$$\lambda = \sum_{i=1}^{i=n} p_i^2$$

Where n is number of species, and p_i is the fraction of individuals in the i-th species. Medvet et al. (2021) then proceed to take the multiplicative inverse (reciprocal) of this, so that higher values indicate greater diversity in the population.

2.4.2 The Need for a New Approach

Each of the discussed methods for measuring diversity has strengths and limitations. Yang et al. (2024) provide a straightforward and computationally efficient approach by calculating a distance and using predefined thresholds to quantify diversity. However, this method assumes linearity, as each individual's dimensions are equally weighted (due to normalisation) and treated as independent. By summing raw, normalised values across dimensions, each contributes additively to the distance and hence does not capture non-linear relationships. Furthermore, by not applying a square or absolute operation, positive and negative values can cancel out, meaning the true diversity may not be fully captured.

Le Goff and Smith (2024) combine component counts with spatial sparsity scores for structural diversity. However, using raw spatial descriptors risks missing deeper morphological relationships and patterns that influence overall design diversity.

Medvet et al. (2021) take inspiration from nature and whilst their method appears effective for their task, it relies on manual classification and fixed categories. This approach could limit its ability to adapt to novel or unusual designs.

These limitations stress the need for a new approach that can capture the complex relationships within morphology, along with offering greater scalability and flexibility. One potential solution lies in the use of Variational Autoencoders (VAEs). However, when working with morphological representations that are sparse, the challenge is not only learning a structured latent space but also handling the unordered nature of the data. This is where representation learning methods, such as PointNet, can provide a foundation.

2.5 3D Representation Learning and Point Cloud

A point cloud is a collection of points in three-dimensional (3D) space, typically represented by `(x, y, z)` coordinates, and sometimes contain attributes such as colour or intensity. They are usually obtained via laser scanning for 3D modelling,

although unlike structured 3D representations such as voxel grids, point clouds lack topological relationships between points. Whilst they effectively capture 3D data, their unordered nature can make processing more complex (Y. Xu et al., 2021).

PointNet is a deep learning architecture designed to process 3D point clouds directly without the need for converting to a full grid space or image like formatting (Anvekar et al., 2022; Molnar & Tamas, 2024). Unlike traditional convolutional neural networks (CNNs), which rely on structured grid data, PointNet operates directly on unordered coordinate sets, making it particularly suitable for this project. Figure 1 illustrates the architecture of PointNet (Qi et al., 2016).

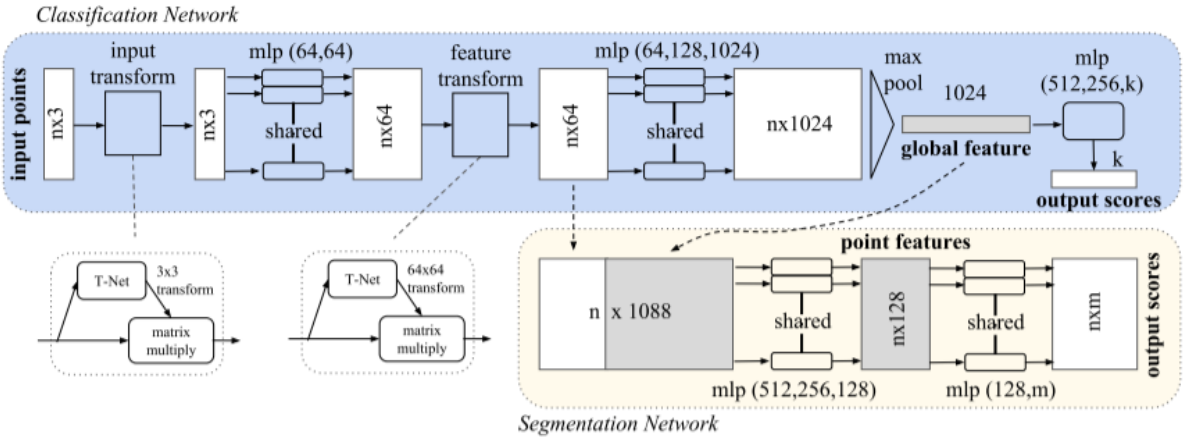


Figure 1. *PointNet architecture*

Note: The diagram shows the original PointNet architecture for 3D point cloud processing, including classification and segmentation modules. Shared MLPs apply the same weights to each point independently, enabling efficient learning. A symmetric aggregation function (e.g., max pooling) allows for order invariance, whilst T-Net modules learn affine transformations to be robust against input misalignment. Reprinted from PointNet (Qi et al., 2016).

Additionally, PointNet includes a transformation network (T-Net) that learns and applies an affine transformation to ensure invariance to input rotations and alignment. Each point is then processed independently through a series of multi-layer perceptrons (MLPs) before the global aggregation step. Since point clouds are unordered sets of points, PointNet must also be invariant to input order. This is achieved using symmetric functions such as max pooling, which aggregate features across all points regardless of their order (Molnar & Tamas, 2024; Qi et al., 2016).

PointNet provides an effective way to extract spatial features from unordered 3D data, but it does not enforce a structured, continuous representation. To address this, VAEs introduce a probabilistic latent space, which not only enables structured

encoding of morphological features but also allows for controlled exploration and potential diversity measurement.

2.6 Variational Autoencoders (VAEs)

2.6.1 Introduction to Variational Autoencoders

To discuss Variational Autoencoders (VAEs), their predecessor, Autoencoders (AE) should first be mentioned. Autoencoders were popularised by Hinton and Salakhutdinov for dimensionality reduction in 2006 and provide the foundation for VAEs.

As described in their work, fundamentally, an Autoencoder consists of two neural networks: an encoder and a decoder. The encoder compresses the input data to a lower-dimensional latent space, whilst the decoder attempts to reconstruct the original from the latent space representation by minimising reconstructed loss (see Figure 2). Operating as an unsupervised framework, the latent space representation enables patterns in the data to be discovered due to its ability to extract meaningful features (Shende, 2023).

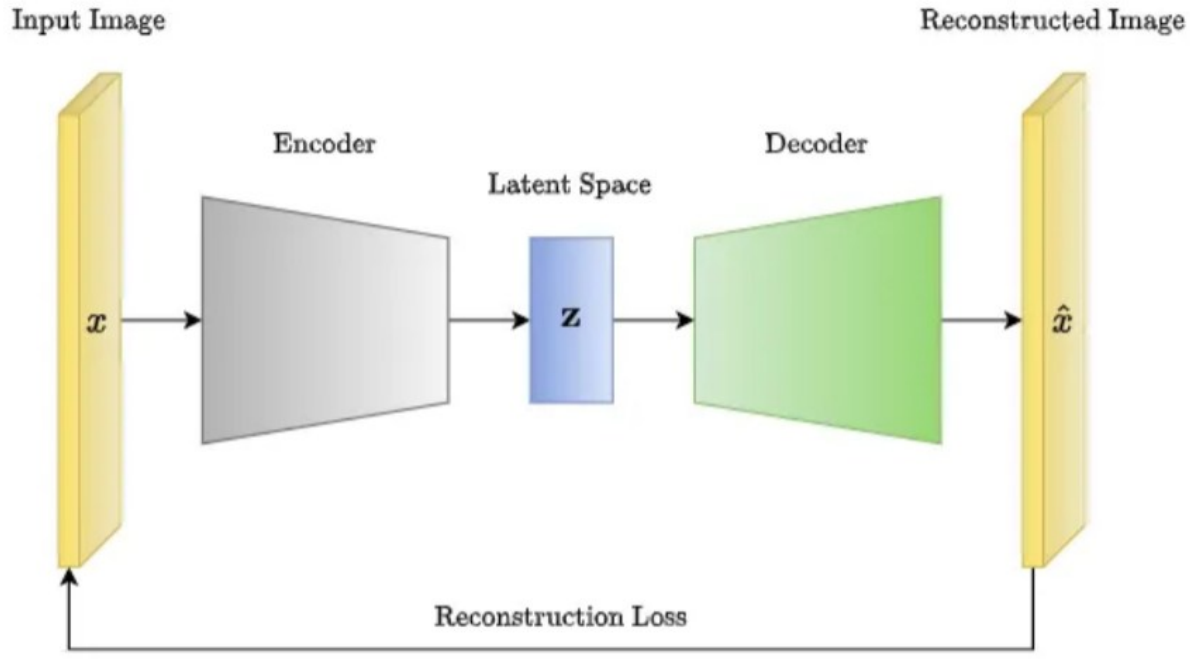


Figure 2. *Autoencoder architecture*
Note: The diagram shows a high-level view of a standard Autoencoder. Input x is encoded into a lower dimensional latent representation z, then decoded to reconstruct the output $\hat{x}$. A reconstruction loss compares the output with the input and updates the network weights. This provides the foundation for more complex models such as Variational Autoencoders (VAEs). Reprinted from Autoencoders, Variational Autoencoders (VAE) and β-VAE (Shende, 2023).

Nonetheless, the Autoencoder has limitations:

- **They are prone to overfitting** - remembering the training data and hence not generalising well (Shende, 2023).
- **The distribution of the points in latent space are undefined** - making latent space sampling difficult (Foster, 2023, Chapter 3).
- **The latent space is not continuous**, instead it is deterministic and unstructured (Foster, 2023, Chapter 3; Shende, 2023). This leads to two consequences –
    1) A given input will always encode to the same fixed point in the latent space, limiting their ability to generalise and generate novel outputs.
    2) Additionally, the unstructured nature of the latent space means similar data groups may be grouped far apart, making Autoencoders unsuitable for diversity measurement.

Because of these limitations, Autoencoders are typically used for simpler reconstruction and transformation tasks, such as cleaning and image denoising, anomaly detection and clustering (Foster, 2023, Chapter 3; Hinton & Salakhutdinov, 2006; Shende, 2023).

Variational Autoencoders (VAEs) are similar, however instead of mapping to a fixed point in the latent space, they learn a probabilistic distribution – typically a Gaussian distribution - over the latent space (Figure 3; Foster, 2023, Chapter 3). Therefore, the latent space is more structured and continuous, allowing for generative modelling and smooth interpolation within the latent space (Shende, 2023).

This mapping to a lower-dimensional, probabilistic space enables the model to organise features in a way that captures essential variations in the data and grouping similar data closer together. Importantly, this not only enables controlled variations in reconstructed outputs, but also presents a potential approach to measure diversity. Exploring structured latent spaces to improve diversity is a concept studied in other generative models. Fernandes et al. (2020) use evolutionary search to explore the latent space of Generative Adversarial Networks (GANs); a type of generative model consisting of a generator that creates new samples and a discriminator that evaluates their plausibility (I. J. Goodfellow et al., 2014). In their study, they use GA

to navigate the latent space, evolving diverse latent vectors, thereby controlling and improving the diversity of generated outputs.

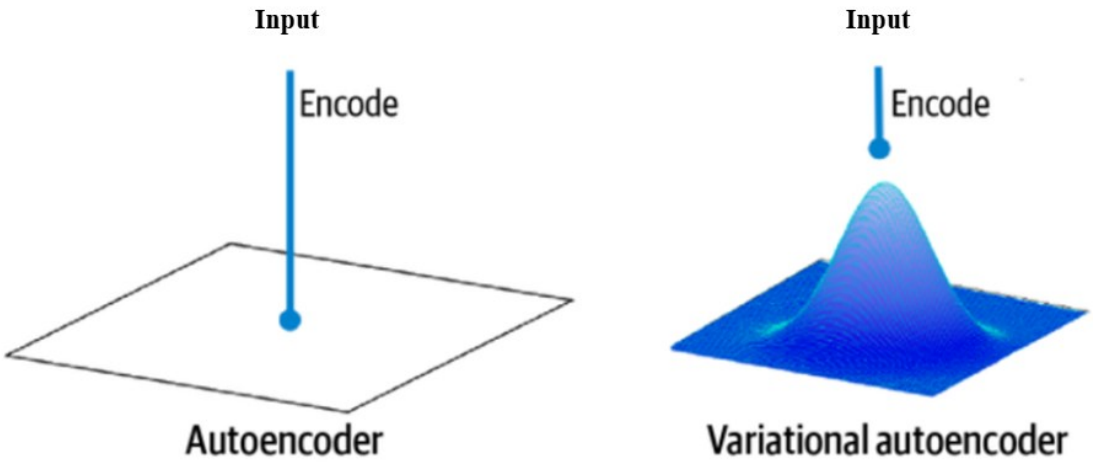


Figure 3. *Visualisation of the difference between AE & VAE encoding*
Note: Autoencoders map each input to a single point in latent space, whereas variational autoencoders map each input to a probability distribution (typically Gaussian). This allows VAEs to sample from a continuous, structured latent space. Adapted from Generative Deep Learning (Foster, 2023, Chapter 3).

2.6.2 Architecture and Operation of Variational Autoencoders

Originally introduced in their 2013 paper, Kingma and Welling proposed VAEs as a probabilistic extension to autoencoders, supporting a structured and continuous latent space. Figure 4 illustrates the architecture, where μ and σ represent the mean and standard deviation vectors respectively. For numerical stability, the log-variance is often used instead of the variance or standard deviation (Yong, 2021). These vectors parameterise the approximate posterior distribution $q_\phi(z|x) = \mathcal{N}(\mu, \sigma^2)$ (typically Gaussian, note the distribution is not explicitly shown in the diagram), which approximates the true posterior $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$, because the true posterior is intractable (Kingma & Welling, 2013). Importantly, μ and σ are not directly learnt, instead, they are outputs of the encoder network, which is parameterised by ϕ (the weights and biases). This structure allows the encoder to dynamically adjust the approximate posterior distribution during training which models the latent variable z.

To sample from approximate posterior whilst maintaining differentiability during backpropagation, the reparameterisation trick is applied (Kingma & Welling, 2013). To do this, instead of sampling directly from the approximate posterior, which introduces randomness and is non-differentiable, a noise vector ε is drawn from a

standard normal distribution $\mathcal{N}(0, 1)$ (known as the prior), which does not depend on learnable parameters. This allows z to be drawn in a differentiable manner:

$$z = \mu + \sigma \odot \varepsilon$$

Here, z is a sampled latent space vector (Foster, 2023, Chapter 3; Yong, 2021). The decoder uses the sampled vector z to produce the reconstructed input $\hat{x}$, which represents the best estimate of the original data x.

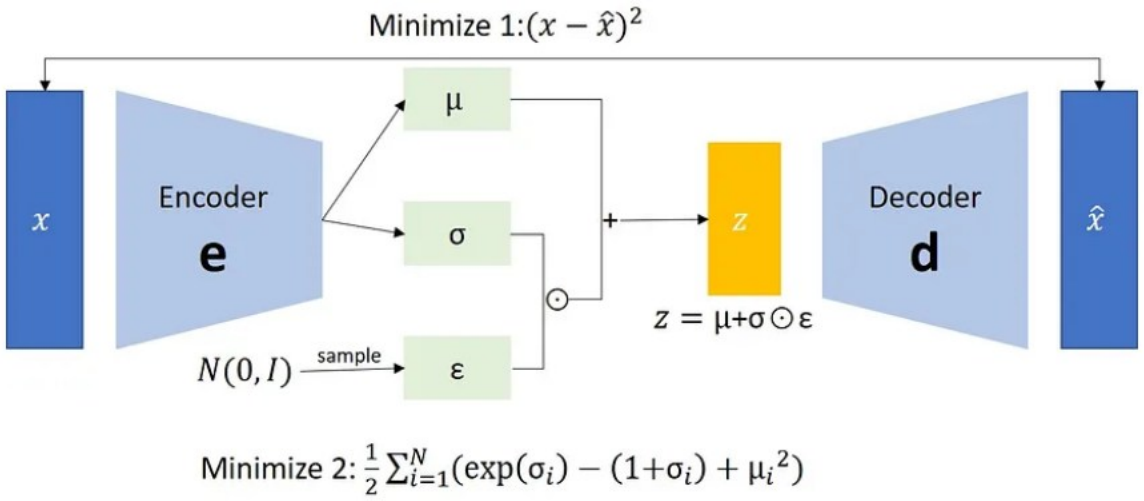


Figure 4. *Variational Autoencoder architecture*
Note: The encoder maps input x to a latent distribution defined by mean μ and standard deviation σ (typically log variance for numerical stability). To allow backpropagation, sampling is performed using the reparameterisation trick, where ε is drawn from a standard normal distribution. The decoder reconstructs $\hat{x}$ from z, and training minimises both reconstruction error (1) and KL divergence (2). Reprinted from Variational Autoencoder (VAE) (Yong, 2021).

The training process optimises two criteria simultaneously:

- **Reconstruction loss** – the difference between the input and reconstructed output (Foster, 2023, Chapter 3).
- **Kullback-Leibler (KL) divergence** – which evaluates how much the approximate posterior distribution $q_\phi(z|x) = \mathcal{N}(\mu, \sigma^2)$ deviates from the chosen prior $p(z) = \mathcal{N}(0, 1)$, commonly a standard Gaussian distribution (Shende, 2023). KL divergence acts as a regularisation term, helping to generate a more meaningful latent space (Pramod, 2024).

The structured and continuous nature of the latent space in a VAE not only enables smooth interpolation for visualisation and exploration but also presents opportunities for enhanced diversity measurement. By extracting, compressing and organising important underlying features and complex relationships of input components, the latent space positions similar inputs closer together and different inputs further apart.

This organisation, in theory, allows variability in inputs to be directly measured within the latent space, supporting the evaluation of diversity.

2.7 Clustering Techniques for Latent Space Exploration

It has been identified that VAEs offer the potential to measure diversity utilising the latent space, but the question remains as to how to approach exploration and evaluation within that space. Taking a distance between points – for example, Euclidean (a method discussed earlier) – could be applicable. Alternatively, looking at how points in the space cluster may be viable.

Many different types and categories of clustering algorithms exist due to no algorithm being capable of addressing every clustering challenge (Figure 5; Oyewole & Thopil, 2023; Saxena et al., 2017; Xu & Tian, 2015). The following algorithms are of particular interest:

- **k-means** – a common partitioning algorithm with low time complexity and high compute efficiency. It iteratively recalculates cluster centroids based on the average distance of data points and reassigns points to the nearest centroid until assignments no longer change. It can be sensitive to outliers, can converge to local optima, and the value of k is predefined.
- **Gaussian Mixture Models (GMM)** – a distribution-based algorithm where the data originates from different Gaussian distributions. It provides probabilistic clustering with high flexibility, however, the performance strongly depends on parameter selection and has high time complexity.
- **Density-Based Spatial Clustering of Algorithms with Noise (DBSCAN)** – a well-known density-based algorithm that groups data points based on high density regions. Unlike k-means, it does not require specifying the number of clusters and can detect outliers as noise. Although performance is sensitive to parameter selection and the algorithm struggles when there is uneven density of the data space.

Jabari et al. (2024) outline a current challenge with k-means, in which increasing the value of k causes the algorithm to converge to a local minimum. They show that to overcome this, particularly with high-dimensional and complex data, that by first applying Principal Component Analysis (PCA) to reduce dimensionality before clustering improves the performance of k-means. Integrating dimensionality

reduction with clustering techniques could assist in exploration of the latent space by preserving key features and reducing noise.

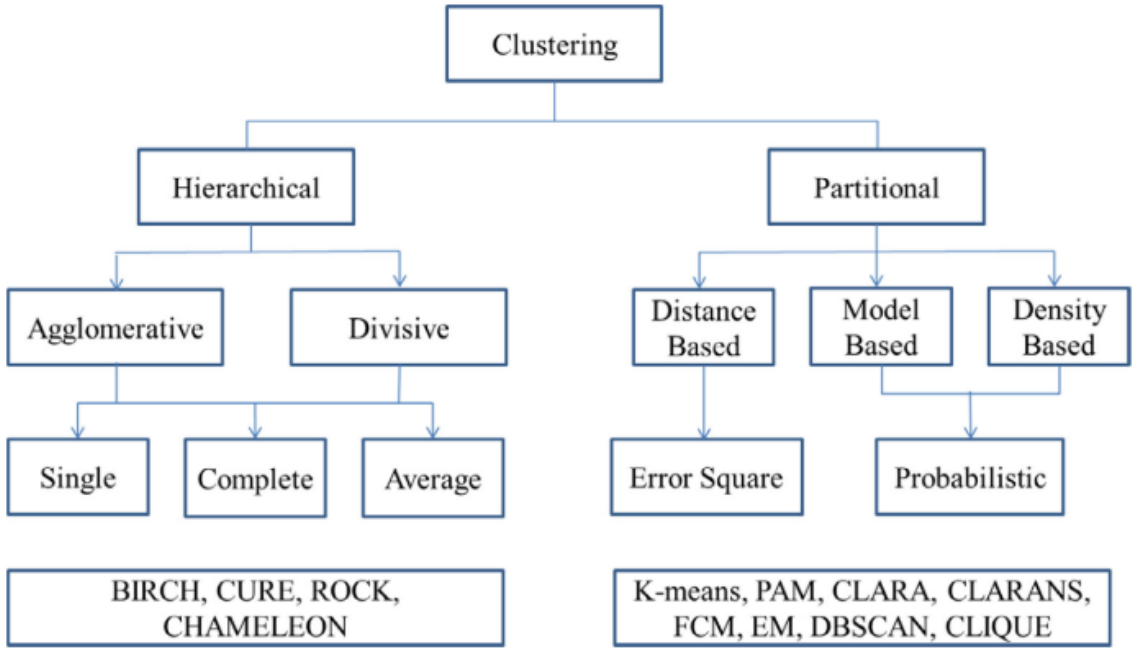


Figure 5. *Clustering approach categories*
Note: A summary of common clustering approaches, including hierarchical and partitional techniques. Reprinted from A Review of Clustering Techniques and Developments *(Fraley & Raftery, 1998, as cited in Saxena et al., 2017)*.

2.8 Dimensionality Reduction Techniques

High dimensionality as discussed, can present challenges for clustering and analysis due to computational complexity and noise sensitivity (Jabari et al., 2024). Dimensionality reduction techniques can assist in discovering the underlying structure of the data, simplifying it to improve interpretability. Even if dimensionality reduction is not required for clustering in this project, it may still be necessary for visualising clusters in the latent space in two dimensions.

Wang et al. (2023) review 24 two-dimensional reduction methods, and benchmark 21 using 110 real and 425 synthetic datasets. Although their work is in the context or preserving biological structures, their work is still relevant to this project, although they advise that the selection of a method should be guided by characteristics of the data and analysis requirements. To evaluate the accuracy of the methods, 16 metrics were selected, categorised into four performance aspects:

1. Global structure preservation.
2. Local structure preservation.
3. Effectiveness for downstream analysis.
4. Similarity to the dataset.

Of the methods tested that are *not* specific to single-cell analysis, they found the best performing were UMAP and t-SNE. Whilst UMAP was also better for downstream analysis tasks – for example, clustering – PCA was found to be beneficial for runtime and memory usage.

Oskolkov (2022, pp. 151–167) outlines these approaches as follows:

- **Principal Component Analysis (PCA)** – a linear dimensionality reduction method that identifies the orthogonal directions (principal components) along which the data varies the most. PCA is used to reduce the number of features and to reduce noise in the data. Although simple and interpretable, it may not effectively represent non-linear structures found in some high-dimensional data.
- **t-distributed Stochastic Neighbour Embedding (t-SNE**) – a non-linear method that first constructs a graph by calculating pairwise probabilities, which represent the likelihood that two points are neighbours in a high-dimensional space. Following this, it attempts to map the high-dimensional graph to a low-dimensional space whilst preserving local structure (nearest neighbour relationships). t-SNE is widely used for data visualisations due to its ability to maintain local structure, although it may misrepresent global relationships and create clusters that are not always meaningful. Furthermore, it can be computationally expensive, sensitive to hyperparameters and limited to producing only two or three-dimensional representations (Mukherjee, 2023).
- **Uniform Manifold Approximation and Projection (UMAP)** – a non-linear, faster alternative to t-SNE. UMAP has the ability to represent data in more than three-dimensions (unlike t-SNE), making it useful for general purpose dimensionality reduction besides visualisations. Additionally, UMAP tends to preserve more global structure compared to t-SNE.

Whilst each method has its strengths and limitations, PCA offers efficiency and interpretability, whereas t-SNE and UMAP are better suited for capturing more of the underlying patterns and provide better visualisation capabilities. However, both t-SNE and UMAP can be sensitive to hyperparameters, meaning they require careful tuning. The choice of method should ultimately be guided by the characteristics of the data and the specific analytical goals.

2.9 Conclusion

This review has explored the history and broader context of evolutionary computation, its application in evolving robotic designs, and the challenges of measuring diversity. Evolutionary algorithms such as Genetic Algorithms and Differential Evolution have proven their effectiveness and adaptability, however the need to maintain diversity within populations remains a challenge for effective exploration and optimisation. Traditional diversity metrics can provide insight, although they often fail to fully capture underlying morphological complexities and relationships. Variational Autoencoders (VAEs) offer the prospect of a more accurate alternative by leveraging a structured latent space to uncover deeper, more meaningful complexities and variations in designs. By combining VAEs with techniques such as dimensionality reduction and clustering, diversity can be measured and visualised more effectively, and in turn advance the field of evolutionary robotics among other areas.

3 Methodology

3.1 Introduction to Methodology

The goal of this study is to determine whether a Variational Autoencoder (VAE) can learn a meaningful latent space that captures diversity between evolved robots. This section outlines the approach taken to develop and evaluate the VAE including data processing, architectural design and custom loss functions to address data sparsity. Additionally, it describes the tuning process, hyperparameter selection and model refinements based on observed limitations. Finally, the evaluation plan is presented, detailing how the VAEs ability to capture diversity is assessed using the model's latent space.

3.2 Data Preparation and Representation

3.2.1 Data Cleaning and Dataset Splits

The dataset used in this study was provided by Leni K. Le Goff and consists of robot designs generated using a compositional pattern-producing network (CPPN). These designs are based on an $11x11x11$ voxel (three-dimensional pixels) grid and consist of empty space, wheel, joint, caster and sensor descriptor values, represented by integer values in the range $[0,4]$, respectively. This project builds upon the existing work of Le Goff & Smith (2024) by exploring diversity measurement through a VAE-learnt latent space. Whilst the VAE training and analysis are independently developed in this study, the use of an EC-based dataset ensures contextual accuracy and relevance, situating this work within an evolutionary framework.

The data was generated without the robot skeleton information due to complexities in the generation process, resulting in extremely sparse data. The dataset consisted of $280{,}375$ robot designs from 28 CSV files, combined into a single data frame before cleaning.

Figure 6 summarises the raw dataset, highlighting its extreme sparsity, with statistics reflecting only the voxel grid space (excluding ID columns).

```
Combined 28 files into a single DataFrame.
Shape of combined DataFrame: (280375, 1335)

Combined full dataset:
Dataset Summary:
--------------------------------------------------
Dimensions: 280375 rows x 1331 columns

Overall Class Counts:
0    371407785
1      1131552
2       372746
3       187756
4        79286

Proportion of Each Class (%):
0    99.53
1     0.30
2     0.10
3     0.05
4     0.02

Average Zeros Per Row: 1324.68
Maximum Non-zero Values in a Row: 8
Minimum Non-zero values in a Row: 0
Average Non-Zero Classes Per Row: 6.32
Rows with Only Zero Values: 1958 (0.70%)
Rows with Exactly 1 Non-Zero Value: 1061 (0.38%)
Rows with Multiple Non-Zero Values: 277356 (98.92%)
Unique Rows: 27018 (9.64)%
```

Figure 6. *Combined raw dataset summary*
Note: Output from a summary script showing the extreme sparsity of the dataset, with over 99% of voxels belonging to class 0 (empty).

Since some rows contained only zeros and provided no meaningful information, along with duplicate rows being present, the dataset was cleaned to remove redundant data. These preprocessing steps resulted in a final dataset of $27{,}017$ robot samples, summarised in Figure 7.

.

```
Cleaned dataset:
Dataset Summary:
--------------------------------------------------
Dimensions: 27017 rows x 1331 columns

Overall Class Counts:
0    35777374
1       79677
2       38218
3       33740
4       30618

Proportion of Each Class (%):
0    99.49
1     0.22
2     0.11
3     0.09
4     0.09

Average Zeros Per Row: 1324.25
Maximum Non-zero Values in a Row: 8
Minimum Non-zero values in a Row: 1
Average Non-Zero Classes Per Row: 6.75
Rows with Only Zero Values: 0 (0.00%)
Rows with Exactly 1 Non-Zero Value: 309 (1.14%)
Rows with Multiple Non-Zero Values: 26708 (98.86%)
Unique Rows: 27017 (100.00)%
```

Figure 7. *Cleaned dataset summary*
Note: Post cleaning summary showing reduced sample count and slightly improved class balance, although the dataset remains extremely sparse.

The cleaned dataset was split into training, validation and test sets, containing $18{,}911$ (70%), $2{,}702$ (10%) and $5{,}404$ (20%) samples, respectively. These were then saved as CSV files for efficient loading. To mitigate excessive training time, smaller ‘toy’ train and validation datasets were created, each containing approximately 20% of the original dataset. Dataset summaries are presented in Figure 8 (full datasets) and Figure 9 (toy datasets).

```
Training dataset:
Dataset Summary:
--------------------------------------------------
Dimensions: 18911 rows x 1331 columns

Overall Class Counts:
0    25043080
1       55602
2       26698
3       23645
4       21516

Proportion of Each Class (%):
0    99.49
1     0.22
2     0.11
3     0.09
4     0.09

Average Zeros Per Row: 1324.26
Maximum Non-zero Values in a Row: 8
Minimum Non-zero values in a Row: 1
Average Non-Zero Classes Per Row: 6.74
Rows with Only Zero Values: 0 (0.00%)
Rows with Exactly 1 Non-Zero Value: 222 (1.17%)
Rows with Multiple Non-Zero Values: 18689 (98.83%)
Unique Rows: 18911 (100.00)%
```

```
Validation dataset:
Dataset Summary:
--------------------------------------------------
Dimensions: 2702 rows x 1331 columns

Overall Class Counts:
0    3578238
1       7898
2       3796
3       3397
4       3033

Proportion of Each Class (%):
0    99.50
1     0.22
2     0.11
3     0.09
4     0.08

Average Zeros Per Row: 1324.29
Maximum Non-zero Values in a Row: 8
Minimum Non-zero values in a Row: 1
Average Non-Zero Classes Per Row: 6.71
Rows with Only Zero Values: 0 (0.00%)
Rows with Exactly 1 Non-Zero Value: 37 (1.37%)
Rows with Multiple Non-Zero Values: 2665 (98.63%)
Unique Rows: 2702 (100.00)%
```

```
Test dataset:
Dataset Summary:
--------------------------------------------------
Dimensions: 5404 rows x 1331 columns

Overall Class Counts:
0    7156056
1      16177
2       7724
3       6698
4       6069

Proportion of Each Class (%):
0    99.49
1     0.22
2     0.11
3     0.09
4     0.08

Average Zeros Per Row: 1324.21
Maximum Non-zero Values in a Row: 8
Minimum Non-zero values in a Row: 1
Average Non-Zero Classes Per Row: 6.79
Rows with Only Zero Values: 0 (0.00%)
Rows with Exactly 1 Non-Zero Value: 50 (0.93%)
Rows with Multiple Non-Zero Values: 5354 (99.07%)
Unique Rows: 5404 (100.00)%
```

Figure 8. *Train, validation and test dataset summaries (full)*
Note: Summary statistics after splitting the cleaned dataset. Class imbalance and sparsity remain similar across all subsets.

```
Toy training dataset:
Dataset Summary:
--------------------------------------------------
Dimensions: 3781 rows x 1331 columns

Overall Class Counts:
0    5007001
1      10951
2       5587
3       4724
4       4248

Proportion of Each Class (%):
0    99.49
1     0.22
2     0.11
3     0.09
4     0.08

Average Zeros Per Row: 1324.25
Maximum Non-zero Values in a Row: 8
Minimum Non-zero values in a Row: 1
Average Non-Zero Classes Per Row: 6.75
Rows with Only Zero Values: 0 (0.00%)
Rows with Exactly 1 Non-Zero Value: 39 (1.03%)
Rows with Multiple Non-Zero Values: 3742 (98.97%)
Unique Rows: 3781 (100.00)%
```

```
Toy validation dataset:
Dataset Summary:
--------------------------------------------------
Dimensions: 541 rows x 1331 columns

Overall Class Counts:
0    716418
1      1616
2       765
3       691
4       581

Proportion of Each Class (%):
0    99.49
1     0.22
2     0.11
3     0.10
4     0.08

Average Zeros Per Row: 1324.25
Maximum Non-zero Values in a Row: 8
Minimum Non-zero values in a Row: 1
Average Non-Zero Classes Per Row: 6.75
Rows with Only Zero Values: 0 (0.00%)
Rows with Exactly 1 Non-Zero Value: 7 (1.29%)
Rows with Multiple Non-Zero Values: 534 (98.71%)
Unique Rows: 541 (100.00)%
```

Figure 9. *Train and validation dataset summaries (toy)*
Note: Summary statistics for reduced sample size toy datasets, used for early experimentation. Class imbalance and sparsity remain similar to the full dataset.

3.2.2 Initial Data Representation (11x11x11 Voxel Grid)

The initial data representation involved reading the CSV file rows into a PyTorch tensor and reshaping into an $(11, 11, 11)$ grid to preserve the natural voxel structure. This explicit voxel representation simplified loss calculations and assisted in the use of evaluation metrics such as F1 score and accuracy.

However, the dataset was extremely sparse, with approximately 99.5% of values consisting of zeros and at most 8 non-zero values per robot. This overwhelming presence of empty space was suspected to be causing the model significant learning difficulties, requiring a new approach for the data representation.

Figure 10 illustrates a typical reconstruction failure when using this natural representation, highlighting the model's inability to learn meaningful features.

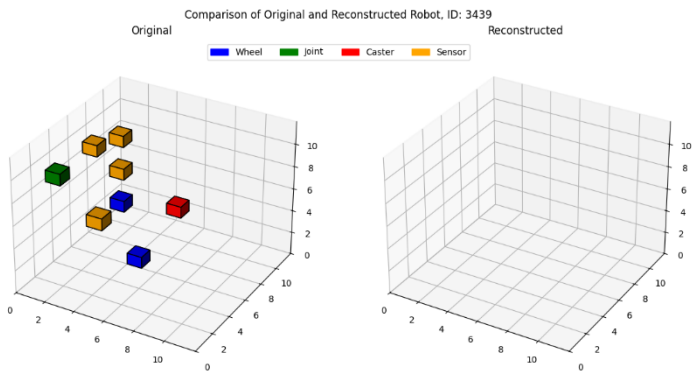


Figure 10. *Typical reconstruction failure with natural data representation*
Note: The left image shows an original input robot in its 11x11x11 grid, where the colours represent blue-wheels, green-joints, red-casters, yellow-sensors. The right image shows the corresponding VAE reconstruction, in which no voxels were produced, illustrating a reconstruction failure caused by data sparsity.

3.2.3 Sparse Representation with Padded Voxels

After multiple architectural and hyperparameter trials (Section 3.8), the model continued to overfit and struggle to learn. To address this, a sparse representation was introduced.

This involved implementing a custom PyTorch dataset class where only non-zero voxel coordinates were extracted and normalised to the range $[0, 1]$ to improve numerical stability. Descriptor values were then one-hot encoded and concatenated to the coordinates to form a single row per voxel. Since robots in the dataset had at most eight voxels, padding rows were added to ensure a consistent input shape for the neural network. Initially, padded voxels were encoded with coordinates $(0, 0, 0)$,

requiring the loss function to differentiate them based on the one-hot descriptor encoding. However, issues with the loss function led to a modification where padded voxels were re-encoded as $(11, 11, 11)$ (normalised to $(1, 1, 1)$). This prevented conflicts with valid coordinate values and allowed padded voxels to be identified using both coordinate and descriptor encodings.

Figure 11 illustrates the sparse representation structure. To prevent positional bias in the model, the voxel rows were shuffled. However, this introduced additional complexity, requiring custom loss functions and penalties (Section 3.4.3).

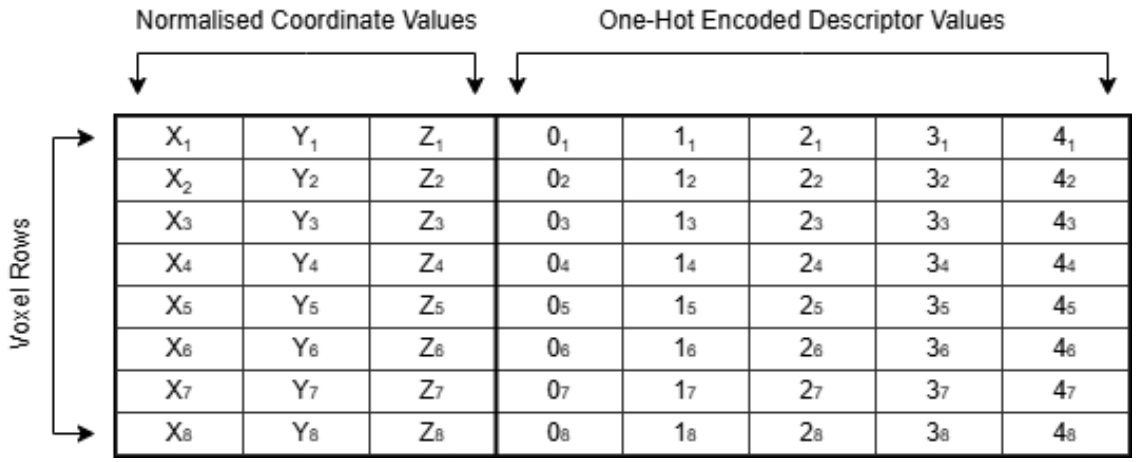


Figure 11. *Sparse data representation structure*
Note: Each robot is represented as a set of voxel rows, where each row includes normalised 3D coordinates (X, Y, Z) and one-hot encoded descriptor values $(0 - 4)$ for component types. Padded voxels are used to enable consistent input size across varying robot designs. Created by the author using diagrams.net.

This modification, combined with an updated model architecture designed to process sparse coordinate encodings (PointNet, Section 3.8.2), led to immediate, albeit minor, improvements. The model began reconstructing at least one voxel per robot, with variations in both location and descriptor type, demonstrating that the model had some capacity to learn meaningful features.

3.3 Initial Model Architecture

The initial model architecture was designed as a baseline for comparison during later enhancements and was based on the VAE structure outlined in Section 2.6.2.

To keep the design simple initially, only fully connected (FC) linear layers were used. The model's encoder took a PyTorch tensor with the shape $(batch, 11, 11, 11)$ as input, which was then flattened to $(batch, 1331)$. It further consisted of 3 FC layers, with ReLU activation functions (Rectified Linear Units; $ReLU(x) = max(0, x)$) applied

after each, except the final layer, ensuring the latent vectors remained unbounded. The final layer was split into 2 parallel FC layers:

- One producing the μ vector (z_{mean}).
- One producing the σ vector (z_{log_var}).

Although these are two separate FC layers, they were treated as a single branching layer, as they operate in parallel. This structure allowed the model to separately learn the mean and variance components of the latent distribution.

As discussed in Section 2.6.2, the log variance was used for numerical stability. This approach allowed for flexibility in the encoder's output, which is typically unconstrained, whilst avoiding the need for explicit constraints to keep σ positive. The reparameterisation trick:

$$z = \mu + \sigma \odot \varepsilon$$

requires σ to be positive, as standard deviation cannot be negative. Treating the encoder's output as the log variance allows σ to be calculated as:

$$\sigma = exp(0.5\ log\ \sigma^2)$$

The encoders outputs; z_{mean} and z_{log_var} (with shape $(batch, latent\_dimension)$, set when determining hyperparameter selection, see Section 3.6) were passed to the sampling function. This function drew a sampled latent space vector z (also with shape $(batch, latent\_dimension)$) whilst allowing for backpropagation by using the reparameterisation trick. The code implementation of this process is provided in Appendix G, Figure G.65.

The decoder was set up to mirror the encoder, systematically reintroducing features in a similar way to how they had been extracted in the encoder. The decoder consisted of three FC layers, with ReLU activation functions applied after layers one and two, and sigmoid activation function (sigmoid; $\sigma(x) = \frac{1}{1+e^{-x}}$) applied after the final layer. This ensured that the output remained within the range $[0,1]$. Figure 12 illustrates the base model's architecture.

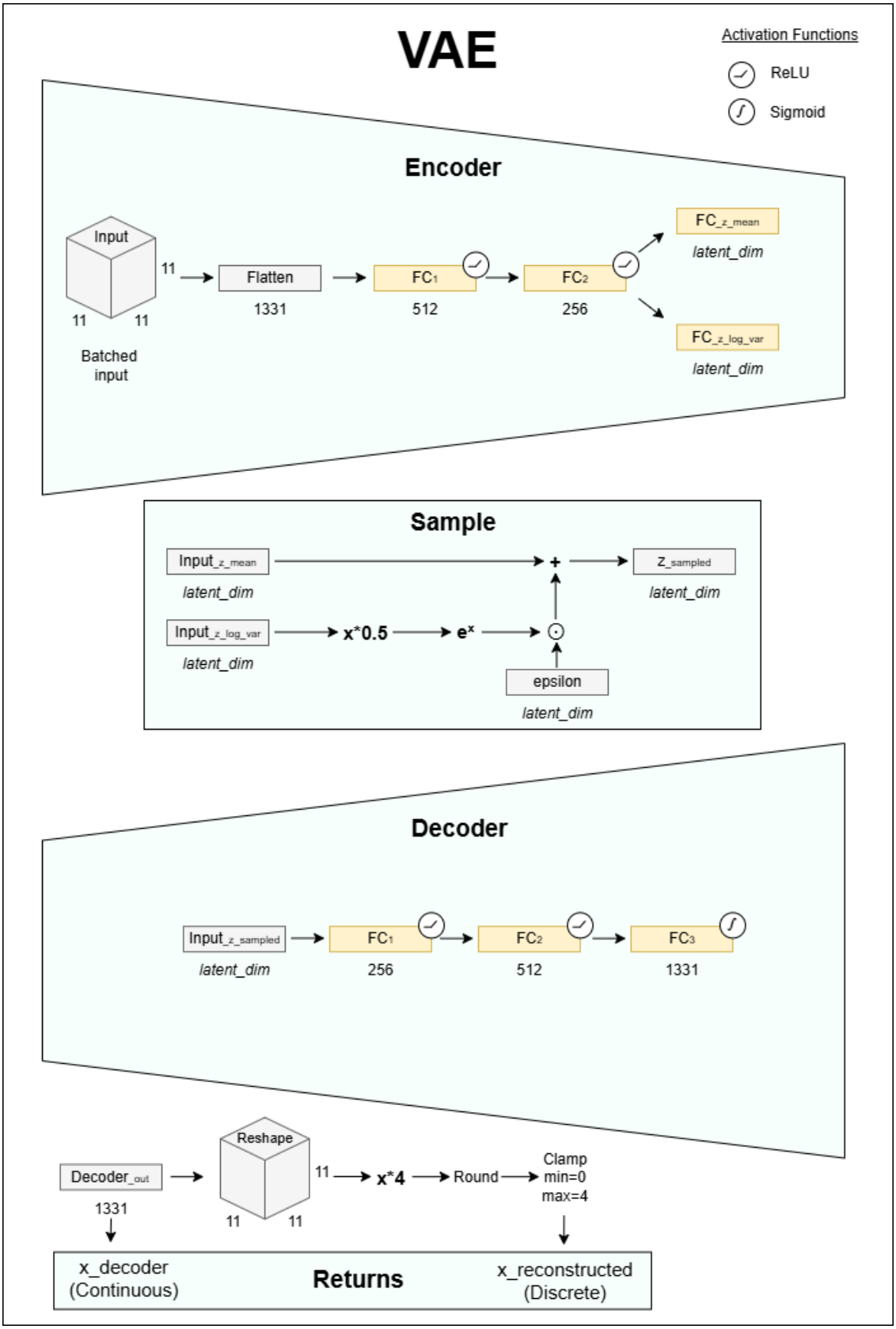


Figure 12. *Base model architecture*

Note: The model consists of an encoder, sampling layer, and decoder. The encoder flattens the $11 \times 11 \times 11$ input into a vector of 1331 elements and passes it through FC layers to predict the latent mean and log variance. The reparameterisation trick samples a latent vector, which is passed through the decoder to reconstruct the input. Outputs include a continuous decoder output ($x_{decoder}$) and a discrete, non-differentiable version ($x_{reconstructed}$) obtained by scaling, rounding, and clamping the output. Created by the author using diagrams.net

Using the output range $[0, 1]$ allowed for different loss functions to be used and meaningfully compared on the same scale. For instance, Binary Cross-Entropy (BCE) loss expects inputs in the range $[0, 1]$, whereas Mean Squared Error (MSE) loss can accept any continuous value range. However, for visualisations and computing metrics such as F1 score and accuracy, adjustments were required to the outputs to reflect their natural grid values. These adjustments included:

- Scaling the decoder output from $[0, 1]$ to $[0, 4]$ to match the original descriptor values.
- Rounding to ensure integer values.
- Clamping to restrict values within the valid descriptor range.

Since rounding and clamping are non-differentiable, they could not be used during loss calculations. To address this, the VAE model returned both:

- $x_{decoder}$ – the original continuous output used for loss calculations.
- $x_{reconstructed}$ – the adjusted output used for visualisations and metrics such as F1 score and accuracy.

Following the transition to the sparse representation, the scaling, rounding and clamping could be removed due to working with normalised coordinates and one-hot encoded descriptor values.

3.4 Loss Functions

3.4.1 Initial Loss Functions

3.4.1.1 Reconstruction Loss

For early training runs, two common loss functions – Mean Squared Error (MSE) and Binary Cross Entropy (BCE) – were selected for comparison. These were chosen because the decoder output was normalised to a continuous range between zero and one. MSE is good for regression tasks, where the goal is to minimise the difference between predicted and true values, whilst BCE is typically used for probabilistic type outputs. Both loss functions are differentiable, which is crucial for optimisation algorithms, and they are commonly used for reconstruction loss in VAEs, making them ideal choices for comparison (Terven et al., 2023).

3.4.1.2 KL Divergence

KL divergence can be used to measure how much the latent space distribution, parameterised by z_{mean} and z_{log_var} differs from a standard normal distribution (Gaussian distribution with $mean$ equals zero and $standard\ deviation$ equals one). KL divergence has the following closed form:

$$D_{kl}[\mathcal{N}(\mu, \sigma) \,||\, \mathcal{N}(0, 1)] = -\frac{1}{2}\sum(1 + log(\sigma^2) - \mu^2 - \sigma^2)$$

which is implemented using PyTorch in Appendix G, Figure G.66 (Foster, 2023, Chapter 3).

3.4.2 Class Weighting for Initial Data Representation

To handle the extreme data sparsity (Section 3.2), a class weighting was introduced for the first training run, as it was assumed that the model would overfit and predict only zeros (empty space). Initially, the weighting applied was the multiplicative inverse of the batch support (calculated from the predictions table outlined in Section 3.5) normalised by the sum of the weights (Appendix G, Figure G.67).

Once the raw reconstruction loss had been calculated, the class weights were applied element wise, before taking the mean to obtain the class weighted reconstruction loss. The weighted reconstruction loss could then be added to the KL divergence to obtain the total loss and then used for backpropagation and adjusting the model's weights.

Applying linear class weighting prevented overfitting to the majority class but caused reconstructions biased toward minority classes (Figure 13). To overcome this, log weighting was introduced, calculating weights as the reciprocal of the log scaled batch support, thereby balancing loss severity with descriptor frequency more effectively (Appendix G, Figure G.68).

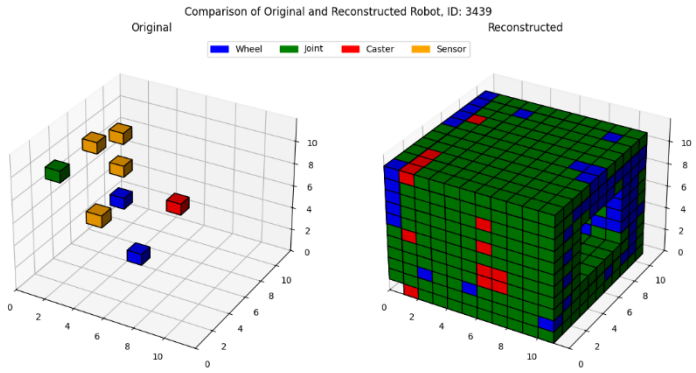


Figure 13. *Reconstructed comparison using multiplicative inverse class weighting*
Note: Original robot (left), VAE reconstruction (right). Overfitting to minority classes can be seen, highlighting the limitations of this weighting strategy.

Figure 14 illustrates applying the weighting using a log scaling did start to produce reconstructions similar to the input. However, the reconstructions were static both in voxel location and descriptor type (colour) regardless of input, indicating the model was lacking sufficient capacity.

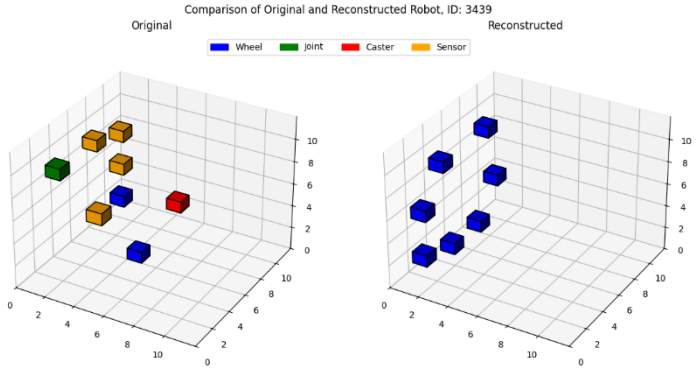


Figure 14. *Reconstructed comparison using log scaled class weighting*
Note: Original robot (left), VAE reconstruction with log scaled weighting (right). Whilst improved, all reconstructions remained static, suggesting insufficient model capacity.

With the transition to the sparse data representation (Section 3.2.3), the class weighting approach was no longer needed. However, voxel shuffling to avoid positional bias presented a significant challenge; standard loss functions rely on a fixed order of input and target values, which was no longer guaranteed. This required the introduction of custom loss functions and penalties to account for voxel order correspondence uncertainty, whilst maintaining differentiability.

3.4.3 Custom Loss Functions and Penalties

Custom loss and penalty functions were introduced to encourage the model to learn voxel spatial coordinates, along with descriptor types accounting for the voxel shuffling operation of the sparse representation. These functions were scaled with

lambda values and combined additively to produce the reconstruction loss and then added to the KL divergence to produce the total loss. Three functions were implemented as a replacement to the standard loss function originally used (Section 3.4.1.1), which was not compatible due to relying on order correspondence between input and target values:

1. `coordinate_matching_loss` (Algorithm 1)
2. `descriptor_matching_loss` (Algorithm 2)
3. `overlap_penalty` (Algorithm 3)

Attention was paid to maintaining differentiability for effective learning and faster convergence. The functions went through various modifications and versions, however only the final functions used are presented here for clarity, see Appendix H for the full implementation.

3.4.3.1 Coordinate Matching Loss

The coordinate matching loss ensured that reconstructed voxels aligned with original spatial positions by penalising clustering and misalignment. This was achieved using entropy-based attention distributions and a distance-based regression loss (MSE) to guide reconstructions toward correct placements.

Entropy is a measure of uncertainty in an entire probability distribution, where nearly deterministic distributions have low entropy, whilst high entropy when closer to uniform (Figure 15; I. Goodfellow, 2016, Chapter 3). Entropy $(H(x))$ can be expressed as:

$$H(x) = -\mathbb{E}_{x\sim P}[log\ P(x)] = -\sum p(x) log\ p(x)$$

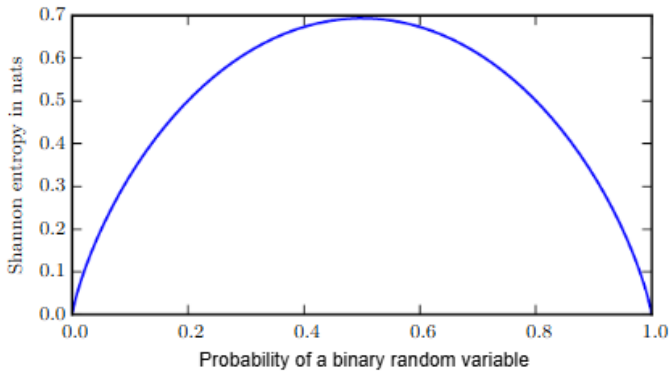


Figure 15. *Entropy of a binary random variable*
Note: The curve shows Shannon entropy (in nats) relative to the probability of a binary variable being true. Entropy is highest when the probability is 0.5, reflecting maximum uncertainty. Adapted from Deep Learning (I. Goodfellow, 2016, Chapter 3).

The basic idea of the loss function is to form a probability distribution over the original voxels based on how much attention they receive from reconstructed voxels. This can be achieved by computing a pairwise distance matrix between reconstructed voxels and original voxels and using this to calculate probable nearest neighbours, which can then in turn be used to form an attention per original voxel distribution.

Entropy measures the uncertainty of the attention distribution. A uniform distribution (where all original voxels receive equal attention) has maximum entropy, whilst clustering of reconstructed voxels around a few original voxels leads to lower entropy. The loss is designed to maximise entropy whilst maintaining spatial accuracy.

Maximum entropy is computed from a uniform distribution, allowing the entropy expression to be rearranged as shown:

$$uniform\ distribution: p(x) = \frac{1}{n}$$

$$H(x) = -\sum(p(x) \cdot log\ p(x))$$

$$H_{max}(x) = -\sum_{1}^{n}(\frac{1}{n} \cdot log(\frac{1}{n}))$$

$$log(\frac{1}{n}) = -log(n)$$

$$H_{max}(x) = log(n)$$

where n is the number of max dataset voxels (number of rows in the sparse data representation).

Calculating the entropy loss encourages the model to give all original voxels equal attention from the reconstructed voxels and allow the model to evenly distribute each reconstructed voxel in a similar way to how the original voxels are spread-out. However, it does not encourage perfect alignment; so long as each reconstructed voxel is near a *different* original voxel, the loss calculated would be close to, or equal to zero. Due to this, the coordinate matching loss consists of two parts:

1. The entropy part (already discussed) which helps avoid clustering of reconstructed voxels around a single original voxel.
2. A regression-based MSE loss, to measure distance to estimated voxels and encourage perfect alignment to original voxels.

The regression part takes the probable nearest neighbour weights matrix, constructed for the entropy calculation, and applies this using matrix multiplication to the original coordinates. This allowed the closest original coordinate to be extracted for each reconstructed coordinate as a target value, overcoming the shuffling operation and no order correspondence. Once target coordinates had been obtained, standard MSE could be applied as a distance-based loss and added to the entropy loss for the total coordinate matching loss. Algorithm 1 shows the pseudocode for the coordinate matching loss function.

Algorithm: Coordinate Matching Loss

Input: Original input x,
Reconstructed output $x\_reconstructed$

$total\_loss \leftarrow 0.0$
$orig\_coords \leftarrow$ batched voxel coordinates of x
$recon\_coords \leftarrow$ batched voxel coordinates of $x\_reconstructed$

foreach $orig, recon$ *in* $(orig\_coords, recon\_coords)$ **do**
  $dist\_matrix \leftarrow$ pairwise distance between $recon$ and $orig$ to form a matrix where each row is distance from reconstructed voxels, and each column is distance from original voxels
  $neighbour\_weights \leftarrow$ softmin of $dist\_matrix$ rows to get probabilities of closest original voxels for each reconstructed voxel
  $attention\_per\_original \leftarrow$ sum columns of $neighbour\_weights$
  $attention\_distribution \leftarrow$ normalise $attention\_per\_original$ by its sum to form distribution
  $entropy \leftarrow$ calculate entropy from $attention\_distribution$
  $max_entropy \leftarrow \log($size of $attention\_distribution)$
  $entropy\_loss \leftarrow (max\_entropy - entropy)^2$
  $target\_coord \leftarrow$ matrix multiplication: $neighbour\_weights$ @ $orig$
  $dist\_loss \leftarrow$ mean squared error between $recon$ and $target\_coord$
  $total\_loss \leftarrow total\_loss + entropy\_loss + dist\_loss$
end

$batch\_loss \leftarrow$ average $total\_loss$ over batch

return $batch\_loss$

Algorithm 1. *Coordinate matching loss*
Note: This is intended to simplify the code to make it more readable. Created by the author using the PseudoCode-AlgorithmTemplate in Overleaf.

3.4.3.2 Descriptor Matching Loss

The descriptor matching loss encourages correct reconstruction of descriptor values by computing a weighted target descriptor class for each voxel based on spatial proximity. A cross-entropy loss is then applied to measure the dissimilarity between predicted and target descriptors.

The nearest original voxels were calculated in the same way as in the coordinate matching loss function, forming a neighbour weights matrix. This matrix could then be applied with matrix multiplication to the original descriptor values, from which the target descriptor class could be extracted.

Cross entropy loss is a multi-class classification loss function and used to measure the dissimilarity between predicted probabilities and target classes (Terven et al., 2023). Once target descriptor classes had been identified, cross entropy loss could then be used to calculate the descriptor loss. Algorithm 2 shows the pseudocode for the descriptor matching loss function.

Algorithm: Descriptor Matching Loss

Input: Original input x,
Reconstructed output $x\_reconstructed$

$total\_loss \leftarrow 0.0$
$orig\_coords \leftarrow$ batched voxel coordinates of x
$orig\_descriptors \leftarrow$ batched descriptor values of x
$recon\_coords \leftarrow$ batched voxel coordinates of $x\_reconstructed$
$recon\_descriptors \leftarrow$ batched descriptor values of $x\_reconstructed$

foreach $orig\_coor, recon\_coor, orig\_desc, recon\_desc$ *in*
$(orig\_coords, recon\_coords, orig\_descriptors, recon\_descriptors)$ **do**
  $dist\_matrix \leftarrow$ pairwise distance between $recon\_coor$ and $orig\_coor$ to form a matrix where each row represents distances from reconstructed voxels, and each column is distance from original voxels
  $neighbour\_weights \leftarrow$ softmin of $dist\_matrix$ rows to get probabilities of closest original voxels for each reconstructed voxel
  $matched\_target\_descriptors \leftarrow$ matrix multiplication: $neighbour\_weights$ @ $orig\_desc$ to get descriptor probabilities for each reconstructed voxel
  $target\_cls \leftarrow$ index of maximum value for each row in $matched\_target\_descriptors$ to get target descriptor class
  $descriptor\_loss \leftarrow$ cross-entropy between $recon\_desc$ (logits) and $target\_cls$
  $total\_loss \leftarrow total\_loss + descriptor\_loss$
end

$batch\_loss \leftarrow$ average $total\_loss$ over batch

return $batch\_loss$

Algorithm 2. *Descriptor matching loss*
Note: This is intended to simplify the code to make it more readable. Created by the author using the PseudoCode-AlgorithmTemplate in Overleaf.

3.4.3.3 Overlap Penalty

The overlap penalty prevents reconstructed voxels from collapsing into the same position by penalising extreme proximity. Due to working with normalised values, the scenario exists for reconstructed voxels collapsing into the same voxel when scaled up for visualisation, resulting in duplicates. Padded voxels on the other hand are allowed to merge into the same voxel.

This penalty works by scaling the reconstructed coordinates in the same way as is done for visualisation (Section 3.7.3) to obtain the natural values. To ensure values are integers, and in the correct range, they are rounded and clamped. The rounding and clamping operations are non-differentiable which can negatively affect backpropagation. To overcome this problem, scaled coordinate values are first masked based on descriptor values, before iterating each to check if any coordinate matches, skipping masked values along with padded voxel encoded coordinate values ($11's$). If matches are found, the matched coordinates are used to index and identify the unscaled reconstructed coordinates, effectively bypassing the differentiability issue.

Pairwise distances are then taken between matching coordinates, followed by taking the log distance to proportionally penalise the model the closer voxels get. The negative sum of the upper triangle of the log distance matrix is then taken, which ensures a positive penalty, whilst avoiding self-distances and double counting distances in each direction. The pseudocode for the overlap penalty function is illustrated in Algorithm 3.

Since coordinates are normalised and only distances close enough to collapse are penalised, distance values should always be less than one. As a result, the logarithm of any such distance will always be negative. A small constant $1e^{-8}$ is added to avoid computing $log(0)$, which is undefined. This behaviour is demonstrated in Appendix G, Figure G.69, which tests the log distance between two points at maximum distance.

Algorithm: Overlap Penalty

Input: Reconstructed output $x\_reconstructed$

$total\_penalty \leftarrow 0.0$
$recon\_coords \leftarrow$ batched voxel coordinates of $x\_reconstructed$
$recon\_mask \leftarrow$ descriptor values mask, True for non-padded voxels, False for padded

foreach $coords, mask$ *in* $(recon\_coords, recon\_mask)$ **do**
  $scaled\_coords \leftarrow$ scale $coords$ to natural expanded grid coordinate range
  $rounded\_coords \leftarrow$ round and clamp $scaled\_coords$ to represent values as visualised (non-differentiable operation)
  $rounded\_coords\_masked \leftarrow$ Mask padded voxels in $rounded\_coords$ with ∞ based on descriptor values to only penalise visualised voxels
  $overlapping\_idx \leftarrow$ initialise empty set for tracking checked coordinates
  foreach i *in range* 0 *to size of* $rounded\_coords\_masked$ **do**
    $current\_coord \leftarrow rounded\_coords\_masked[i]$
    if $current\_coord$ *is padded (any* ∞ *or padded coordinate values) or already found* $(i$ *in* $overlapping\_idx)$ **then**
      Continue to next iteration
    end
    $matches \leftarrow$ check for exact matches of all coordinate values between $current\_coord$ and $rounded\_coords\_masked$ to form boolean vector
    $match\_indices \leftarrow$ indices of True values in $matches$
    if $match\_indices$ *contains more than one element (not counting self-matching)* **then**
      $dist \leftarrow$ pairwise distance between unscaled $coords$ to maintain differentiability using $match\_indices$ to identify
      $scaled\_dist \leftarrow \log(dist + 1e^{-8})$ to proportionally discourage model the closer voxels get. $1e^{-8}$ used to avoid $\log(0)$
      $penalty \leftarrow$ calculate penalty by taking the negative sum of the upper triangle of $scaled\_dist$ to not double count $(a \leftarrow b \ \& \ b \rightarrow a)$ and excluding diagonal to not include self-distances
      $total\_penalty \leftarrow total\_penalty + penalty$
      Add $match\_indices$ to $overlapping\_idx$
    end
  end
end

$batch\_penalty \leftarrow$ average $total\_penalty$ over batch

return $batch\_penalty$

Algorithm 3. *Overlap penalty*
Note: This is intended to simplify the code to make it more readable. Created by the author using the PseudoCode-AlgorithmTemplate in Overleaf.

3.5 Metrics

Whilst loss functions guide optimisation, classification metrics provide additional insight into reconstruction accuracy and model performance. To compute these metrics, a prediction table was generated by comparing the original input (x) and the reconstructed output ($x_{reconstructed}$). In this table:

- Each row represents a descriptor label class.
- Each column represents a reconstructed (predicted) descriptor value.
- Diagonal values indicate correct predictions (true positives), whilst off-diagonal values represent misclassifications.

The pseudocode for constructing the prediction table is provided in Algorithm 4.

Algorithm: Compute Prediction Table

Input: Original input x,
Adjusted VAE output $x\_reconstructed$,
Number of classes $NUM\_CLASSES$

$x\_flat \leftarrow$ reshape x into 1D vector
$x\_reconstructed\_flat \leftarrow$ reshape $x\_reconstructed$ into 1D vector
$prediction\_table \leftarrow NUM\_CLASSES \times NUM\_CLASSES$ zeros matrix

foreach *index and class_label in* x_flat **do**
  row index $\leftarrow class\_label$
  column index $\leftarrow x\_reconstructed\_flat[index]$
  Increment $prediction\_table[row\_index, column\_index]$
end

return $prediction\_table$

Algorithm 4. *Compute prediction table*
Note: This is intended to simplify the code to make it more readable. Created by the author using the PseudoCode-AlgorithmTemplate in Overleaf.

The following metrics were then computed from the prediction table:

- $Accuracy = \frac{Correct\ predictions}{All\ predictions} = \frac{Sum\ of\ diagonal\ elements}{Sum\ of\ all\ elements\ in\ the\ table}$
- $Recall = \frac{TP}{TP + FN} = \frac{Diagonal\ elements}{Sum\ of\ corresponding\ row}$
- $Precision = \frac{TP}{TP + FP} = \frac{Diagonal\ elements}{Sum\ of\ corresponding\ column}$
- $F1\ score = 2\frac{(Precision \cdot Recall)}{(Precision + Recall)}$
- $Batch\ support = TP + FN = sum\ row\ elements$

where TP is true positives, FN is false negatives and FP is false positives.

Batch-support-weighted averages were used to track recall, precision and F1 score, whilst accuracy was averaged across all batches per epoch.

After moving to the sparse representation (Section 3.2.3) the prediction table was no longer necessary. Since descriptor values were one-hot encoded, the confusion components (TP, FN, FP) could be directly computed by comparing predictions and targets using `argmax`, as shown in the code implementation provided in Appendix G, Figure G.70. However, after transitioning to the sparse representation, these metrics became less informative as they no longer accounted for coordinate positions. Therefore, model performance evaluation placed greater emphasis on custom loss functions and penalties, along with visual inspection of reconstructions.

Additionally, for the sparse representation, the mean Euclidean distance between original coordinates and reconstructed coordinates was measured. This metric was tracked to provide an indication of coordinate accuracy. However, since distances were computed based on dataset order, this metric did not account for voxel shuffling, an oversight during development. The implementation can be seen in Appendix G, Figure G.71.

3.6 Training Setup

To maintain common configurations in a single location for ease of adjustments, a configuration file was created. This included:

- **Directory paths**,
- **High-level training configurations** – epochs, patience (for early stopping), scheduler patience (for learning rate adjustments) and number of classes (descriptor values),
- **High-level data and model configurations** – such as expanded (natural) grid size, model input dimensions and dataset maximum voxels (when the sparse representation was used),
- **Seeds for repeatability**,
- **Setting the PyTorch training device** - CPU/GPU.

In addition to these high-level settings, a class was created for tracking performance history. Hyperparameters were set up and adjusted in a grid search python file and checkpointing methods were implemented to store model weights for loading.

3.6.1 History Tracking

A class called `TrainingHistory` was developed for tracking model performance, hyperparameters and metrics. It also logged the number of epochs run and the model architecture used.

Methods were included to save and load a history object, along with a `__str__` method to print important information. Objects were saved using PyTorch's `save()` function to store the dictionary state as a `.pth` file. This made saving straightforward, and due to some attributes containing PyTorch tensors, avoided unnecessary conversions and potential loss of information.

A method to roll back history to a specific epoch was also included. This was useful for continuing training from a given point or when loading an earlier checkpoint. History tracking was kept separate from model checkpointing so that even if a model file was missing, the training history could still be loaded and reviewed.

3.6.2 Checkpointing and Early Stopping

Checkpointing was triggered by an improvement in either:

- Weighted average F1 score.
- Total loss.

A method was incorporated in `TrainingHistory` to check for improvement in these metrics and reset an epoch counter whenever an improvement occurred.

If the current epoch ever reached the epochs setting in the config file, a *final* model file would be checkpointed. Similarly, if the epoch improvement counter reached the patience setting in the config file, a *terminated* model file would be checkpointed. When patience was reached, a Boolean termination flag was returned as `True` from the method to signal early stopping.

All checkpointed files were stored with the current epoch as part of the filename for easy identification. By default, and to save storage space, the `train_val` function responsible for directing the full training cycle calls a function to prune old checkpoint files whenever there is an improvement. This function maintains the latest file for both F1 and total loss checkpoints ensuring one of each remains, and does not affect final or terminated model checkpoint files.

A checkpoint file consists of:

- Model's state dictionary – the parameters/weights.
- Model's `__dict__` – the model's attributes which includes the model's name.
- Model's class name.
- The state dictionary of the optimizer (a PyTorch object) – parameters.
- Optimizer's defaults – initialisation arguments.
- Optimizer's class name.
- Current epoch.
- Scheduler state dictionary and `__dict__` (if using).

Storing all the above, instead of just the state dictionaries, allows full checkpoints to be loaded without the requirement to initialise and pass model, optimizer or scheduler objects to the load function.

Additionally, the load checkpoint function accommodates loading from a checkpoint path, or from a `TrainingHistory` object. An option is included to specify whether to load the model for last updated, last improved, best weighted average F1 or best total loss model.

3.6.3 Grid Search for Hyperparameter Tuning

To effectively explore various hyperparameter settings whilst tuning the model, a grid search approach was implemented. Hyperparameter configurations were set up in two different ways as a list of dictionaries:

- By using `for in` statements from specific hyperparameter list variables of settings – to test all possible permutations of settings.
- Manually – to test specific combinations.

The configuration dictionary specified settings for the following hyperparameters:

- Batch size.
- Latent dimension.
- Optimizer.
- Learning rate.
- Weight decay.
- Beta.

Beta (β) is used for scaling the influence of KL divergence in the total loss (standard VAEs have a β value of one). Using a β scaling means the VAE is a variant known as a *β-VAE*. Higher β values constrain the latent space, which can reduce reconstruction quality. In contrast, lower β values allow greater flexibility, potentially at the cost of a meaningful latent representation (Shende, 2023).

In early training runs where different loss functions were being compared, the loss to be used was selected as part of the grid search parameters. In later runs, when custom losses were used, *lambda* hyperparameters scaled the influence of each loss and penalty.

Each configuration was iterated over, automatically initialising a VAE model, criterion (loss function), optimizer and scheduler (when used) before training and testing for a set number of epochs. After training each configuration, plots were generated, metrics were logged and the model's name recorded in a search file.

Following training of all the specified configurations, the grid search could commence. This involved loading the history of each configuration listed in the search file and calculating a *trade-off score*:

$$score = loss\_f1\_tradeoff \cdot lowest\_loss + (1 - loss\_f1\_tradeoff) \cdot (1 - highest\_f1\_average)$$

where $loss\_f1\_tradeoff$ is a scaling factor. When set to one, it allows searching for the epoch with the lowest loss. Whilst when set to zero, it returns the highest F1 weighted average epoch. Alternatively a balance of the two could be found (as they may not occur in the same epoch) when between the values of zero and one. Note the F1 average is inverted multiplicatively to form a minimisation problem. The history with the lowest trade-off score was then returned from the grid search function.

Training using this grid search approach allowed for efficient setup, training and comparison of different configurations to explore how they affected a model, assisting in architecture modification decisions. This was made more effective by incorporating a timer to calculate the average training time for each configuration and using this to estimate and output the grid search training completion time.

3.7 Training Visualisation and Decision Making

During model design and training, three different types of visualisations were used to assist in decision making:

- Metric trend plots.
- Latent space cluster graphs.
- Reconstruction comparison visualisations.

All plots created were saved with the training run name, along with the specific hyperparameters used as part of the filename for easy identification. This section covers each type used and provides some examples.

3.7.1 Training Metrics Visualisation

One of the primary plots used for decision making were the metric plots. Two types were developed:

- Metric vs epoch trend line plot.
- Loss trade-off scatter plot.

3.7.1.1 Metric vs Epoch Trend Line Plot

The metric vs epoch trend plot could be used to show the different metrics stored in `TrainingHistory` over epochs, an example is shown in Figure 16. Key metrics include beta-scaled KL divergence, total loss, accuracy, F1 weighted average and learning rate.

The function has been developed to support up to six different metrics and can be used to plot either train metrics, validation metrics or both together. The legend has been added as a subplot to maintain clear visualisations.

To support metrics with dramatically different scales, an automated method determines appropriate y-axis limits. The process begins by sorting the maximum values of each metric and computing the global scale ratio between the largest and smallest values. If this ratio is within a set threshold (default equals ten), a single y-axis scale is used. Otherwise, a progressive search is initiated to determine an optimal split.

The search begins with a midpoint heuristic, initially dividing values into two groups. The ratio between the last element of the small value group and the first element of

the large value group is then checked. If this ratio exceeds the threshold, elements are progressively shifted from the large group to the small group until a valid split is found. If no suitable split is found, the search reattempts from the start of the list.

If a valid split is found, the maximum value from each group is returned. Otherwise, the function defaults to a single maximum scale. These values define the y-axis upper limits, which are scaled by 10% to prevent points from touching the plot boundary. When two values are returned, a twin y-axis is used, with larger values plotted on the right. Both axes always start at zero to ensure clarity when comparing values across different plots.

X-axis ticks were calculated by taking the range of one to the final epoch plus one with a step size of $max((epochs\ //\ 7), 1)$. Using max ensured at least one tick was shown, whilst floor division maintained integer ticks, and a divisor of seven was found to work well for a wide range of epoch values.

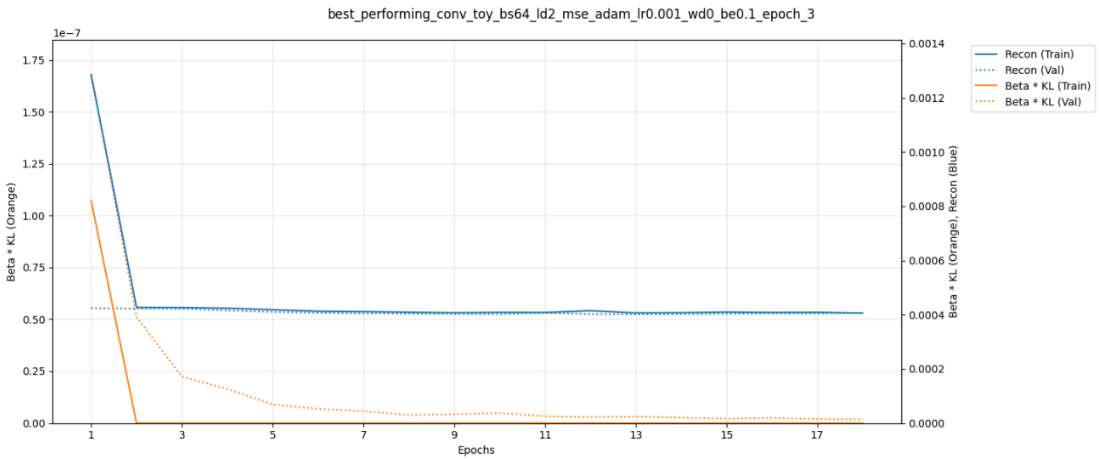


Figure 16. *Example metric vs epoch trend line plot*
Note: Demonstrates one option for the metric vs epoch trend plot. Up to six metrics are supported, and with twin y-axes used when value ranges exceed a ratio threshold. The larger scale is always plotted on the right.

3.7.1.2 Loss Trade-Off Scatter Plot

The second type of metric plot implemented was the loss trade-off scatter plot shown in Figure 17. This was used to plot specific and important metrics that can influence each other, plotting from either train or validation histories, or both. Options include:

- KL divergence vs reconstruction loss.
- Beta-scaled KL divergence vs reconstruction loss.
- Total loss vs weighted F1 average.

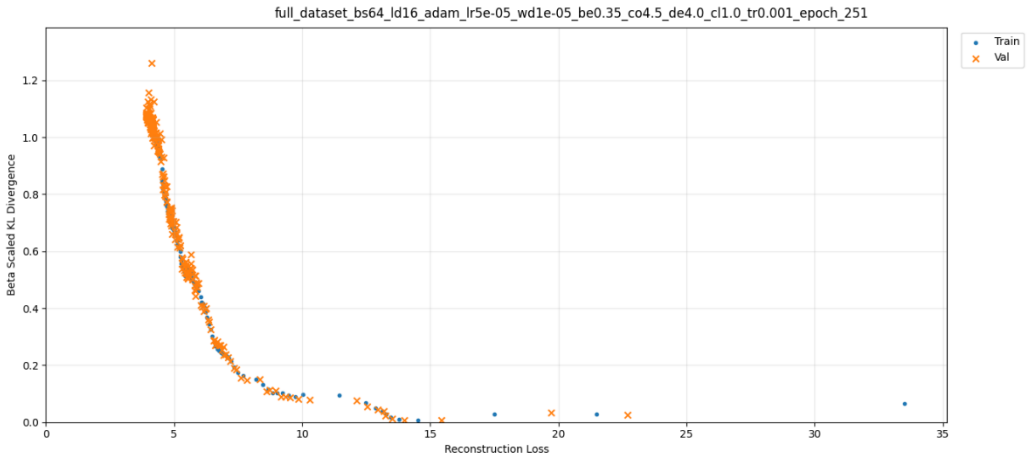


Figure 17. *Example loss trade-off scatter plot*
Note: Demonstrates one option for plotting loss trade-off scatter plots. Here beta-scaled KL divergence is plotted against reconstruction loss.

3.7.2 Latent Space Visualisation During Training

To gain an understanding of how the latent space was behaving during training, PCA and UMAP were used to reduce sampled latent vectors to a two-dimensional representation. This was followed by k-means clustering for visualisation.

First, a sampled latent vector was extracted via a forward pass of the model's encoder and sampling layers using the training data. This vector was then normalised using scikit-learn's `StandardScaler` before applying PCA and UMAP. Next, the validation data was used to extract a sample vector in the same way and then normalised using the scaler fitted on the training data sample vector and finally transformed using the trained reducer models. Normalisation and training were done in this way to prevent data leakage and maintain consistent scaling.

Note, UMAP requires that `n_neighbours` is specified, which balances how UMAP concentrates on local vs global structure (*Basic UMAP Parameters*, 2018). The default value of 15 was used here, with the intention to later tune if time.

The k-means clustering could then be applied to both the PCA and UMAP data to predict cluster labels. Due to the requirement to specify the number of clusters for k-means, the elbow method was used to determine the optimal k (*Elbow Method for Optimal Value of k in KMeans*, 2025; scikit-learn developers, 2025a). As shown in Figure 18, the optimal elbow point was found at $k = 5$. Although this value may not be optimal for all models, it was fixed across training runs to ensure consistency in comparisons.

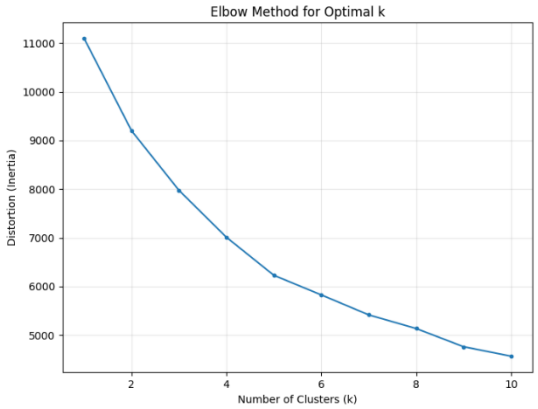


Figure 18. *Elbow method for optimal k-means k value*
Note: The elbow point was identified at $k = 5$, which was fixed across all training runs to ensure consistency in comparisons.

With validation latent samples extracted, dimensionality reduced, and cluster labels identified, a scatter plot could be created. Example plots are shown in Figure 19 and Figure 20. Well separated clusters indicate the model has learnt to differentiate between different underlying factors in the data and imply a meaningful latent space leading to better generalisation. Tightly packed clusters suggest strong similarity between encoded features, indicating that the model has consistently captured shared characteristics. Whereas overlapping clusters may indicate poor disentanglement, where the model struggles to separate distinct features in the latent space (Mathieu et al., 2018; Yadav, 2024).

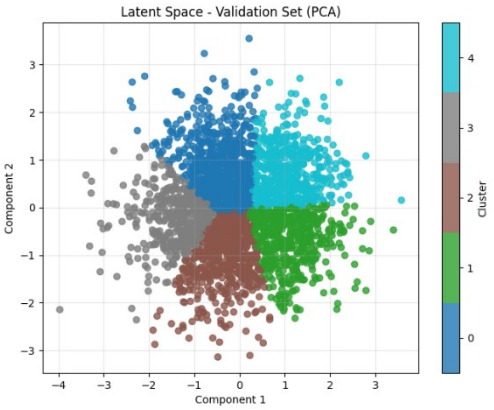


Figure 19. *Example training PCA plot*
Note: Shows the latent space projection during training using PCA. Each point represents a robot sample. Colours denote unsupervised cluster labels obtained through k-means.

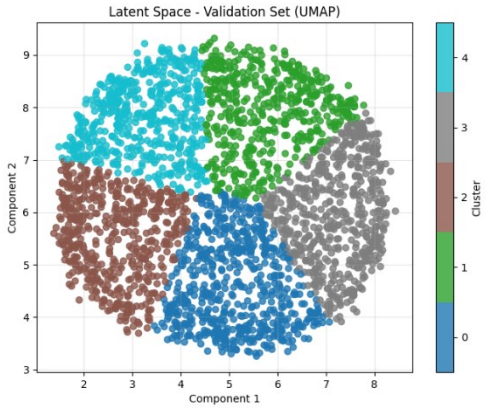


Figure 20. *Example training UMAP plot*
Note: Used in a similar way during training as Figure 19, but projected using UMAP.

To further assess latent space structure, additional metrics were computed and logged in `TrainingHistory`:

- **Silhouette score** – using the PCA and UMAP data with the k-means labels. Calculated using scikit-learn's `silhouette_score()` function, which computes the mean silhouette coefficient using the mean intra-cluster distance and the mean nearest-cluster distance for each sample (scikit-learn developers, 2025c).
- **Euclidean pairwise distance between normalised validation latent vector sample points –**
    - Mean distance.
    - Standard deviation of distances.

An oversight during development led to self-distances being included in pairwise Euclidean distance calculations, slightly diluting mean values. However, this inconsistency was kept throughout training to maintain comparability.

3.7.3 Robot Reconstruction and Comparison

Initial code to visualise the robots was provided by Leni K. Le Goff which visualised a single robot from the CSV files based on its row index. This visualisation script was modified to include a legend and visualise based on robot ID. Further functionality was added to visualise directly from PyTorch tensor grid data, which was extended to plot the original and reconstructed robots in separate subplots. An example is shown in Figure 21.

Whilst stored visualisations and therefore those presented here are two-dimensional (2D), when viewed with matplotlib during program execution, visualisations were three-dimensional (3D) allowing for finer inspection. Because 3D projections are viewed in 2D here, some optical illusions may occur, making background and foreground voxels appear closer together than they actually are.

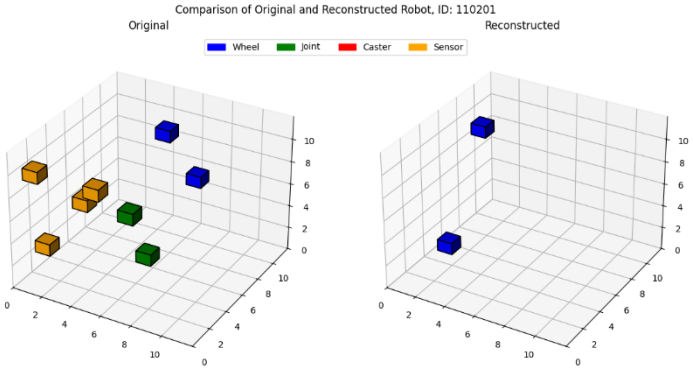


Figure 21. *Example reconstruction comparison visualisation*
Note: The left image shows the original robot, in this example for ID 110201, with colours indicating blue-wheels, green-joints, red-casters, and yellow-sensors. The right image shows the corresponding VAE reconstruction, which in this case produces only a subset of components.

3.8 Iterative Model Refinements

This section outlines the key modifications made to improve reconstruction quality and latent space structure. Whilst various minor adjustments were tested, only the most significant changes are discussed.

3.8.1 Convolutional Model

The initial FC architecture (Section 3.3) struggled to capture spatial relationships, leading to poor reconstructions (Figure 13 and Figure 14) and weak latent space clustering. To address this, the model was restructured using 3D convolutional layers, whilst attempting to maintain the number of trainable layers (six) and parameters ($1{,}629{,}495 \rightarrow 1{,}825{,}285$ assuming latent dimension equals two) similar for comparison. This was expected to capture more spatial information due to convolution being a locally connected network.

Metrics were found to be similar between the FC model and the convolutional model. Reconstructions however, were mostly non-existent, although the latent space clustering was slightly tighter in the convolutional model. This suggested the latent dimensionality was too small and/or the model lacked sufficient capacity. Given the

assumption of convolution performing better than FC layers in addition to the improved latent space, the convolutional model was adopted as the new baseline.

To refine the convolutional model, various layer configurations were tested, including pooling layers, skip connections and batch normalisation.

- Pooling layers improved latent space clustering but caused a loss of spatial information in reconstructions.
- Skip connections, inspired by ResNet, a former state-of-the-art (SOTA) model, were not found to outperform previous experiments (He et al., 2015).
- Batch normalisation was introduced to stabilise training but had limited impact.

In addition, Leaky ReLU activation functions were trialled which can prevent the dying ReLU problem (Mastromichalakis, 2020). Further to these experiments, a deeper network with larger latent space dimensionality was trained in an attempt to improve capacity.

Among all tested models, the best performing model was achieved by the deep model trained on the toy dataset. The model summary for this is shown in Figure 22, whilst Figure 23 to Figure 26 display the performance visualisations. This model included Leaky ReLU activations, but did not include pooling, batch normalisation or skip connection layers and was using the hyperparameters as illustrated in Table 1.

Table 1. *Deep convolution model hyperparameters*

Batch Size	Latent Dimension	Loss function	Optimizer	Learning rate	β
64	8	MSE	Adam	$1e^{-3}$	0.1

```
Model summary:
==========================================================================================================================
Layer (type:depth-idx)                   Input Shape               Output Shape              Param #
==========================================================================================================================
VAE                                      [1, 11, 11, 11]           [1, 11, 11, 11]           --
├─Encoder: 1-1                           [1, 11, 11, 11]           [1, 8]                    --
│    └─Conv3d: 2-1                       [1, 1, 11, 11, 11]        [1, 64, 11, 11, 11]       1,792
│    └─LeakyReLU: 2-2                    [1, 64, 11, 11, 11]       [1, 64, 11, 11, 11]       --
│    └─Conv3d: 2-3                       [1, 64, 11, 11, 11]       [1, 128, 6, 6, 6]         221,312
│    └─LeakyReLU: 2-4                    [1, 128, 6, 6, 6]         [1, 128, 6, 6, 6]         --
│    └─Conv3d: 2-5                       [1, 128, 6, 6, 6]         [1, 256, 6, 6, 6]         884,992
│    └─LeakyReLU: 2-6                    [1, 256, 6, 6, 6]         [1, 256, 6, 6, 6]         --
│    └─Conv3d: 2-7                       [1, 256, 6, 6, 6]         [1, 256, 6, 6, 6]         1,769,728
│    └─LeakyReLU: 2-8                    [1, 256, 6, 6, 6]         [1, 256, 6, 6, 6]         --
│    └─Flatten: 2-9                      [1, 256, 6, 6, 6]         [1, 55296]                --
│    └─Linear: 2-10                      [1, 55296]                [1, 8]                    442,376
│    └─Linear: 2-11                      [1, 55296]                [1, 8]                    442,376
├─Sample: 1-2                            [1, 8]                    [1, 8]                    --
├─Decoder: 1-3                           [1, 8]                    [1, 11, 11, 11]           --
│    └─Linear: 2-12                      [1, 8]                    [1, 55296]                497,664
│    └─Conv3d: 2-13                      [1, 256, 6, 6, 6]         [1, 256, 6, 6, 6]         1,769,728
│    └─LeakyReLU: 2-14                   [1, 256, 6, 6, 6]         [1, 256, 6, 6, 6]         --
│    └─Conv3d: 2-15                      [1, 256, 6, 6, 6]         [1, 128, 6, 6, 6]         884,864
│    └─LeakyReLU: 2-16                   [1, 128, 6, 6, 6]         [1, 128, 6, 6, 6]         --
│    └─ConvTranspose3d: 2-17             [1, 128, 6, 6, 6]         [1, 64, 11, 11, 11]       221,248
│    └─LeakyReLU: 2-18                   [1, 64, 11, 11, 11]       [1, 64, 11, 11, 11]       --
│    └─Conv3d: 2-19                      [1, 64, 11, 11, 11]       [1, 1, 11, 11, 11]        1,729
==========================================================================================================================
Total params: 7,137,809
Trainable params: 7,137,809
Non-trainable params: 0
Total mult-adds (G): 1.50
==========================================================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 3.59
Params size (MB): 28.55
Estimated Total Size (MB): 32.14
==========================================================================================================================
```

Figure 22. *Deep model summary*

Note: Summary of the deep VAE model trialled during development. The model uses 3D convolutions and linear layers to process the full $11x11x11$ voxel grid input, with a latent space dimension of 8. Total parameters: 7.1 million.

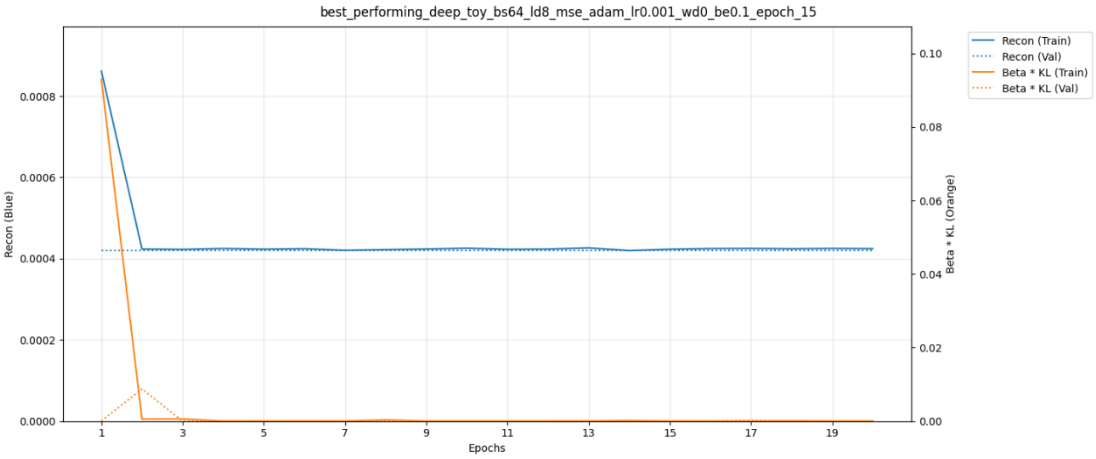


Figure 23. *Deep model toy loss trend*
Note: Training and validation reconstruction loss (blue) and beta scaled KL divergence (orange) for the deep model trained on the toy dataset with $\beta = 0.1$. The model quickly overfits, with minimal KL contribution.

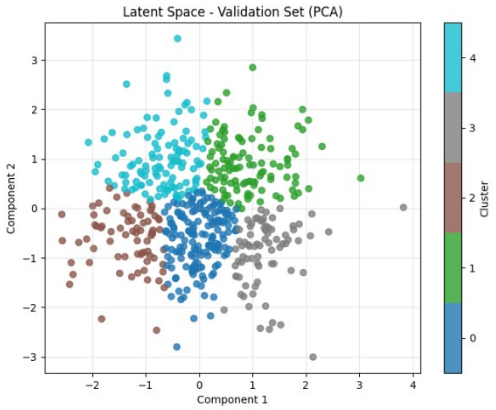


Figure 24. *Deep model toy latent PCA*
Note: PCA projection of the latent space learnt by the deep model. Colours indicate clusters assigned by k-means, showing some loose structure.

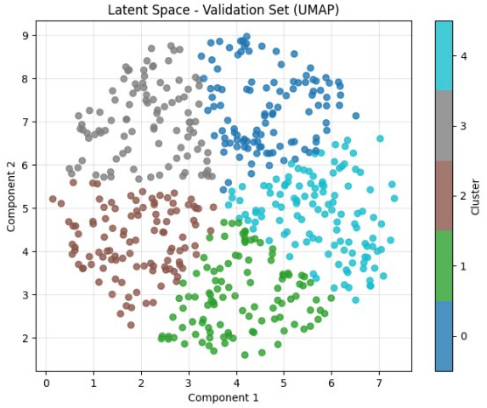


Figure 25. *Deep model toy latent UMAP*
Note: UMAP projection of the latent space learnt by the deep model. Colours indicate clusters assigned by k-means, showing clearer structure and separation than PCA.

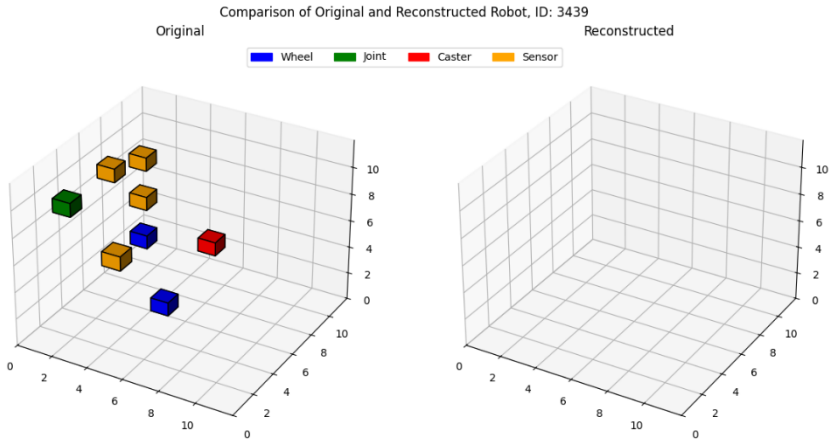


Figure 26. *Deep model toy reconstruction comparison*
Note: The left image shows the original robot for ID 3439. The right image shows the VAE reconstruction, which failed to reproduce any voxels.

Looking at the model's performance visualisations, it showed clear signs of overfitting and continually reconstructed zeros (empty space). A major challenge for this project was how best to tackle the data sparsity. At this stage of development, considerations for alternative data representations were strongly considered.

3.8.2 PointNet Inspired Model

To tackle the overfitting and learning problems, the data was encoded as a sparse representation (Section 3.2.3). Additionally, a model inspired by PointNet was implemented (Section 2.5). These changes required the introduction of custom losses (Section 3.4.3), along with adding a transformed original to the comparison visualisations to observe the performance of T-Net (Section 2.5).

The model adapted the original PointNet design to accommodate sparse voxel data with descriptor values. Unlike the standard PointNet, which uses max pooling to extract global features, this implementation uses 1D adaptive pooling to reduce information loss. Additionally, rather than processing descriptors alongside coordinate values, they were explicitly processed through an independent branch and later recombined with coordinate features. This approach ensured descriptor information actively contributed to feature extraction.

Figure 27 illustrates the architecture at this stage. 1D convolution and pooling layers operate over the sequence length whilst keeping the number of channels the same. This allowed coordinate and descriptor features to be extracted from voxels to learn shared weights after first transposing.

Furthermore, a regularising term was introduced to the reconstruction loss as outlined by Qi et al. (2016):

$$L_{reg} = ||I - AA^T||_F^2$$

where A is the feature alignment matrix and I is the identity matrix. This was to encourage feature transformation via T-net to be orthogonal and prevent information loss in the input. This regularisation term was scaled with `lambda_reg` to control its influence. An orthogonal matrix is one where the product of a matrix and its transpose results in the identity matrix, which has ones on the diagonal, and zeros everywhere else. According to Sydsæter et al. (2010, pp. 151–152), orthogonal matrices have the property of preserving distances and angles.

Whilst the move to PointNet and a sparse data representation did show improvement, there were limited layout variations and numbers of voxels in the reconstructions. Targeted hyperparameter tuning experiments were performed in an attempt to advance the model, including adjusting lambda scaling factors for the transformation matrix and custom losses. Despite this effort, only limited progress was made leading to deeper investigation of the model's implementation. Figure 28 to Figure 32 displays the best performing model's plots at this time (trained on the full dataset). This used the hyperparameters and settings shown in Table 2 and Table 3.

Table 2. *PointNet V1 best performing model hyperparameters*

Batch Size	Latent Dimension	Optimizer	Learning Rate (LR)	β	Coordinate Lambda	Descriptor Lambda	Collapse Lambda	Reg Lambda (Transformation)
64	16	Adam	$1e^{-3}$	0.4	5.0	3.5	0.5	$1e^{-3}$

Table 3. *PointNet V1 best performing model settings*

Epochs Run For / Set	Early Stop Patience	Scheduler Patience (Epochs with no improvement before adjusting LR)	Scheduler Factor (to reduce learning rate by)
63 / 400	20	10	0.2

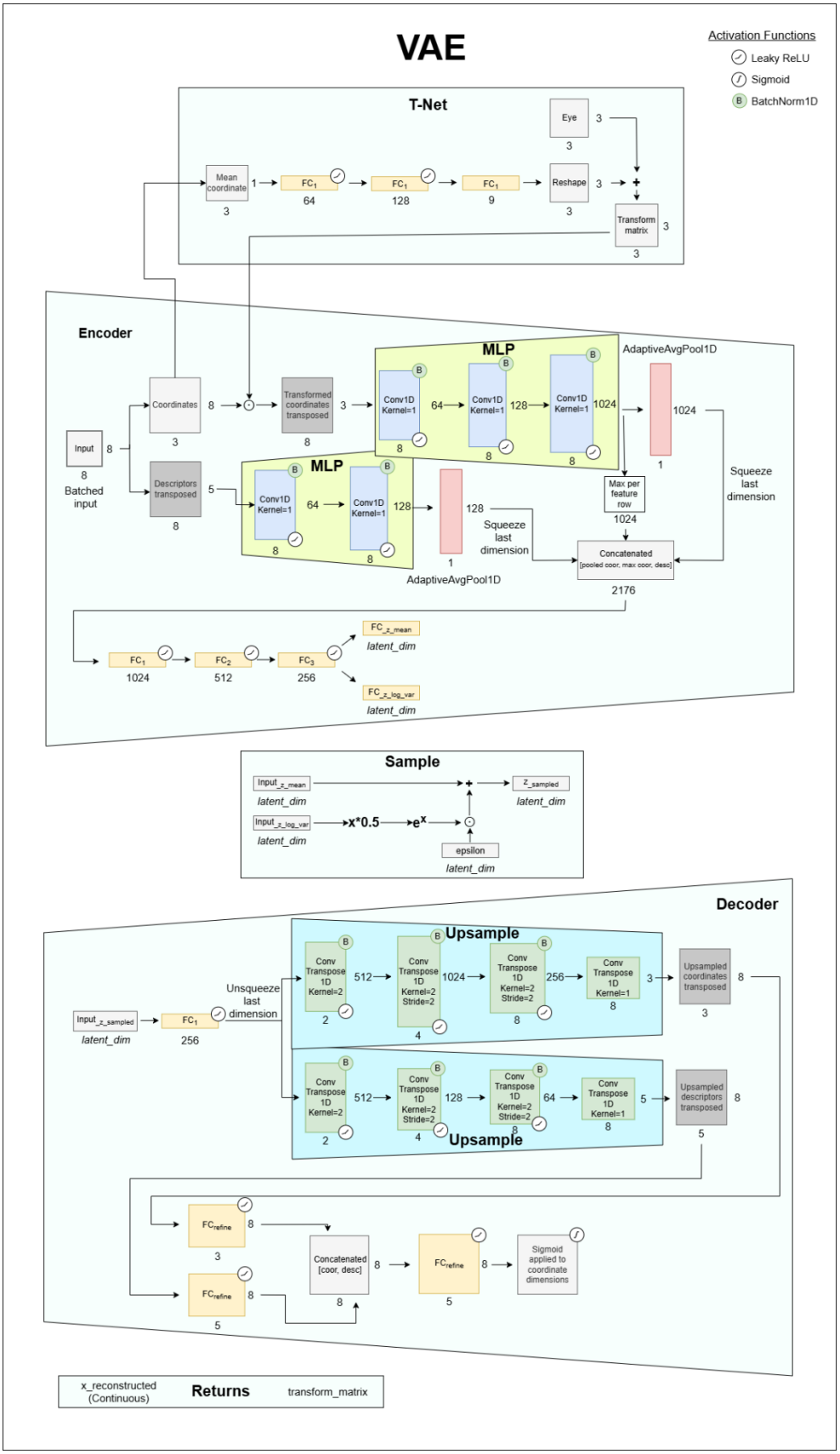


Figure 27. *PointNet V1 inspired architecture*

Note: The architecture includes a T-Net transformer, an encoder with separate coordinate and descriptor branches, reparameterisation sampling, and a decoder featuring upsampling branches. This figure represents a near-final version, minor differences exist from the final implementation (Section 4.1). Created by the author using diagrams.net.

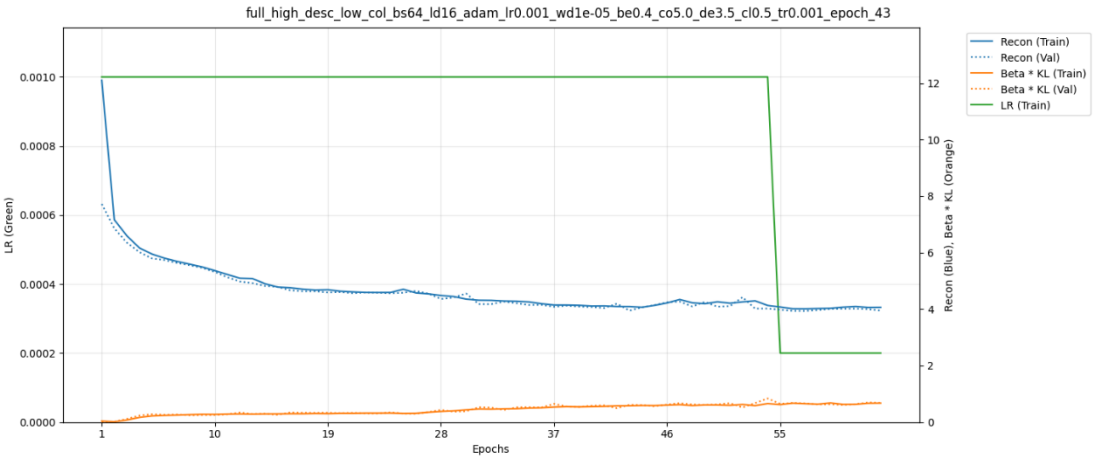


Figure 28. *PointNet V1 best performing model loss trend*
Note: Illustrates reconstruction loss (blue), beta scaled KL divergence (orange) and learning rate (green).

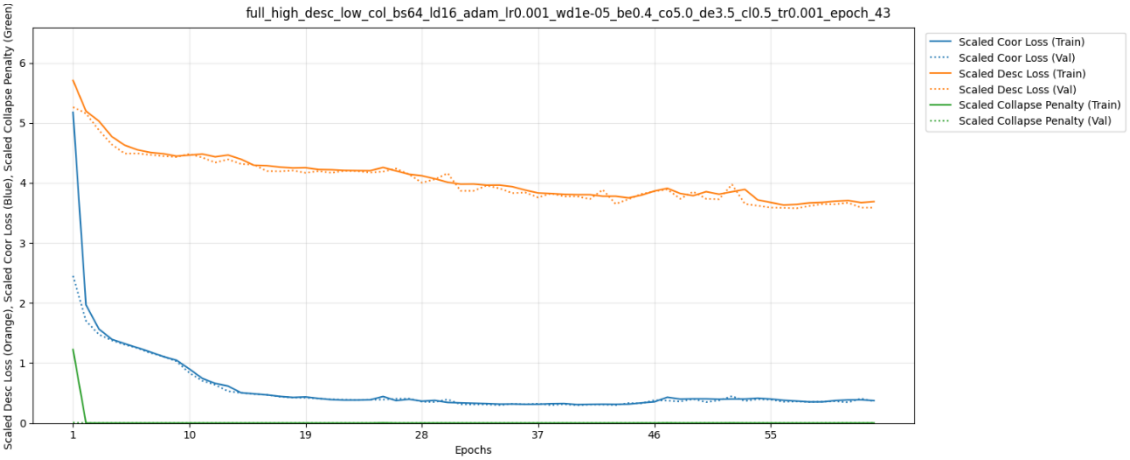


Figure 29. *PointNet V1 best performing model scaled loss component trends*
Note: Illustrates scaled loss components. Coordinate loss is shown in blue, descriptor loss is shown in orange, and collapse penalty is shown in green.

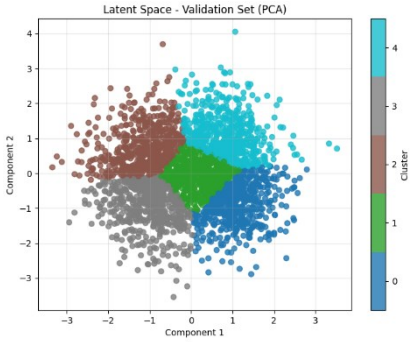


Figure 30. *PointNet V1 best performing model latent PCA*
Note: PCA projection of the latent space learnt by the PointNet V1 model. Colours indicate clusters assigned by k-means, showing clear structure.

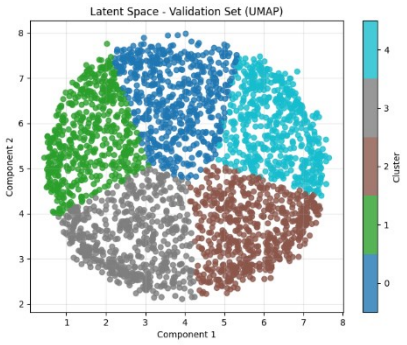


Figure 31. *PointNet V1 best performing model latent UMAP*
Note: UMAP projection of the latent space learnt by the PointNet V1 model. Clearer separation can be seen in comparison to PCA.

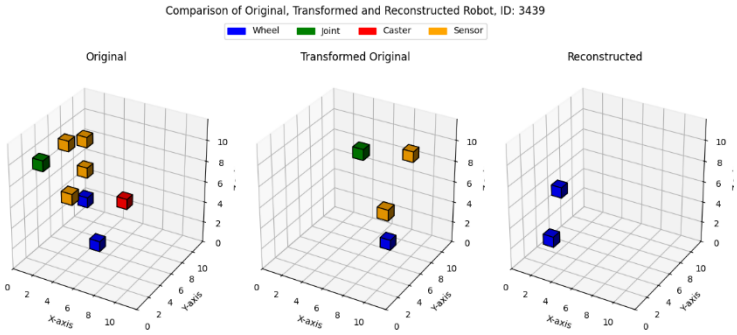


Figure 32. *PointNet V1 best performing model reconstruction*
Note: The left panel displays the original robot sample, whilst the centre shows the original structure after being passed through the learnt transformer (T-Net). The right panel shows the VAE's output, illustrating limited reconstruction quality.

3.8.3 Key Issues and Final Model Adjustments

Several issues were identified during model development that impacted learning and reconstruction quality. The main concern was that concatenating the coordinate and descriptor feature branches in the encoder prevented any interaction between them. This was particularly problematic for descriptors, as their loss function relied on spatial proximity.

To address this, several approaches were explored:

- **Additive Combination** – instead of concatenation, the outputs of the coordinate and descriptor branches were additively combined. To ensure they had the same dimensions for this operation, an additional 1D convolutional layer was introduced to the descriptor multi-layer perceptron (MLP). Whilst this led to slight improvements in reconstruction quality, the impact was limited.

- **Cross-Attention Mechanisms** – initially, coordinate features were used as queries whilst descriptor features functioned as keys and values, allowing spatial features to selectively attend to descriptor information. This resulted in extremely static layouts. The roles were then reversed (descriptors as queries, coordinates as keys/values), leading to a marginal improvement. However, the benefits were negligible, and the model was reverted to the previous best performing configuration with additive combination.

Whilst investigating these modifications, a bug in the custom loss functions was discovered. Padded voxel coordinates had been changed from $(0, 0, 0)$ (a valid position) to $(11, 11, 11)$ ensuring they were correctly included in loss calculations. However, an oversight in the overlap penalty function (Section 3.4.3.3) meant that these values were not properly accounted for in scaling adjustments.

Additionally, it was identified that the entropy-based coordinate matching loss was only enforcing distribution similarity, rather than exact alignment. This created the risk of local minima where voxels aligned in a one-to-one manner but were still spatially incorrect. To mitigate this, a distance-based MSE loss was introduced (Section 3.4.3.1).

Once these core issues were addressed, the best-performing model was selected (presented in the Results Section). This was trained using different beta scaling values to evaluate their effect on latent space structuring and diversity measurement. Since beta controls the balance between reconstruction and regularisation, this comparison was essential for assessing the quality of the learnt representations.

A final issue was discovered late in the process – the LeakyReLU activation was being applied to the final descriptor logits in the decoder. This allowed negative values, and prevented a proper probability distribution for descriptor classification. To correct this, softmax was introduced to ensure valid class probabilities. However, due to time constraints, extensive tuning was not possible. Whilst reconstruction quality remained good, both loss values and silhouette scores worsened compared to the previous model. Therefore, the earlier model was selected for final evaluation.

The full metrics table comparing beta runs before and after this change is provided in Appendix I.

3.9 Evaluation Plan

The goal of the evaluation is to determine whether the model captures meaningful diversity in the latent space. To achieve this, datasets were created based on two grouping criteria: component-based diversity and spatial-based diversity. Originally, the evaluation was planned to use the best robots evolved for different landscapes. However, due to data unavailability, an alternative approach was developed.

The task of creating datasets for diversity measurement analysis required grouping samples into distinct categories based on their component composition and spatial distribution. Specifically, two types of groupings were defined:

- **Component-based groups:**
    1. `component dominance`.
    2. `component_moderate_dominance`.
    3. `component_variety`.
- **Spatial-based groups:**
    1. `spatial_compact`.
    2. `spatial_moderate_spread`.
    3. `spatial_spread_dispersed`.

To ensure the model was evaluated on previously unseen data, the test dataset (Section 3.2.1) was used for categorisation. Each sample was processed in its natural grid representation, iterated over, and assigned to one of three groups for each category.

3.9.1 Component-Based Categorisation

For component-based datasets, the non-zero descriptor values were extracted from each sample. The relative frequency of each component type was then computed as:

$$Relative\ Frequency\ = \frac{f_i}{\sum f}$$

where f_i refers to the descriptor frequency for a specific descriptor type, and $\sum f$ is the total number of descriptors in the sample. This describes the dominance of each component type in the sample.

The assignment of samples to component-based datasets was decided as follows:

- **`component_dominance`** – if *any* component had a relative frequency equal to 100% capturing single type samples.
- **`component_moderate_dominance`** – if *any* component had a relative frequency greater than 55% and less than 65% (not inclusive). This dataset captures samples with majority, but not total dominance.
- **`component_variety`** – If *no* descriptor type had a relative frequency above 38%, ensuring that no single component dominates.

This did mean that some samples did not fall into any dataset and were therefore discarded. The choice to use these thresholds was twofold:

1. The gap between set thresholds helped to produce more distinct and separable datasets.
2. Threshold values were found to produce an acceptable number of samples per dataset, whilst producing datasets with varying descriptor types.

The generated datasets were verified by manual inspection to ensure they were fit for purpose.

3.9.2 Spatial-Based Categorisation

The spatial-based datasets relied on two measures:

- **Bounding box volume.**
- **Mean distance to centroid.**

The calculations for these are detailed below.

3.9.2.1 Bounding Box Volume Calculation

The bounding box volume was calculated as follows:

Let S be the set of coordinate indexes (zero indexed) in a sample,

$$X = \{x_i | (x_i, y_i, z_i) \in S\}$$

$$Y = \{y_i | (x_i, y_i, z_i) \in S\}$$

$$Z = \{z_i | (x_i, y_i, z_i) \in S\}$$

$$V_{box} = (max(X) - min(X) + 1) \cdot ((max(Y) - min(Y) + 1) \cdot ((max(Z) - min(Z) + 1)$$

This means that a sample with a single voxel has a bounding box volume of one, avoiding division by zero when normalised.

3.9.2.2 Mean Distance to Centroid Calculation

The centroid was calculated from the mean (x, y, z) coordinate by taking the mean for each position column. The Euclidean distance to the centroid from each coordinate was then computed using PyTorch's built in `vector_norm()` function and then taking the mean.

3.9.2.3 Normalisation and Spatial Score

Both the bounding box and mean distance were normalised ensuring they contribute equally. The bounding box was normalised by dividing it with the maximum volume of the grid space:

$$Normalised\ bounding\ box = \frac{V_{box}}{max\ grid\ volume}$$

Whilst the mean Euclidean distance was divided by the grid space diagonal:

$$Normalised\ mean\ Euclidean\ centroid\ distance\ = \frac{d_{mean}}{max\ grid\ diagonal\ distance}$$

The `spatial_score` was then calculated as the sum of these two normalised values.

Threshold parameters determined the dataset assignment:

- **`spatial_compact`** – if the score was less than 0.2. Capturing samples that are highly clustered and spatially compact.
- **`spatial_moderately_spread`** – if the score was greater than 0.4 and less than 0.5. Capturing moderately spread-out samples.
- **`spatial_spread_dispersed`** – if the score was greater than 1.0. Capturing highly dispersed samples.

The decision to use these threshold values was based on observation and trial-and-error, ensuring that samples were categorised distinctively.

3.9.3 Visualising Latent Vectors from Grouped Datasets

Once the datasets for evaluation had been collected, they could then be visualised for analysis. This was done using dimensionality reduction to visualise dataset samples as single points, providing information about the variance of a dataset, and how it was captured in the latent space representation.

Each of the trained beta models (Section 3.8.3) could be visualised in this way for comparison. Analysing data from the different beta models allowed for the effects of beta to be assessed, as well evaluating how well the models can capture diversity.

3.9.3.1 Dimensionality Reduction Plots

To construct the dimensionality reduction plots, first the trained and checkpointed model was loaded. Next, the collected data was passed through only the encoder, avoiding sampling using the reparametrisation layer, as this introduces noise with epsilon. The z_{mean} vectors were then collected as these represent the central values of the latent distribution for each dimension.

Each vector was normalised based on the entire dataset category to standardise the samples using scikit-learns `StandardScaler`. PCA and UMAP models were trained and fitted using these normalised vectors to reduce the dimensionality from the 16 latent dimensions to 2 for visualising. Dataset centroids could be computed by taking the mean of the reduced data.

Each of the two grouping categories (component and spatial) were plotted separately for manual evaluation to see if variabilities in the datasets could be observed in the visualisations. These were plotted alongside their centroids, from which Euclidean distances between each could be calculated and displayed in the legend. Furthermore, for the PCA plots, eigenvectors could be plotted from the `components_` attribute which represents the direction of maximum variance in the data for each principal component (Amritesh, 2024; scikit-learn developers, 2025b). These could be scaled by the `explained_variance_` attribute, after first converting it to standard deviation by taking the square root so it was in the original units (Starbuck, 2023).

3.9.4 Evaluation Metrics

Each plot included ellipses fitted to datasets within each category, adapting code by Imelfort (2013) to apply in 2D. This code is based on the Khachiyan's algorithm, which is an approximation algorithm for computing the Minimum Volume Enclosing Ellipsoid (MVEE) (Todd & Yıldırım, 2007).

Following the fitting of an ellipse using the adapted algorithm, a *Shapely* ellipse was created for geometric calculations. Shapely is a Python library for manipulation and

analysis of planar geometric objects which could be used to calculate ellipse areas and overlaps (Gillies, 2025).

Further to the ellipse metrics, centres of mass were calculated for the PCA and UMAP plots. These represent the mean of the reduced vector for each dataset. These centres were selected over the ellipse centres as they should be less sensitive to outliers and provide more information. Distances between these centres were calculated and displayed to provide a quantitative comparison measure between datasets.

Ellipse areas, centroid distances, and overlap percentages were calculated to provide quantitative comparisons across different beta values and dataset groupings. These are presented in Section 4 Results for analysis of the model.

3.9.5 Interpreting Latent Representation

The selection of a beta model for further exploration was based on a combination of reconstruction quality and visual inspection of the latent space structure. The model that provided the clearest separation between groups whilst maintaining morphological variation was selected for in depth analysis, as presented in Section 4.

To gain a general understanding of the latent representation, reconstructions could be overlaid on the PCA and UMAP projections. This provided an insight into the mappings between original robots and their position in the latent space encoding.

PCA maps linearly and is good at capturing global structure, however local relationships may be lost (Cristian et al., 2024; Schmitz et al., 2021). In contrast, UMAP focuses on local structure and preserving nearby points in the global space. Considering this information, a comparison between where specific points are mapped to could provide valuable insight. Hence, the final visualisations involved selecting points from both the PCA and UMAP projections based on if they were in the outer regions or towards the overall centre of mass. These could then be compared to see if both plots agree or disagree.

The purpose of these final visualisations was to explore how individual robots are positioned in the latent space according to different projection techniques, and

whether consistent patterns emerge between PCA and UMAP. These observations do not aim to evaluate model performance directly, but rather to provide tools for interpreting what structural or component-based information may be encoded by the model. These visualisations can be seen in the Results Section.

4 Results

This section presents the final model architecture and evaluation metrics from training the Variational Autoencoder (VAE). This includes model loss trends, reconstruction results, and visualisations from exploring the structure of the latent space. Multiple models were trained using different beta (β) values to control the strength of regularisation in the latent space. These models were evaluated using test data to extract latent representations, which were visualised and analysed using dimensionality reduction. Based on this evaluation, one model was selected for detailed exploration of the latent space structure.

First, the final model architecture is presented, followed by each model's training metrics and loss trends, shown to document model convergence over time. Next, reconstruction outputs are presented to compare the original robot morphologies and their VAE-generated reconstructions for model selection. Finally, the latent representation plots are explored for the chosen beta model.

4.1 Final Model Architecture and Summary

The final model included a LeakyReLU activation in the last layers of both the descriptor and coordinate outputs, with a sigmoid activation applied to coordinates to constrain their values between 0 and 1 (Section 3.8.3). The model architecture was based on a PointNet-inspired design, using multi-layer perceptrons (MLPs) with batch normalisation and LeakyReLU activations (Section 3.8.2). An additional 1D convolutional layer was introduced in the encoder's descriptor branch to allow for additive combination of features. To maintain architectural symmetry, a corresponding FC layer was added to the decoder.

The final model summary, including layer shapes and parameter counts, is shown in Figure 33, whilst Figure 34 presents a visual overview of the architecture. Additionally, the code for the final VAE model can be found in Appendix J.

Model summary:

Layer (type:depth-idx)	Input Shape	Output Shape	Param #
VAE	[1, 8, 8]	[1, 8, 8]	--
├─Encoder: 1-1	[1, 8, 8]	[1, 16]	--
│ └─TNet: 2-1	[1, 8, 3]	[1, 3, 3]	--
│ │ └─Sequential: 3-1	[1, 3]	[1, 9]	9,737
│ └─Sequential: 2-2	[1, 3, 8]	[1, 1024, 8]	--
│ │ └─Conv1d: 3-2	[1, 3, 8]	[1, 64, 8]	256
│ │ └─BatchNorm1d: 3-3	[1, 64, 8]	[1, 64, 8]	128
│ │ └─LeakyReLU: 3-4	[1, 64, 8]	[1, 64, 8]	--
│ │ └─Conv1d: 3-5	[1, 64, 8]	[1, 128, 8]	8,320
│ │ └─BatchNorm1d: 3-6	[1, 128, 8]	[1, 128, 8]	256
│ │ └─LeakyReLU: 3-7	[1, 128, 8]	[1, 128, 8]	--
│ │ └─Conv1d: 3-8	[1, 128, 8]	[1, 1024, 8]	132,096
│ │ └─BatchNorm1d: 3-9	[1, 1024, 8]	[1, 1024, 8]	2,048
│ │ └─LeakyReLU: 3-10	[1, 1024, 8]	[1, 1024, 8]	--
│ └─Sequential: 2-3	[1, 5, 8]	[1, 1024, 8]	--
│ │ └─Conv1d: 3-11	[1, 5, 8]	[1, 64, 8]	384
│ │ └─BatchNorm1d: 3-12	[1, 64, 8]	[1, 64, 8]	128
│ │ └─LeakyReLU: 3-13	[1, 64, 8]	[1, 64, 8]	--
│ │ └─Conv1d: 3-14	[1, 64, 8]	[1, 128, 8]	8,320
│ │ └─BatchNorm1d: 3-15	[1, 128, 8]	[1, 128, 8]	256
│ │ └─LeakyReLU: 3-16	[1, 128, 8]	[1, 128, 8]	--
│ │ └─Conv1d: 3-17	[1, 128, 8]	[1, 1024, 8]	132,096
│ │ └─BatchNorm1d: 3-18	[1, 1024, 8]	[1, 1024, 8]	2,048
│ │ └─LeakyReLU: 3-19	[1, 1024, 8]	[1, 1024, 8]	--
│ └─AdaptiveAvgPool1d: 2-4	[1, 1024, 8]	[1, 1024, 1]	--
│ └─AdaptiveAvgPool1d: 2-5	[1, 1024, 8]	[1, 1024, 1]	--
│ └─LeakyReLU: 2-6	[1, 1024]	[1, 1024]	--
│ └─Sequential: 2-7	[1, 1024]	[1, 128]	--
│ │ └─Linear: 3-20	[1, 1024]	[1, 512]	524,800
│ │ └─LeakyReLU: 3-21	[1, 512]	[1, 512]	--
│ │ └─Linear: 3-22	[1, 512]	[1, 256]	131,328
│ │ └─LeakyReLU: 3-23	[1, 256]	[1, 256]	--
│ │ └─Linear: 3-24	[1, 256]	[1, 128]	32,896
│ │ └─LeakyReLU: 3-25	[1, 128]	[1, 128]	--
│ └─Linear: 2-8	[1, 128]	[1, 16]	2,064
│ └─Linear: 2-9	[1, 128]	[1, 16]	2,064

```
├─Sample: 1-2                            [1, 16]                   [1, 16]                   --
├─Decoder: 1-3                           [1, 16]                   [1, 8, 8]                 --
│    └─Sequential: 2-10                  [1, 16]                   [1, 256]                  --
│    │    └─Linear: 3-26                 [1, 16]                   [1, 128]                  2,176
│    │    └─LeakyReLU: 3-27              [1, 128]                  [1, 128]                  --
│    │    └─Linear: 3-28                 [1, 128]                  [1, 256]                  33,024
│    │    └─LeakyReLU: 3-29              [1, 256]                  [1, 256]                  --
│    └─Sequential: 2-11                  [1, 256, 1]               [1, 3, 8]                 --
│    │    └─ConvTranspose1d: 3-30        [1, 256, 1]               [1, 512, 2]               262,656
│    │    └─BatchNorm1d: 3-31            [1, 512, 2]               [1, 512, 2]               1,024
│    │    └─LeakyReLU: 3-32              [1, 512, 2]               [1, 512, 2]               --
│    │    └─ConvTranspose1d: 3-33        [1, 512, 2]               [1, 1024, 4]              1,049,600
│    │    └─BatchNorm1d: 3-34            [1, 1024, 4]              [1, 1024, 4]              2,048
│    │    └─LeakyReLU: 3-35              [1, 1024, 4]              [1, 1024, 4]              --
│    │    └─ConvTranspose1d: 3-36        [1, 1024, 4]              [1, 256, 8]               524,544
│    │    └─BatchNorm1d: 3-37            [1, 256, 8]               [1, 256, 8]               512
│    │    └─LeakyReLU: 3-38              [1, 256, 8]               [1, 256, 8]               --
│    │    └─ConvTranspose1d: 3-39        [1, 256, 8]               [1, 3, 8]                 771
│    └─Sequential: 2-12                  [1, 256, 1]               [1, 5, 8]                 --
│    │    └─ConvTranspose1d: 3-40        [1, 256, 1]               [1, 512, 2]               262,656
│    │    └─BatchNorm1d: 3-41            [1, 512, 2]               [1, 512, 2]               1,024
│    │    └─LeakyReLU: 3-42              [1, 512, 2]               [1, 512, 2]               --
│    │    └─ConvTranspose1d: 3-43        [1, 512, 2]               [1, 128, 4]               131,200
│    │    └─BatchNorm1d: 3-44            [1, 128, 4]               [1, 128, 4]               256
│    │    └─LeakyReLU: 3-45              [1, 128, 4]               [1, 128, 4]               --
│    │    └─ConvTranspose1d: 3-46        [1, 128, 4]               [1, 64, 8]                16,448
│    │    └─BatchNorm1d: 3-47            [1, 64, 8]                [1, 64, 8]                128
│    │    └─LeakyReLU: 3-48              [1, 64, 8]                [1, 64, 8]                --
│    │    └─ConvTranspose1d: 3-49        [1, 64, 8]                [1, 5, 8]                 325
│    └─Sequential: 2-13                  [1, 8, 3]                 [1, 8, 3]                 --
│    │    └─Linear: 3-50                 [1, 8, 3]                 [1, 8, 3]                 12
│    │    └─LeakyReLU: 3-51              [1, 8, 3]                 [1, 8, 3]                 --
│    └─Sequential: 2-14                  [1, 8, 5]                 [1, 8, 5]                 --
│    │    └─Linear: 3-52                 [1, 8, 5]                 [1, 8, 5]                 30
│    │    └─LeakyReLU: 3-53              [1, 8, 5]                 [1, 8, 5]                 --
│    └─Sequential: 2-15                  [1, 8, 8]                 [1, 8, 8]                 --
│    │    └─Linear: 3-54                 [1, 8, 8]                 [1, 8, 8]                 72
│    │    └─LeakyReLU: 3-55              [1, 8, 8]                 [1, 8, 8]                 --
│    └─Sigmoid: 2-16                     [1, 8, 3]                 [1, 8, 3]                 --
==========================================================================================================================
Total params: 3,277,731
Trainable params: 3,277,731
Non-trainable params: 0
Total mult-adds (M): 13.11
==========================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.47
Params size (MB): 13.11
Estimated Total Size (MB): 13.58
==========================================================================================================================
```

Figure 33. *Final model summary*
Note: The summary details layer types, shapes of data at each stage, and parameter counts for each component. Total parameters: 3.3 million.

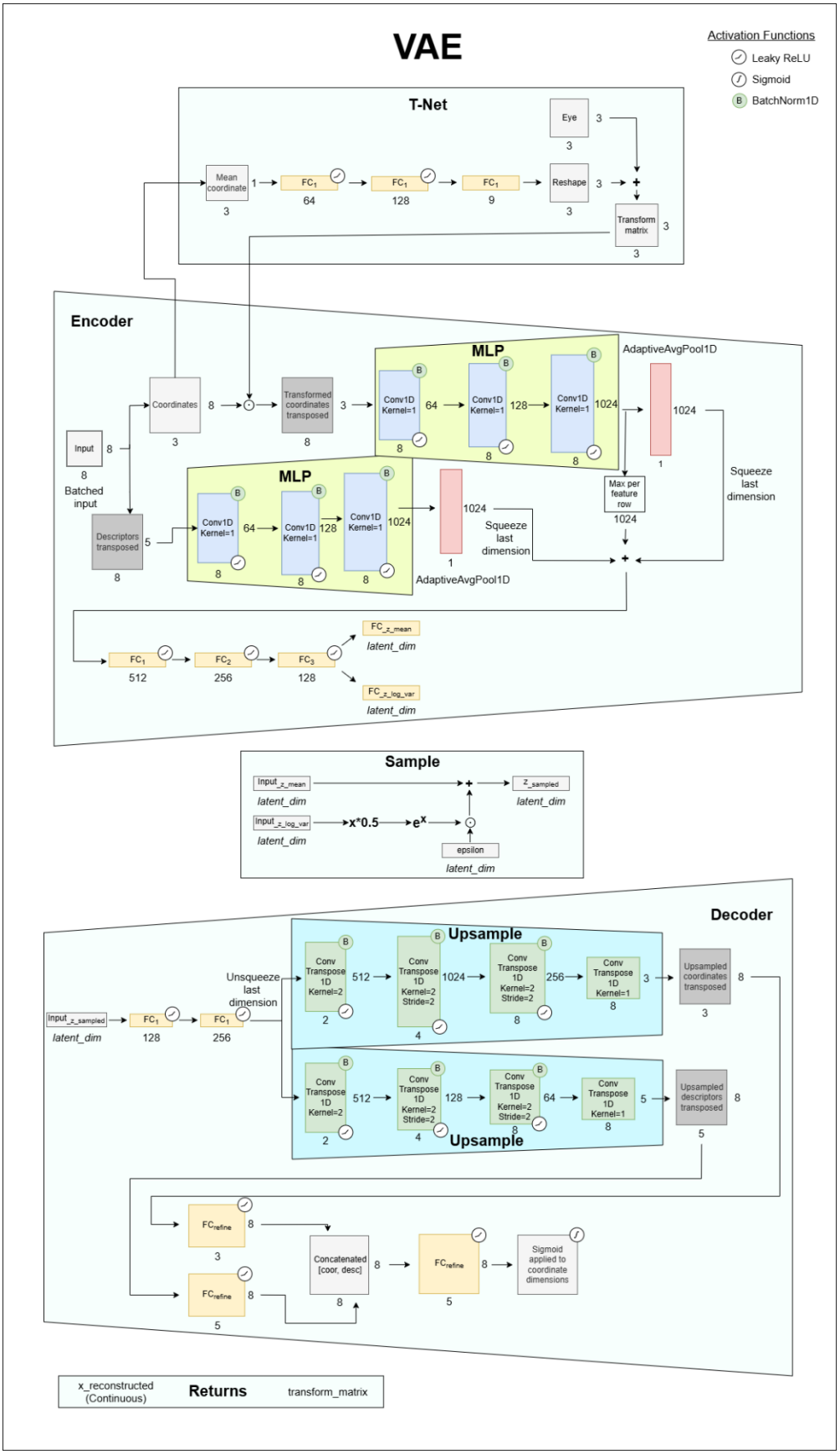


Figure 34. *Final model architecture*

Note: Illustrates the complete architecture of the implemented final PointNet-inspired VAE model, including the T-Net transformer, encoder branches for coordinate and descriptor, reparameterisation sampling, and decoder with separate coordinate and descriptor upsampling branches. Created by the author using diagrams.net.

4.2 Beta Model Comparison and Selection

To explore the effect of latent space regularisation, the model was trained using six different beta values: $\beta = 0.1, 0.3, 0.5, 1.0, 1.5$ and 2.0. For each configuration, the model checkpoint with the lowest validation loss was selected for evaluation. Table 4 to Table 6 list the training hyperparameters, model settings, and the selected checkpoint epochs for each beta configuration.

Table 4. *Evaluation model hyperparameters*

Batch Size	Latent Dimension	Optimizer	Learning Rate (LR)	Coordinate Lambda	Descriptor Lambda	Collapse Lambda	Reg Lambda (Transformation)
64	16	Adam	$1e^{-4}$	2.5	5.0	0.3	$1e^{-3}$

Table 5. *Evaluation model training settings*

Epochs Set	Early Stop Patience	Scheduler Patience (Epochs with no improvement before adjusting LR)	Scheduler Factor (to reduce learning rate by)
500	31	10	0.5

Table 6. *Beta models termination and selection epoch*

β	Epochs Run	Selected Model Epoch
0.1	125	67
0.3	152	121
0.5	225	194
1.0	183	152
1.5	158	127
2.0	186	155

4.2.1 Model Loss Trends

Reconstruction loss component trends across training epochs are shown in Figure 35 to Figure 40. These include coordinate loss, descriptor loss, and collapse penalty values, plotted separately for training and validation sets.

Figure 41 to Figure 46 show total reconstruction loss compared with the beta scaled KL divergence for each model. Together, these figures present a complete overview of model behaviour across different beta values.

Accuracy and F1 score plots have been omitted due to limitations after transitioning to the sparse data representation (Section 3.5). Additionally, an oversight during development resulted in these metrics not being updated correctly, leading to a one-to-one correspondence for descriptors and targets. Therefore, they did not provide a complete representation of model performance. However, for reference, the best accuracy and F1 scores across models ranged between 0.37 and 0.53, with performance trends closely following total loss trends. Due to custom losses being used, the loss components effectively describe both voxel alignment and descriptor accuracy.

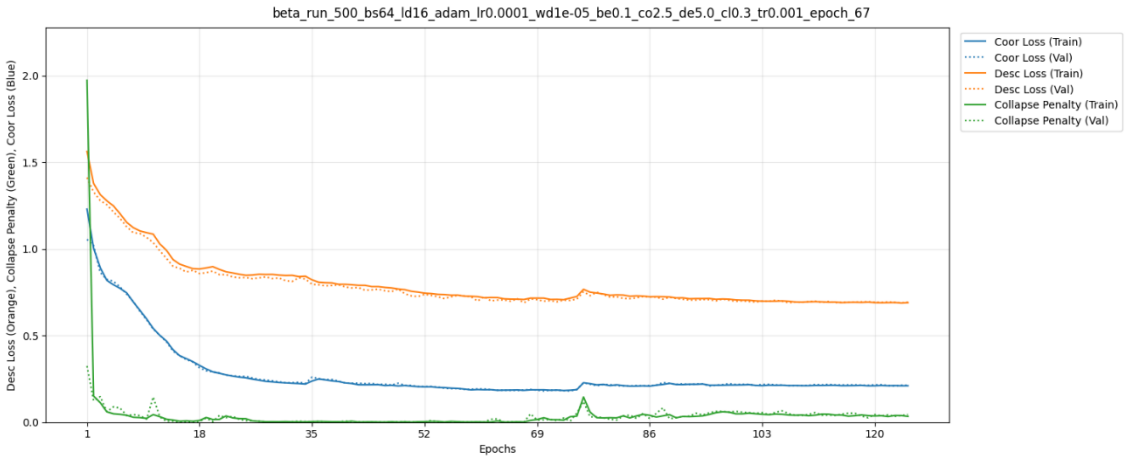


Figure 35. *β=0.1 model raw reconstruction loss component trends*

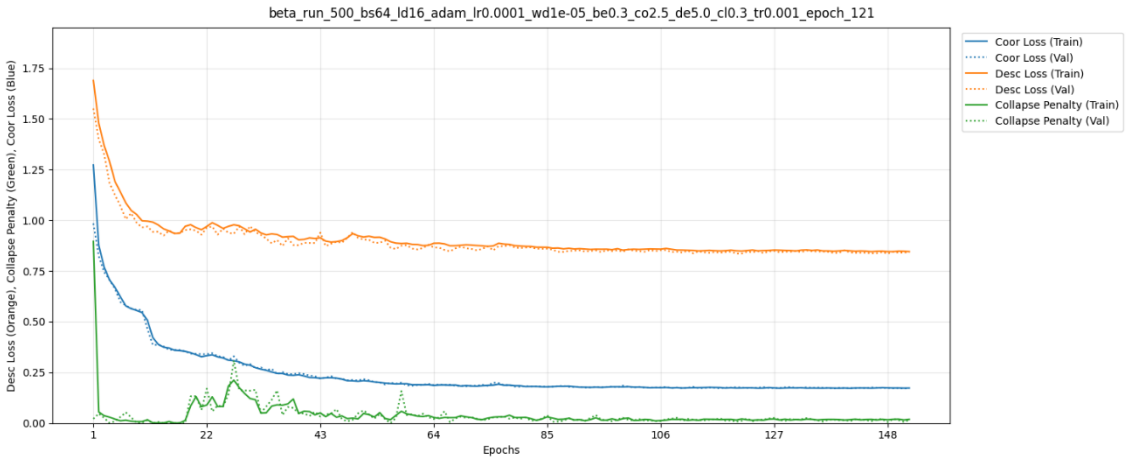


Figure 36. *β=0.3 model raw reconstruction loss component trends*

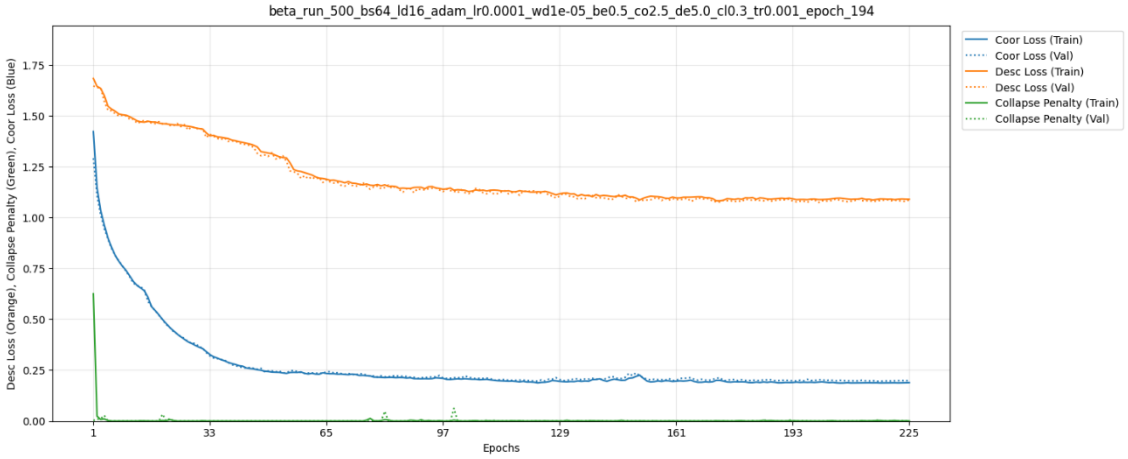


Figure 37. *β=0.5 model raw reconstruction loss component trends*

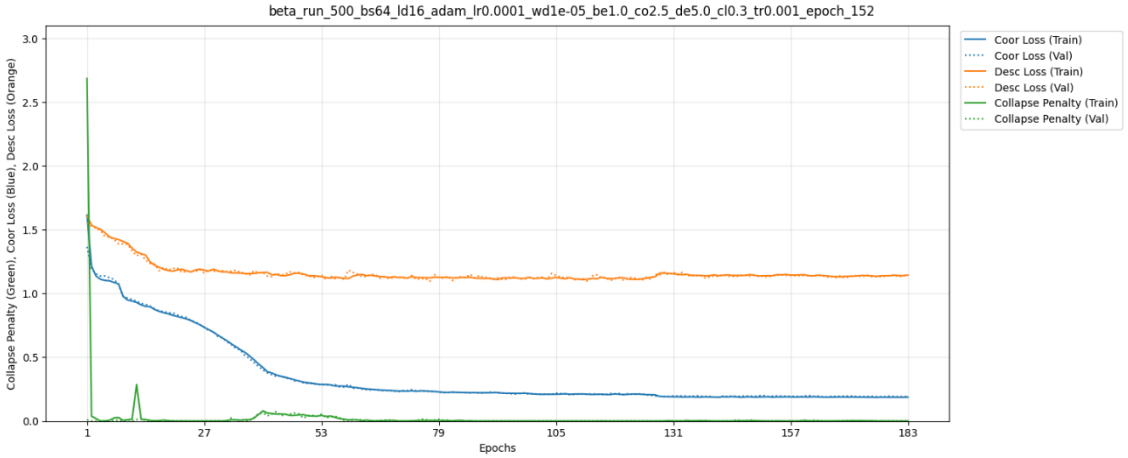


Figure 38. *β=1.0 model raw reconstruction loss component trends*

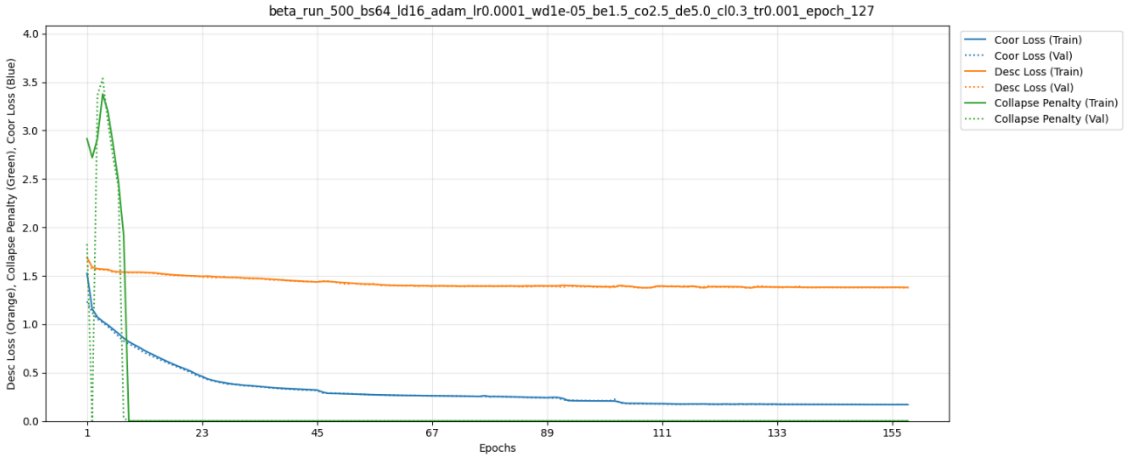


Figure 39. *β=1.5 model raw reconstruction loss component trends*

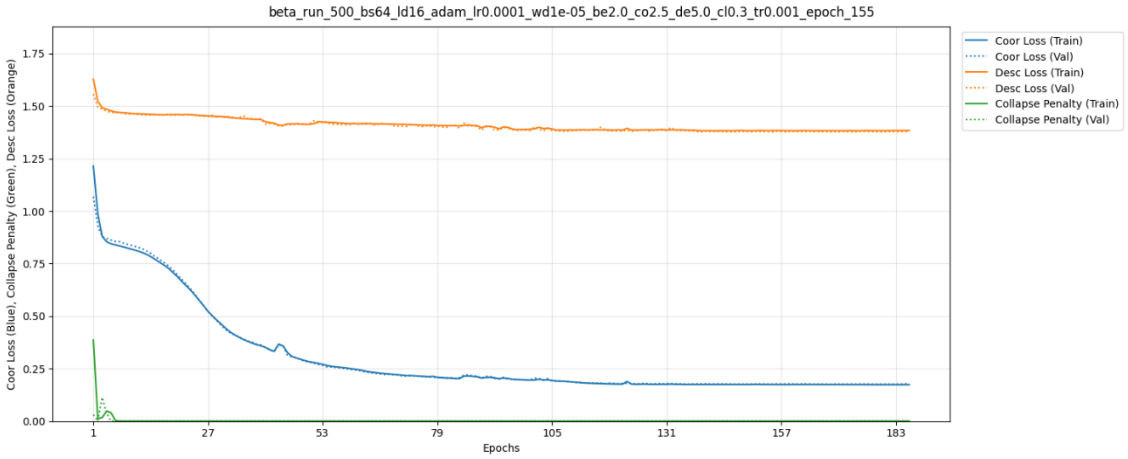


Figure 40. *β=2.0 model raw reconstruction loss component trends*
Note: Collapse penalty (green) drops sharply in early epochs. Coordinate loss (blue) is similar for all models, whilst descriptor loss (orange) generally increases with higher beta values.

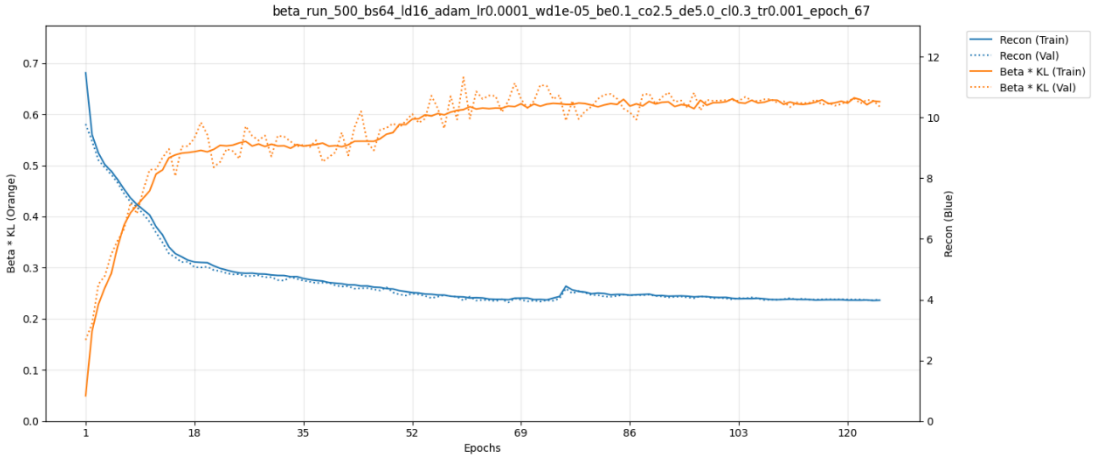


Figure 41. *β=0.1 model scaled reconstruction loss and beta-scaled KL divergence trends*
Note: KL divergence (orange) increases early then plateaus at a lower value compared to the $\beta = 0.3$ model.

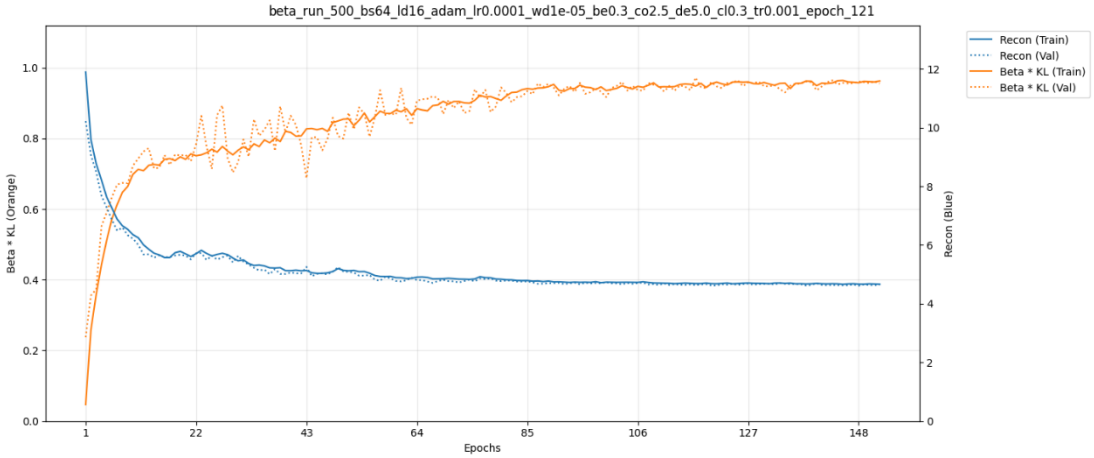


Figure 42. *β=0.3 model scaled reconstruction loss and beta-scaled KL divergence trends*
Note: KL divergence (orange) peaks at a higher value than both *β=0.1 and β=0.5.*

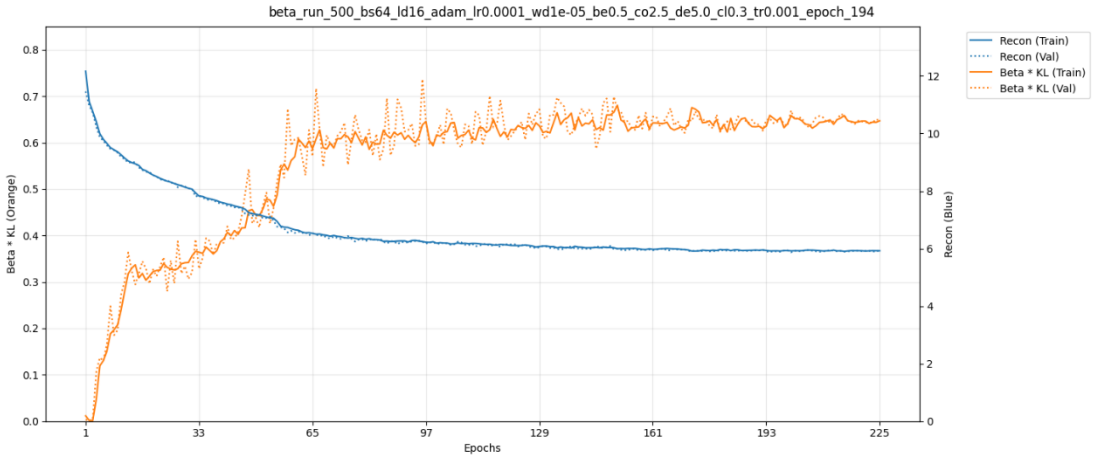


Figure 43. *β=0.5 model scaled reconstruction loss and beta-scaled KL divergence trends*

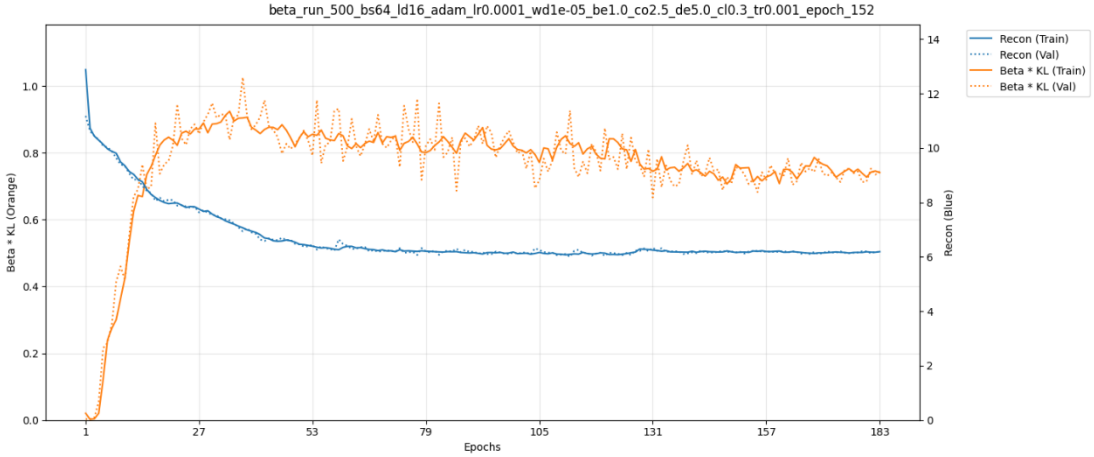


Figure 44. *β=1.0 model scaled reconstruction loss and beta-scaled KL divergence trends*
Note: KL divergence (orange) rises and falls gradually, indicating a transition towards stronger regularisation at $\beta \geq 1.0$.

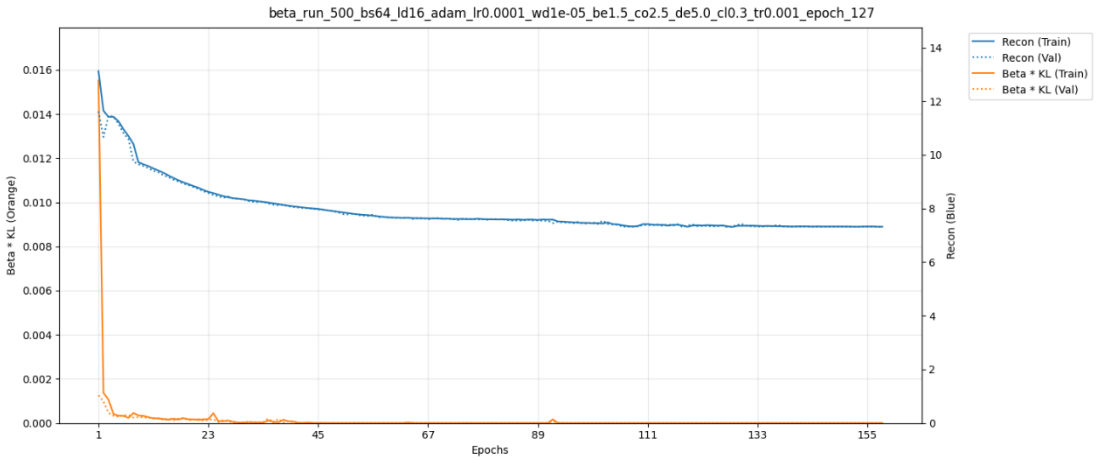


Figure 45. *β=1.5 model scaled reconstruction loss and beta-scaled KL divergence trends*

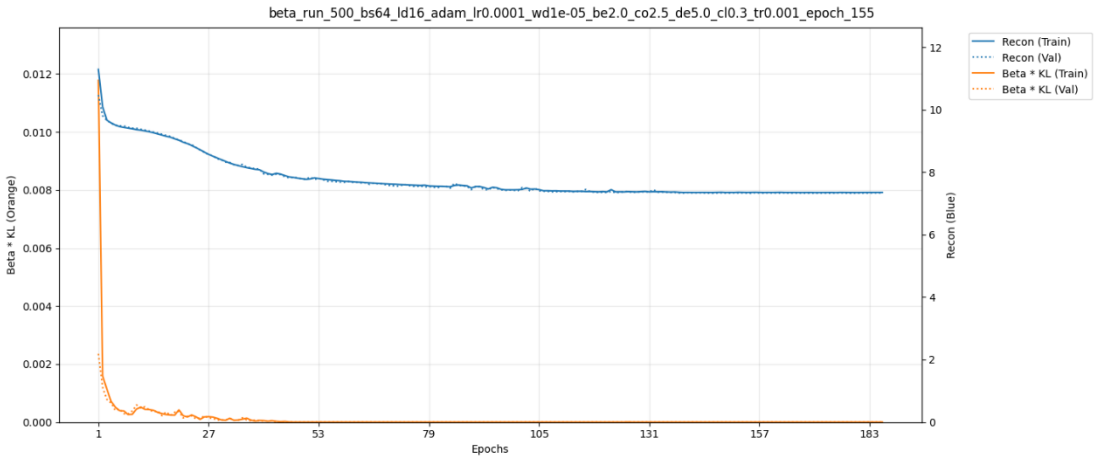


Figure 46. *β=2.0 model scaled reconstruction loss and beta-scaled KL divergence trends*
Note: KL divergence (orange) collapses in high beta value models.

Across all models, coordinate loss converges to similar values, regardless of beta. In contrast, descriptor loss increases with higher beta values, suggesting that descriptor reconstruction becomes more difficult under stronger regularisation. The collapse penalty decreases sharply and remains near zero early in training across all models, indicating that outputs do not collapse into the same spatial region after initial optimisation. Stabilisation across all loss components typically occurs within the first 50 epochs.

KL divergence trends vary by beta. For $\beta < 1.0$, it increases during early epochs and then plateaus. At $\beta = 1.0$, it rises and falls more gradually. For $\beta > 1.0$, KL divergence collapses rapidly and stays near zero, indicating over regularisation.

Among the models with $\beta \leq 1.0$, the $\beta = 0.3$ model offered the best trade-off between latent structure and reconstruction stability. The $\beta = 0.1$ model achieved slightly lower total reconstruction loss, but its KL divergence plateaued at a lower value, indicating weaker regularisation and reduced utilisation of the latent space. Whereas the $\beta = 0.5$ model showed a similar KL divergence to $\beta = 0.1$, but with increased reconstruction loss, suggesting that regularisation began to dominate at the expense of reconstruction quality. In contrast, the $\beta = 0.3$ model achieved the highest KL divergence (~ 1.0) among the three, suggesting more effective latent encoding without compromising loss stability. Descriptor loss for the $\beta = 0.3$ model remained lower than in higher beta variants, and coordinate loss maintained the same convergence trend seen across all models. These trends support $\beta = 0.3$ as

the most balanced configuration for maintaining latent structure whilst preserving reconstruction performance.

4.2.2 Reconstruction Outputs

To evaluate how well each model preserved structure and component descriptors, robot morphologies from each grouped dataset (Section 3.9) were reconstructed using each trained beta model. Three random samples were selected from each group in both the component-based and spatial-based datasets. The original and reconstructed robots are shown in Figure 47 (component-based) and Figure 48 (spatial-based).

The reconstructions show that low to mid-range beta values ($\beta = 0.1, 0.3, 0.5$) preserve greater structural variety and more accurate component information. In contrast, reconstructions from higher beta models ($\beta = 1.5, 2.0$) are identical regardless of input and consist of three wheels, indicating a latent space collapse in the latent representation. These results, along with loss trends and latent space visualisations (Sections 4.2.1 and 4.2.3), informed the selection of the beta model for further inspection (Section 4.3).

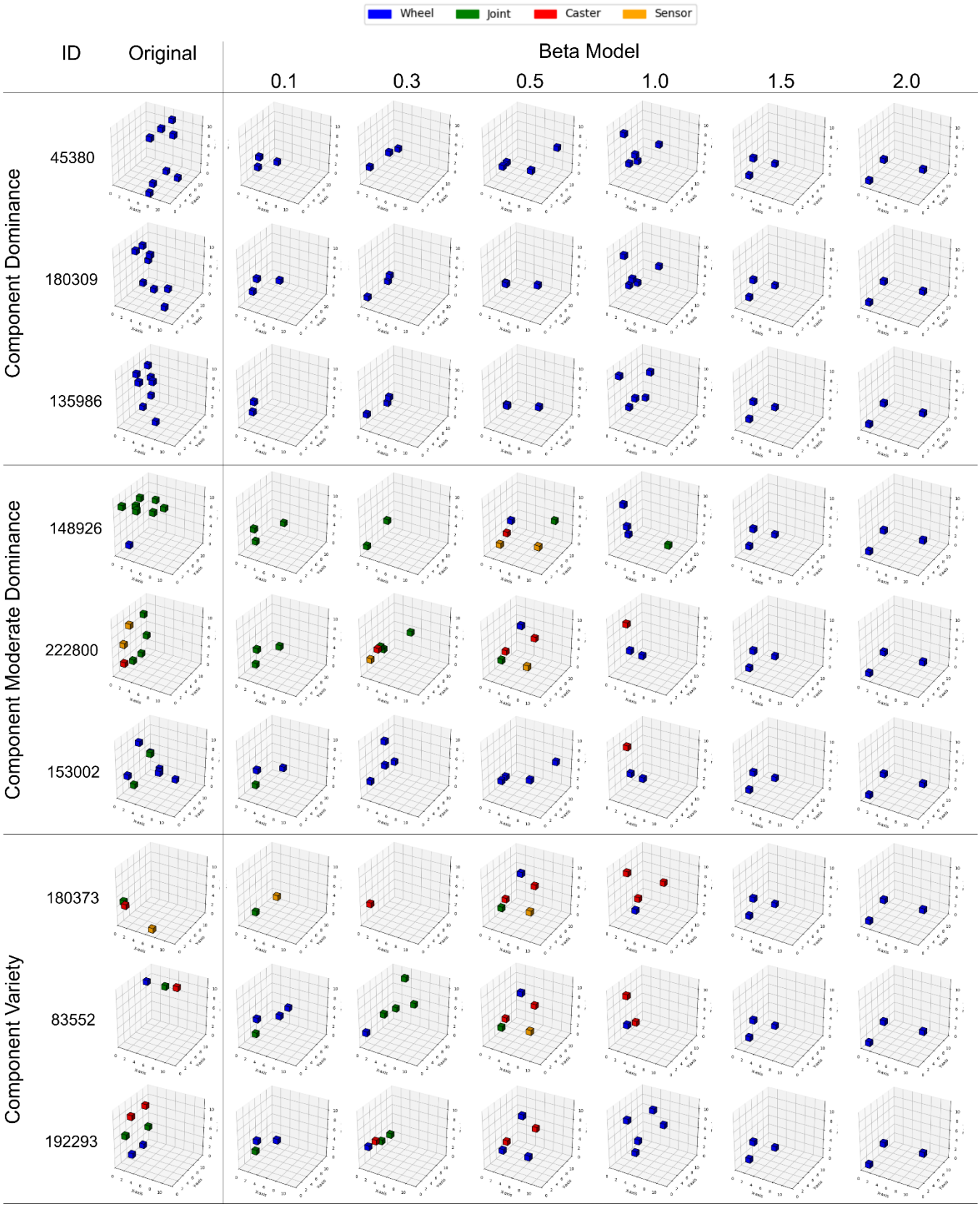


Figure 47. *Component category model reconstruction comparison*
Note: Reconstructions from models with different beta values demonstrate more static designs with increasing beta, highlighting the trade-off between reconstruction quality and latent space regularisation. Whilst not perfect, $\beta = 0.3$ and 0.5 models show the best reconstruction quality. Created by the author using diagrams.net.

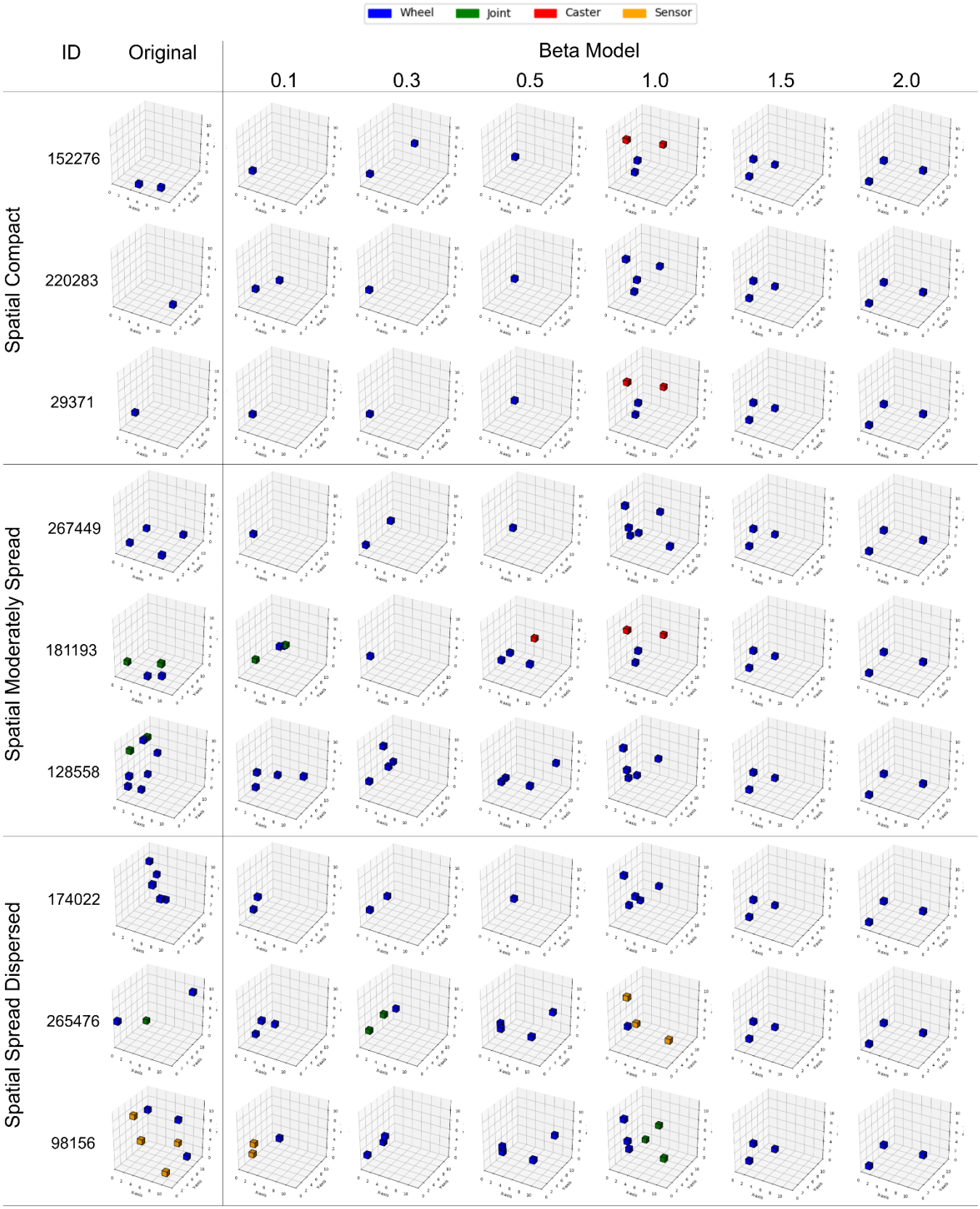


Figure 48. *Spatial category model reconstruction comparison*
Note: Similar to the component reconstructions, static reconstructions can be seen with higher beta values. Signs of static pattens can also be seen in the $\beta = 0.5\ and\ 1.0$ models, suggesting the start of latent space collapse. Created by the author using diagrams.net.

4.2.3 Latent Space Representations

To explore how each beta model organises the data in the latent space, the z_{mean} vectors were extracted from the encoder. This avoided the sampling layer which introduces noise with epsilon (ε). The z_{mean} was used instead of z_{log_var}, as it describes the central tendency of each latent dimension's distribution. These vectors were then processed using PCA and UMAP. The projections were applied to each of the grouped datasets – component-based (defined by descriptor dominance) and spatial-based (defined by bounding box volume and distance to centroid scores) – as described in Section 3.9.3.

Each dataset category was evaluated using an equal number of samples across its three groups. To ensure balance, the minimum group size (187 samples) was used as the maximum sample count for the remaining groups, which were selected randomly.

Figure 49 shows the PCA and UMAP plots for each beta model. Each dataset was visualised with ellipses fitted to their distributions, whilst centres of mass were calculated from the reduced vectors of each group. In the PCA projections, eigenvectors were plotted to show the directions of maximum variance, with their lengths indicating their eigenvalues (converted to standard deviation $\sqrt{\lambda}$ and scaled for visibility), representing explained variance. UMAP was run with default parameters (e.g. `n_neighbours` $= 15$). Although tuning of these parameters was planned, the default configuration was retained due to time constraints.

Quantitative metrics summarising the centroid distances, ellipse areas, and overlap percentages are provided in Table 7 to Table 10. For the highest beta values ($\beta = 1.5$ and 2.0), PCA and UMAP projections could not be computed due to lack of variance in the latent vectors, further indicating over regularisation and latent space collapse.

Both PCA and UMAP projections show distinct clustering patterns across datasets, with varying degrees of separation depending on the beta value. Cluster boundaries typically appear clearer in the UMAP plots. Across all overlap values in Table 7, the greatest separation is observed between the most distinct datasets (dominant vs. variety in the component-based case, and compact vs. dispersed in the spatial-

based case). The least separation consistently occurs between the moderate datasets and the expansive sets (variety, dispersed). Studying the projections reveals that in almost all models, the expansive sets (purple) are almost full subsets of the moderate sets (blue). Whereas the focused sets (green - dominant, compact) typically occupies an area which the others do not.

The evaluation datasets used here were constructed to reflect distinct types of variation. The component-based datasets capture different levels of descriptor dominance but do not consider spatial configuration. Similarly, the spatial-based datasets reflect variation in physical distribution but do not account for descriptor type or variety. These sets were designed to assess whether differences in morphology correspond to separation in the latent space, as visualised through PCA and UMAP. Although the VAE was not explicitly trained to separate these categories, the goal was to observe whether latent structure reflected such differences.

Table 7. *PCA metrics*

	Centroid Distances Component	Centroid Distances Spatial	Ellipse Areas Component	Ellipse Areas Spatial	Overlap Component	Overlap Spatial
Dataset / β	Dom-Mod, Mod-Var, Dom-Var	Com-Mod, Mod-Dis, Com-Dis	Dom, Mod, Var	Com, Mod, Dis	Dom-Mod, Mod-Var, Dom-Var	Com-Mod, Mod-Dis, Com-Dis
0.1	2.01, 0.58, 2.38	2.62, 1.13, 2.9	173.6, 157.2, 81.66	100.61, 87.82, 43.0	74.94%, 92.15%, 75.14%	84.09%, 80.65% 70.32%
0.3	1.78, 0.46, 2.22	1.32, 0.66, 1.97	147.91, 205.24, 47.89	91.38, 176.62, 116.62	87.89%, 100%, 87.59%	99.56%, 100%, 81.30%
0.5	1.97, 1.27, 3.21	2.47, 0.82, 2.61	126.38, 75.4, 16.36	114.34, 64.48, 42.95	58.05%, 81.32%, 74.26%	97.59%, 98.21%, 99.27%
1.0	1.84, 1.48, 3.24	2.1, 1.36, 3.44	51.36, 53.54, 23.3	37.91, 58.83, 36.4	70.73%, 92.94%, 63.05%	60.40%, 99.93%, 26.92%

Table 8. *UMAP metrics*

	Centroid Distances Component	Centroid Distances Spatial	Ellipse Areas Component	Ellipse Areas Spatial	Overlap Component	Overlap Spatial
Dataset / β	Dom-Mod, Mod-Var, Dom-Var	Com-Mod, Mod-Dis, Com-Dis	Dom, Mod, Var	Com, Mod, Dis	Dom-Mod, Mod-Var, Dom-Var	Com-Mod, Mod-Dis, Com-Dis
0.1	4.25, 3.53, 7.56	3.19, 2.7, 5.82	88.81, 115.5, 50.33	60.04, 108.56, 60.78	89.72%, 87.02%, 64.69%	86.70%, 97.40%, 43.97%
0.3	4.61, 1.63, 6.06	4.83, 2.78, 7.41	64.44, 101.0, 60.51	103.31, 75.04, 55.75	93.18%, 91.59%, 57.75%	90.15%, 82.88%, 85.45%
0.5	1.47, 3.63, 5.05	4.73, 2.39, 5.66	288.73, 229.75 98.47	204.52, 171.11, 157.92	97.05%, 94.23%, 90.73%	92.66%, 93.07%, 86.47%
1.0	4.45, 3.43, 7.82	3.58, 3.92, 7.48	76.1, 152.67, 75.46	60.08, 121.97, 91.09	71.29%, 92.81%, 23.38%	40.45%, 86.11%, 25.10%

Table 9. *PCA summary statistics*

β	Mean Euclid Comp	Mean Euclid Spatial	Mean Area Comp	Mean Area Spatial	Mean Overlap Comp	Mean Overlap Spatial	PC1 Eigen $\sqrt{\lambda}$ Comp	PC1 Eigen $\sqrt{\lambda}$ Spatial	PC2 Eigen $\sqrt{\lambda}$ Comp	PC2 Eigen $\sqrt{\lambda}$ Spatial
0.1	1.66	2.22	137.49	77.14	80.74%	78.35%	2.5293	2.7826	1.5615	1.5911
0.3	1.49	1.32	133.68	128.21	91.83%	93.62%	2.5711	2.4277	1.5177	1.5803
0.5	2.15	1.97	72.71	73.92	71.21%	98.36%	2.2038	2.3434	1.5592	1.4390
1.0	2.19	2.30	42.73	44.38	75.57%	62.42%	2.2106	2.0862	1.5870	1.5398

Table 10. *UMAP summary statistics*

β	Mean Euclid Comp	Mean Euclid Spatial	Mean Area Comp	Mean Area Spatial	Mean Overlap Comp	Mean Overlap Spatial
0.1	5.11	3.90	84.88	76.46	80.48%	76.02%
0.3	4.10	5.01	75.32	78.03	80.84%	86.16%
0.5	3.38	4.26	205.65	177.85	94.00%	90.73%
1.0	5.23	4.99	101.41	91.05	62.49%	50.55%

Note: Overlap percentages reflect how much the smaller ellipse overlaps with the larger one. A value of 100% indicates that the smaller ellipse lies entirely within the larger ellipse.

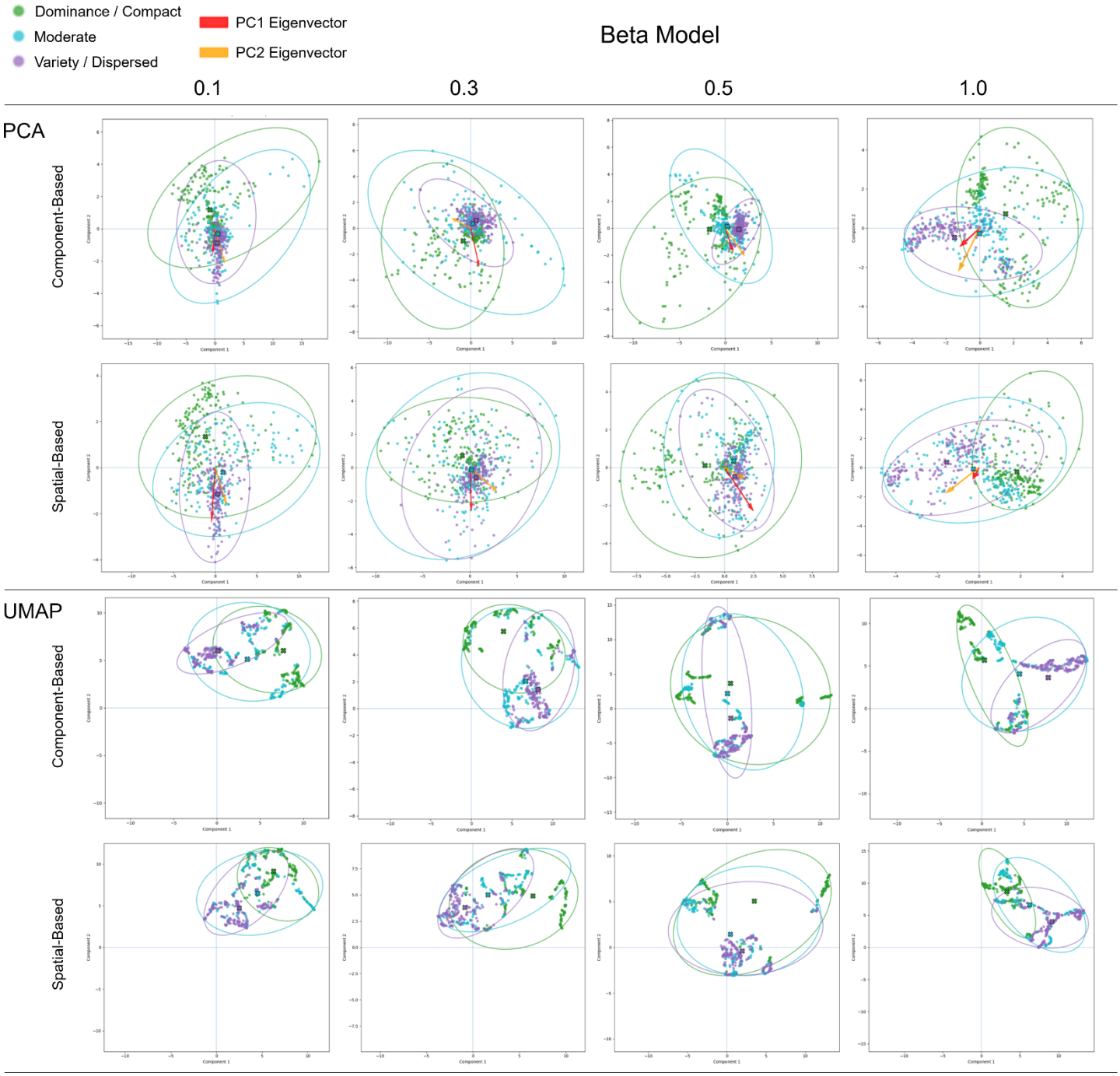


Figure 49. *PCA and UMAP comparisons*

Note: Each beta model is evaluated using PCA and UMAP for both component-based and spatial-based datasets. Ellipses represent dataset spread, with centres of mass shown as crosses. PCA plots include PC1 and PC2 eigenvectors to indicate direction and magnitude. Created by the author using diagrams.net

.

4.3 Exploration of the $\beta = 0.3$ Model's Latent Space

To explore what structural or component-based information the latent space may be encoding, further analysis was carried out using the $\beta = 0.3$ model. This model was selected based on its balanced performance in terms of latent structure and reconstructions as seen in Figure 47 to Figure 49. The goal of this section is to explore how robot morphologies are distributed in the latent space and whether meaningful patterns can be identified through dimensionality reduction and sample comparison. The full PCA and UMAP plots for the model and its closest contender ($\beta = 0.5$) can be found in Appendix K.

Figure 50 to Figure 53 show original robot examples mapped into each projection, with samples selected to reflect both clustered regions and bordering points. Figure 50 and Figure 51 present overlays for the component-based datasets, whilst Figure 52 and Figure 53 present overlays for the spatial-based sets

To further explore the consistency between PCA and UMAP, a subset of robots was selected and their distances to the overall projection centroid were computed. Each robot was then classified as either *central* or *outer region* based on whether its distance was above or below half the maximum distance observed across the full projection dataset. These classifications, along with the robot IDs and their distances, are shown in Figure 54. These were then compared across the two projections to assess agreement. Samples where both PCA and UMAP placed a robot in a similar region (central or outer edges) are marked in blue, whilst samples where the projections disagree are marked in red.

Figure 55 presents this comparison for the component-based datasets where the overall centre of mass is shown as a grey cross, and the corresponding original robot morphologies are illustrated in Figure 56. A similar comparison is shown for spatial-based datasets in Figure 57 and Figure 58.

Together, these visualisations provide a basis for interpreting how the VAE organises morphologies within the latent space, and for identifying whether meaningful relationships have been captured. Interpretation of these findings is presented in Section 5 Discussion and Evaluation.

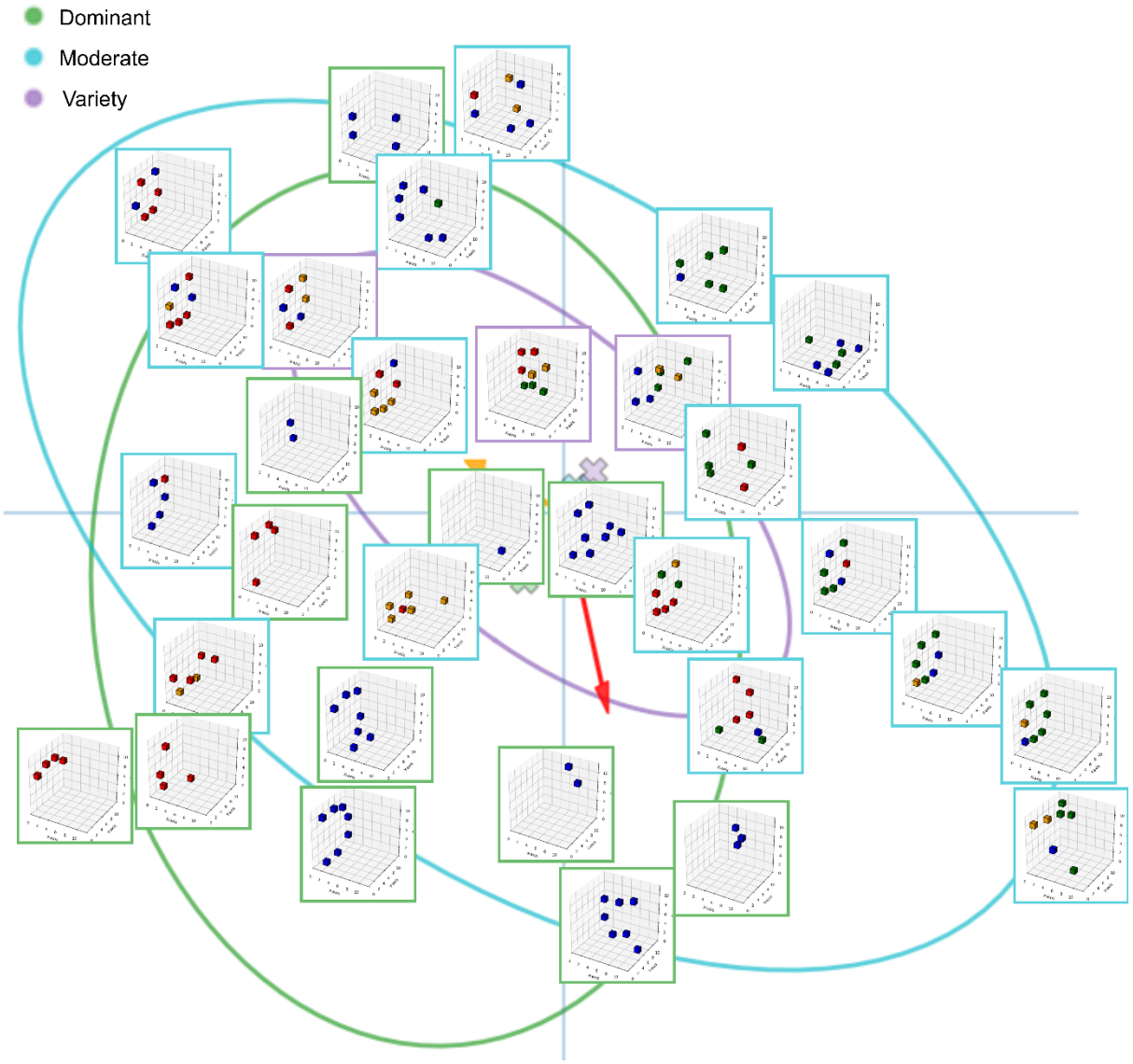


Figure 50. *β=0.3 model component PCA, original robots overlayed*
Note: A similar structural pattern can be observed along the PC2 maximum variance diagonal (yellow arrow - lower right to upper left) consisting of vertically stacked, circular pattens with some descriptor variation. Created by the author using diagrams.net.

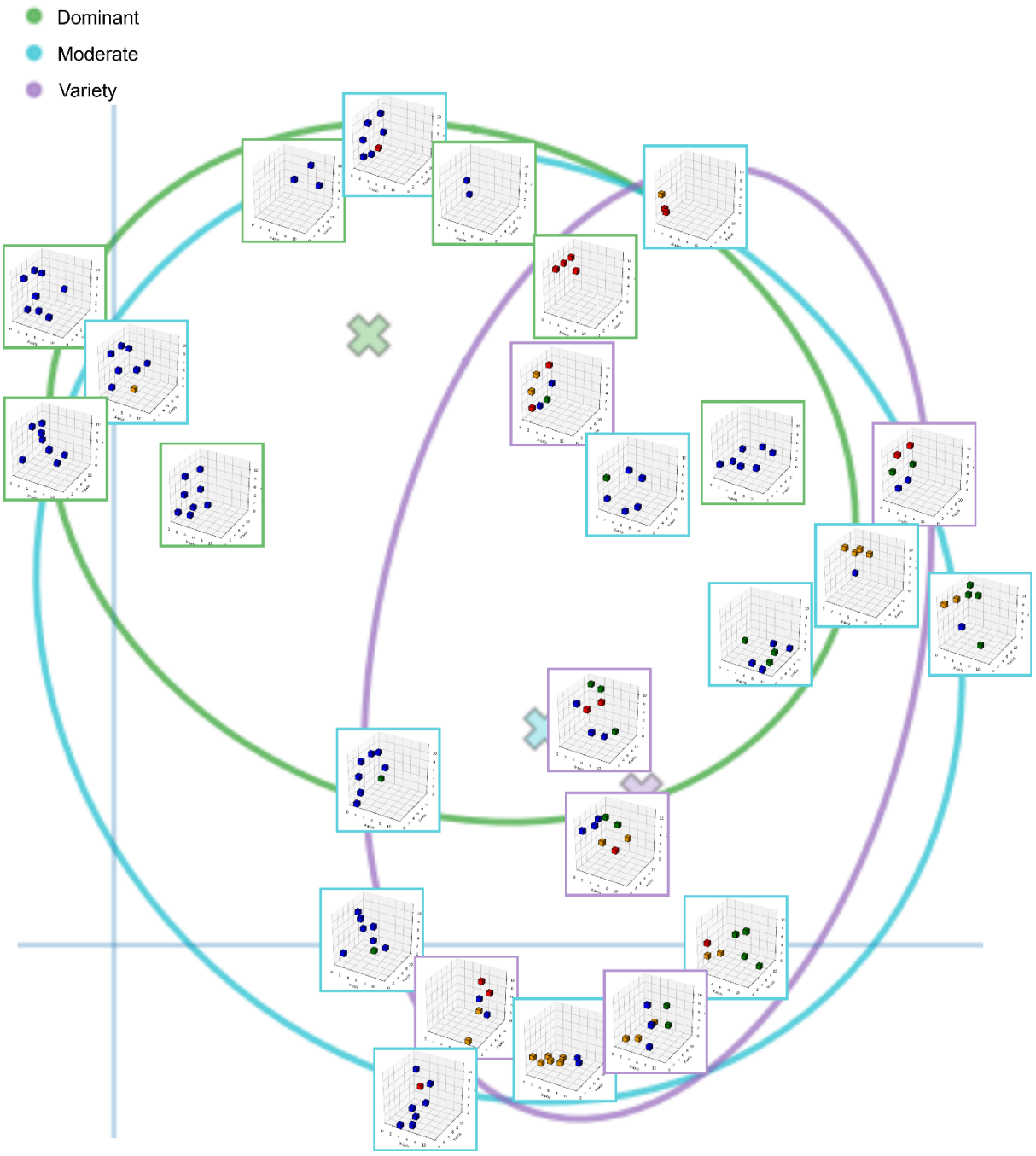


Figure 51. *β=0.3 model component UMAP, original robots overlayed*
Note: Structural similarity appears to cluster together, although these may be in different orientations. To the left appears to be more wheels dominant (blue), whereas more variety appears to the right. Created by the author using diagrams.net.

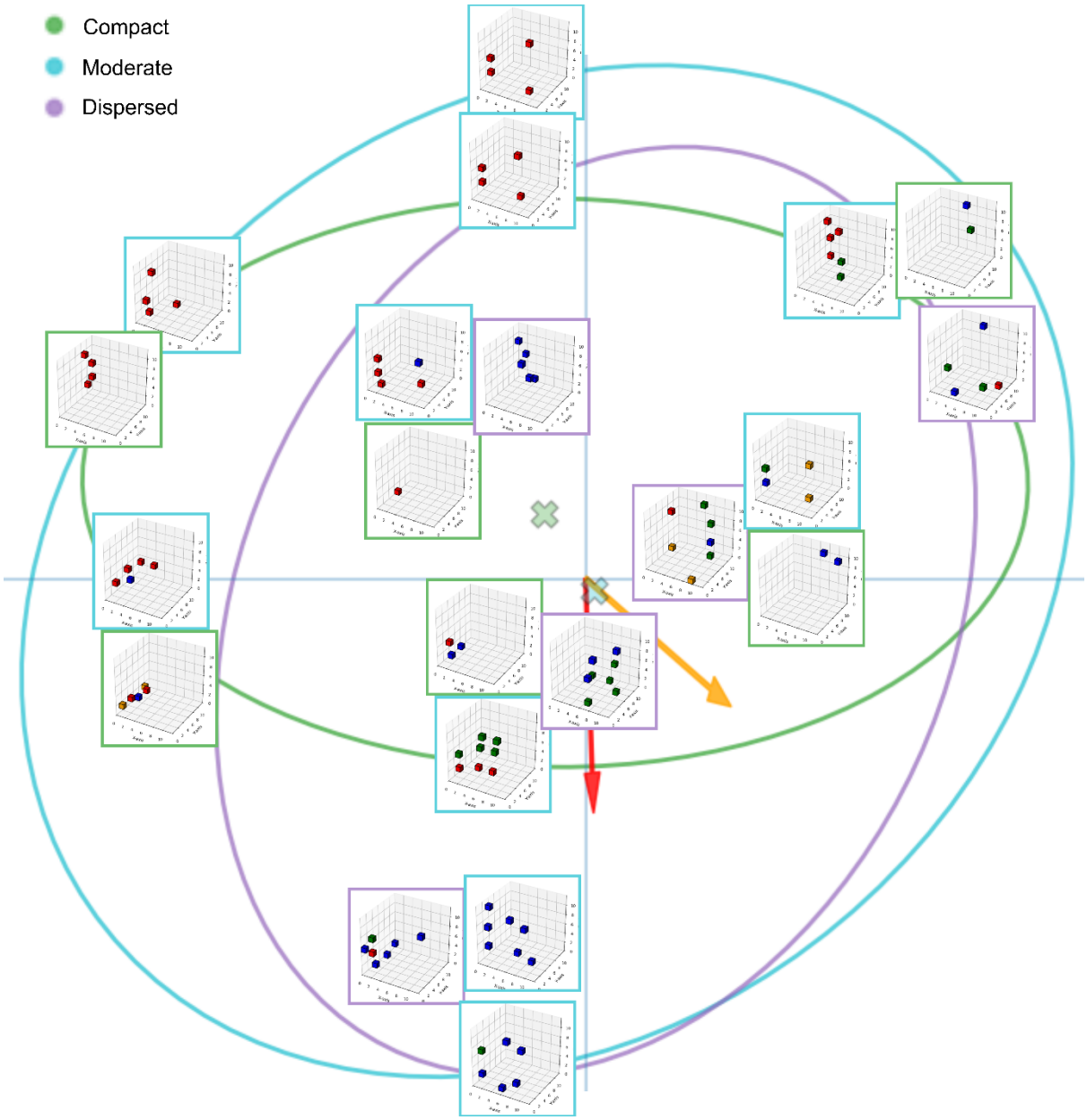


Figure 52. *β=0.3 model spatial PCA, original robots overlayed*
Note: More caster components (red) and more clustered designs appear in the upper left corner, whereas more component variety and dispersed designs appear towards the lower right corner. Created by the author using diagrams.net.

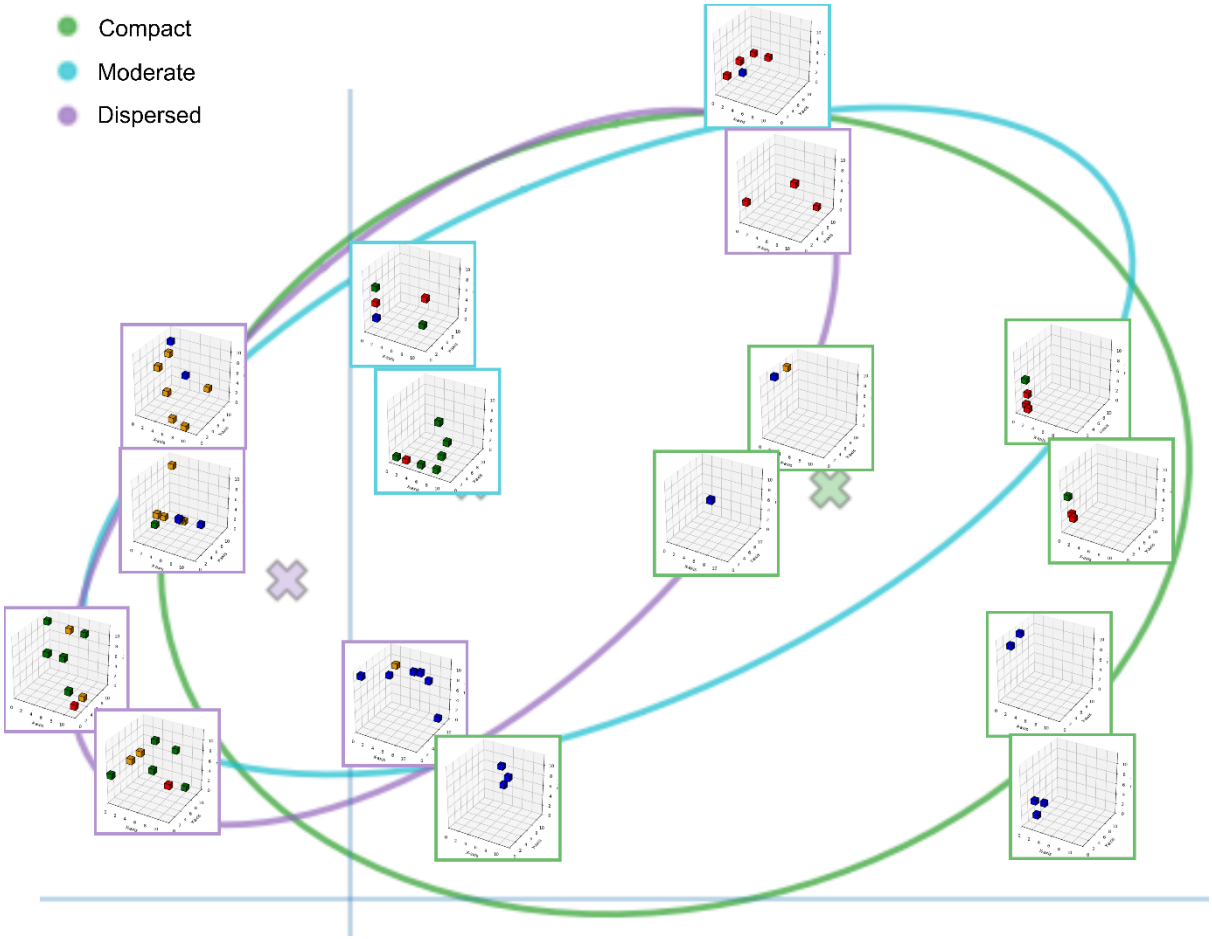


Figure 53. *β=0.3 model spatial UMAP, original robots overlayed*
Note: Simpler and more clustered designs appear on the right, whereas more dispersed designs and greater voxel numbers appear on the left. Created by the author using diagrams.net.

```
PCA Projection - Component Based: Beta 0.3
Distances to Overall Centre:
81445: Central, 2.9625494480133057
210406: Outer region, 10.81870174407959
176868: Central, 1.0230637788772583
213295: Central, 1.362777829170227
30678: Central, 4.5593695640563965
195043: Outer region, 6.399326801300049
230604: Outer region, 6.287214756011963
242949: Central, 1.386569619178772
123338: Outer region, 12.048810005187988
151167: Outer region, 7.527970790863037
152089: Central, 0.3625156879425049
30383: Central, 0.9312426447868347
172427: Central, 0.42008325457572937
81347: Central, 0.6924715042114258
123954: Central, 2.962503433227539
233356: Central, 2.528770923614502

Maximum distance for all data points
(not just those selected): 12.048810005187988
```

```
UMAP Projection - Component Based: Beta 0.3
Distances to Overall Centre:
81445: Outer region, 4.931463718414307
210406: Outer region, 3.9761288166046143
176868: Outer region, 7.602010726928711
213295: Outer region, 4.829528331756592
30678: Central, 3.270777702331543
195043: Outer region, 5.214235305786133
230604: Central, 3.708897590637207
242949: Central, 1.1434065103530884
123338: Outer region, 6.927539348602295
151167: Central, 2.305649518966675
152089: Central, 3.691009759902954
30383: Central, 2.495617151260376
172427: Outer region, 3.9535348415374756
81347: Central, 2.4434316158294678
123954: Outer region, 4.516946792602539
233356: Outer region, 6.198480129241943

Maximum distance for all data points
(not just those selected): 7.602010726928711
```

```
PCA Projection - Spatial Based: Beta 0.3
Distances to Overall Centre:
46710: Central, 1.0565909147262573
235270: Central, 2.5195276737213135
83192: Central, 0.7428305149078369
193546: Outer region, 9.509340286254883
214744: Outer region, 9.418404579162598
222164: Central, 0.2404121607542038
125196: Central, 2.7684998512268066
18577: Outer region, 5.471858501434326
224389: Outer region, 9.604000091552734
270583: Central, 2.547292470932007
131331: Central, 0.9338268041610718
86971: Outer region, 5.416409015655518
244764: Central, 1.4133020639419556
100293: Central, 2.071793794631958

Maximum distance for all data points
(not just those selected): 9.604000091552734
```

```
UMAP Projection - Spatial Based: Beta 0.3
Distances to Overall Centre:
46710: Central, 3.0543932914733887
235270: Central, 3.2702698707580566
83192: Outer region, 7.707158088684082
193546: Outer region, 5.865175724029541
214744: Outer region, 5.78991174697876
222164: Outer region, 7.3605146408081055
125196: Central, 1.081725001335144
18577: Outer region, 4.69426965713501
224389: Central, 2.5260169506073
270583: Outer region, 6.377876281738281
131331: Outer region, 4.275487422943115
86971: Central, 0.48658671975135803
244764: Central, 3.1299474239349365
100293: Outer region, 4.806952476501465

Maximum distance for all data points
(not just those selected): 7.740355014801025
```

Figure 54. *Robot distances to overall centroid*
Note: These classifications are intended only a guide. Classification can be misleading when samples are close to the decision boundary.

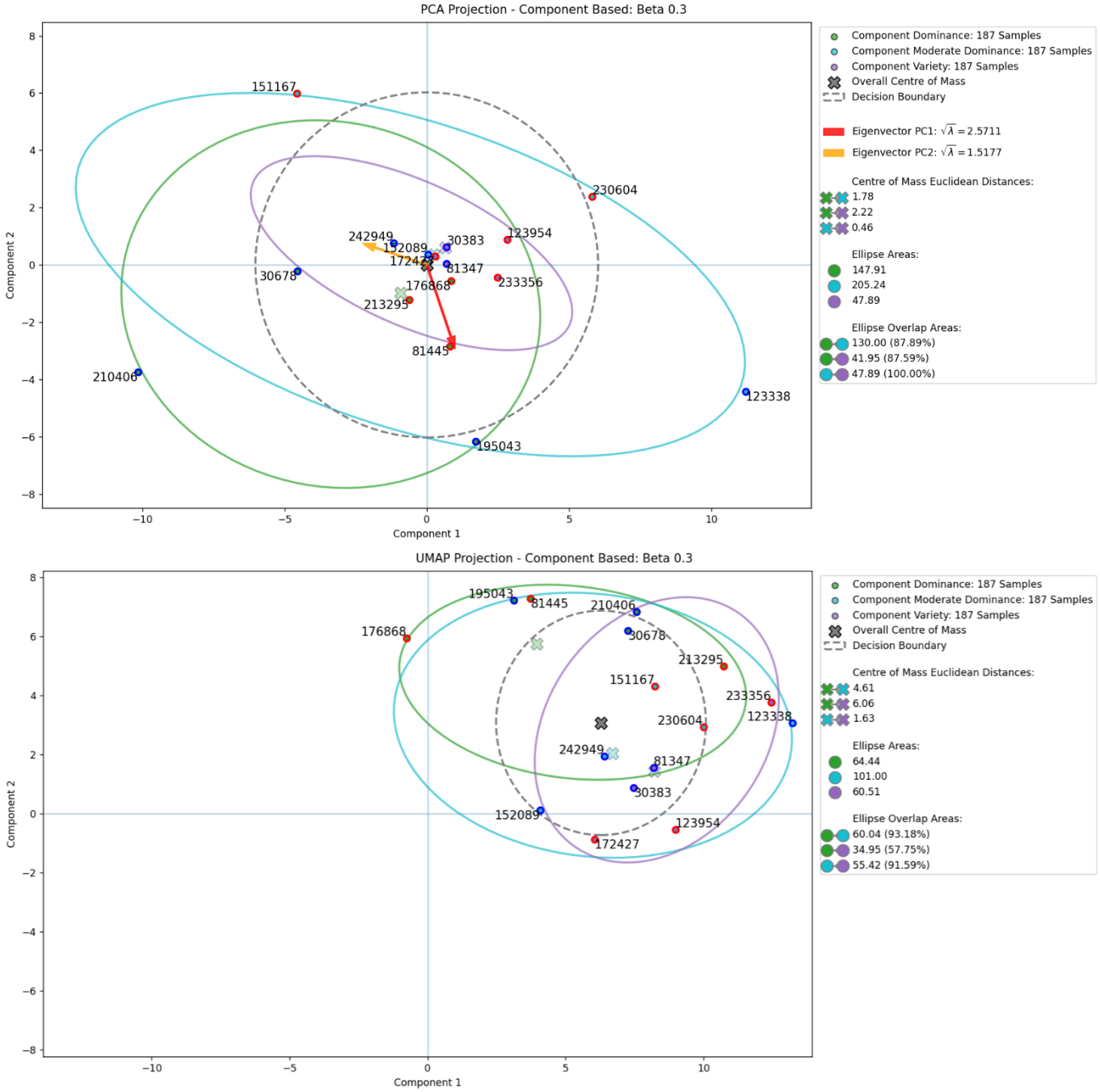


Figure 55. *β=0.3 model PCA and UMAP component point comparison*
Note: PCA (global relationships) and UMAP (local relationships) may position similar robots differently, highlighting the different relationships preserved by different reduction techniques (e.g., Samples 210406 and 30678). Created by the author using diagrams.net.

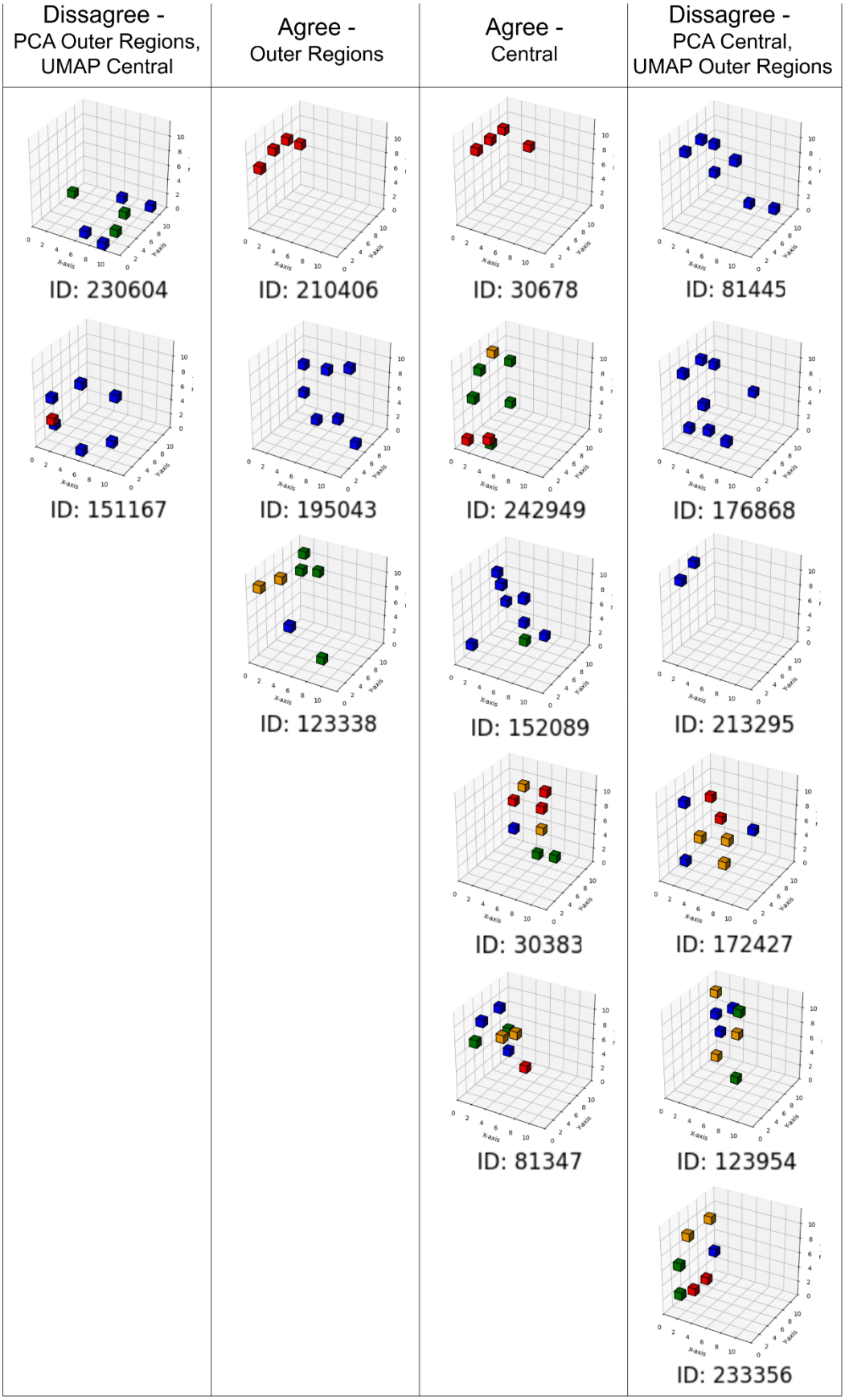


Figure 56. *β=0.3 model component point comparison original robots*
Note: Samples 210406 and 30678 appear similar in design, note their position in the projections in Figure 55.
Created by the author using diagrams.net.

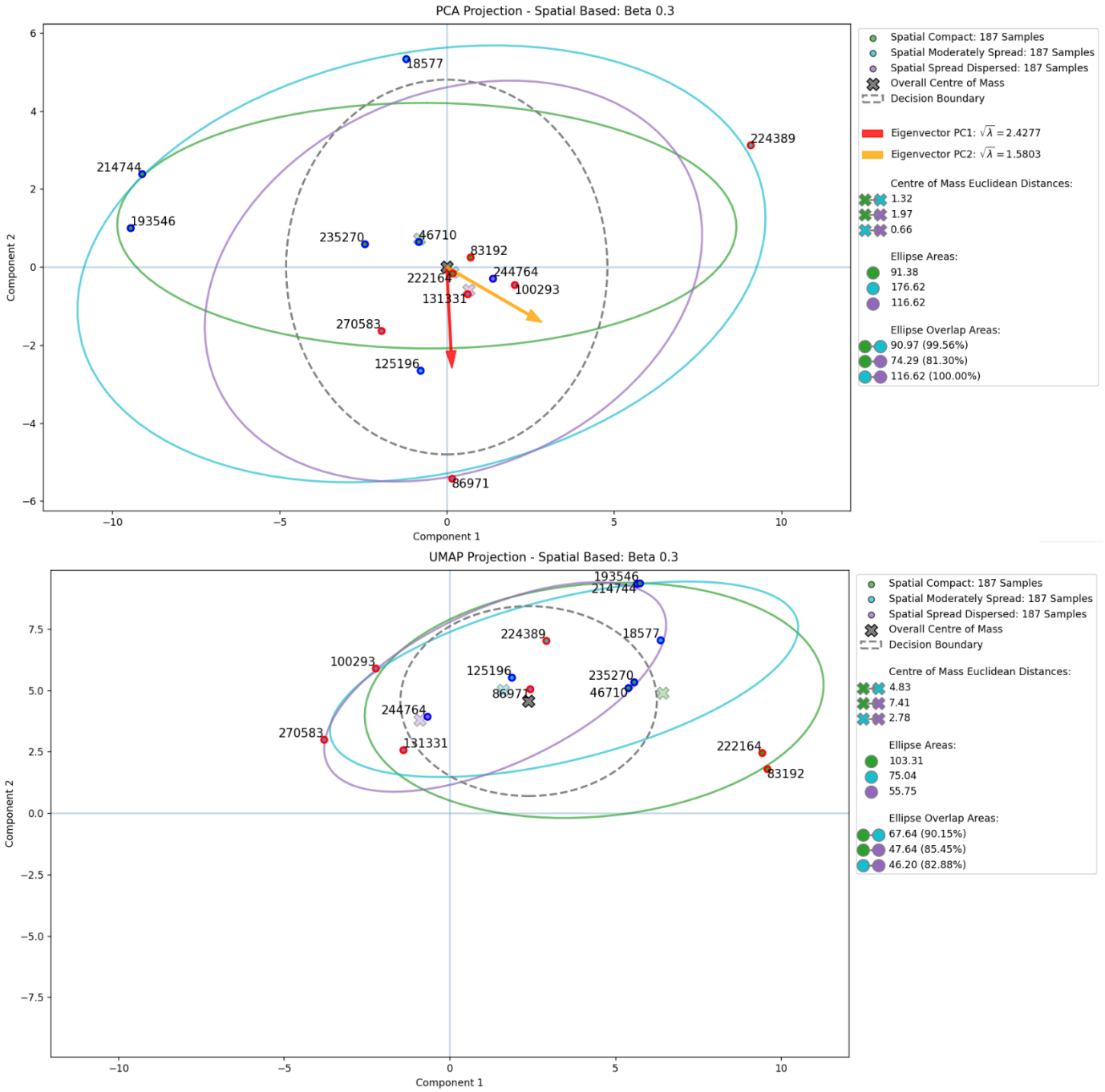


Figure 57. *β=0.3 model PCA and UMAP spatial point comparison*
Note: Samples 193546 and 214744 are almost in the same location in UMAP, they are also close together in PCA. Created by the author using diagrams.net.

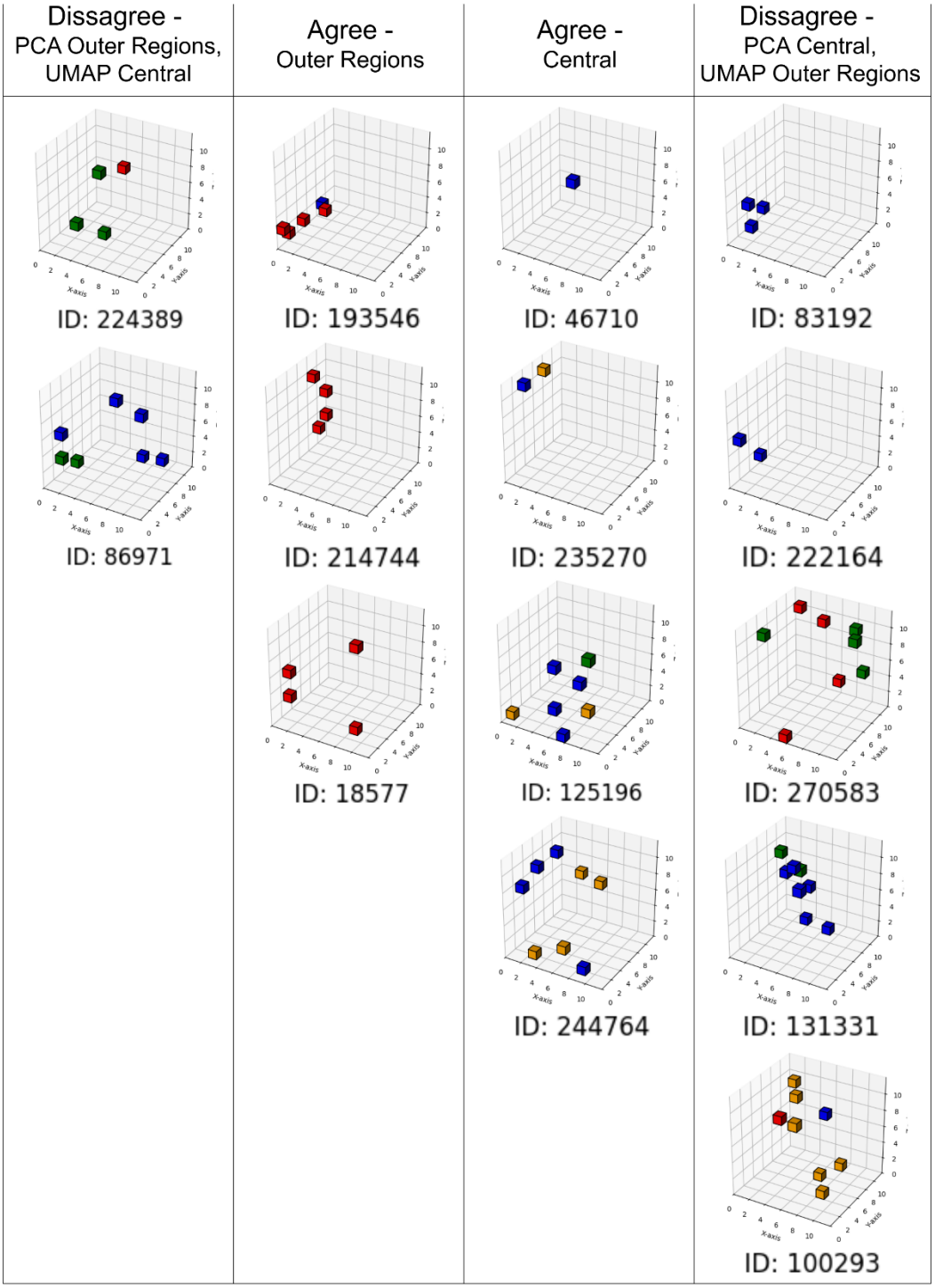


Figure 58. *β=0.3 model spatial point comparison original robots*
Note: Samples 193546 and 214744 appear to be similar but in different orientations. Created by the author using diagrams.net.

5 Discussion and Evaluation

This project set out to explore whether a Variational Autoencoder (VAE) trained on sparse 3D robot phenotype descriptor values could capture meaningful diversity in its latent space. To evaluate this, test data was grouped into separate datasets based on component dominance and spatial distribution, representing two distinct aspects of diversity. Latent vectors from the encoder were then analysed using PCA and UMAP to visualise the underlying structure of the latent space, and quantitative metrics were used to assess the separation between groups.

Although the model was trained without supervision regarding these groupings, the aim was to investigate whether the learnt latent representations aligned with observable morphological differences. A range of beta values was explored to assess the impact of regularisation strength on reconstruction quality and latent space organisation. Loss trends across models, including KL divergence and reconstruction losses, were also examined to understand trade-offs and support model selection.

Based on visual and quantitative comparisons, the β=0.3 model was selected for more detailed investigation. This section presents and interprets the findings from reconstruction comparisons, latent space projections, and dataset specific evaluations.

5.1 Effects of Beta Scaling

Beta (β) plays an important role in the VAEs performance by controlling the balance between reconstruction quality and latent space regularisation. Lower beta values ($\beta = 0.1$ to 0.5) preserved reconstruction detail, suggesting latent representations retained more meaningful structural and component information. In contrast, models trained with high beta values ($\beta = 1.5$ and 2.0) showed signs of latent space collapse. Their reconstructions became identical regardless of input (Figure 47 and Figure 48), whilst PCA and UMAP projections could not be computed due to zero variance in their z_{mean} vectors.

Loss trends also support this interpretation. For $\beta > 1.0$, KL divergence collapsed to near zero early in training, indicating that the latent space was not being effectively

utilised. Furthermore, descriptor loss increased with higher beta values, suggesting that component reconstruction became more difficult under strong regularisation. These patterns highlight the need for careful tuning of beta to maintain a meaningful latent space without overwhelming the reconstruction objective.

5.2 Understanding the Learnt Latent Space

5.2.1 Latent Space Structure

To evaluate whether the VAE captured meaningful morphological variation, PCA and UMAP were applied to the latent vectors (z_{mean}) for all test set robots (Figure 49). These were grouped into predefined component-based and spatial-based categories (Section 3.9).

Clustering was observed in both PCA and UMAP projections across the category groupings. PCA showed partial separation between datasets, whilst UMAP revealed clearer boundaries and more distinct clusters. These patterns were especially apparent when focusing on central groupings and overlooking outliers. In some cases, outliers may reflect valid similarities between samples that were not accounted for in the original dataset categories. For instance, samples grouped by component dominance may still have spatial similarities with those from other groups, and vice versa. Additionally, the model may be capturing more abstract or complex relationships that go beyond the manually defined categories.

A clear trend emerged when studying the centres of mass for each dataset in the projections (Figure 49). In both categories, these centres formed a logical progression if a line were to be drawn to connect them — stretching from focused designs (dominant, compact) to more expansive ones (variety, dispersed), with moderate datasets consistently in between. For models with $\beta = 0.1$ and $\beta = 0.3$, this progression aligned closely with the angular bisector between the PC1 and PC2 eigenvectors, suggesting a balanced contribution from both dimensions. In higher beta models, however, this alignment shifted and became more perpendicular, indicating a change in how the latent space was structured under stronger regularisation.

The $\beta = 0.3$ model was chosen for deeper analysis as it showed clear dataset separation and meaningful latent space structure across both component-based and

spatial-based projections (Figure 49). It also maintained good reconstruction quality (Figure 47 and Figure 48) and offered a reasonable trade-off between regularisation and detail retention. Loss trends (Figure 36 and Figure 42) further support this selection, with the $\beta = 0.3$ model achieving the highest KL divergence among the best performing models, indicating strong latent space utilisation without overwhelming the reconstruction objective. Although the $\beta = 0.5$ model was a strong contender based on its projections, it showed signs of over regularisation and latent collapse in the reconstructions, making $\beta = 0.3$ the more suitable choice.

5.2.2 Clustering and Dataset Separation in $\beta = 0.3$ Model

For the $\beta = 0.3$ model, PCA and UMAP projections showed clear evidence of latent structure aligned with the predefined dataset groupings (Figure 49). This section considers both component-based and spatial-based categories together, as several similar trends were observed across them.

In the PCA plots, the focused (dominant/compact) and expansive (variety/dispersed) datasets were visually separated by the x-axis. For the component-based projection, dominant samples appeared below the axis, whilst variety samples were above. In the spatial-based projection, dispersed samples were below and compact samples above. This vertical separation occurred along the y-axis, which corresponds to the direction of maximum variance as indicated by the red PC1 eigenvector. In both cases, moderate samples occupied a more central position between the two extremes, suggesting they share characteristics with both focused and expansive groups.

UMAP revealed even stronger separation. In the component-based plot, clusters were arranged vertically, with the dominant group positioned at the top, whilst the variety and moderate datasets formed tight clusters closer to the bottom. In the spatial-based UMAP, the separation was horizontal. The dispersed group was clearly defined on the left, whilst the compact group appeared to the right, with moderate samples distributed in between. These patterns suggest that the model has learnt to encode relevant spatial and component-based features in a consistent way.

Ellipses were fitted to highlight the general shape and spread of each dataset, although they are sensitive to outliers, particularly in cases where the model has learnt underlying similarities between samples from different groupings. In some cases, the fitted ellipses become larger or distorted. This may not fully reflect the core distribution, especially when moderate sets share features with both extremes.

Quantitative metrics support these visual interpretations. In PCA, centroid distances across the component sets were 1.78, 0.46, and 2.22, whilst for the spatial sets they were 1.32, 0.66, and 1.97 (Table 7). UMAP projections showed even greater centroid separation, with distances of 4.61, 1.63, and 6.06 for component groups, and 4.83, 2.78, and 7.41 for spatial groups (Table 8). These values confirm the clear group separation seen in UMAP and align with expected relationships between groups.

Ellipse areas and overlaps further reinforce the grouping separation and similarities. Moderate datasets showed the highest overlap with both extremes, reflecting their intermediate position. In component-based PCA, the moderate group also had the largest ellipse area (205.24), suggesting greater internal variance. In contrast, the variety group formed a small, tight cluster (47.89), which is unexpected given its high descriptor diversity and may suggest that the model learnt a simplified representation for these designs.

Together, these patterns indicate that the VAE has learnt to differentiate both component and spatial categories, with the moderate groups showing overlap across boundaries. This suggests that the model captures diversity in a continuous way, even without supervision on these categories.

5.2.3 Interpretations from Overlaid Visualisations

To explore what the VAE may be encoding in its latent space, original robot morphologies were overlaid onto the PCA and UMAP projections for the $\beta = 0.3$ model (Figure 50 to Figure 53). This approach allowed spatial locations in the projections to be mapped with specific input structures, enabling qualitative interpretation of how the model internally represents the samples.

At this stage, the focus shifted away from predefined groupings and instead, the goal was to examine whether the latent space had learnt meaningful structural or descriptor-based relationships on its own. The overlaid visualisations offer insight into how the VAE organises morphologies based on features it has independently identified as relevant.

5.2.3.1 PCA Projections

In the PCA plots, some broad spatial patterns could be observed across both the component-based and spatial-based overlays. Robots located in the lower region of the projection tend to show more symmetric layouts. These designs often had higher voxel counts and appeared either in a horizontal line or resemble vertical backward ‘C’ forms. Blue (wheel) and red (caster) descriptors were particularly common in the lower left region. In contrast, the upper-right region contained designs that tended to have spread in both horizontal and vertical directions. These robots also showed increased presence of yellow (sensor) and green (joint) descriptors.

The lower-right and upper-left corners showed another form of structure. These regions featured more rounded or centrally clustered designs, with the lower-right showing green and yellow components, and the upper-left leaning more toward red and blue. Overall, the distribution in PCA suggests that the VAE has encoded multiple overlapping traits, however, there does appear to be differences between the regions of the projections.

One notable pattern appears in the component-based PCA projection, where a series of robots are aligned diagonally along the PC2 eigenvector axis. These designs consist of six or seven voxels arranged in a vertically stacked, circular pattern, where the primary variation is in descriptor type rather than structure. In contrast, the diagonal running from lower left to upper right appeared to reflect spatial differences in layout. This suggests that PCA may be capturing structural and compositional variation along different directions in the latent space. An example of this repeated vertical arrangement is shown in Figure 59.

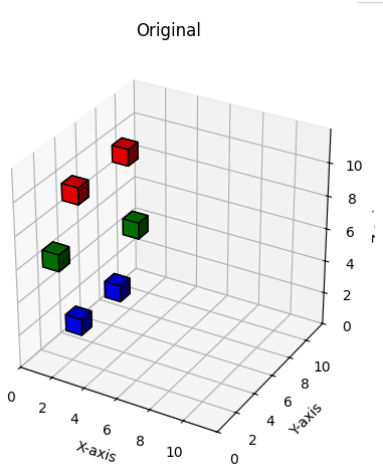


Figure 59. *Repeated pattern in PCA projection along PC2 maximum variance*
Note: Highlighting the repeated vertically stacked, circular patterns identified along PCA's PC2 axis.

5.2.3.2 UMAP Projections

In UMAP, the structural separation appeared even more distinct. Robots on the left side were generally more dispersed and occupied more space in the original 3D grid. These designs often used more voxels and had looser, more complex forms. On the right side, robots tended to be simpler, smaller, and more compact with voxels clustered closer together. Some of the designs on the right showed more symmetry and vertical reach.

Along the vertical axis of the UMAP projections, additional trends emerged. Designs located toward the bottom tended to be narrow and tubular, stretching primarily along one axis, although their orientations varied. Those near the top appeared more clustered toward one corner of the grid. These differences did not consistently reflect a single trait like size or descriptor diversity but seemed to reflect more subtle structural patterns.

Additionally, the descriptor types seem to vary with position, except for the wheels (blue) which seems to be more spread around. Red appears more dominant in the upper right, whilst joints (green) can be seen along the central horizontal portion, and sensors (yellow) can be found in the far reaches to the left or lower regions. This suggests that component placement is not random, but may be tied to the spatial or structural features the model has learnt to encode.

Taken together, these observations support the hypothesis that the VAE is capturing more than just surface level and obvious traits. The model appears to organise

morphologies based on combinations of spatial spread, component usage, symmetry, and possibly shape complexity. These patterns emerge independently of the predefined dataset categories. This pattern of organisation indicates that the latent space has learnt meaningful internal structure and could be used as a tool for assessing morphological diversity in a data-driven way.

5.2.4 PCA and UMAP Agreement

To explore how the latent space encodes structural and compositional features, comparisons were made between PCA and UMAP projections using selected robot samples. Initially, robots were categorised based on agreement or disagreement between their relative positions in each projection (Figure 55 to Figure 58). This approach was intended to highlight whether the latent space treated certain morphologies as typical or distinct in terms of both global (PCA) and local (UMAP) relationships (Cristian et al., 2024; Schmitz et al., 2021). However, this method proved to be slightly ambiguous in interpretation, as boundary placements could result in disagreement due to small shifts in projection, rather than genuine differences in the underlying encoding.

To address this limitation and focus on more meaningful comparisons, pairwise Euclidean distances were computed between the selected samples in both PCA and UMAP projection spaces. These are reported in full in Appendix L. A threshold distance of 2.5 was used to identify pairs that were close together in both projections. By analysing these samples, the aim was to understand which morphological features had led to these samples being positioned similarly in the latent space, and whether those similarities reflected structural, descriptor-based, or more abstract relationships.

In the component-based projections, several samples were found to be positioned close together. Samples 30383 and 81347 were located within the central region in both PCA and UMAP and were among the most similar pairs identified (PCA: 0.58, UMAP: 0.98). Visual inspection confirms that these robots share a similar number of voxels, similar component types, and similar overall shapes, although they differ slightly in rotation. This suggests that the latent space captures both component composition and approximate spatial layout, whilst being tolerant to orientation.

Samples 30383 and 172427 (PCA: 0.51, UMAP: 2.26) and 152089 and 172427 (PCA: 0.25, UMAP: 2.21) also appear similar, all being high in voxel numbers, moderately compact with diagonally spread. These samples are however, less similar than 30383 and 81347, particularly in descriptor composition and symmetry. PCA identifies them as highly similar, whereas UMAP places them further apart. These results reinforce that PCA preserves broader structural similarities, whereas UMAP may be more sensitive to small local differences such as voxel positioning or symmetry.

In the spatial-based projections, similar insights can be seen. Samples 193546 and 214744 had some separation in PCA, whilst being almost identical in UMAP (PCA: 1.43, UMAP: 0.09). These samples visually resemble one another in composition, differing only in orientation – 193546 being horizontal, whilst 214744 is vertical. Samples 83192 and 222164 (PCA: 0.66, UMAP: 0.67) form another highly similar pair, being compact, few blue voxels, and positioned in the corner of the grid. These examples suggest that the VAE latent space encodes structure in a way that reflects both spatial density and overall form, whilst being invariant to axis flips or mirroring. This could most likely be attributed to the T-Net component of the VAE model (Section 2.5 and Figure 34).

Rather than replacing the agreement classification, the pairwise distance analysis provides a more reliable foundation for interpreting the latent space structure. The original agreement projections (Figure 55 and Figure 57) and robot visuals (Figure 56 and Figure 58) remain useful for reference, whilst the similarity approach enables more targeted comparisons. Together, these approaches show that the VAE encodes abstract morphological traits such as how spread-out or compact a robot is, the variety of components used, and overall symmetry, whilst remaining tolerant to changes in orientation. This behaviour aligns with the expected influence of the T-Net module (Section 2.5) and supports the interpretation that the latent space preserves both global and local relationships in a meaningful and structured way.

5.3 VAE as a New Diversity Measurement Tool

Across the evaluation methods used – dataset separation, overlaid robot visualisations, and comparative PCA/UMAP analysis – the VAE demonstrated

meaningful structure in its latent space. The observed clustering patterns and the differences between local and global relationships suggest that the model learnt to encode both component-based and spatial-based diversity, as well as more complex morphological traits.

Whilst some of the latent traits remain difficult to interpret, all forms of analysis show some degree of separation and structure, supporting the idea that VAEs can capture and organise morphological diversity. These findings indicate that unsupervised latent representations can be used as tools for measuring diversity, although interpreting them remains a challenge.

Compared to existing approaches, the VAE framework offers a fundamentally different perspective. Methods such as that used by Yang et al. (2024) rely on predefined thresholds and assume linear relationships between features, whilst others like Le Goff & Smith (2024) and Medvet et al. (2021) manually define categories or calculate component counts. In contrast, the VAE learns a non-linear, continuous latent space directly from morphological data without requiring predefined features or species labels. It encodes variation implicitly, capturing both structural and spatial traits along with more complex relationships. These latent representations can then be analysed through projection, considering clustering, overlap, or spread as indicators of morphological diversity.

Beyond simply measuring diversity, VAEs could play a valuable role in EC frameworks. Their latent representations could be used to maintain population diversity, compare individuals, or identify underexplored regions of the solution space during the evolutionary process. This could help address common challenges in EC such as premature convergence or limited exploration, which were highlighted in the literature review (Section 2.3; Eiben & Smith, 2015; Le Goff & Smith, 2024; Medvet et al., 2021).

Additionally, the model’s unsupervised nature means it is scalable and potentially generalisable across domains. Although understanding and interpreting how the model learns and what is encoded remains a challenge, the results from this project suggest that VAEs could help improve both evaluation and exploration for EC tasks and similar application areas.

5.4 Limitations

Whilst this project presents promising evidence that VAEs can be utilised as tools for capturing morphological diversity, several limitations should be acknowledged. Due to time constraints, the model could not be fully optimised. Hyperparameter tuning and architectural refinement were limited, meaning that the final model may not represent the best performing configuration. Additionally, the dataset only encoded component types and did not include the underlying skeleton or structure of the robots. This limited the model's ability to learn deeper relationships involving how components were connected, along with adding to data sparsity.

To accommodate sparsity and address difficulties with the model learning, the data representation was adapted to a sparse voxel format. This introduced complexity in both model design and loss calculation, requiring the creation of custom loss functions to handle the lack of direct one-to-one correspondence between original and reconstructed voxels. Additionally, the sparse representation and lack of correspondence limited the usefulness of metrics like F1 score and accuracy. This led to reliance on loss metrics and visual inspection to evaluate model performance. These are discussed in detail in the methodology (Section 3).

PointNet architecture was introduced to address sparsity adopting the T-Net module with it, which may have introduced unintended limitations. Whilst T-Net is designed to standardise input alignment for stability, this transformation could suppress spatial differences that are important for measuring diversity. As it was not tested due to time constraints, future work should investigate the effect of excluding or modifying T-Net to assess its impact on encoding meaningful morphological variation.

An implementation issue was also discovered late in development, where the final activation layer of the decoder used LeakyReLU instead of the intended Softmax function. Although a Softmax version was trained, time constraints prevented sufficient tuning, and its performance was not good enough compared to the existing model to use for evaluation.

In addition to these practical limitations, there are also limitations in explainability. Like many deep learning models, VAEs function as black boxes. Although latent space projections help with analysis, understanding exactly what features and

relationships are being captured remains a difficult challenge. There is also the potential for hidden biases in the data or model architecture, which could impact its generalisability.

Finally, limitations exist in the evaluation metrics themselves. The ellipses used for visualising grouped datasets were constructed using Shapely from polygon approximations using the minimum volume enclosing ellipsoid (MVEE). This method is an approximation and can be sensitive to outliers due to the method attempting to enclose all of the dataset. This can inflate or distort ellipse area and overlap calculations. Therefore, these metrics should be interpreted as rough indicators rather than concrete measures of dataset spread or separation.

5.4.1 Future Work

Several extensions could build upon the foundations of this project. One direction would be to apply the VAE framework to robot populations evolved for specific tasks or terrains. The best performing individuals could be selected for VAE processing and comparison. Such an approach would allow evaluation of task-relevant diversity using real-world data, allowing clearer assessment of its effectiveness within an evolutionary context.

Removing or modifying the T-Net module could be a valuable direction for further work. Adjusting the architecture in this way may help preserve spatial differences, allowing the model to better capture morphological variation relevant to diversity measurement.

Future work could also explore the use of more descriptive and complete morphological data that includes the robot’s skeleton rather than just component types. Including this additional structure could reduce sparsity and help the model learn deeper structural relationships.

One valuable exploration path would be to use the latent space to identify underrepresented or empty regions in an evolutionary population’s distribution. Sampling from these gaps has the potential to generate novel robot morphologies, helping to increase population diversity and guide effective exploration within evolutionary systems.

Incorporating the VAE directly into the evolutionary loop could be explored. The latent space could be used to maintain morphological diversity, avoiding premature convergence, or informing selection. Automating this process may help improve the overall efficiency and adaptability of evolutionary algorithms.

This project focused only on the mean latent vector (z_{mean}) for projection and comparison. Future work could study how the log variance vector (z_{log_var}) contributes to the learnt representation, for instance by developing similarity metrics that account for both mean and variance. Such metrics could provide a more complete understanding of how the VAE encodes morphological representations.

Finally, extensions could explore automated design of VAE architectures, drawing on recent advances in evolutionary architecture search (Chen et al., 2021). This automation could identify configurations suited to morphological data and reduce manual effort, whilst improving diversity sensitivity.

6 Conclusion

This project set out to investigate whether a Variational Autoencoder (VAE) could be used to learn meaningful latent representations of evolved robot morphologies. The primary aim was to determine whether such a representation could be used to support the measurement of morphological diversity in a way that reflects learnt structural relationships, rather than relying on raw data or manually defining diversity categories or features. The project also posed two central research questions – first, whether a VAE could effectively encode sparse robot morphologies in an evolutionary context, and second, whether the learnt latent space could meaningfully reflect and separate diverse designs.

To explore these questions, a custom VAE architecture was designed specifically for the challenges of sparse, voxel-based (3D pixels) morphological data. Much of the project was dedicated to this design process, which proved significantly more complex than anticipated. Whilst the VAE framework itself is well established, applying it to morphology data in this form introduced several unique difficulties. Robots generated through a CPPN as part of an evolutionary system often contained only a few active voxels within a 3D grid. This extreme sparsity, along with

unavoidable imbalances in voxel number, type, and position due to the diversity inherent in the generated morphologies, introduced complexity. Additionally, the transformation from dense voxel grids to sparse point representations, whilst beneficial for reducing data sparsity, introduced unordered input, which created difficulties for learning.

Overcoming these challenges required a series of architectural and loss function design decisions. Inspiration was drawn from PointNet – a neural network for processing point-cloud data – but was adapted to account for both voxel coordinate and component descriptor encodings, along with learning a compressed representation. Custom loss functions were developed to address the key difficulties of unordered input and target uncertainty. These included a coordinate entropy-based loss to promote well distributed reconstructions, a descriptor matching loss to handle the target uncertainty for components, and a collapse penalty to avoid duplicate reconstructed voxels. Considerable experimentation was required to balance these losses and determine suitable values for the VAE's regularisation parameter (β). The final model, selected with $\beta = 0.3$, provided a promising balance between reconstruction quality and latent organisation.

Once trained, the model was analysed from three key angles. First, it was tested for its ability to separate predefined datasets, each grouped according to thresholds and spatial or component-based categories. These datasets had been constructed to reflect different aspects of morphological variation, although notably, each was defined without consideration of the other category. Clear separation was observed in the 2D latent space projections (via PCA and UMAP dimensionality reduction techniques), indicating that the model had successfully encoded high-level and meaningful differences across the categories. Second, individual robots were visualised by overlaying their morphologies in the projected latent space. This visualisation provided insight into what the model had learnt and confirmed that distinct spatial or structural features often clustered together, with similar patterns emerging along common planes. Third, PCA and UMAP outputs were compared to evaluate how the latent space captured morphologies from global and local relationship perspectives.

Whilst the projections showed the ability to handle rotational symmetry by placing similarly structured but differently oriented robots close together, they also occasionally highliighted less apparent relationships. Patterns observed in the projections suggest that the latent space is capturing more complex relationships beyond simple spatial transformations. These likely reflect interactions between spatial structures and component types.

These results provide strong evidence that the VAE was able to encode meaningful information about robot morphology. Whilst several limitations remain, the project provides a valuable foundation for future work. The learnt representation offers several methods toward quantifying morphological diversity in a flexible, data-driven way. Beyond supporting maintaining diversity in evolutionary populations, it may also be applied to tasks such as morphological novelty search or generative design.

In conclusion, this project successfully shows that a VAE can encode sparse morphological representations in a structured latent space and that this space can be used to analyse diversity between evolved robot designs. The architectural challenges were addressed through careful design and experimentation, and the resulting model provides insight into how learnt representations may capture structural and compositional variation. Whilst further development is needed to explore the full potential of this approach, this study demonstrates the value of representation learning within an evolutionary context and provides a foundation for new methods of quantifying and analysing morphological diversity.

References

Alhijawi, B., & Awajan, A. (2024). Genetic algorithms: theory, genetic operators, solutions, and applications. *Evolutionary Intelligence*, *17*(3), 1245–1256. https://doi.org/10.1007/s12065-023-00822-6

Amritesh. (2024). *PCA Examples Implementation Using Numpy & Sklearn*. Kaggle. https://www.kaggle.com/code/kelixirr/pca-examples-implementation-using-numpy-sklearn

Anvekar, T., Tabib, R. A., Hegde, D., & Mudengudi, U. (2022). VG-VAE: A Venatus Geometry Point-Cloud Variational Auto-Encoder [Proceeding]. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2977–2984. https://doi.org/10.1109/CVPRW56347.2022.00336

Bäck, T. H. W., Kononova, A. V., van Stein, B., Wang, H., Antonov, K. A., Kalkreuth, R. T., de Nobel, J., Vermetten, D., de Winter, R., & Ye, F. (2023). Evolutionary Algorithms for Parameter Optimization—Thirty Years Later. *Evolutionary Computation*, *31*(2), 81–122. https://doi.org/10.1162/evco_a_00325

Basic UMAP Parameters. (2018). Read the Docs. https://umap-learn.readthedocs.io/en/latest/parameters.html

Cambridge University Press & Assessment. (n.d.). *Morphology*. Retrieved January 24, 2025, from https://dictionary.cambridge.org/dictionary/english/morphology

Chen, X., Sun, Y., Zhang, M., & Peng, D. (2021). Evolving Deep Convolutional Variational Autoencoders for Image Classification. *IEEE Transactions on Evolutionary Computation*, *25*(5), 815–829. https://doi.org/10.1109/TEVC.2020.3047220

Cicirello, V. A. (2024). Evolutionary Computation: Theories, Techniques, and Applications. *Applied Sciences*, *14*(6), 2542. https://doi.org/10.3390/app14062542

Cristian, P.-M., Aarón, V.-J., Armando, E.-H. D., Estrella, M.-L. Y., Daniel, N.-R., David, G.-V., Edgar, M., Paul, S.-C. J., & Osbaldo, R.-A. (2024). Diffusion on

PCA-UMAP Manifold: The Impact of Data Structure Preservation to Denoise High-Dimensional Single-Cell RNA Sequencing Data. *Biology*, *13*(7), 512. https://doi.org/10.3390/biology13070512

Eiben, A. E., & Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-44874-8

Elbow Method for optimal value of k in KMeans. (2025, February 7). GeeksforGeeks. https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/

Emmerich, M., Shir, O. M., & Wang, H. (2018). Evolution Strategies. In *Handbook of Heuristics* (pp. 1–31). Springer International Publishing. https://doi.org/10.1007/978-3-319-07153-4_13-1

Fernandes, P., Correia, J., & Machado, P. (2020). *Evolutionary Latent Space Exploration of Generative Adversarial Networks*. 595–609. https://doi.org/10.1007/978-3-030-43722-0_38

Floreano, D. M. C. (2008). *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. MIT Press.

Foster, D. (2023). *Generative deep learning: teaching machines to paint, write, compose, and play* (K. Friston, Ed.; Second edition.) [Book]. O'Reilly Media, Incorporated.

Gandomi, A., Alavi, A., & Ryan, C. (2015). *Handbook of Genetic Programming Applications* (A. H. Gandomi, A. H. Alavi, & C. Ryan, Eds.). Springer International Publishing. https://doi.org/10.1007/978-3-319-20883-1

Gillies, S. (2025, January 31). *The Shapely User Manual*. Shapely. https://shapely.readthedocs.io/en/stable/manual.html

Goodfellow, I. (2016). *Deep learning* (Y. Bengio & A. Courville, Eds.) [Book]. The MIT Press.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative Adversarial Networks*. https://doi.org/10.48550/arXiv.1406.2661

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. https://doi.org/10.48550/arXiv.1512.03385

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–507. https://doi.org/10.1126/science.1127647

Imelfort, M. (2013). *Ellipsoid*. GitHub. https://github.com/minillinim/ellipsoid/blob/master/ellipsoid.py

Jabari, I. K. O., Shofiyah, S, P. K., Putriwijaya, N. N., & Yudistira, N. (2024). *Learning-Augmented K-Means Clustering Using Dimensional Reduction*. https://doi.org/10.1145/3626641.3627239

Jin, Y. (2023). *Computational Evolution of Neural and Morphological Development*. Springer Nature Singapore. https://doi.org/10.1007/978-981-99-1854-6

Jong, K. De. (2009). Evolutionary computation. *WIREs Computational Statistics*, *1*(1), 52–56. https://doi.org/10.1002/wics.5

Kingma, D. P., & Welling, M. (2013). *Auto-Encoding Variational Bayes*. https://doi.org/10.48550/arXiv.1312.6114

Le Goff, L. K., Buchanan, E., Hart, E., Eiben, A. E., Li, W., De Carlo, M., Winfield, A. F., Hale, M. F., Woolley, R., Angus, M., Timmis, J., & Tyrrell, A. M. (2023). Morpho Evolution With Learning Using a Controller Archive as an Inheritance Mechanism [Article]. *IEEE Transactions on Cognitive and Developmental Systems*, *15*(2), 507–517. https://doi.org/10.1109/TCDS.2022.3148543

Le Goff, L. K., & Smith, S. C. (2024). *Efficient and Diverse Generative Robot Designs using Evolution and Intrinsic Motivation*. https://doi.org/10.48550/arXiv.2411.18423

Liu, L., Fei, T., Zhu, Z., Wu, K., & Zhang, Y. (2023). A Survey of Evolutionary Algorithms. *2023 4th International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE)*, 22–27. https://doi.org/10.1109/ICBAIE59714.2023.10281260

Ma, Z., Wu, G., Suganthan, P. N., Song, A., & Luo, Q. (2023). Performance assessment and exhaustive listing of 500+ nature-inspired metaheuristic algorithms. *Swarm and Evolutionary Computation*, *77*, 101248. https://doi.org/10.1016/j.swevo.2023.101248

Mastromichalakis, S. (2020). *ALReLU: A different approach on Leaky ReLU activation function to improve Neural Networks Performance*. https://doi.org/10.48550/arXiv.1605.09332

Mathieu, E., Rainforth, T., Siddharth, N., & Teh, Y. W. (2018). *Disentangling Disentanglement in Variational Autoencoders*. https://doi.org/10.48550/arXiv.1812.02833

Medvet, E., Bartoli, A., Pigozzi, F., & Rochelli, M. (2021). Biodiversity in evolved voxel-based soft robots. *Proceedings of the Genetic and Evolutionary Computation Conference*, 129–137. https://doi.org/10.1145/3449639.3459315

Molnar, S., & Tamas, L. (2024). Variational autoencoders for 3D data processing [Article]. *The Artificial Intelligence Review*, *57*(2), 42. https://doi.org/10.1007/s10462-023-10687-x

Mukherjee, A. K. (2023, August 16). *Feature Dimensionality Reduction Techniques Part II: t-SNE*. Medium.

Oskolkov, N. (2022). *Dimensionality Reduction*. https://doi.org/10.1007/978-3-030-88389-8_9

Oyewole, G. J., & Thopil, G. A. (2023). Data clustering: application and trends. *Artificial Intelligence Review*, *56*(7), 6439–6475. https://doi.org/10.1007/s10462-022-10325-y

Pigozzi, F., Camerota Verdù, F. J., & Medvet, E. (2023). How the Morphology Encoding Influences the Learning Ability in Body-Brain Co-Optimization. *Proceedings of the Genetic and Evolutionary Computation Conference*, 1045–1054. https://doi.org/10.1145/3583131.3590429

Pramod, O. (2024, June 16). *Autoencoders Explained*. Medium. https://medium.com/@ompramod9921/autoencoders-explained-872f77de0d79#:~:text=On%20the%20other%20hand%2C%20in,similar%20to%20a%20known%20distribution.

Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2016). *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. https://doi.org/10.48550/arXiv.1612.00593

Saxena, A., Prasad, M., Gupta, A., Bharill, N., Patel, O. P., Tiwari, A., Er, M. J., Ding, W., & Lin, C.-T. (2017). A review of clustering techniques and developments. *Neurocomputing*, *267*, 664–681. https://doi.org/10.1016/j.neucom.2017.06.053

Schmitz, S., Weidner, U., Hammer, H., & Thiele, A. (2021). EVALUATING UNIFORM MANIFOLD APPROXIMATION AND PROJECTION FOR DIMENSION REDUCTION AND VISUALIZATION OF POLINSAR FEATURES. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *V-1–2021*, 39–46. https://doi.org/10.5194/isprs-annals-V-1-2021-39-2021

scikit-learn developers. (2025a). *KMeans*. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

scikit-learn developers. (2025b). *PCA*. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

scikit-learn developers. (2025c). *silhouette_score*. Scikit-Learn . https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

Shende, R. (2023, April 19). *Autoencoders, Variational Autoencoders (VAE) and β-VAE*. https://medium.com/@rushikesh.shende/autoencoders-variational-autoencoders-vae-and-%CE%B2-vae-ceba9998773d

Slowik, A., & Kwasnicka, H. (2020). Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, *32*(16), 12363–12379. https://doi.org/10.1007/s00521-020-04832-8

Starbuck, C. (2023). Descriptive Statistics. In *The Fundamentals of People Analytics* (p. 103). Springer International Publishing. https://doi.org/10.1007/978-3-031-28674-2_7

Sudholt, D. (2020). *The Benefits of Population Diversity in Evolutionary Algorithms: A Survey of Rigorous Runtime Analyses* (pp. 359–404). https://doi.org/10.1007/978-3-030-29414-4_8

Sydsæter, K., Strøm, A., & Berck, P. (2010). Special matrices. Leontief systems. In *Economists' Mathematical Manual* (pp. 151–152). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-28518-2_22

Terven, J., Cordova-Esparza, D. M., Ramirez-Pedraza, A., Chavez-Urbiola, E. A., & Romero-Gonzalez, J. A. (2023). *Loss Functions and Metrics in Deep Learning*. https://doi.org/10.48550/arXiv.2307.02694

Thomson, S. L., Le Goff, L. K., Hart, E., & Buchanan, E. (2024). Understanding fitness landscapes in morpho-evolution via local optima networks [Article]. *ArXiv.Org*. https://doi.org/10.48550/arxiv.2402.07822

Todd, M. J., & Yıldırım, E. A. (2007). On Khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids. *Discrete Applied Mathematics*, *155*(13), 1731–1744. https://doi.org/10.1016/j.dam.2007.02.013

Wang, K., Yang, Y., Wu, F., Song, B., Wang, X., & Wang, T. (2023). Comparative analysis of dimension reduction methods for cytometry by time-of-flight data. *Nature Communications*, *14*(1), 1836. https://doi.org/10.1038/s41467-023-37478-w

Xu, D., & Tian, Y. (2015). A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*, *2*(2), 165–193. https://doi.org/10.1007/s40745-015-0040-1

Xu, Y., Tong, X., & Stilla, U. (2021). Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry. *Automation in Construction*, *126*, 103675. https://doi.org/10.1016/j.autcon.2021.103675

Yadav, A. (2024, October 8). *A Comprehensive Guide to Latent Space in Machine Learning*. Medium. https://medium.com/biased-algorithms/a-comprehensive-guide-to-latent-space-in-machine-learning-b70ad51f1ff6#:~:text=Interpreting%20Latent%20Representations&text=When%20you%20look%20at%20a,they%20share%20some%20underlying%20characteristics.

Yang, Q., Chu, S.-C., Pan, J.-S., Chou, J.-H., & Watada, J. (2024). Dynamic multi-strategy integrated differential evolution algorithm based on reinforcement learning for optimization problems. *Complex & Intelligent Systems*, *10*(2), 1845–1877. https://doi.org/10.1007/s40747-023-01243-9

Yong, R. (2021, July 8). *Variational Autoencoder(VAE)*. Medium. https://medium.com/geekculture/variational-autoencoder-vae-9b8ce5475f68

Yu, X., & Gen, M. (2010). *Introduction to Evolutionary Algorithms* (Mitsuo. Gen & SpringerLink (Online service), Eds.) [Book]. Springer London. https://doi.org/10.1007/978-1-84996-129-5

Appendix A: Personal Appraisal

At the start of the project, I was excited by the challenge and saw it as a good opportunity to apply the PyTorch skills I had developed over the summer. My goal was to investigate whether a variational autoencoder (VAE) could be used to encode robot morphologies in a way that supports diversity measurement. I began by creating a first version of the Gantt chart to plan the main project stages. Early on, I spent a lot of time reading into and understanding relevant areas such as evolutionary computation and neural network architectures, particularly VAEs. This initial phase took longer than I expected, and by the end of the first trimester, I was behind where I had hoped to be.

I used the Christmas break to catch up by completing the literature review and laying down most of the initial codebase. At the beginning of trimester two, I reassessed my timeline and created a second version of the Gantt chart, helping me regain control of the workload and focus my effort on the most important parts. From that point on, I made steady progress and kept track of everything through regular diary entries and commented commits to GitHub. I broke the work down into manageable tasks and adjusted my attention as needed when new challenges arose.

One of the biggest challenges was getting the model to learn anything useful from the data. The robots in the dataset were extremely sparse, and early model versions struggled to make meaningful reconstructions. After some-trial-and-error, I realised that the core problem was how the data was structured. This led me to redesign both the data representation and the model architecture, switching to a sparse input format and adapting a different processing approach that could handle unordered data.

These changes also meant I needed to develop custom loss functions, since standard ones could no longer be applied. This was one of the most difficult but rewarding parts of the project. I had to think carefully about how to measure similarity between unordered voxel positions and descriptor types, and how to encourage the model to align its reconstructions. I iterated through several designs, debugging and adjusting as I went. This process improved my ability to design and test ideas

independently and taught me how to approach complex technical problems in a more structured way.

Throughout the project, I developed a range of skills. On the technical side, I became much more confident working with PyTorch, managing data pipelines, building and training neural networks, and using high-performance computing resources (HPC). On the soft skills side, I learnt to manage a long-term project, respond to setbacks, adapt plans when needed, and document my work carefully. I made a deliberate effort to keep my code clean and commented, and although not everything is perfect, I am happy with the overall quality of the implementation.

Looking back, I think the project went well overall. There were points where I felt stuck or behind, but I managed to work through them by adjusting my approach and sticking with the problem until I found a solution. More time for tuning and experimentation before the evaluation phase would have been helpful, but I am confident that the final model is a good representation of the work I set out to do. This was one of the most technically demanding projects I have done, but despite the difficulties, it helped me grow as both a developer and a researcher, especially in applying machine learning to sparse and challenging data.

Appendix B: Initial Project Overview (IPO)

Andrew Taison - IPO 40538519

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: Exploring Variational Autoencoders (VAEs) for Encoding Robot Morphologies, Measuring Diversity, and Generating Novel Forms in an Evolutionary Framework

Overview of Project Content and Milestones

This project aims to develop a Variational Autoencoder (VAE) to effectively encode and reconstruct robot morphologies, intending to capture important and complex underlying features. By doing so, it intends to improve how diversity is measured in evolutionary robotics. Current methods of measuring diversity (for comparing two or more robots) are based on quantifying morphological features (number of wheels, sensors etc.), which may not effectively capture the true complexity of the robots (e.g. it is not sensitive to transformations such as rotation or size). Using a VAE will enable measurement through analysis of clustering in the probabilistic latent space, potentially allowing for more complex and meaningful features to be considered. If time allows, the project will investigate latent space gaps for generating novel robot forms via the decoder, creating diverse additions to the evolutionary population.

Research Questions:

1. Technical Question: Can a VAE effectively encode robot morphologies in an evolutionary framework?

2. Analytical Question: Can the latent space of a VAE be used to measure diversity across generations of robot morphologies?

3. Exploratory Question (Time conditional): Can the latent space of a VAE be used to identify gaps and, by utilising the decoder, generate diverse and novel robot forms from these gaps?

Milestones:

- Literature Review: Research relevant material regarding VAE's, evolutionary robotics and diversity measurement techniques.
- Identify suitable datasets, tools and techniques.
- Develop methodology.
- VAE Implementation: Build, train and test with minimal reconstruction loss.
- Analysis of the latent space: Visualise how robots are grouped and clustered.
- Diversity Measurement: Evaluate performance of how morphological variations are captured. Demonstrating how the VAE / latent space can be leveraged to improve diversity measurement.
- Explore Latent Space Gaps (optional): Investigate gaps and potential to generate novel robot forms, contributing to increased diversity in the population.
- Collect and analyse results on effectiveness of diversity measurement.
- Poster Creation.

Andrew Taison - IPO 40538519

The Main Deliverable(s):

- Detailed literature review containing background information and knowledge to provide sufficient understanding of the project.
- Implementation of a VAE for encoding robot morphologies with minimal loss.
- Analysis of the VAE's latent space:
  - Visualisations demonstrating how the VAE organises and clusters robots based on key morphological features.
  - Quantitative and qualitative evaluations of the performance of the VAE determining how well it captures morphological variations (e.g. sensitivity to transformations such as rotation, size or differences in features such as number of wheels).
- Key findings and insights on how VAE's offer improvements for measuring diversity. This should also include any limitations (e.g. potential loss of information or biases).
- A concluding section discussing the projects overall outcomes, applications and future work.
- A project poster summarising the overall project.

The Target Audience for the Deliverable(s):

- Academic Community:
  - Researchers, individuals or teams working in fields related to machine learning, evolutionary robotics, or data science may be interested in the findings.
  - Students researching similar topics may find the project relevant.
- Professionals in Robotics or Data Science.

The work may be applicable to those trying to measure or produce increased diversity in their data or projects.

The Work to be Undertaken:

- Literature Review:
  - Research and investigate key concepts – VAE's, robot morphologies, evolutionary computation, and diversity metrics.
  - Compare various dimensionality reduction, clustering and visualisation techniques, particularly in relevance to the latent space of a VAE.
  - Explore alternative generative models and examine existing work on diversity measurement and robotic evolution.
- Identify suitable datasets:
  - Obtain or generate datasets of robot morphologies.
  - Preprocessing – normalise, clean and ensure compatibility with VAE.
  - Understand structure of the data.
- Identify suitable tools and techniques for experiments considering:
  - Structure of VAE.
  - Visualisation and clustering.
  - Validation criteria and performance metrics.
- Implement a VAE model and train focusing on effectively encoding robot morphologies with minimal reconstruction loss.
- Analysis of the latent space:
  - Visualise how robots are grouped and clustered.
  - Evaluate performance of how morphological variations are captured.

    - Compare diversity measurement with other methods (such as feature based methods).
- Generate Novel Morphologies (Optional – time permitting):
    - Explore latent space to identify gaps and generate novel forms.
    - Visualise in simulator.
- Create a project poster.

Additional Information / Knowledge Required:

Significant time has been dedicated over the summer to learning PyTorch and implementing a basic VAE. However, for the success of this project, it will be important to gain a deeper understanding of how to effectively design and structure (particularly the layers of) VAE models.

Additionally, expanding knowledge of clustering and visualisation techniques will be essential for effective analysis of the latent space.

Understanding the process of running programs on the HPC (High Performance Computing) system will be important for training models efficiently. Furthermore, familiarity with CoppeliaSim simulator will be necessary to visualise robots.

Information Sources that Provide a Context for the Project:

The initial idea for this project stems from collaboration between Simon Smith and Leni Le Goff, who are researchers at Edinburgh Napier University. They have been working on developing an evolutionary framework to automatically produce robot bodies (morphologies) whilst simultaneously training controllers (essentially the robot brains). Le Goff et al. (2023) propose a framework for generating and evolving various body plans and robot structures whilst simultaneously training controllers. By utilising a controller archive and combining with body creation, it can allow for better robots to be generated faster. In their paper, they indicate that the evolutionary process may converge too quickly, "proceeding along the 'easiest' path" resulting in successful robots being of a similar type. This can be at the detriment of exploring diverse options. The controller archive is populated using the best found controllers for certain body types. Currently, in their framework, type is defined using characteristics (e.g. number of wheels, or sensors). This could lead to transformational aspects such as rotation or size being underrepresented. Using a VAE offers the potential to overcome this and capture more complex and important features in diversity.

Shende (2023) summarises three of the different types of generative models used in unsupervised machine learning including autoencoders, variational autoencoders, and beta-variational autoencoders.

Kingma & Welling (2013) outlines the concept of VAE's and introduces the parametrisation trick, which enables efficient differentiation and optimisation using stochastic gradient decent.

The Importance of the Project:

The project aims to help improve the measurement of diversity by utilising VAE's. This seeks to provide a deeper understanding of diversity allowing complexities to be captured which simpler models and methods may not. This will hopefully lead to a more informative understanding of diversity within robot morphologies leading to better decisions in robot

design and selection. Furthermore, the technique should also help improve exploration of the search space and potentially accelerate convergence. These improvements may also lead to the development of more effective distance functions for measuring similarity or diversity among robot morphologies.

The Key Challenge(s) to be Overcome:

Challenges that must be addressed for the projects success include:

- Dataset Selection: Choosing the appropriate dataset(s) will be vital. This requires considering the number of features and ensuring the datasets sufficiently represent well defined, diverse robot morphologies.
- Model Design and Tuning: Designing and tuning the VAE model could present a significant challenge. Difficulties in creating an effective model could prove detrimental to the project. Key considerations include latent space size, architecture choices (number and types of layers and activation functions), and hyperparameter tuning (learning rates etc.).
- Performance Evaluation: Determining appropriate metrics to inform on the performance of the VAE and how well it captures morphological variations and measuring diversity could be complex. This is an important aspect of the project to determine its effectiveness compared to other diversity measurement methods.
- Familiarity With Required Tools: This project will utilise a lot of new tools, resources, and techniques (PyTorch, clustering techniques, HPC, CoppeliaSim etc.). Enough time must be dedicated to learning these and becoming familiar with them for the success of the project.

References

Kingma, D. P., & Welling, M. (2013). *Auto-Encoding Variational Bayes*. https://doi.org/10.48550/arXiv.1312.6114

Le Goff, L. K., Buchanan, E., Hart, E., Eiben, A. E., Li, W., De Carlo, M., Winfield, A. F., Hale, M. F., Woolley, R., Angus, M., Timmis, J., & Tyrrell, A. M. (2023). Morpho Evolution With Learning Using a Controller Archive as an Inheritance Mechanism [Article]. *IEEE Transactions on Cognitive and Developmental Systems*, *15*(2), 507–517. https://doi.org/10.1109/TCDS.2022.3148543

Shende, R. (2023, April 19). *Autoencoders, Variational Autoencoders (VAE) and β-VAE*. https://medium.com/@rushikesh.shende/autoencoders-variational-autoencoders-vae-and-%CE%B2-vae-ceba9998773d

Appendix C: Interim Review

SOC10101 Honours Project (40 Credits)

Week 9 Report

Student Name: Andrew Dickinson

Supervisor: Simon Smith

Second Marker: Peter Chapman

Date of Meeting: 14-1-25

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **yes**

If not, please comment on any reasons presented

Andrew, as ever, keeps meticulous records of his time.

Please comment on the progress made so far

Andrew has made excellent progress, even if he does not believe it. This is a difficult problem, and some rescoping of it may be necessary. But, this is not a reflection on the progress made, or the efforts of Andy.

Is the progress satisfactory? **yes**

Can the student articulate their aims and objectives? **yes**

If yes then please comment on them, otherwise write down your suggestions.

As mentioned above, there is inherent difficulty in doing actual research. Some rescoping is necessary to fit an honours project, and to alleviate Andrew's stress.

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Does the student have a plan of work? **yes**

If yes then please comment on that plan otherwise write down your suggestions.

A clear plan, which is still ambitious, but should be achievable.

Does the student know how they are going to evaluate their work? **yes**

If yes then please comment otherwise write down your suggestions.

Yes, but some of it is dependent on how the remaining work goes. Clear pathways exist, however.

Any other recommendations as to the future direction of the project

I expect Andrew's work to be of high quality, given the level of questions/discussions had at this meeting.

Signatures: Supervisor Simon Smith Second Marker Peter Chapman

Student Andrew Dickinson

The student should submit a copy of this form to Moodle immediately after the review meeting; A copy should also appear as an appendix in the final dissertation.

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Appendix D: Diary Sheets

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 22/09/2024 **Last diary date:**

Objectives:

Work to complete Initial Project Overview (IPO).
Construct a Gantt chart to gain a better understanding of the complete project timeline.

Start reviewing papers.

Progress:

Project registration submitted – This includes having decided on a project title.
I emailed the registration to Simon for feedback, and he has confirmed he is happy that it is a good summary / outline of the project (see comments below).

Supervisor's Comments:

Hi Andrew,

This is a very good summary of what we want to do. Congratulations. I think that the third goal that you added is on point, and hopefully we can have time to do it. If we do, I'm sure we can make this a substantial contribution to the field, i.e. write a paper (but this would be focusing too much ahead of the current goals, 1 and 2).

I will contact you to have a meeting next week, as this one is being fully booked with teaching and its preparations.

Best regards,
Simon

Version 2 Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 29/09/2024 **Last diary date:** 22/09/2024

Objectives:

Make any required changes and submit IPO once received feedback from Simon. Following submission, email Peter (2nd marker) to provide feedback.

Schedule regular weekly meetings with Simon.

Start reviewing papers and constructing literature review – aim for 1 paper a day (5 – 7 per week), along with around 400 words (although this early on, there may not be much to write about, review each week).
Main task is to start outlining relevant literature and collecting in Mendeley.

Progress:

IPO and Gantt completed in draft – sent to Simon for feedback.
Includes having outlined research questions – these may require refining potentially as the project continues.

Reviewed a couple of papers, but not as many as I would have liked.

Supervisor's Comments:

We did not get around to scheduling a meeting for this week due to Simon being busy. However, I don't think this is much of an issue this week as the main task was working on creating the IPO and Gantt. We are trying to schedule one for next week.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 08/10/2024 **Last diary date:** 29/09/2024

Objectives:

Schedule regular weekly meetings with Simon.

Continue reviewing papers and constructing literature review – aim for 1 paper a day (5 – 7 per week). Need to review weekly schedule to make more time for this task.
Main task is to start outlining relevant literature and collecting in Mendeley.

Progress:

Submitted IPO and I have emailed Peter to inform him of its submission - Simon was happy with the overview.

Unfortunately, we have been unable to schedule a meeting, however Simon and I have been keeping in contact via email. At this stage during early research this is probably sufficient.

I have started sourcing and summarising papers, though not quite at the pace I would like to be aiming for – managed around 5 papers this week which is the lower end of my target. Will need to look for more opportunities to make time to review more papers

Furthermore, I have roughly outlined the literature review research areas based on the IPO set out (as much as I can in brief before really delving into the research). The thinking here is to help guide my research and sourcing of papers.

Supervisor's Comments:

Hi Andy,

Sorry for the late reply, but I will be very busy until next week. I have looked over your IPO and it is again very well written. If the complain about the length, say that I approve this length to capture the complexity of the project. I think it is realistic, as the last part has been defined as optional. The first two questions you will be able to answer properly with the time, data and computational power that we have.

I will send you a invitation to meet next week.

Best regards,
Simon

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 15/10/2024 **Last diary date:** 08/10/2024

Objectives:

Continue to source papers, try to get more focused. Look for examples of what people are currently doing with, or how they are using relevant technologies regarding evolutionary computation, morph evolution, and variational autoencoders (ideally 2-3 for each).

Produce slide for meeting with Simon detailing what has been done in the last week and plan for the next.

Progress:

I have been continuing to collect papers and started getting more specific in my sources. I still feel like the progress is slow. However, saying this, I managed to find and briefly summarise around 15 papers (although none of these have been in-depth reads).

Managed to have a meeting online with Simon, and we have scheduled these weekly on Fridays going forward.

Supervisor's Comments:

Simon believes my research template/guide is possibly too complicated. I should cut it back (for now) and focus on finding sources that demonstrate the current state of areas of research. This should be with regard for evolutionary computation, morph evolution, and variational autoencoders. Find out what people are currently using these things for, and some papers which talk about how these work (possibly around 2-3 papers for each).

Produce a slide (basic, nothing pretty) that details what I have done each week and what my plan is for the next week to take to each meeting.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 20/10/2024 **Last diary date:** 15/10/2024

Objectives:

Start to go through the sources that have been collected in more detail in preparation to start writing.

Produce slide for meeting with Simon detailing what has been done in the last week and plan for the next.

Progress:

Continued to source papers, didn't produce a slide as not a lot to put on there.

Supervisor's Comments:

Slide is a reminder for Simon and I as to where we are and to keep talking points during the meeting – so try to produce one regardless of how much completed in last week.

EC is a large field, won't be expected to know or read everything, probably just need to focus on the important areas – EC, Morph, VAE. Be conservative about paper selection, only use most representative, i.e. don't want 100 papers.

Simon working on talking to Leni about datasets (obtaining & format)

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 26/10/2024 **Last diary date:** 20/10/2024

Objectives:

Now I have most sources, continue to go through them in more detail and preparing to start writing ASAP.

Start considering experiments – even if just thinking about them for now.

Add objectives to future meeting slides.

Progress:

Believe I have now finished collecting most sources required. I have been through these and chose what I believe are the most appropriate for each section of the literature review.

Started going through the sources in detail and utilising Mendeley to add annotations and notes.

Created a slide for the meeting, added concerns (time it is taking me to get through papers, clash with other coursework's, initial aims are fast approaching, effects of postponing the interim meeting)

Supervisor's Comments:

Add objectives for the next week to slides for next time and going forward.

Don't need to go too deep into morph evolution, focus on what others are doing.

Either aim literature review for around a month's time, or work in parallel with starting to think about experiments and practical work – don't wait for literature review to be finished before starting working on the next part of project.
Shouldn't be a problem postponing Interim Meeting to later in the trimester.

Simon will setup a meeting with Leni to discuss datasets – features/creation to help understand it etc.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 01/11/2024 **Last diary date:** 26/10/2024

Objectives:

Continue to work through sources and populate the topic board.

Progress:

Continued going through sources in more detail – though this is extremely slow going.

I have created a topic board/map in which each of my intended sections in the literature review are columns, and each cell is a topic/talking point to form paragraphs. The intention is to drop quotes or even just the reference to sources that relate to each of these topics hopefully making writing faster and more efficient.

I have also considered major tasks for the project and potential challenges which may take time. These were added to the meeting slide for discussion with Simon.

Leni joined the meeting so we could discuss the dataset, the one I currently have is a sparse matrix without skeleton information. He is going to try to produce a dataset which includes the skeleton information along with a description of its format and potentially also investigate how we can visualise the robots from these matrices.

Supervisor's Comments:

Happy with the topic board, however there is a lot of detail in there. Probably can prune some topics if finding it too much, won't need to mention everything.

We discussed what the experiments might look like from the project major tasks I had identified. For trialling optimisers, probably can just stick with using Adam as it has been a go-to in the ML field for at least the past 5 years. Convolution might work, might want to look at 3D convolution. Can cut off hyperparameter tuning, the VAE won't have to be optimal, just good enough to see that the robots can be encoded and the VAE can be used, its more important to address the clustering and diversity aspect. For clustering, stick to one or two, probably start with t-SNE. We will probably want to do graphical analysis looking at distances. Will want to first plot a lot of robots and then work out how we can quantify these.

Don't wait for the final dataset to be ready to test things can be run ok on the HPC, however for now, continue working on the literature review.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 08/11/2024 **Last diary date:** 01/11/2024

Objectives:

Aim to complete a section of the literature review each week. Not sure how realistic this is currently, especially with other coursework pressures, so review progress each week along with implications of doing it slower.

Run a test on the HPC to check can run programs ok.

Contact Leni either this week or next to request an update about dataset (might be better to give him another week before requesting update).

At some point (doesn't have to be this week, but sooner rather than later), send an email to Simon and Peter to set a date for the interim meeting.

Progress:

Topic board was not as affective as I imagined and seemed to just be adding more work, so this has now been scrapped. Sticking to highlighting and annotating in Mendeley.

Started writing literature review, slow, but making progress.

Supervisor's Comments:

Happy with progress.

Discussed interim, Simon is fine with this being towards the end of the trimester as there will be more to review and give feedback on. Though ensure to send an email out about this sooner rather than later to set a date before it's too late.

Probably fine to contact Leni about the dataset, though fine to give him another week before getting an update.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 15/11/2024 **Last diary date:** 08/11/2024

Objectives:

Aim to complete a section of the literature review each week. Review next week to check progress.

Contact Leni either this week or next to request an update about dataset, Cc Simon.

At some point (doesn't have to be this week, but sooner rather than later), send an email to Simon and Peter to set a date for the interim meeting.

Progress:

Unfortunately, I did not manage to get much done this week due to coursework difficulties which hindered working on this project.

However, I did manage to get programs running on the HPC (after a lot of faffing) due to the compute nodes not having an internet connection. Certain libraries required downloading and then moving to the HPC, this also included pretrained models such as ResNet50.

Supervisor's Comments:

Although not much progress was achieved this week, Simon is still happy with the progress so far (especially now that I can run things on the HPC).

We can stop meetings for a while until coursework pressures has calmed down. If I have anything specific to discuss, I can email him to arrange a meeting.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 06/12/2024 **Last diary date:** 15/11/2024

Objectives:

Try to make any progress if coursework and exam pressures allows.

Progress:

Emailed Leni about the dataset and spoke to Simon about it during the meeting.
Spoke to Simon about the questions I have for Leni regarding the dataset structure.

Emailed Peter (2nd Marker) and Simon about postponing interim meetings until next trimester – both are ok with this.

Coursework has been heavily dominating my time, and so no progress has been made for this project.

Supervisor's Comments:

Spoke to Leni, he has nearly finished the code for visualising the robot from the dataset, but no ETA, will ask today.

The dataset is generated from a CPPN output which is strings of data that is then split into different streams, 1; to build the robot in the simulator, and 2; for encoding the robot to extract features (macro numbers). There is no way to go between as are not the same processes. It will probably not be possible to do visualisation from the encoding values.

When time allows to continue work on the project, prioritise working on the code, will have to work on lit review / written report in parallel where possible.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 13/12/2024 **Last diary date:** 06/12/2024

Objectives:

Try to make any progress if possible (though still working on coursework).

Progress:

Leni has sent me the dataset, this is of descriptor values rather than encoding values, and does not contain the skeleton, only the key components (0 = nothing, 1 = wheel, 2 = joint, 3 = caster, 4 = sensor). This should be fine for the project.

Leni has also sent me a small program for visualising a single robot using matplotlib. I adapted this slightly by adding a legend and allowing a robot to be visualised by its ID rather than its index in the CSV file.

Supervisor's Comments:

No meeting.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 03/01/2025 **Last diary date:** 13/12/2024

Objectives:

Continue to build pipeline.

Will need to go back to working on lit review and written report but prioritise getting codebase setup to start training.

Progress:

Started setting up the pipeline –

- Set up project as a python package.
- Config file for easy access to training/model/hyperparameter settings.
- Combine all available dataset csv files into a single data frame and generate unique ID for each robot.
- Split combined data frame into train, validate, test sets and save as CSV files.
- Create data loaders.
- Adapted robot visualisation to visualise from either a file or a torch tensor of grid data.
- VAE model architecture – decoder output from a sigmoid output, VAE outputs both decoder output and reconstructed output (which is not differentiable due to scaling, rounding and clamping operations).
- VAE loss function (reconstruction loss and KL divergence) which uses the decoder output.
- Metric functions (prediction table, accuracy, recall, precision, F1 score) which use the reconstructed output.
- Single train loop (one epoch).

Supervisor's Comments:

No meeting.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 10/01/2025 **Last diary date:** 03/01/2025

Objectives:

Build plotting functions to analyse models once trained.

Will need to go back to working on lit review and written report but prioritise getting codebase setup to start training.

Progress:

Continued building pipeline –

- Added checkpointing functionality to save and load models, optimizers and schedulers.
- Built a TrainingHistory class to track history and metrics, saves, loads and rolls back. This allows history to be loaded even if the checkpoint doesn't exist or the model architecture has changed.
- Test loop (one epoch), default prunes old checkpoints when improvement in average weighted F1 score or total loss to save memory.
- Train_val, full training cycle, checkpointed when theres an improvement in average weighted F1 score or total loss, as well as the final model (either from reaching the maximum epoch, or terminating via early stopping).
- Grid search to define training configurations to test and then search for best performing based on a score calculated by balancing loss and weighted F1 score by a defined trade-off score.
- Added timing of training loops to output an estimated completion time.
- Function to summarise dataset – found some datasets contained just 0's (no robot exists providing no information), and some robots were duplicated. Found 0's makes up around 99.5% of the dataset (extremely sparse).
- Function to clean data – remove rows containing only 0's and remove duplicate rows. This brought down the total number of samples from 280375 to 27017. When split into train / val / test sets, produced sizes 18911 / 2702 / 5404 respectfully.
- Added class weighting to loss function to account for sparsity of data descriptor values (lots of 0's).

Run on HPC –

- Job script added.
- Code tested and adjusted to run on HPC using GPU.
- Started training a base model for comparisons later.

Supervisor's Comments:

No meeting.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 17/01/2025 **Last diary date:** 10/01/2025

Objectives:

Continue to develop model and write as much of the literature review as possible.

Progress:

Developed plotting functions –

- Plot metrics – Plots defined metric over epochs, can plot multiple metrics from multiple training histories on the same plot.
- Plot loss trade offs – Plots either KL divergence vs reconstruction loss, beta scaled KL divergence vs reconstruction loss, or total loss vs weighted F1 average.
- Adapted robot visualisation to save image.
- Functionality to analyse latent space – includes functions to sample from the latent space, train both PCA and UMAP dimensionality reduction models, perform kmeans clustering, plot the latent space clustering from both models, calculate the silhouette score, and calculate the Euclidean pairwise distance. There is also functionality to use the elbow method to find the optimal K value.
- Added function to visualise and compare original robot against the reconstructed robot.

Training –

- Finished training base model.
- Generated various plots for evaluating.
- Analysed latent space and generated plots for the model with the best total loss, the best weighted F1 score, and the best performing model using a trade-off loss/F1 value of 0.7. Generated robot comparisons for 3 samples from the 3 models.
- Noticed minority descriptor values (1-4) were being heavily predicted in the reconstruction. Changed class weighting to use logarithmic scaling to help give minority classes a boost so the model pays more attention to them.
- Started retraining base configuration with only the logarithmic scaled class weighting change.
- Created toy train and val datasets (20% of original) to reduce training time.

Continued writing literature review.

Supervisor's Comments:

Meeting postponed until Tuesday.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 24/01/2025 **Last diary date:** 17/01/2025

Objectives:

Continue to develop model and write as much of the literature review as possible (priority).
Trial model with latent dimension of 8.
Look into potential other loss functions and maybe cut down to using just one.
Look into using workers and Nvidia SMI to see how much of the GPU is being used.

Email Peter and cc Simon to arrange Interim.

Progress:

Training –

- Reconstructed output much better using logarithmic scaled class weighting, and much faster using toy dataset.
- Trained model with 3D convolutional layers which seems to have better latent space clustering.
- Tried adding pooling layers but appears to loose spatial information.
- Removed rounding and clamping operations from VAE (was originally returning two different outputs from the VAE due to these operations being non-differentiable). These operations are still carried out outside of the VAE when creating a prediction table and when visualising for comparison.
- Adjusted plotting function to avoid the potential for train/validate date for the same metric to be plotted against different scales.
- Values are normalised for BCE, but also for other losses to make them comparable in the grid search.
- Retrained convolution model without pooling layers to gain comparative results going forward.
- Added convolutional skip connection in the hope of capturing additional features. Trained models with the skip connection in both encoder and decoder, and just in encoder.

Continued writing literature review.

Supervisor's Comments:

1st Meeting back was held on Tuesday.

Prototype seems to be working even if latent space not so good yet.
Aim to have good latent space over good reconstruction, however, to know if the latent representation makes sense, need to be able to see that using reconstructed output.
The data we have (11x11x11) is not large compared to normal images or VAE's.
Maybe use just one loss function to reduce training time.
Trial around 8 for the latent dimension – looking for the latent space to resemble the features of

the robot (number / position of objects). There are maybe other entropy / loss functions that are better suited to this task. Look up transfer of patterns and what loss is used in GAN's, along with what loss is used in graph networks. Can also set workers when running on the HPC to run faster, look into Nvidia SMI to see how much of GPU using, want around 98-99%. Need to setup interim meeting, email Peter and cc Simon. On track but trying to do too many things. Don't go too deep in literature review as lots to cover.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 31/01/2025 **Last diary date:** 24/01/2025

Objectives:

Continue to develop model and complete literature review as soon as possible.
Start to consider or move onto latent space exploration and measuring.

Prepare for interim meeting.

Progress:

Training –

- Model without skip connection seems to perform better.
- Started to train deeper model with 10 trainable layers with using LeekyReLU.
- Changed gird search parameters, now trialling with latent dim 4 and 8, and beta of 0.1 and 0.5
- Haven't trained much this week, as focus has been on the literature review.

Nearly completed writing literature review.

Scheduled interim meeting for Tuesday – Have sent over the repository link to the code and diary entries to Peter and Simon. Sending literature review later today regardless of if it is fully completed or not.

Supervisor's Comments:

Even if the model trained isn't great, just need a model where the latent space could make sense. Continue onto next part, measuring. May want to consider using sparse matrix data representation if time.

Going to send more data from recent experiments where robots have been generated from five different terrains (e.g. hill, rough, corridor). For each terrain there has been 20 runs.
Take the best (maybe 20%) robots from each terrain and pass them through the encoder. Look at how latent vectors are different – generate UMAP. Quantify area of robots in UMAP, larger equals more diverse. Check for overlapping – generate circles around each terrain.

Low priority, however, would be nice to know the difference between the clusters in UMAP plot. Have a look at UMAP documentation to see if its possible to display the ID of each robot along with number of wheels, sensors etc.

Prepare for the interim meeting.

Going to keep Friday presentation meetings but also adding a 1-1 meeting on Wednesdays.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 07/02/2025 **Last diary date:** 31/01/2025

Objectives:

Continue to develop mode, deadline to have something reasonable working by next week.
Start to consider or move onto latent space exploration and measuring.

If model isn't working next week, need to discuss alternative directions for project.

Discuss evaluation next week.

Progress:

Training –

- Deep model overfitting though appears to be leveraging larger latent space.
- Added batch norm layers in an attempt to stabilise training, though still overfitting and nothing in the reconstruction.
- Setup sparse representation, data now represented as an 8x8 grid, max non-zero voxels in the dataset is 8, by (x, y, z, one-hot descriptors). The order is shuffled to avoid positional bias.
- F1 score now does not consider coordinates (due to the way the data is encoded), so loss is a better indicator which does consider coordinates and descriptors.
- Sparse-to-dense function to convert back to full grid.
- Class weighting removed, no longer required now using sparse representation.
- Changed model to PointNet inspired model (simplified version using fully connected layers).
- Added alpha value to balance descriptor loss and coordinate loss.
- Added penalty for duplicate coordinates adjusted for padded voxels in the original data.

Completed writing literature review (full draft).
Attended interim meeting.

Supervisor's Comments:

Both Simon and Peter seem happy with progress and the overall plan.
Discussed timeframe and contingency plan should the model still struggle to learn.
Set a deadline for next week to get the model working reasonably and then should move on to the latent space analysis or decide new direction for project.
2-3 weeks should probably be the deadline for implementation and experiments to allow time to focus on writing.

Recommends one-hot encoding for sparse matrix

Evolution experiments currently running where robots are being trained for different terrains. The best 20% of these should provide a good dataset for evaluating.

Version 2 Edinburgh Napier University

When evaluating the latent space, look at measuring stats such as mean and standard deviation (assume Gaussian). Will discuss further next week.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 14/02/2025 **Last diary date:** 07/02/2025

Objectives:

Finish model training ASAP and move onto evaluation.

Let Simon know when training is complete and put loss functions into simple terms to communicate them.

Progress:

Training –

- Redefined model architecture to be more inline with PointNet (1D Conv layers and BatchNorm). Major changes made to entire codebase to accommodate. Instant improvement in what is being reconstructed, though voxels get lost in the reconstructed descriptor space and only a single voxel being produced. This voxel is moved around slightly but appears to be drawn to a specific location. Descriptor value is changed, though model seems to prefer blue (wheels).
- Applying transformation matrix to original input coordinates before calculating loss, thinking behind it being it will then match what the model is trying to learn.
- Using a regularising term for transformation matrix to encourage orthogonality - based on https://medium.com/@itberrios6/point-net-for-classification-968ca64c57a9
- Now normalising sampled latent vectors for latent space analysis.
- Enhanced visualisation function to include transformed descriptor space alongside the current original and reconstructed.
- Experimented with alpha, penalties and beta values, along with other hyperparameters.
- Transformation matrix seems overly aggressive and collapsing the structure into near 2D and disregarding some voxels. Voxels are getting lost in the transformation process, further to the model's reconstruction losing voxels. Removed application of transformation to original input prior to calculating the loss. Network was possibly cheating by adjusting the transformation model.
- Trialled longer training runs (task is complex and multi-objective), the model managed to produce 2 or 3 voxels constructed in various ways, but they tended to remain in a specific corner of the space.
- Inspected the reconstructed output closer and found the coordinates are not duplicates but are very close. Meaning when the sparse grid is converted back to a dense one for visualisations, the coordinates are collapsing to duplicates. The way the coordinate loss is calculated also assumes a 1-to-1 correspondence, which is not the case due to shuffling of the sparse representation matrix. Further to this, the way the duplicate and pad penalty is calculated and applied is non-differentiable and so the model isn't learning directly. The descriptor loss is also being affected by the shuffling; CrossEntropyLoss is order dependant. The shuffling is best to try to maintain to avoid a positional bias.
- Added custom loss/penalty functions to overcome the issues (which were a headache to put together), particularly the shuffled voxels. Kept discovering problems and correcting them after initiating a training run.
- Added a coordinate matching loss, loosely inspired by the Hungarian algorithm and

utilising information theory. It consists of two parts; the first part penalises for multiple reconstructed points clustering around a single original point. This is done by using cdist to calculate a distance matrix which calculates pairwise distance from reconstructed coordinates to original coordinates (rows represent reconstructed, columns are original). Padded voxels are masked out (based on descriptor designation) to ignore the corresponding coordinates in the penalty calculation. Applies softmin (differentiable) to work out nearest neighbour probability - also scales the distance matrix to encourage points to focus on a single point. Takes the sum of the columns to calculate the attention each original point receives. This is then normalised to create a probability distribution over the original points. Clustered points will have low entropy, and perfect alignment has max entropy as perfect alignment gives each original point equal attention – uniform distribution. The clustering penalty is calculated by taking the entropy from max entropy. The second part calculates a distance penalty by scaling the neighbour probability (likelihood) by the masked distance matrix. This does encourage the model to under predict, which is offset by the padded penalty.

- Added a padding penalty which uses softmax (differentiable) to get descriptor class probabilities. Smooth L1 (differentiable everywhere) is used to calculate the difference between original padded probabilities and the reconstructed padded probabilities and takes the mean reduction as penalty. Calculates the difference in padded voxels and penalises stronger when more reconstructed padded voxels than original to balance coordinate matching loss.
- Added overlap penalty to stop normalised coordinates collapsing to a single point when scaled up to the full grid space. Coordinates are scaled up to the grid space range, then the padded coordinates are masked. This scaled up matrix is used to find matching coordinates (overlaps), and their indices are then used to create a cdist distance matrix. The penalty is calculated as the negative sum of the log of the upper triangle not including the diagonal (self). Log is used to allow the penalty to scale as coordinates get closer.
- Added a descriptor matching loss. Calculates the pairwise distance between original and reconstructed points. Padded voxels are then masked, before obtaining the most likely descriptor value based on nearest original voxel. Loss is calculated by taking the cross entropy (model must output raw logits).
- Initial training runs look much more promising. Though haven't had the time to properly tune due to finding issues in the loss/penalty functions. Had to pretty much start from scratch on the hyperparameter tuning.

Supervisor's Comments:

Evolution experiments that would have been good to use as an evaluation dataset has failed.
Need to discuss and consider a new evaluation approach.
- May need to be manual evaluation and comparing the robots that the model finds differences in.

Let Simon know when training is finished and put loss functions into simple terms.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 21/02/2025 **Last diary date:** 14/02/2025

Objectives:

Finish plots and evaluation work.
Focus fully on the writing.

Look at component visualisations and see if there is any obvious overlaps between the datasets.

See supervisor comments below for more details:
Plot PCA has eigenvectors.
Look at robots to reason overlapping.
Get Euclidean distance between each centre of mass.
Plot bounding area and get blob area and overlap.
Look for features in the actual robots to try to help explain the plots.

Plot each dataset so each set is still in colour and the rest are in grey.

Only if time – towards end of project:
Visualise what changes when varying one latent space dimension (look into ipywidgets for sliders).

Progress:

Training –

- Model was still clustering and had difficulties balancing the loss functions and penalties. Suspect bias in the losses.
- Simplified by encoding padded voxels as (11,11,11) instead of (0,0,0) – a genuine coordinate. Then instead of masking out padded voxels based on descriptor values, padded coordinates formed part of the calculation based on entropy. This means the padded penalty could be removed as the coordinate matching loss function should push coordinates to the correct number and location – perfect alignment has max entropy (uniform distribution).
- Get some spatial and voxel variation, seems to be an improvement, but descriptor loss is flatlining – poor learning.
- Experimented with ranging scaling values for the losses and penalties, but descriptor loss still seems to have poor learning. Suspected collapse loss is potentially overpowering spatial learning, setup experiments with it set to 0 scaling (entropy-based coordinate matching should take care of collapses and pull them apart).
- Found when coordinate scaling is high, coordinate loss drops effectively, but also results in limited spatial or colour variation, potentially falling into local minima. Seems to require sufficient descriptor balance. Without collapse penalty, spatial alignment seems to be more effective, though still suffers from clustering (so setting to 1.0 – lower value).
- Ran a full dataset run to see if more data would help learning, though it has extremely limited number of voxels being reconstructed and layout variation. Then ran a longer

full dataset run with higher coordinate lambda and lower collapse, still limited variation.

- Realised a potential why descriptor loss may be flat, not passing any coordinate information back to the descriptors in the model, just concatenating the tensors rather than combining additively or multiplicatively. Assuming the descriptors would benefit from spatial information due to the nearest neighbour being used for comparison in the loss. Coordinates may also benefit from descriptor information (if e.g. components lower down are more likely to be wheels, higher up may be sensors).
- Added another 1D conv layer to encoder for the descriptor mlp to make it the same shape as coordinate mlp output and replace the concatenation with additive combination followed by LeakyReLU activation. Changed linear steps in refining fc layer to be smaller and smoother, along with an intermediary layer in the decoder to mirror before upsampling. Quick runs showed that descriptor emphasis performed the best, encouraging better spatial awareness, though still limited.
- Tested extreme lambdas, minimal variation all round. Suspect when descriptor loss higher than coordinate loss, falling into local minima.
- Explored higher coordinate scaling, descriptor loss remains fairly flat. Coordinate loss drops well until below descriptor loss where it seems to stabilise, followed by beta kl starting to rise when it crosses. Potential high coordinate loss is pushing model to the extreme.
- Trialled higher descriptor scaling and low coordinate scaling, which produced more variation in layout, moderate variation in voxel numbers, and some variation in colour (descriptor type). However, descriptor loss still struggles to learn.
- Tried adding a cross-attention mechanism to allow the coordinate representations to selectively attend to descriptor information. Thinking was so coordinates can learn to focus on the most relevant descriptor features (using coordinates features as key, descriptor features as keys and values). This led to extremely static layouts.
- Attempted reversing the cross-attention query and keys/values – using descriptor features as query, coordinate as keys/values. Slight improvement in layout, but negligible, still fairly static. Suspecting an issue with the losses.
- Inspecting the gradients, it looks like much higher lambda values are required. Tried a lot of quick runs and modifications, though seemed to be making the model worse. Discovered some values were going to nan. Reducing learning rate seemed to stop this (for short runs), though suspect deeper problem, but no time to investigate. Added gradient clipping as some safety.
- Added penalty in coordinate matching function for difference in number of padded voxels to original based on coordinates rather than descriptors. Hoping to have it help overcome the entropy local minima.
- Trained on full dataset for much longer with a learning rate scheduler – early stopping triggered after 183 epochs.
- Reverted to best model prior to attention mechanism, which combines the pooled outputs additively. Training with same configuration for comparison. Performance much better, variability in voxel numbers (3 and 4), spatial layout and colour. Though not perfect, this should be good enough to move forward to evaluation.

Evaluation -

- Added functionality to group test data into diverse sets – 3 based on component dominance, and 3 based on spatial variability.
- Started working on plotting the datasets by passing them through the encoder and extracting the mean vector (central tendency of the distribution – sampled latent vector would introduce noise with epsilon). Plotting with centre of masses for each dataset.

Supervisor's Comments:

Quick chat on Tuesday: Discussed plotting options for evaluation, starting with the mean vectors should be a good start.

Find centre of masses, get distance between.

Thursday Meeting: Results look good, may not be as sharp as we would like, but we are seeing clusters of robots in the spatial plot that we would expect to cluster together. Can see separation between the spatial datasets in UMAP plot.
Component plot is harder to find separation.

Look at component visualisations and see if there is any obvious overlaps between the datasets.

PCA plot points are more spread around. PCA has eigenvectors (2 of) which give directional information. Look into how to plot those with the same colours as the datasets and see if they give us any more information.

With the UMAP, the green and brown sets overlap slightly, look at shape and form in the datasets and see if you can work out any reasoning as to how and why.

Get some quantitative measures from the plots – Euclidean distance between each centre of mass.
SciPy or Open CV should have tools to help with bounding area and get the blob area.
We want 2 or 3 statistical measures (area, overlap, distance).
Look for features in the actual robots to try to help explain these.

Plot each dataset so each set is still in colour and the rest are in grey in addition to current plots.

Only if time:
Get the latent space of a specific robot, modify 1 dimension at a time (high and low) and pass both the original and the modified through the decoder to see if you can identify what changes (this may be multiple elements). Look into ipywidgets to use a slider to interact with the VAE and change dimension values.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 28/02/2025 **Last diary date:** 21/02/2025

Objectives:

Focus fully on the writing – aim for minimum 2.5k-3k per week.

Progress:

Evaluation -

- Reworked spatial dataset grouping criteria. Now taking the box volume and normalising by the total grid space volume. Taking the mean distance to the sample centroid – calculated by taking the mean coordinate - and normalising by the grid space diagonal. The normalised volume and normalised distanced are added together to produce the spatial score, samples are then grouped based on thresholds. This seems to produce better representative datasets.
- Attempted to add ellipses to plots, though this is proving to be extremely difficult. OpenCV does not have good documentation, it does however have a function to fit an ellipse to data. This returns the ellipse as major axis and minor axis, but matplotlib ellipse (to draw the ellipse) takes width and height, so cannot identify the orientation. Furthermore, the ellipse produced does not look like it is the correct size in any orientation. Further investigation required if time – though this has taken up a lot of time already.
- Added Euclidean distances to plot legend between dataset centre of masses.
- Plotting each dataset in colour with the others grey.

Supervisor's Comments:

No meeting this week – email update.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 07/03/2025 **Last diary date:** 28/02/2025

Objectives:

Focus fully on the writing.
Finish methodology as a minimum

Progress:

Writing –

- Writing methodology

Training –

- Noticed part (if not the) problem with my losses, entropy only encourages reconstructed coordinates to align to a single original coordinate but not match exactly spatially. So long as each original coordinate had attention from one reconstructed voxel (ie no clustering), then the penalty could be zero. Kept the entropy loss to discourage clustering to a single original voxel, added a MSE loss based on nearest neighbour and removed padded voxel difference penalty (as entropy penalty should encourage the correct number of voxels).
- Trained full dataset using previous best hyperparameters – best performing model to date, reconstructions are much, much better. Started some quick tuning runs for lambda values.
- Training full dataset runs with best quick tune run hyperparameter settings with different beta values for evaluation.

Supervisor's Comments:

Happy with ellipses as you had them even if they don't fully encompass the data.
If time, see if can get the overlap areas.
Can see some ellipses are similar, see if this similarity can be identified in the datasets.

Put plots in white for report

Look again at PCA eigenvectors, see if can normalise on full dataset, but create different pca models for each dataset to get dataset eigenvectors. Then plot these together.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 14/03/2025 **Last diary date:** 07/03/2025

Objectives:

Focus fully on the writing.
Aim to have as much of the report completed as possible.

Progress:

Writing –

- Completed methodology in draft.

Training –

- Noticed part applying LeakyReLU as final step of decoder to descriptor features. This could lead to negative values, and unbounded upper values when the values they represent are one-hot encoded. Though the model appears to work currently, retrained removing LeakyReLU, and instead applying softmax, though no time to tune other hyperparameters. Reconstruction quality remained good, but loss values and silhouette scores worsened compared to previous model, so reverted to previous model that had been tuned.

Supervisor's Comments:

Put plots in white for report.
Change background colour for architecture plots to white.

Good that you noticed activation function bug, and fine to stick with previous model that's been tuned.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 21/03/2025 **Last diary date:** 14/03/2025

Objectives:

Focus fully on the writing.
Aim to have as much of the report completed as possible.

Progress:

Writing –

- Managed to change all plots to have white background – had to regenerate all plots as inverting them produced messy, unusable plots.
- Changed background for VAE architecture diagram.

Code –

- Changed the way ellipses are generated – found some code https://github.com/minillinim/ellipsoid/blob/master/ellipsoid.py based on Khachiyan's algorithm to fit Minimum Volume Enclosing Ellipsoid (MVEE). Adapted this to work for 2D data. This works much better than the previous method, and gets the rotation correct. It is still an approximation method that fits an ellipse with a tolerance, but good for purpose.
- Computing areas and overlap using shapely. First need to construct a shapely ellipse (for geometric calculations) by using the centre, radii and rotation values calculated from MVEE. Shapely has built in methods to compute area and intersection.
- Unable to perform PCA or UMAP using models trained with beta = {1.5, 2.0} due to these produce static latent space vectors with no variance. Beta value has a huge influence on how meaningful the latent space is and how much information the VAE can capture.
- Added plots for latent space features. Plotting log variance vector against mean vector for 3 different robots for each dataset. Drawing ellipses around each dataset category.
- Changed evaluation dataset thresholds to create a gap between the datasets making them more distinct. Now using:

  Component:
  - Dominance = Must now all be same descriptor type.
  - Moderate = Any descriptor must have dominance between 55-65% (not inclusive)
  - Variety = All descriptors must have dominance less than 38%

  Spatial:
  - Compact = Spatial score less than 0.2
  - Moderate = Spatial score between 0.4 and 0.5
  - Dispersed = Spatial score greater than 1.0

  This produced good datasets for evaluation, verified by manual inspection. This does mean some samples do not fall into any dataset and therefore discarded. Though a good number of samples (187) is still collected for each dataset – using lowest number in any dataset to set maximum for all to make it fair and balanced.

Supervisor's Comments:

Add line to diagrams to say 'created in draw.io' to state you haven't copied them from somewhere. Diagrams all look good. Try to work out how they correlate with the data. Why do samples/datasets shown to be similar/different? Send each chapter as you complete it for reviewing.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 28/03/2025 **Last diary date:** 21/03/2025

Objectives:

Double check over dissertation, add abstract and appendices.
Complete poster.

Progress:

Writing –

- Completed dissertation main text in draft.
- Added note to figures to say which tools have been used to create them.

Code –

- Added some quick and messy code to generate PCA and UMAP comparison plots along with plotting decision boundary and overall centre of mass for all plotted datasets.

Supervisor's Comments:

Change contributions to bullet points in introduction.
Be careful of 'cherry picking' results, try to base discussion in observations, e.g. reconstruction loss was bad, so clustering is bad etc.

Quickly outlined important considerations for the poster.
People will interact in 2 ways; through you and questions, and through isolated interpretation.
Also start thinking about a 'lift pitch' for when people look at your poster and ask about it.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Andrew Taison **Supervisor:** Simon Smith

Date: 31/03/2025 **Last diary date:** 28/03/2025

Objectives:

Final proofreading and addressing any last-minute feedback regarding the poster or dissertation report.
Submit, relax... and if you've read all of this — I owe you a drink!

Progress:

Writing –

- Rewrote discussion section and made minor changes to other sections.
- Added appendices and self-appraisal.
- Added abstract.
- Addressed Simons feedback.
- Double checked word count and cut some sections out.
- Asked as many people as I could to proofread the dissertation.

Code –

- Added some more (quick) code to calculate and print Euclidean distances between selected points in the PCA / UMAP comparison plots. This allowed similar points to be identified to make it easier to decide which samples to look at to identify traits the model had encoded as similar.

Poster –

- Completed poster in draft.

Supervisor's Comments:

Email feedback:

- Too many details in lit review for EC.
- Put more explanation in image captions.
- Change 'Shannon entropy' to 'entropy'.

Appendix E: Gantt Chart

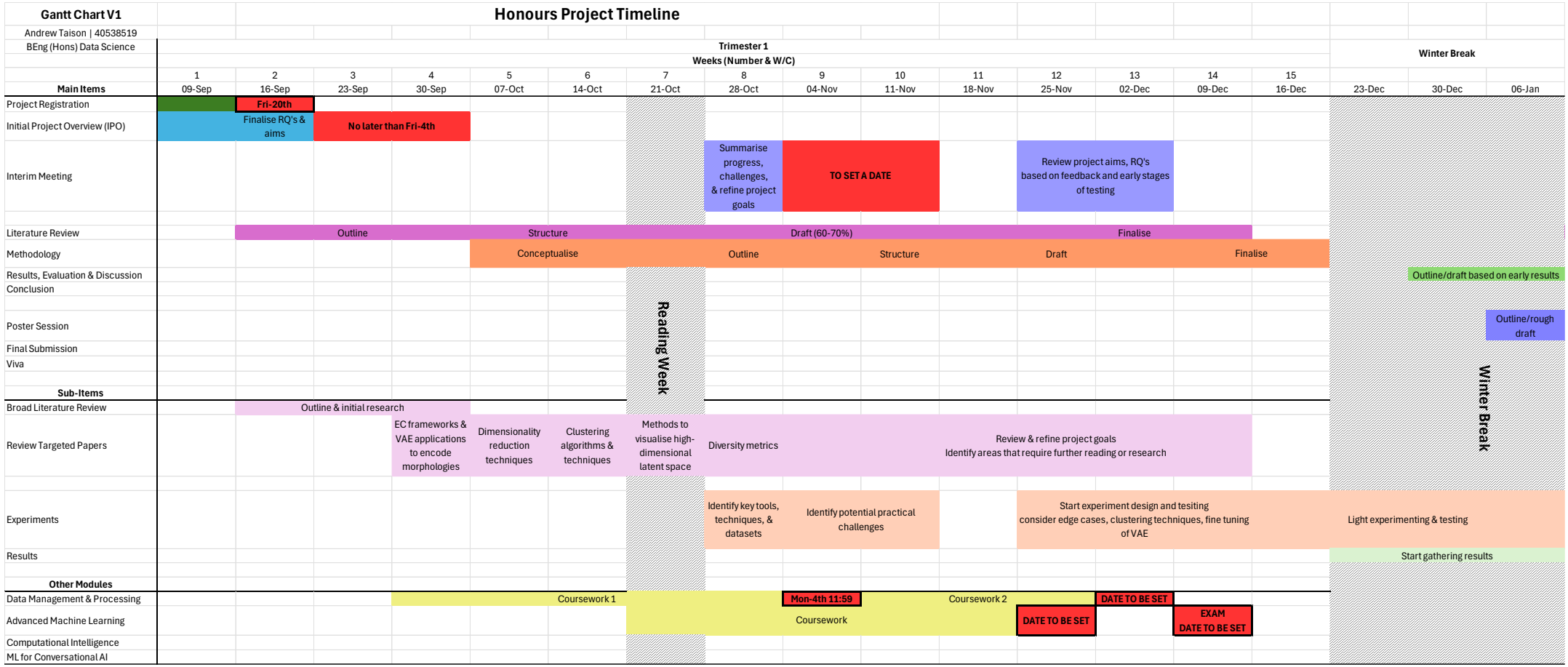


Figure E.60. *Gantt chart project plan, version 1, first trimester*
Note: Developed near the beginning of the project and was slightly ambitious.

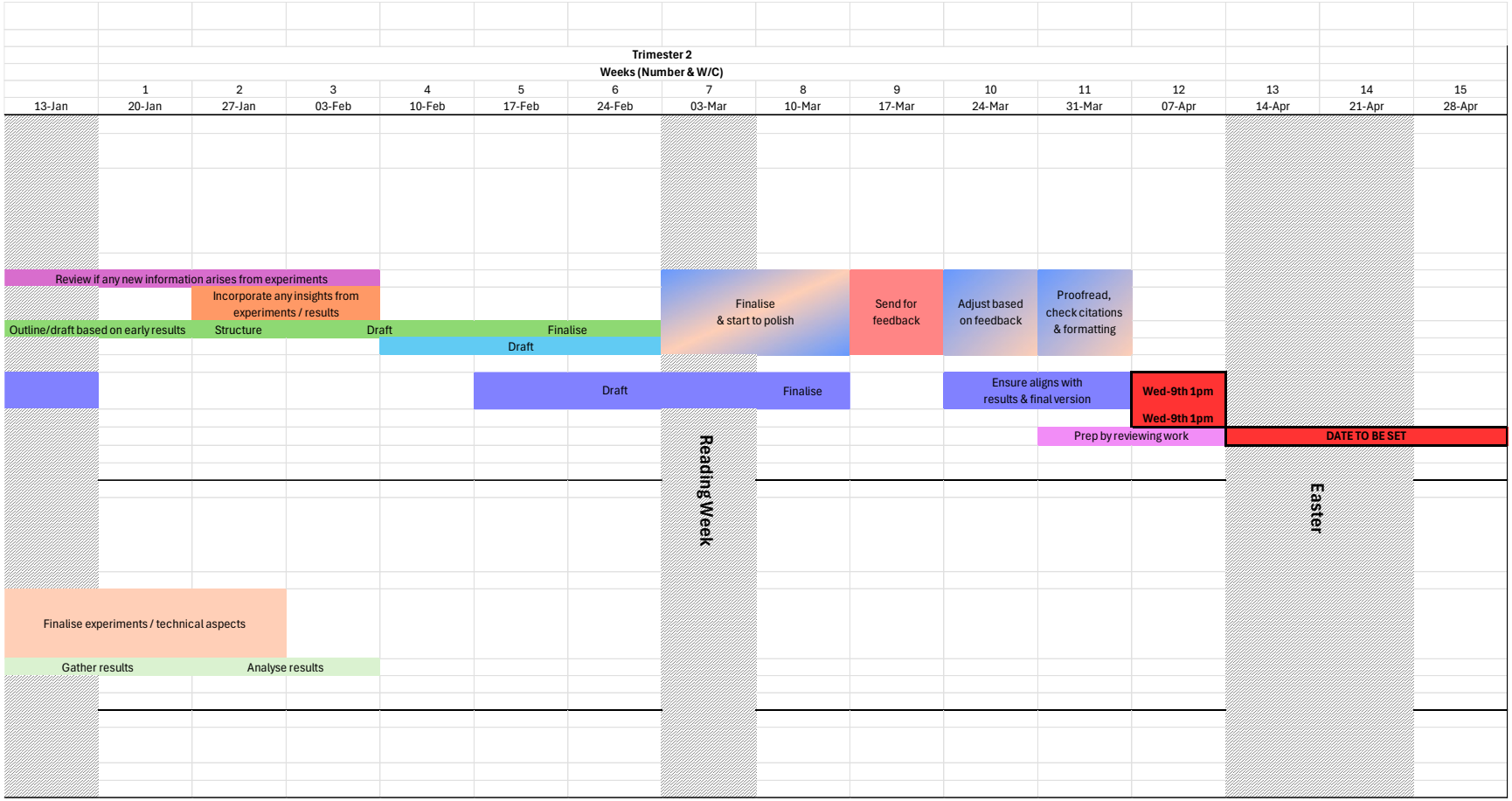


Figure E.61. *Gantt chart project plan, version 1, second trimester*
Note: Due to being so far behind the initial plan come the end of the Christmas break, this was replaced by version 2 (Figure E.62).

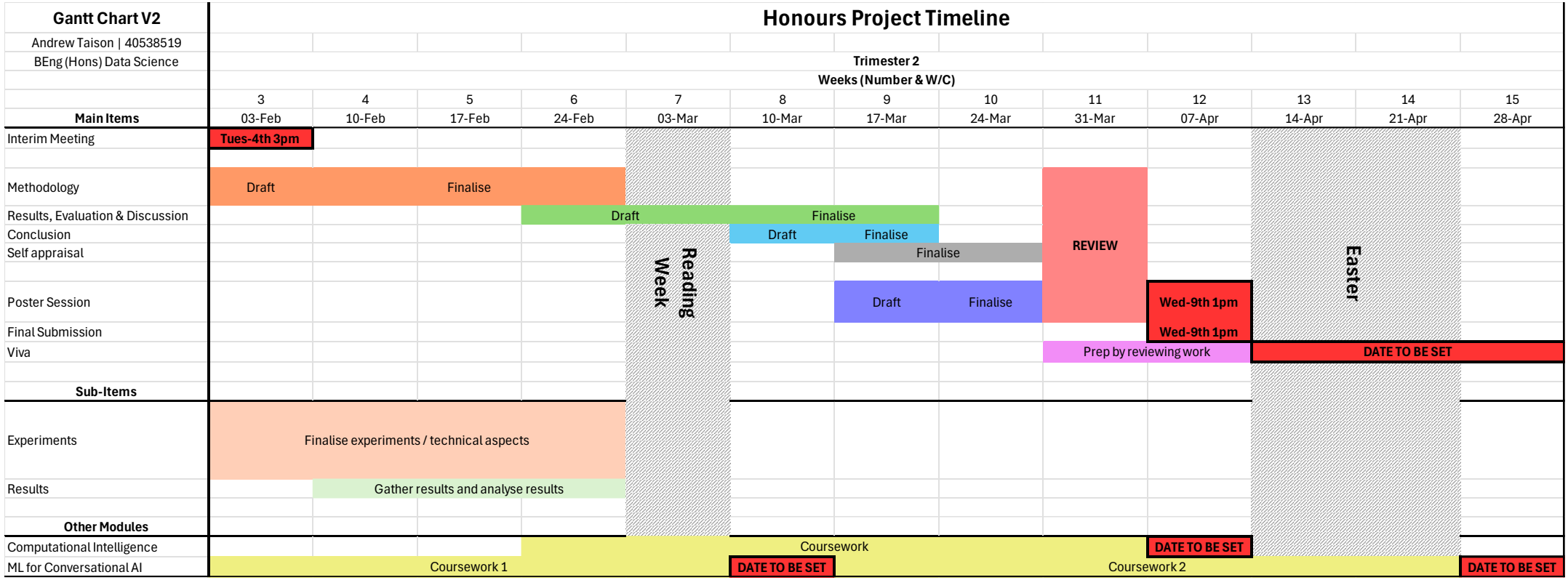


Figure E.62. *Gantt chart project plan, version 2, second trimester*
Note: Developed to replace Figure E.61. This plan was followed more closely.

Appendix F: Version Control and Project Management

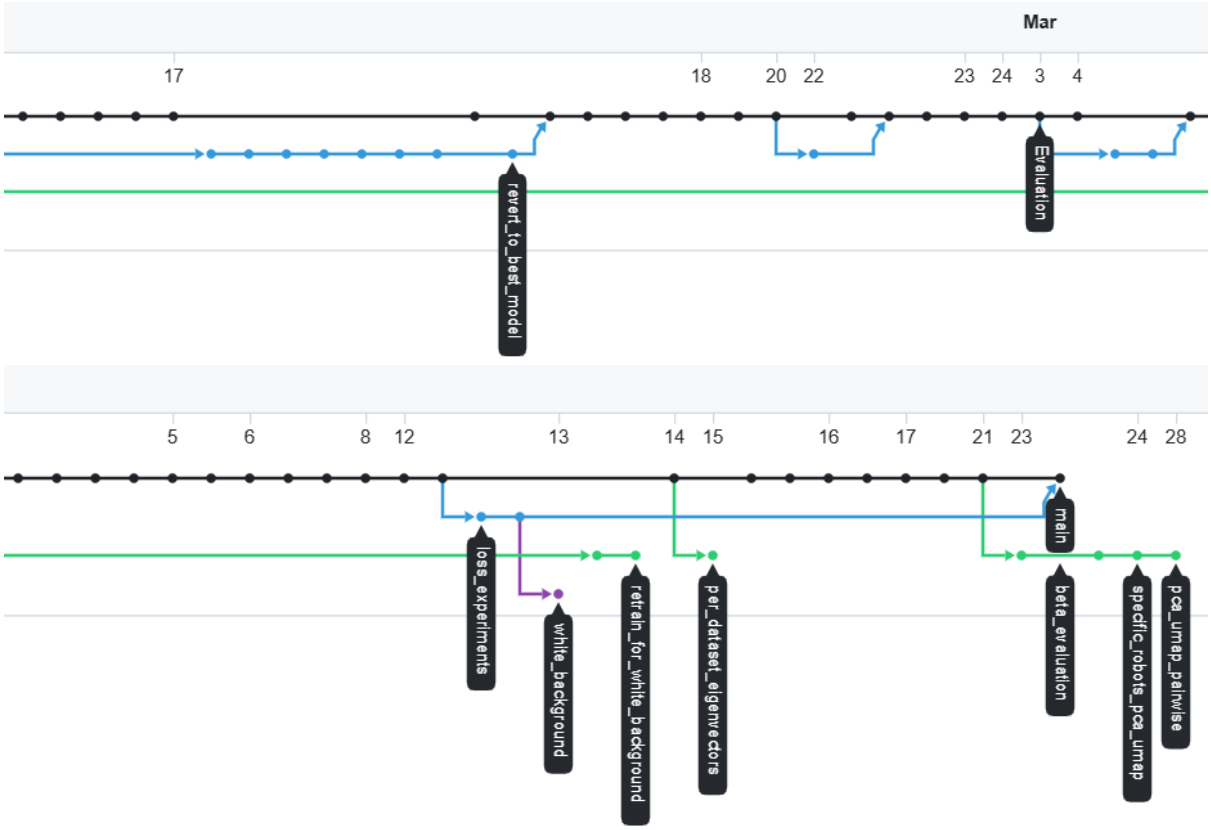


Figure F.63. *GitHub network graph (part)*
Note: Shows active use of multiple branches throughout the project, demonstrating version control.

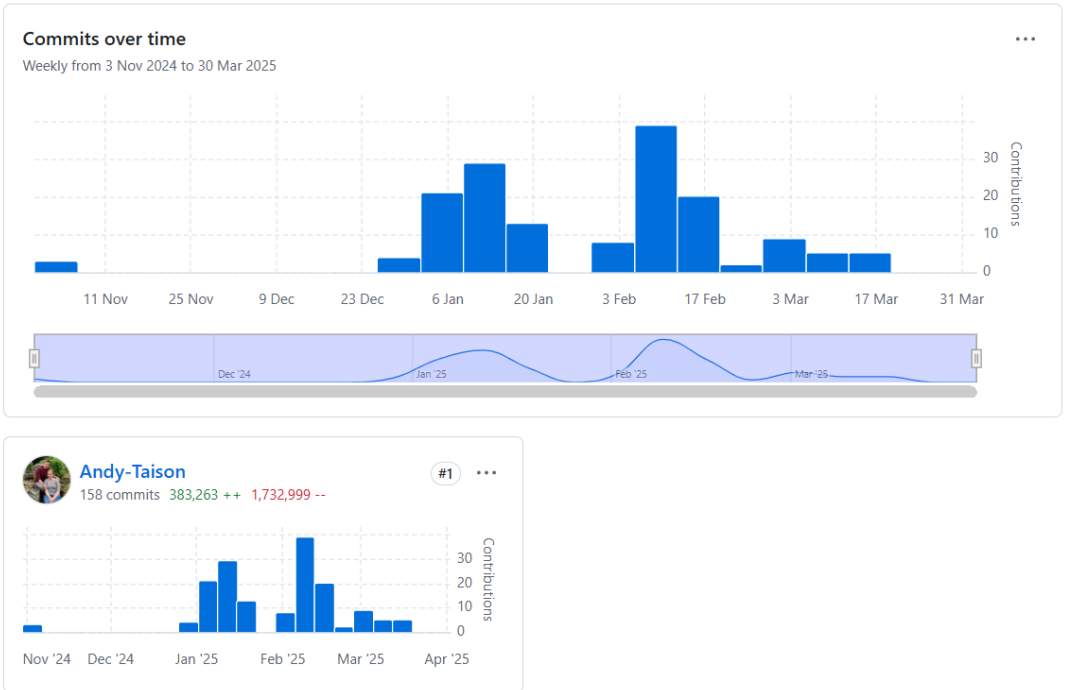


Figure F.64. *GitHub commit chart*
Note: Displays commit activity across the project timeline. Reflects regular progress and good project management practices.

Appendix G: Code Snippets

```python
class Sample(nn.Module):
    """
    Reparameterisation trick to allow backpropagation and sampling.
    """
    def forward(self, z_mean: torch.Tensor, z_log_var: torch.Tensor) -> torch.Tensor:
        """
        Forward pass.

        :param z_mean: Latent space mean with shape (batch_size, latent_dim)
        :param z_log_var: Log variance of latent space with shape (batch_size, latent_dim)
        :return: Sampled latent vector with shape (batch_size, latent_dim)
        """
        epsilon = torch.randn_like(z_mean)  # Samples from a standard normal distribution

        return z_mean + torch.exp(0.5 * z_log_var) * epsilon
```

Figure G.65. *Reparameterisation trick implementation*
Note: PyTorch-based function for sampling from the latent distribution. Implements the reparameterisation trick to enable backpropagation.

```python
kl_div = -0.5 * torch.sum(1 + z_log_var - z_mean.pow(2) - z_log_var.exp()) / x.shape[0]  # Averaged across batch size
```

Figure G.66. *KL divergence implementation*
Note: Computes the KL divergence between the learnt latent distribution and a standard normal distribution using PyTorch. The calculation is averaged across the batch.

```python
def compute_class_weights(batch_support: torch.Tensor) -> torch.Tensor:
    """
    Calculates class weights using batch support, used to account for
    class imbalance (descriptor values are sparse).

    :param batch_support: Batch support 1D tensor
    :return: Class weights tensor
    """
    # Inverses batch support to obtain weights
    weights = 1.0 / batch_support
    weights[torch.isinf(weights)] = 1  # Replaces inf with 1

    # Normalises
    weights /= weights.sum()

    return weights
```

Figure G.67. *Multiplicative inverse batch support class weights*
Note: This PyTorch implementation computes class weights using the multiplicative inverse of batch support, normalised across the batch. Infinite values caused by division by zero are replaced with one to ensure numerical stability.

```python
# Use logarithmic scaling to boost minority class weighting
weights = 1.0 / torch.log(batch_support + 1)
```

Figure G.68. *Log scaled class weights*
Note: Improvement on the linear class weights seen in Figure G.67, balancing loss severity with descriptor frequency more effectively.

```python
a = torch.tensor([0.863, 0.863, 0.863])
b = torch.tensor([0.96, 0.96, 0.96])

print(f"Tensor a: {a}")
print(f"Tensor b: {b}")

scaled_a = a * (EXPANDED_GRID_SIZE)
print(f"\nScaled a: {scaled_a}")
scaled_b = b * (EXPANDED_GRID_SIZE)
print(f"Scaled b: {scaled_b}")

rounded_a = torch.clamp(torch.round(scaled_a), min=0, max=EXPANDED_GRID_SIZE)
print(f"\nRounded and clamped a: {rounded_a}")
rounded_b = torch.clamp(torch.round(scaled_b), min=0, max=EXPANDED_GRID_SIZE)
print(f"Rounded and clamped b: {rounded_b}")

stacked = torch.stack((a,b))
print(f"\nStacked for cdist:\n{stacked}")
dist = torch.cdist(stacked, stacked)
print(f"\nPairwise distance between non-scaled a & b:\n{dist}")

log_dist = torch.log(dist + 1e-8)
print(f"\nlog(dist + 1e-8):\n{log_dist}")
```

```
Tensor a: tensor([0.8630, 0.8630, 0.8630])
Tensor b: tensor([0.9600, 0.9600, 0.9600])

Scaled a: tensor([9.4930, 9.4930, 9.4930])
Scaled b: tensor([10.5600, 10.5600, 10.5600])

Rounded and clamped a: tensor([9., 9., 9.])
Rounded and clamped b: tensor([11., 11., 11.])

Stacked for cdist:
tensor([[0.8630, 0.8630, 0.8630],
        [0.9600, 0.9600, 0.9600]])

Pairwise distance between non-scaled a & b:
tensor([[0.0000, 0.1680],
        [0.1680, 0.0000]])

log(dist + 1e-8):
tensor([[-18.4207,  -1.7837],
        [ -1.7837, -18.4207]])
```

Figure G.69. *Maximum log distance for collapsing voxels*
Note: Demonstrates that log distance values for collapsing voxels remain negative. A small constant $1e^{-8}$ is added to avoid undefined values from $log(0)$. The distance penalty is only calculated for voxels that would collapse to the same position. For demonstration, the distance between points that would scale to $(9, 9, 9)$ and $(11, 11, 11)$ is used – a diagonal separation beyond the collapse threshold. As coordinates are normalised, the resulting distances never reach or exceed 1, keeping the log negative and allowing its sign to be flipped for use as a penalty.

```python
def descriptor_metrics(x: torch.Tensor, x_reconstructed: torch.Tensor) -> tuple[float, torch.Tensor, torch.Tensor, torch.Tensor]:
    """
    Calculates metrics for descriptor values.

    :param x: Input tensor
    :param x_reconstructed: Reconstructed tensor, scaled to have original descriptor values
    :return: accuracy, recall, precision, f1
    """
    preds = torch.argmax(x_reconstructed[:, :, 3:], dim=-1)
    targets = torch.argmax(x[:, :, 3:], dim=-1)

    tp = (preds == targets) & (targets != 0)
    fp = (preds != targets) & (preds != 0)
    fn = (preds != targets) & (targets != 0)

    correct = (preds == targets).sum().item()
    total = preds.numel()
    accuracy = correct / total

    recall = tp.sum().item() / (tp.sum().item() + fn.sum())
    recall[torch.isnan(recall)] = 0  # Replace nan values from division by zero

    precision = tp.sum().item() / (tp.sum().item() + fp.sum())
    precision[torch.isnan(precision)] = 0  # Replace nan values from division by zero

    f1 = 2 * (precision * recall) / (precision + recall)
    f1[torch.isnan(f1)] = 0  # Replace nan values from division by zero

    return accuracy, recall, precision, f1
```

Figure G.70. *Sparse representation metric calculations implementation*
Note: This PyTorch implementation computes true positives, false positives, and false negatives by comparing one-hot encoded descriptor predictions and targets using argmax. After transitioning to the sparse format, these metrics became less informative as they did not account for coordinate mismatch, leading to a greater focus on reconstruction loss and visual analysis.

```python
"""
Coordinate metrics
"""
def euclidean_distance(x: torch.Tensor, x_reconstructed: torch.Tensor) -> float:
    """
    Calculates the average Euclidean distance between original input coordinates and reconstructed coordinates.

    :param x: Input tensor with shape (batch_size, *input_dim)
    :param x_reconstructed: Reconstructed input with shape (batch_size, *input_dim)
    :return: Mean Euclidean distance of reconstructed coordinates to original input coordinates
    """
    distance = torch.linalg.vector_norm(x[:, :, :3] - x_reconstructed[:, :, :3], dim=-1).mean().item()

    return distance
```

Figure G.71. *Sparse representation coordinates mean Euclidean distance implementation*
Note: PyTorch function used to calculate the average Euclidean distance between original and reconstructed voxel coordinates. This metric was used to estimate coordinate reconstruction accuracy. However, it assumes voxel order alignment between input and reconstruction, which is not guarenteed when inputs are shuffled. Due to this, its effectiveness was limited.

Appendix H: VAE and Custom Losses Full Implementation

losses.py

```python
"""
Defines loss functions
"""

import torch
import torch.nn.functional as F
from ..config import COORDINATE_DIMENSIONS, DEVICE, EXPANDED_GRID_SIZE


def coordinate_matching_loss(x: torch.Tensor, x_reconstructed: torch.Tensor) -> torch.Tensor:
    """
    Calculates a penalty based on spatial alignment between original and reconstructed coordinates.
    Penalty is the difference between max entropy (perfect alignment) and the entropy over
    the original point attention distribution. Padded voxels are encoded as a coordinate outside
    the expanded grid (EXPANDED_GRID_SIZE), and contribute to the loss.
    A distance based MSE loss is calculated between each reconstructed coordinate and its estimated
    original target coordinate. It is estimated due to the shuffling of original voxels and the sparse
    representation having no inherent ordering.

    :param x: Original input
    :param x_reconstructed: Reconstructed
    :return: Penalty averaged over batch
    """
    total_penalty = torch.tensor(0.0, device=DEVICE)

    orig_coords = x[:, :, :COORDINATE_DIMENSIONS]
    recon_coords = x_reconstructed[:, :, :COORDINATE_DIMENSIONS]

    for orig, recon in zip(orig_coords, recon_coords):

        # Pairwise distance matrix between all original and reconstructed coordinates
        dist_matrix = torch.cdist(recon, orig, p=2)  # (rows: distances from reconstructed, columns: distances from original)

        # Gets probability of the closest point, softmin is differentiable
        neighbour_weights = F.softmin(dist_matrix * 100, dim=1)  # Scaling distance encourages model to focus on a single point

        # Sum of attention each original point (columns) receives from all reconstructed points
        attention_per_original = neighbour_weights.sum(dim=0)  # (num_original_points)

        # Normalise attention to form a probability distribution over original points
        attention_distribution = attention_per_original / (attention_per_original.sum() + 1e-8)  # Avoid division by 0

        # Shannon entropy for attention distribution (amount of uncertainty) - clustered points will have lower entropy
        entropy = -(attention_distribution * torch.log(attention_distribution + 1e-8)).sum()  # log(0) is undefined

        # Maximum entropy (perfect alignment)
        # Happens when distribution is uniform: p(x) = 1/n
        # H(x)max = -sum_{i..N}((1/n)*log(1/n))
        # log(1/n) = -log(n)
        # H(x)max = log(n)
        max_entropy = torch.log(torch.tensor(attention_distribution.size(0), device=DEVICE))

        # Clustered points will have lower entropy
        entropy_penalty = (max_entropy - entropy) ** 2  # Avoid negatives

        # Matrix multiplication to get predicted coordinates
        matched_target_coord = torch.matmul(neighbour_weights, orig)  # (recon_voxels, coordinate_dim)

        # MSE loss between reconstructed coordinates and matched target coordinates
        loss = F.mse_loss(recon, matched_target_coord, reduction='mean')

        total_penalty += entropy_penalty + loss

    batch_penalty = total_penalty / x.size(0)  # Averaged over batch

    return batch_penalty


```

```python
def descriptor_matching_loss(x: torch.Tensor, x_reconstructed: torch.Tensor) -> torch.Tensor:
    """
    Calculates a loss for descriptor values based on nearest neighbour.
    Calculates the pairwise distance between original and reconstructed points.
    The most likely descriptor value based on nearest original voxel is then obtained.
    Loss is calculated by taking the cross entropy (model must output raw logits).

    :param x: Original input
    :param x_reconstructed: Reconstructed
    :return: Descriptor loss averaged over batch
    """
    total_loss = torch.tensor(0.0, device=DEVICE)

    orig_coords = x[:, :, :COORDINATE_DIMENSIONS]
    orig_descriptors = x[:, :, COORDINATE_DIMENSIONS:]

    recon_coords = x_reconstructed[:, :, :COORDINATE_DIMENSIONS]
    recon_descriptors = x_reconstructed[:, :, COORDINATE_DIMENSIONS:]

    for orig_coor, recon_coor, orig_desc, recon_desc in zip(orig_coords, recon_coords, orig_descriptors, recon_descriptors):

        # Pairwise distance matrix between all original and reconstructed coordinates
        # (rows: distances from reconstructed, columns: distances from original)
        dist_matrix = torch.cdist(recon_coor, orig_coor, p=2)

        # Get probability of the closest point, softmin is differentiable
        neighbour_weights = F.softmin(dist_matrix * 100, dim=1)  # Scaling distance helps focus on a single point

        # Matrix multiplication to get predicted descriptors
        matched_target_descriptors = torch.matmul(neighbour_weights, orig_desc)  # (recon_voxels, descriptor_dim)

        # Extract the most likely class index for each matched descriptor
        target_cls = matched_target_descriptors.argmax(dim=-1)  # (recon_voxels)

        # Apply CrossEntropyLoss
        descriptor_loss = F.cross_entropy(recon_desc, target_cls, reduction='mean')

        total_loss += descriptor_loss

    batch_loss = total_loss / x.size(0)  # Average over batch

    return batch_loss


def overlap_penalty(x_reconstructed: torch.Tensor) -> torch.Tensor:
    """
    Calculates penalty for overlapping non-padded voxels that would collapse into
    one another when scaled to the full descriptor matrix.
    The penalty is calculated from the pairwise distance between matching coordinates.
    The negative sum of the log of the upper triangle of the distance matrix (excluding the diagonal)
    is calculated as the penalty and then averaged over the batch.

    :param x_reconstructed: Reconstructed
    :return: Overlap penalty averaged over batch
    """
    total_penalty = torch.tensor(0.0, device=DEVICE)

    recon_coords = x_reconstructed[:, :, :COORDINATE_DIMENSIONS]

    # Mask for non-padded voxels from descriptor values
    recon_mask = (x_reconstructed[:, :, COORDINATE_DIMENSIONS:].argmax(dim=-1) != 0)  # (Batch_size, num_voxels)

    for coords, mask in zip(recon_coords, recon_mask):
        # Scale, round, and clamp to grid space range
        scaled_coords = coords * EXPANDED_GRID_SIZE
        rounded_coords = torch.clamp(torch.round(scaled_coords), min=0, max=EXPANDED_GRID_SIZE)

        # Mask padded voxels with 'inf' based on descriptor values
        rounded_coords[~mask, :] = float('inf')

        # Tracks what has been found to not double count
        overlapping_idx = set()

        # Iterate over each coordinate set (x,y,z) and check for matching
        for i in range(rounded_coords.size(0)):
            current_coord = rounded_coords[i]

            # Skip if padded (masked with inf (descriptor) or scaled coordinate padded value) or already found overlapping
            if torch.isinf(current_coord).any() or (rounded_coords[i] == EXPANDED_GRID_SIZE).any() or i in overlapping_idx:
                continue

            # Find matching coordinates (overlaps)
            matches = (rounded_coords == current_coord).all(dim=1)
            match_indices = matches.nonzero(as_tuple=True)[0]  # Returns indices of True values as a 1D tensor

            # Calculate pairwise distance, and take the negative sum of the log of the upper triangle of
            # the distance matrix (excluding the diagonal)
            if match_indices.numel() > 1:
                dist = torch.cdist(coords[match_indices, :], coords[match_indices, :])

                total_penalty += -torch.triu(torch.log(dist + 1e-8), diagonal=1).sum()  # Avoid log(0)

                # Add to set to not check matched coordinates
                overlapping_idx.update(match_indices.tolist())

    batch_penalty = total_penalty / x_reconstructed.size(0)  # Averaged over batch

    return batch_penalty


```

```python
class VaeLoss:
    def __init__(self, lambda_coord: float, lambda_desc: float, lambda_collapse: float, lambda_reg: float = 0.001):
        """
        Initialise VAE Loss with lambda scaling terms.

        :param lambda_coord: Scaling factor of coordinate matching loss
                (clustering around original points, and nearest neighbour weighted pairwise distance)
        :param lambda_desc: Scaling factor of descriptor loss
                (cross entropy loss based on raw logits and nearest original voxel descriptor)
        :param lambda_collapse: Scaling factor of overlap penalty
                (negative sum of the log of the upper triangle of the distance matrix (excluding diagonal)
                of reconstructed voxel coordinates that would collapse into a single point when scaled)
        :param lambda_reg: Scaling factor of transformation regularising term
        """
        self.lambda_coord = lambda_coord
        self.lambda_desc = lambda_desc
        self.lambda_collapse = lambda_collapse
        self.lambda_reg = lambda_reg

    def __call__(self, x: torch.Tensor, x_reconstructed: torch.Tensor, z_mean: torch.Tensor, z_log_var: torch.Tensor,
                 beta: float, transform_matrix: torch.Tensor) -> tuple[torch.Tensor, dict[str:float]]:
        """
        Calculates VAE loss (reconstruction loss + beta * KL divergence)
        Reconstruction loss is the sum of coordinate match loss, descriptor loss, padded penalty,
        collapse penalty, and transform regularising term (all scaled).

        KL divergence is averaged across batch size.

        Returned loss parts dictionary contains the following keys with float values:
        - 'recon_loss'
        - 'coor_match_loss'
        - 'scaled_coor_match_loss'
        - 'desc_loss'
        - 'scaled_desc_loss'
        - 'collapse_penalty'
        - 'scaled_collapse_penalty'
        - 'transform_reg'
        - 'kl'
        - 'beta_kl'

        :param x: Input tensor with shape (batch_size, *input_dim)
        :param x_reconstructed: Decoder output with shape (batch_size, *input_dim)
        :param z_mean: Latent space mean with shape (batch_size, latent_dim)
        :param z_log_var: Log variance of latent space with shape (batch_size, latent_dim)
        :param beta: Beta to scale KL divergence
        :param transform_matrix: Transformation matrix used to calculate regularisation term to encourage orthogonality
        :return: Total loss, Loss parts dictionary
        """
        # Calculate losses and penalties
        coor_match_loss = coordinate_matching_loss(x, x_reconstructed)
        desc_loss = descriptor_matching_loss(x, x_reconstructed)
        collapse_penalty = overlap_penalty(x_reconstructed)

        # Regularisation term to encourage orthogonality - based on
        # https://medium.com/@itberrios6/point-net-for-classification-968ca64c57a9
        batch_size = transform_matrix.size(0)
        identity = torch.eye(transform_matrix.size(-1)).to(DEVICE)
        transform_reg = torch.linalg.norm(identity - torch.bmm(transform_matrix, transform_matrix.transpose(2, 1)))
        transform_reg = transform_reg / batch_size

        recon_loss = (
                self.lambda_coord * coor_match_loss +
                self.lambda_desc * desc_loss +
                self.lambda_collapse * collapse_penalty +
                self.lambda_reg * transform_reg
        )

        kl_div = -0.5 * torch.sum(1 + z_log_var - z_mean.pow(2) - z_log_var.exp()) / x.shape[0]  # Averaged across batch size

        total_loss = recon_loss + beta * kl_div

        loss_parts = {
            'recon_loss': recon_loss.item(),
            'coor_match_loss': coor_match_loss.item(),
            'scaled_coor_match_loss': (self.lambda_coord * coor_match_loss).item(),
            'desc_loss': desc_loss.item(),
            'scaled_desc_loss': (self.lambda_desc * desc_loss).item(),
            'collapse_penalty': collapse_penalty.item(),
            'scaled_collapse_penalty': (self.lambda_collapse * collapse_penalty).item(),
            'transform_reg': transform_reg.item(),
            'scaled_transform_reg': (self.lambda_reg * transform_reg).item(),
            'kl': kl_div.item(),
            'beta_kl': (beta * kl_div).item()
        }

        return total_loss, loss_parts

```

Figure H.72. *Implementation of VAE loss and custom losses*

Appendix I: Beta Run Results – With and Without Softmax

Table I.11. *Beta training run results for evaluation models*

Name	Batch Size	Latent Dim	Learning Rate	Lambda Coord	Lambda Desc	Lambda Collapse	β	Recon Loss	KL
beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be0.1_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	0.1	3.9068	6.2604
beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be0.3_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	0.3	4.6199	3.1570
beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be0.5_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	0.5	5.8815	1.3046
beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be1.0_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	1	6.1184	0.7355
softmax_beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be0.1_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	0.1	6.6908	1.9015
beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be1.5_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	1.5	7.3012	0.0000
beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be2.0_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	2	7.3266	0.0000
softmax_beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be0.3_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	0.3	7.7081	0.0003
softmax_beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be1.5_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	1.5	7.7106	0.0000
softmax_beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be2.0_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	2	7.7146	0.0000
softmax_beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be0.5_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	0.5	7.7217	0.0000
softmax_beta_run_500_bs64_ld16_adam_lr0.0001_wd1e-05_be1.0_co2.5_de5.0_cl0.3_tr0.001	64	16	0.0001	2.5	5	0.3	1	7.7466	0.0000

Scaled KL	Total Loss	Avg. Weighted F1	Avg. Euclid Dist	Scaled Coord	Scaled Desc	Scaled Collapse	Epochs Run (Rolled back)	PCA Sil	UMAP Sil	Pairwise Mean	Pairwise STD	k Used
0.6260	4.5329	0.5332	0.7480	0.4637	3.4430	0	125	0.3308	0.3852	5.5803	0.9765	5
0.9471	5.5670	0.5002	0.7410	0.4375	4.1789	0.0034	152	0.3037	0.3762	5.5607	0.9897	5
0.6523	6.5338	0.4103	0.7735	0.4934	5.3880	0	225	0.3118	0.3704	5.5466	0.9913	5
0.7355	6.8539	0.4083	0.7625	0.4811	5.6373	0	183	0.3126	0.3706	5.5597	1.0016	5
0.1901	6.8810	0.5049	0.7671	0.2932	6.3975	0	107	0.3060	0.3600	5.5401	0.9918	5
0.0000	7.3012	0.3747	0.7937	0.4317	6.8696	0	158	0.2949	0.3650	5.5614	0.9878	5
0.0000	7.3266	0.3754	0.8170	0.4409	6.8857	0	186	0.3071	0.3550	5.5576	0.9832	5
0.0001	7.7082	0.3761	0.7713	0.3711	7.3370	0	117	0.3053	0.3690	5.5694	1.0027	5
0.0000	7.7106	0.3761	0.7723	0.3801	7.3306	0	73	0.3044	0.3573	5.5474	0.9852	5
0.0000	7.7146	0.3766	0.8008	0.3971	7.3175	0	130	0.3092	0.3721	5.5799	0.9959	5
0.0000	7.7218	0.3746	0.7726	0.3833	7.3384	0	75	0.3090	0.3567	5.5475	0.9890	5
0.0000	7.7466	0.3752	0.7889	0.3924	7.3542	0	275	0.3015	0.3642	5.5564	0.9882	5

Appendix J: Final VAE Model Implementation Code

model.py

```
"""
Defines VAE model modules
"""

import torch
import torch.nn as nn
from ..config import DEVICE, NUM_CLASSES, MAX_VOXELS, COORDINATE_DIMENSIONS


class TNet(nn.Module):
    """
    Transformation Net.
    Should only pass the coordinates to TNet.
    Learns a transformation matrix that is applied to each point, helping to make the network invariant to geometric
    variations e.g. rotation, translation and scaling.
    """

    def __init__(self, coordinate_dimensions: int):
        """
        :param coordinate_dimensions: Number of coordinate_dimensions
        """
        super(TNet, self).__init__()
        self.coord = coordinate_dimensions  # Only applied to coordinates

        self.fc = nn.Sequential(
            nn.Linear(self.coord, 64),
            nn.LeakyReLU(),
            nn.Linear(64, 128),
            nn.LeakyReLU(),
            nn.Linear(128, self.coord * self.coord)
        )

        # Initialise as identity matrix for stability
        self.eye = torch.eye(self.coord).flatten()

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Forward pass.

        :param x: Input tensor with shape (batch_size, coordinate_dimensions)
        :return: Transformation matrix (batch_size, coordinate_dimensions, coordinate_dimensions)
        """
        batch_size = x.size(0)
        transform = self.fc(x.mean(dim=1))  # Learns transformation matrix from average global features
        transform = transform.view(batch_size, self.coord, self.coord) + self.eye.to(DEVICE).view(1, self.coord, self.coord)

        return transform


class Encoder(nn.Module):
    """
    Encoder module.
    """

    def __init__(self, latent_dim: int, coordinate_dimensions: int = 3):
        """
        :param latent_dim: Dimensionality of latent space
        :param coordinate_dimensions: Number of coordinate dimensions
        """
        super(Encoder, self).__init__()
        self.latent_dim = latent_dim
        self.coord = coordinate_dimensions
        self.tnet = TNet(self.coord)

        # MLP for coordinates
        self.spatial_mlp = nn.Sequential(
            nn.Conv1d(self.coord, 64, kernel_size=1),
            nn.BatchNorm1d(64),
            nn.LeakyReLU(),
            nn.Conv1d(64, 128, kernel_size=1),
            nn.BatchNorm1d(128),
            nn.LeakyReLU(),
            nn.Conv1d(128, 1024, kernel_size=1),
            nn.BatchNorm1d(1024),
            nn.LeakyReLU()
        )

```

```
        # MLP for descriptor values (less complex than spatial, so do not need to go as deep)
        self.descriptor_mlp = nn.Sequential(
            nn.Conv1d(NUM_CLASSES, 64, kernel_size=1),
            nn.BatchNorm1d(64),
            nn.LeakyReLU(),
            nn.Conv1d(64, 128, kernel_size=1),
            nn.BatchNorm1d(128),
            nn.LeakyReLU(),
            nn.Conv1d(128, 1024, kernel_size=1),
            nn.BatchNorm1d(1024),
            nn.LeakyReLU()
        )

        self.global_pool = nn.AdaptiveAvgPool1d(1)

        self.combined_activation = nn.LeakyReLU()

        # Reduction before latent space
        self.fc = nn.Sequential(
            nn.Linear(1024, 512),
            nn.LeakyReLU(),
            nn.Linear(512, 256),
            nn.LeakyReLU(),
            nn.Linear(256, 128),
            nn.LeakyReLU()
        )

        self.z_mean_fc = nn.Linear(128, self.latent_dim)
        self.z_log_var_fc = nn.Linear(128, self.latent_dim)

    def forward(self, x: torch.Tensor) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
        """
        Forward pass.

        :param x: Input tensor with shape (batch_size, *input_dim)
        :return:
            z_mean - Latent space mean with shape (batch_size, latent_dim),
            z_log_var - Log variance of latent space with shape (batch_size, latent_dim)
            transform_matrix - Transforms coordinates for geometric invariance
        """
        # Split coordinates and descriptors
        coord = x[:, :, :self.coord]  # (B, N, coordinates)
        desc = x[:, :, self.coord:]  # (B, N, descriptors)

        # T-Net used for learning spatial alignment matrix - coordinates only
        transform_matrix = self.tnet(coord)
        coord_transformed = torch.bmm(coord, transform_matrix)

        # Transpose for input to 1D convolutional layers
        # (batch, num voxels, coordinate features) -> (batch, coordinate features, num voxels)
        coord_transposed = coord_transformed.transpose(1, 2)
        # (batch, num voxels, one-hot descriptors) -> (batch, one-hot descriptors, num voxels)
        desc_transposed = desc.transpose(1, 2)

        # Per point features - spatial and descriptors processed separately
        spatial_features = self.spatial_mlp(coord_transposed)
        desc_features = self.descriptor_mlp(desc_transposed)

        # Global feature extraction
        pooled_spatial = self.global_pool(spatial_features).squeeze(-1)
        max_spatial = torch.max(spatial_features, dim=2)[0]
        pooled_desc = self.global_pool(desc_features).squeeze(-1)

        # Concatenate global, local, and descriptor features
        combined_features = self.combined_activation(pooled_spatial + max_spatial + pooled_desc)

        reduced_features = self.fc(combined_features)

        z_mean = self.z_mean_fc(reduced_features)
        z_log_var = self.z_log_var_fc(reduced_features)

        return z_mean, z_log_var, transform_matrix


```

```python
class Decoder(nn.Module):
    """
    Decoder module.
    """

    def __init__(self, latent_dim: int, max_voxels: int, coordinate_dimensions: int = 3):
        """
        :param latent_dim: Dimensionality of latent space
        :param max_voxels: Maximum number of voxels per robot in full dataset
        :param coordinate_dimensions: Number of coordinate dimensions
        """
        super(Decoder, self).__init__()
        self.num_voxels = max_voxels
        self.coord = coordinate_dimensions

        # Mirrors encoders mean/logvar branches
        self.fc = nn.Sequential(
            nn.Linear(latent_dim, 128),
            nn.LeakyReLU(),
            nn.Linear(128, 256),
            nn.LeakyReLU()
        )

        # Upsample coordinates
        self.coord_deconv = nn.Sequential(
            nn.ConvTranspose1d(256, 512, kernel_size=2),  # 1 -> 2
            nn.BatchNorm1d(512),
            nn.LeakyReLU(),
            nn.ConvTranspose1d(512, 1024, kernel_size=2, stride=2),  # 2 -> 4
            nn.BatchNorm1d(1024),
            nn.LeakyReLU(),
            nn.ConvTranspose1d(1024, 256, kernel_size=2, stride=2),  # 4 -> 8
            nn.BatchNorm1d(256),
            nn.LeakyReLU(),
            nn.ConvTranspose1d(256, self.coord, kernel_size=1)  # (B, coord dim, num_voxels) 8 -> 8
        )

        # Refine coordinates
        self.coord_fc = nn.Sequential(
            nn.Linear(self.coord, self.coord),
            nn.LeakyReLU()
        )

        # Upsample descriptors
        self.desc_deconv = nn.Sequential(
            nn.ConvTranspose1d(256, 512, kernel_size=2),  # 1 -> 2
            nn.BatchNorm1d(512),
            nn.LeakyReLU(),
            nn.ConvTranspose1d(512, 128, kernel_size=2, stride=2),  # 2 -> 4
            nn.BatchNorm1d(128),
            nn.LeakyReLU(),
            nn.ConvTranspose1d(128, 64, kernel_size=2, stride=2),  # 4 -> 8
            nn.BatchNorm1d(64),
            nn.LeakyReLU(),
            nn.ConvTranspose1d(64, NUM_CLASSES, kernel_size=1)  # (B, num descriptors, num_voxels)  8 -> 8
        )

        # Refine descriptors
        self.desc_fc = nn.Sequential(
            nn.Linear(NUM_CLASSES, NUM_CLASSES),
            nn.LeakyReLU()
        )

        # Combined refining
        self.combined_fc = nn.Sequential(
            nn.Linear(self.coord + NUM_CLASSES, self.coord + NUM_CLASSES),
            nn.LeakyReLU()  # This should be removed for the softmax model
        )

        # Normalise coordinates
        self.sigmoid = nn.Sigmoid()

        # Probability for one-hot descriptors - This is what was intended to be applied, but no time to tune for evaluation
        # self.softmax = nn.Softmax(dim=-1)

```

```
    def forward(self, z: torch.Tensor) -> torch.Tensor:
        """
        Forward pass.
        Normalise coordinates using sigmoid.
        Descriptor values are raw logits.

        :param z: Sampled latent vector with shape (batch_size, latent_dim)
        :return: Reconstructed input with shape (batch_size, *input_dim)
        """
        upsampled_latent = self.fc(z).unsqueeze(-1)  # (B, 256, 1)

        upsampled_coord = self.coord_deconv(upsampled_latent)  # (B, coord dim, num_voxels)
        upsampled_desc = self.desc_deconv(upsampled_latent)  # (B, num descriptors, num_voxels)

        transposed_coord = upsampled_coord.transpose(1, 2)  # (B, num_voxels, coord dim)
        transposed_desc = upsampled_desc.transpose(1, 2)  # (B, num_voxels, descriptor_dim)

        refined_coord = self.coord_fc(transposed_coord)
        refined_desc = self.desc_fc(transposed_desc)

        combined = torch.cat((refined_coord, refined_desc), dim=2)

        x_reconstructed = self.combined_fc(combined)

        # Normalises coordinates
        x_reconstructed[:, :, :self.coord] = self.sigmoid(x_reconstructed[:, :, :self.coord])

        # Apply Softmax to descriptors - This is what was intended to be applied, but no time to tune for evaluation
        # x_reconstructed[:, :, self.coord:] = self.softmax(x_reconstructed[:, :, self.coord:])

        # Raw logits for descriptors
        return x_reconstructed


class Sample(nn.Module):
    """
    Reparameterization trick to allow backpropagation and sampling.
    """

    def forward(self, z_mean: torch.Tensor, z_log_var: torch.Tensor) -> torch.Tensor:
        """
        Forward pass.

        :param z_mean: Latent space mean with shape (batch_size, latent_dim)
        :param z_log_var: Log variance of latent space with shape (batch_size, latent_dim)
        :return: Sampled latent vector with shape (batch_size, latent_dim)
        """
        epsilon = torch.randn_like(z_mean)

        return z_mean + torch.exp(0.5 * z_log_var) * epsilon


class VAE(nn.Module):
    """
    Implements encoder, sampling (reparameterization trick) and decoder modules for complete VAE architecture.
    """

    def __init__(self, input_dim: tuple[int, int], latent_dim: int, model_name: str, max_voxels: int = MAX_VOXELS,
                 coordinate_dimensions: int = COORDINATE_DIMENSIONS):
        """
        :param input_dim: Dimensions of input tensor (and reconstructed), (e.g., (8, 8))
        :param latent_dim: Dimensionality of latent space
        :param model_name: Name of model instance, used to identify models and name saved files, ENSURE UNIQUE to avoid overwriting
        :param max_voxels: Maximum number of voxels per robot in full dataset
        :param coordinate_dimensions: Number of coordinate dimensions
        """
        super(VAE, self).__init__()
        self.name = model_name
        self.input_dim = input_dim
        self.latent_dim = latent_dim

        self.encoder = Encoder(latent_dim, coordinate_dimensions)
        self.sampling = Sample()
        self.decoder = Decoder(latent_dim, max_voxels, coordinate_dimensions)

```

```
    def forward(self, x: torch.Tensor) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor, torch.Tensor]:
        """
        Forward pass.

        :param x: Input tensor with shape (batch_size, *input_dim)
        :return:
            x_reconstructed - Reconstructed input with shape (batch_size, *input_dim),
            z - Sampled latent vector with shape (batch_size, latent_dim),
            z_mean - Latent space mean with shape (batch_size, latent_dim),
            z_log_var - Log variance of latent space with shape (batch_size, latent_dim)
            transform_matrix - Transforms coordinates for geometric invariance
        """
        z_mean, z_log_var, transform_matrix = self.encoder(x)
        z = self.sampling(z_mean, z_log_var)
        x_reconstructed = self.decoder(z)

        return x_reconstructed, z, z_mean, z_log_var, transform_matrix

```

Figure J.73. *Final model implementation*
Note: Code implementation corresponds directly to the final architecture visualised in Figure 34.

Appendix K: $\beta = 0.3$ and $\beta = 0.5$ PCA and UMAP Plots

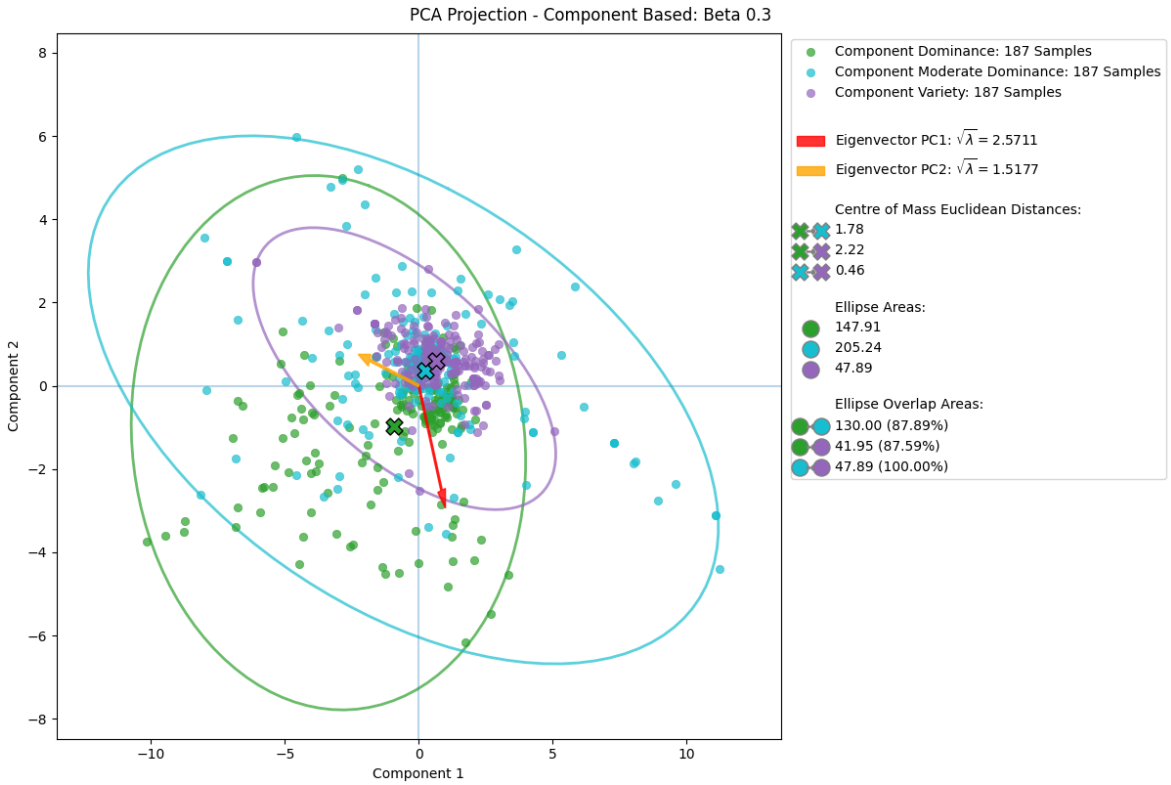


Figure K.74. *β=0.3 model component PCA projection*

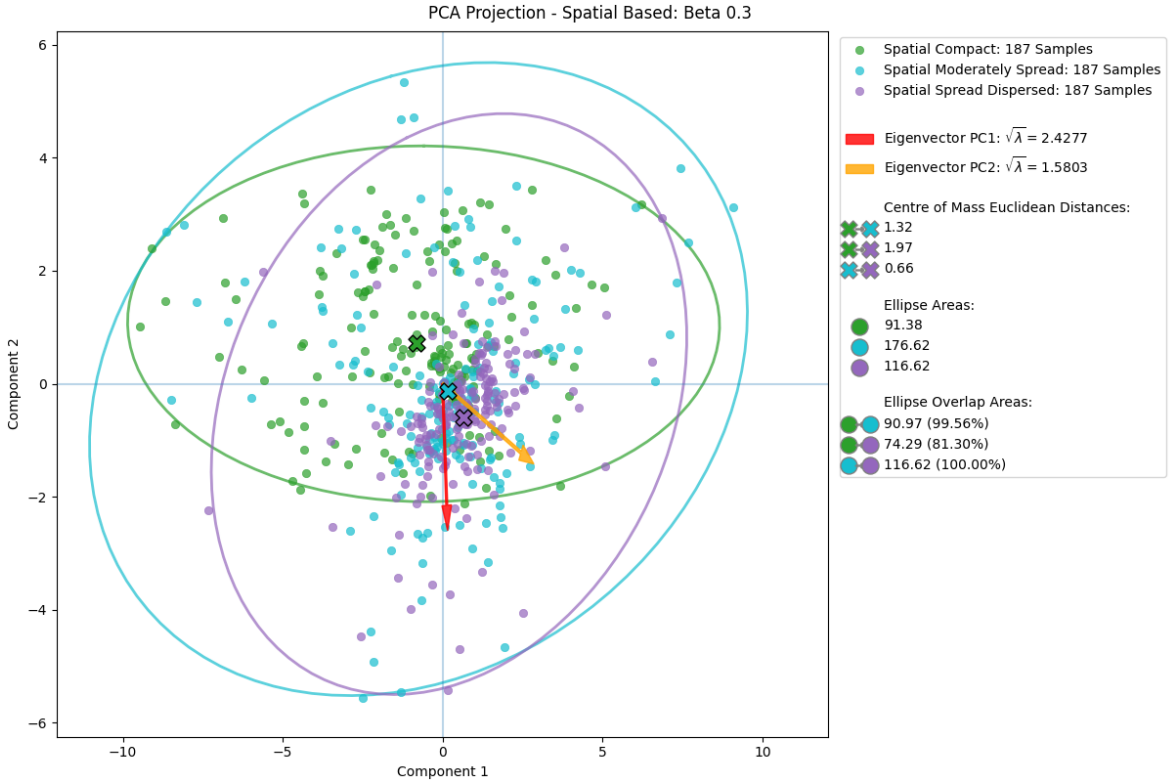


Figure K.75. *β=0.3 model spatial PCA projection*

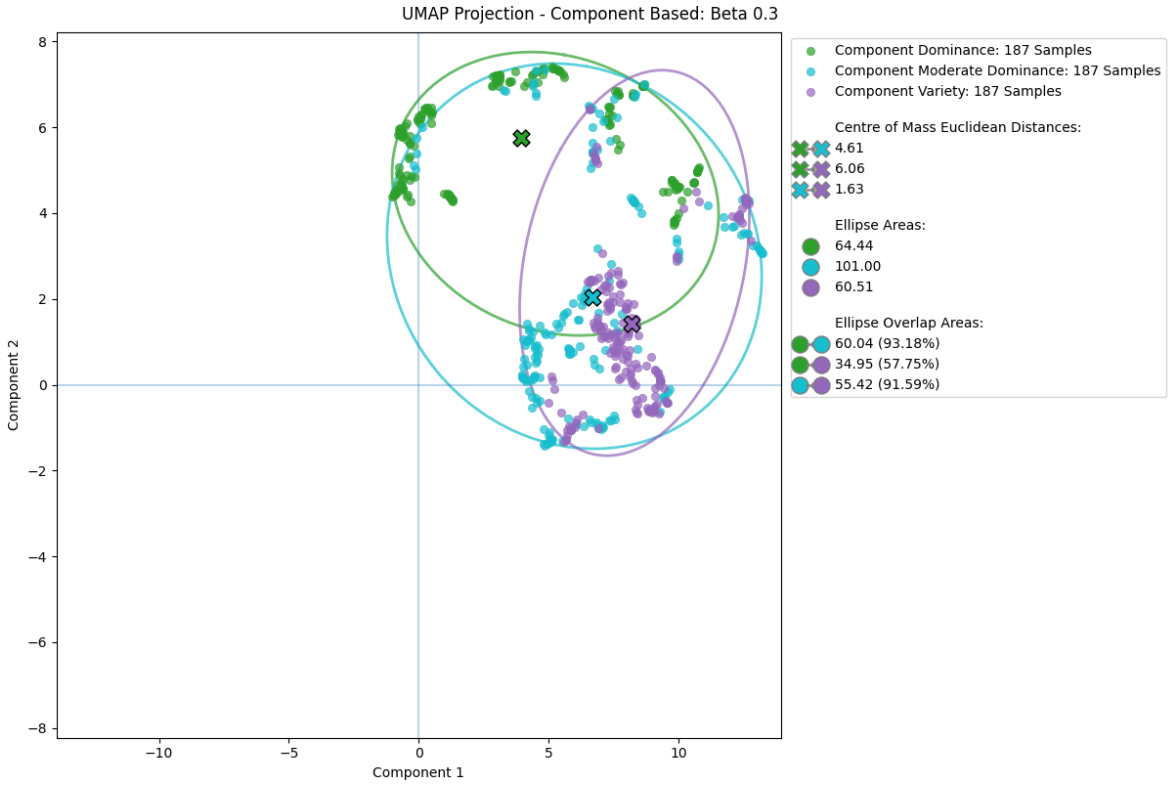


Figure K.76. *β=0.3 model component UMAP projection*

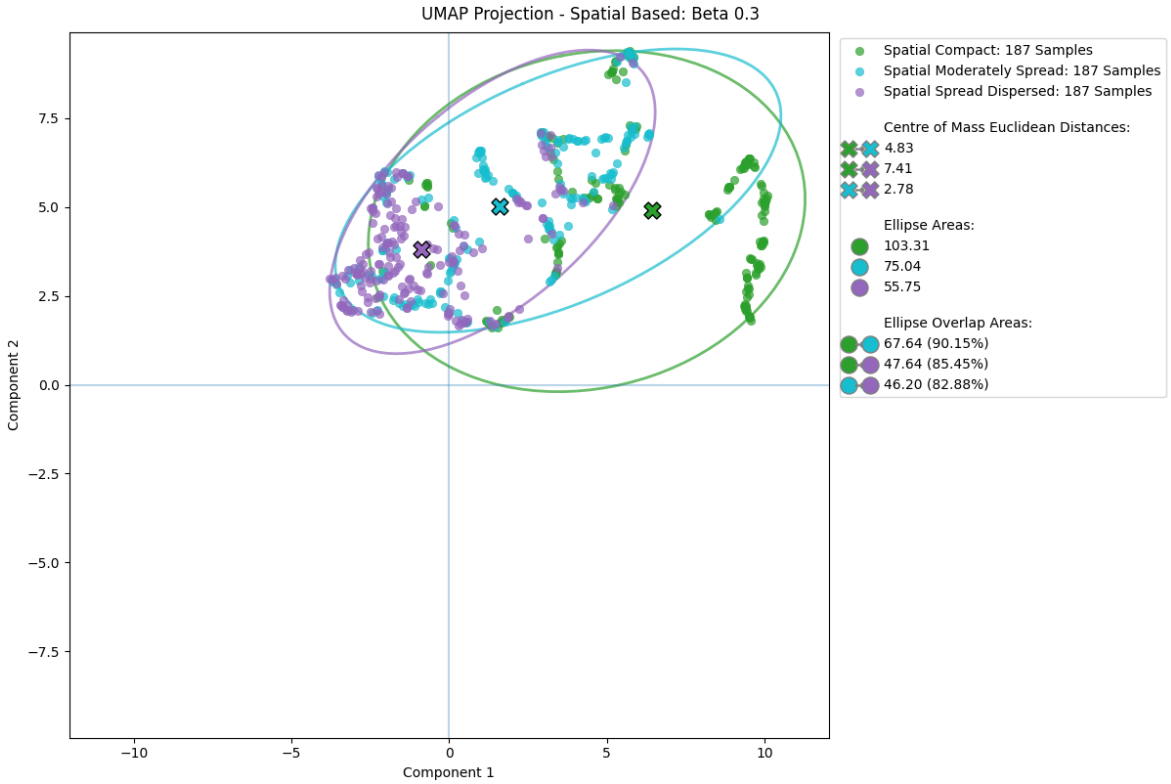


Figure K.77. *β=0.3 model spatial UMAP projection*

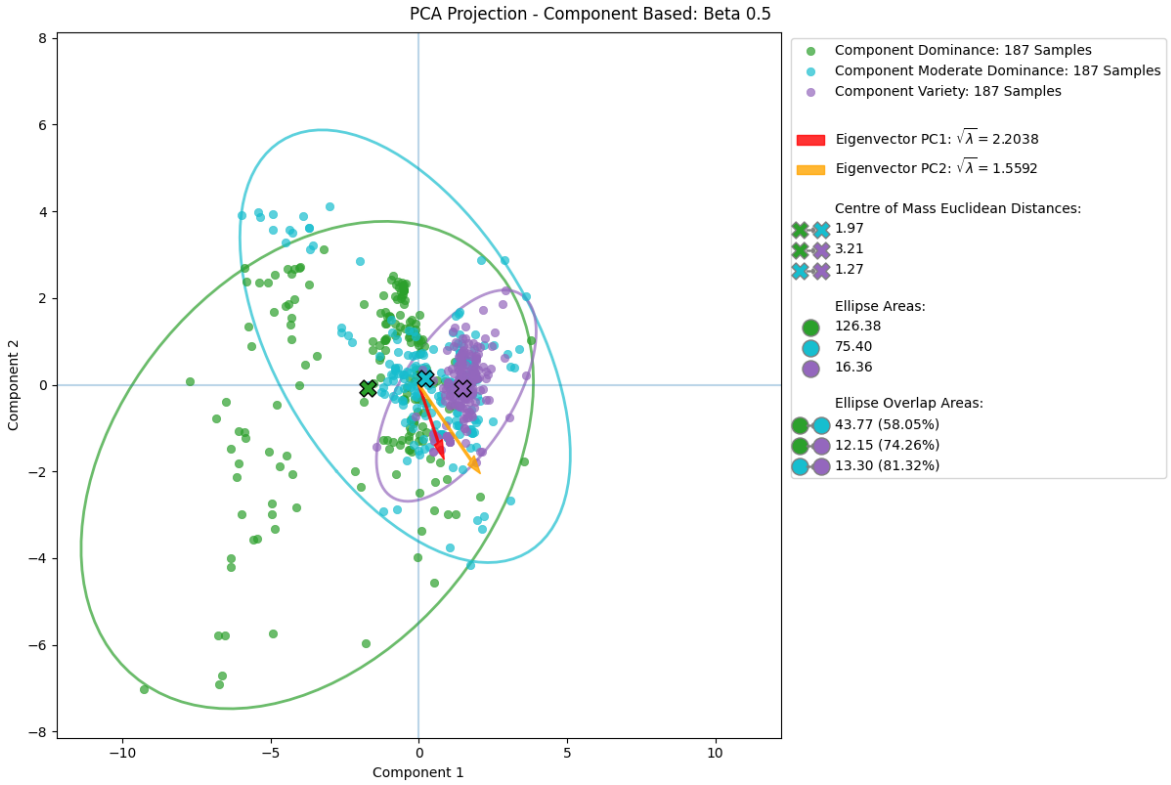


Figure K.78. *β=0.5 model component PCA projection*

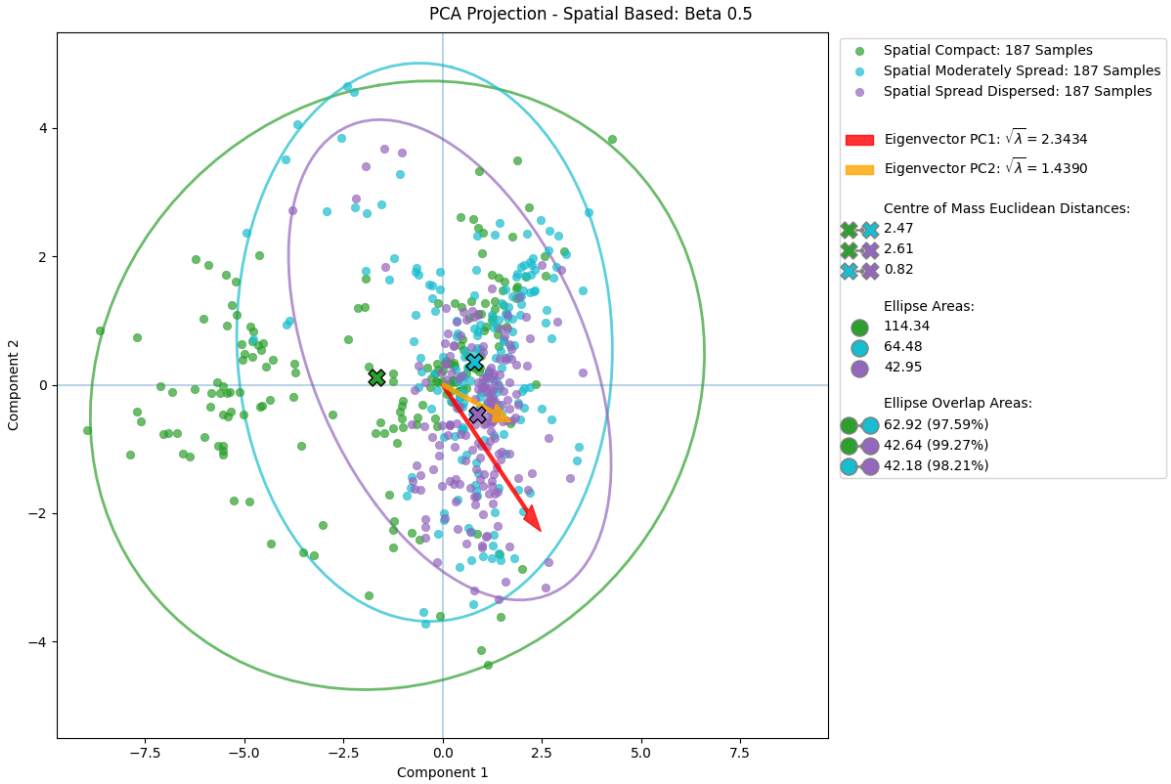


Figure K.79. *β=0.5 model spatial PCA projection*

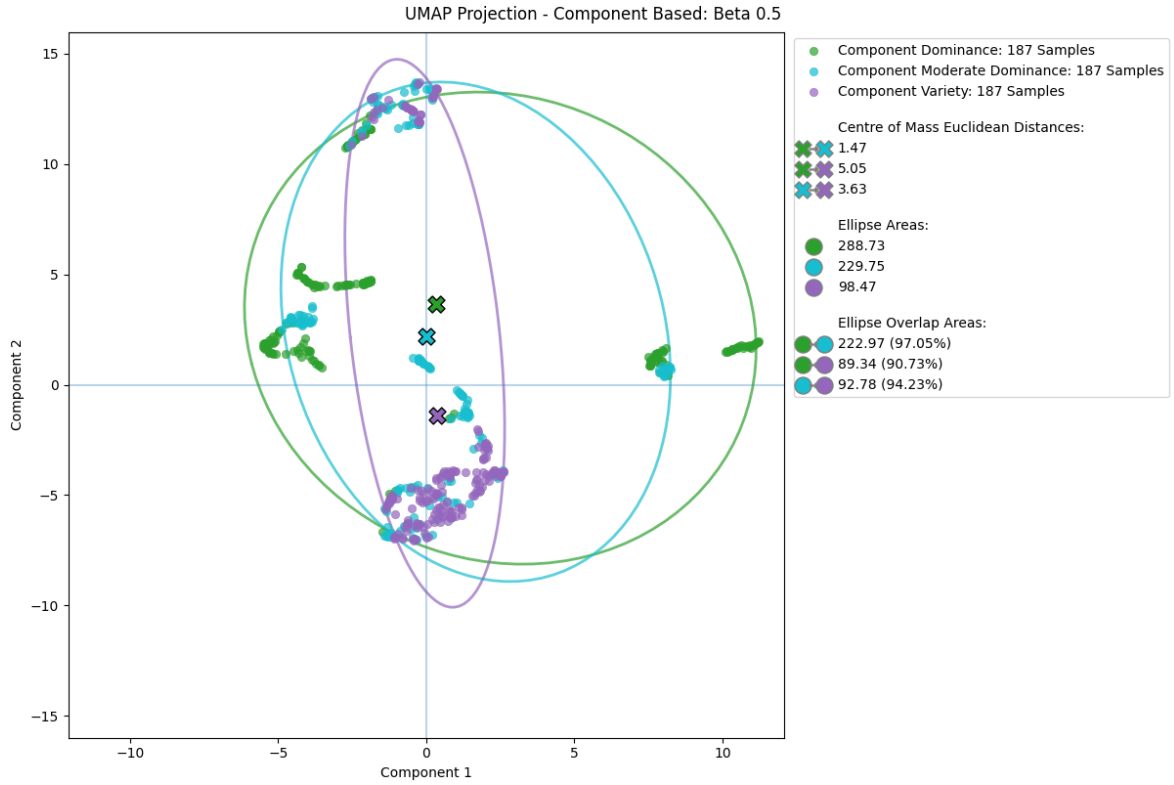


Figure K.80. *β=0.5 model component UMAP projection*

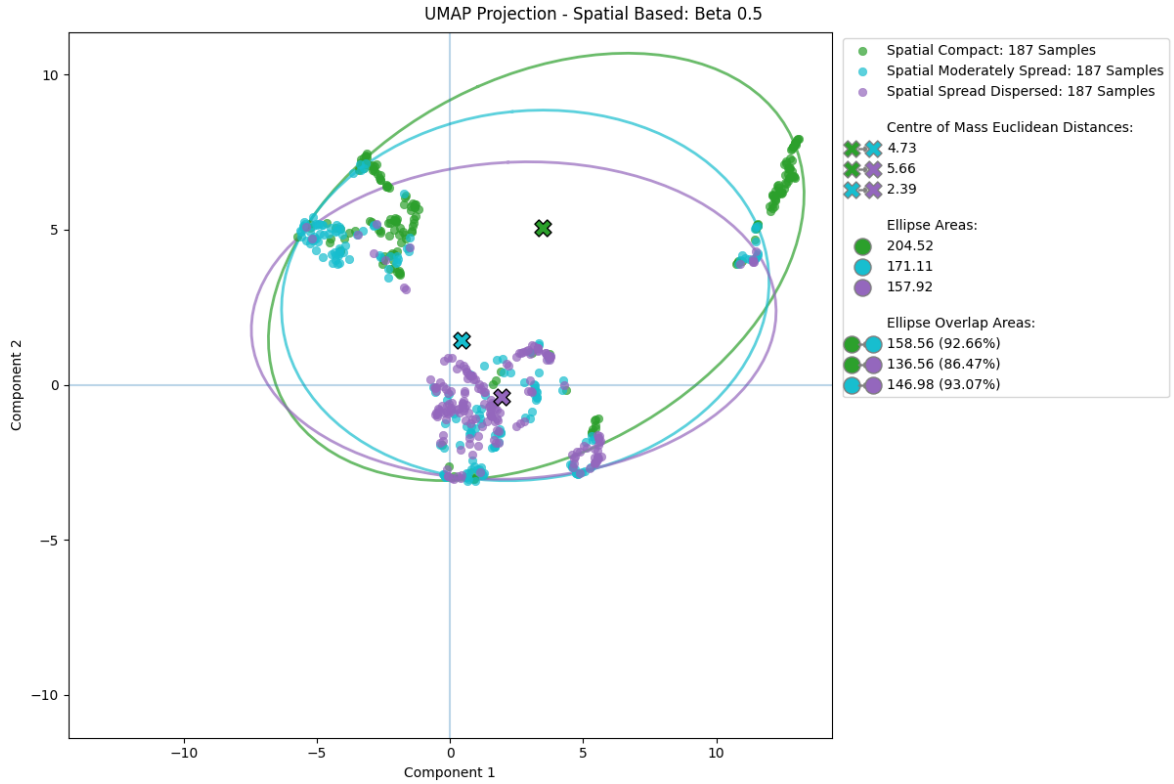


Figure K.81. *β=0.5 model spatial UMAP projection*

Appendix L: PCA and UMAP Point Similarity

```
PCA Projection - Component Based: Beta 0.3
Pairwise distances:
IDs:      81445,   210406,   176868,   213295,   30678,   195043,   230604,   242949,   123338,   151167,   152089,   30383,   172427,   81347,   123954,   233356
tensor([[ 0.0000, 10.9990,  2.2995,  2.1650,  5.9716,  3.4368,  7.2436,  4.1153, 10.5177, 10.3416,  3.2988,  3.4669,  3.1977,  2.8913,  4.2409,  2.9314],
        [10.9990,  0.0000, 11.4666,  9.8677,  6.6091, 12.1209, 17.1033, 10.0605, 21.3745, 11.2091, 10.9911, 11.6919, 11.1965, 11.4829, 13.7782, 13.0631],
        [ 2.2995, 11.4666,  0.0000,  1.6182,  5.4258,  5.6780,  5.7594,  2.4096, 11.0465,  8.4962,  1.2223,  1.1776,  1.0304,  0.6142,  2.4320,  1.6302],
        [ 2.1650,  9.8677,  1.6182,  0.0000,  4.0645,  5.4690,  7.3699,  2.0563, 12.2478,  8.2159,  1.7091,  2.2536,  1.7693,  1.8108,  4.0299,  3.1957],
        [ 5.9716,  6.6091,  5.4258,  4.0645,  0.0000,  8.6401, 10.6946,  3.5393, 16.3118,  6.2080,  4.6372,  5.3195,  4.8713,  5.2518,  7.4649,  7.0463],
        [ 3.4368, 12.1209,  5.6780,  5.4690,  8.6401,  0.0000,  9.4780,  7.5029,  9.6480, 13.6787,  6.7346,  6.8555,  6.6255,  6.2876,  7.1296,  5.7678],
        [ 7.2436, 17.1033,  5.7594,  7.3699, 10.6946,  9.4780,  0.0000,  7.1590,  8.6789, 10.9922,  6.1161,  5.4157,  5.9070,  5.6371,  3.3460,  4.3696],
        [ 4.1153, 10.0605,  2.4096,  2.0563,  3.5393,  7.5029,  7.1590,  0.0000, 13.4081,  6.2344,  1.2691,  1.8609,  1.5161,  1.9850,  3.9865,  3.8413],
        [10.5177, 21.3745, 11.0465, 12.2478, 16.3118,  9.6480,  8.6789, 13.4081,  0.0000, 18.8976, 12.1425, 11.6537, 11.8986, 11.4236,  9.9144,  9.5822],
        [10.3416, 11.2091,  8.4962,  8.2159,  6.2080, 13.6787, 10.9922,  6.2344, 18.8976,  0.0000,  7.2749,  7.5205,  7.4720,  7.9371,  8.9870,  9.5474],
        [ 3.2988, 10.9911,  1.2223,  1.7091,  4.6372,  6.7346,  6.1161,  1.2691, 12.1425,  7.2749,  0.0000,  0.7009,  0.2482,  0.7201,  2.8307,  2.5722],
        [ 3.4669, 11.6919,  1.1776,  2.2536,  5.3195,  6.8555,  5.4157,  1.8609, 11.6537,  7.5205,  0.7009,  0.0000,  0.5143,  0.5763,  2.1463,  2.0815],
        [ 3.1977, 11.1965,  1.0304,  1.7693,  4.8713,  6.6255,  5.9070,  1.5161, 11.8986,  7.4720,  0.2482,  0.5143,  0.0000,  0.4827,  2.6044,  2.3254],
        [ 2.8913, 11.4829,  0.6142,  1.8108,  5.2518,  6.2876,  5.6371,  1.9850, 11.4236,  7.9371,  0.7201,  0.5763,  0.4827,  0.0000,  2.2968,  1.8622],
        [ 4.2409, 13.7782,  2.4320,  4.0299,  7.4649,  7.1296,  3.3460,  3.9865,  9.9144,  8.9870,  2.8307,  2.1463,  2.6044,  2.2968,  0.0000,  1.3694],
        [ 2.9314, 13.0631,  1.6302,  3.1957,  7.0463,  5.7678,  4.3696,  3.8413,  9.5822,  9.5474,  2.5722,  2.0815,  2.3254,  1.8622,  1.3694,  0.0000]])

UMAP Projection - Component Based: Beta 0.3
Pairwise distances:
IDs:      81445,   210406,   176868,   213295,   30678,   195043,   230604,   242949,   123338,   151167,   152089,   30383,   172427,   81347,   123954,   233356
tensor([[ 0.0000,  3.8576,  4.6795,  7.3579,  3.7031,  0.6054,  7.6280,  5.9851, 10.3778,  5.3999,  7.1855,  7.4252,  8.4898,  7.2735,  9.4379,  9.4023],
        [ 3.8576,  0.0000,  8.3612,  3.6597,  0.7098,  4.4492,  4.5992,  5.0382,  6.7906,  2.6176,  7.5730,  5.9595,  7.8582,  5.3274,  7.5262,  5.7722],
        [ 4.6795,  8.3612,  0.0000, 11.5110,  8.0228,  4.0852, 11.1585,  8.2031, 14.2576,  9.1324,  7.5746,  9.6639,  9.6380,  9.9703, 11.7029, 13.3773],
        [ 7.3579,  3.6597, 11.5110,  0.0000,  3.6560,  7.9122,  2.1792,  5.2865,  3.1441,  2.5781,  8.2369,  5.2339,  7.4903,  4.2656,  5.8076,  2.1177],
        [ 3.7031,  0.7098,  8.0228,  3.6560,  0.0000,  4.2644,  4.2479,  4.3442,  6.7143,  2.1171,  6.8642,  5.3173,  7.1698,  4.7357,  6.9596,  5.7205],
        [ 0.6054,  4.4492,  4.0852,  7.9122,  4.2644,  0.0000,  8.0929,  6.2192, 10.9043,  5.8768,  7.1725,  7.6902,  8.6101,  7.6069,  9.7310,  9.9386],
        [ 7.6280,  4.5992, 11.1585,  2.1792,  4.2479,  8.0929,  0.0000,  3.7251,  3.2243,  2.2329,  6.5507,  3.2476,  5.4760,  2.2680,  3.6306,  2.5897],
        [ 5.9851,  5.0382,  8.2031,  5.2865,  4.3442,  6.2192,  3.7251,  0.0000,  6.9046,  2.9961,  2.9527,  1.5056,  2.8308,  1.8338,  3.5819,  6.3126],
        [10.3778,  6.7906, 14.2576,  3.1441,  6.7143, 10.9043,  3.2243,  6.9046,  0.0000,  5.1316,  9.6015,  6.1419,  8.1733,  5.2440,  5.5745,  1.0309],
        [ 5.3999,  2.6176,  9.1324,  2.5781,  2.1171,  5.8768,  2.2329,  2.9961,  5.1316,  0.0000,  5.9027,  3.5110,  5.6206,  2.7596,  4.9183,  4.2487],
        [ 7.1855,  7.5730,  7.5746,  8.2369,  6.8642,  7.1725,  6.5507,  2.9527,  9.6015,  5.9027,  0.0000,  3.4825,  2.2103,  4.3594,  4.9454,  9.1272],
        [ 7.4252,  5.9595,  9.6639,  5.2339,  5.3173,  7.6902,  3.2476,  1.5056,  6.1419,  3.5110,  3.4825,  0.0000,  2.2565,  0.9825,  2.0764,  5.7446],
        [ 8.4898,  7.8582,  9.6380,  7.4903,  7.1698,  8.6101,  5.4760,  2.8308,  8.1733,  5.6206,  2.2103,  2.2565,  0.0000,  3.2322,  2.9407,  7.8939],
        [ 7.2735,  5.3274,  9.9703,  4.2656,  4.7357,  7.6069,  2.2680,  1.8338,  5.2440,  2.7596,  4.3594,  0.9825,  3.2322,  0.0000,  2.2433,  4.7916],
        [ 9.4379,  7.5262, 11.7029,  5.8076,  6.9596,  9.7310,  3.6306,  3.5819,  5.5745,  4.9183,  4.9454,  2.0764,  2.9407,  2.2433,  0.0000,  5.5345],
        [ 9.4023,  5.7722, 13.3773,  2.1177,  5.7205,  9.9386,  2.5897,  6.3126,  1.0309,  4.2487,  9.1272,  5.7446,  7.8939,  4.7916,  5.5345,  0.0000]])

Pair: 242949 and 30383  |  PCA: 1.86, UMAP: 1.51
Pair: 242949 and 81347  |  PCA: 1.99, UMAP: 1.83
Pair: 152089 and 172427  |  PCA: 0.25, UMAP: 2.21
Pair: 30383 and 172427  |  PCA: 0.51, UMAP: 2.26
Pair: 30383 and 81347  |  PCA: 0.58, UMAP: 0.98
Pair: 30383 and 123954  |  PCA: 2.15, UMAP: 2.08
Pair: 81347 and 123954  |  PCA: 2.30, UMAP: 2.24
```

Figure L.82. *Pairwise matrices for selected samples for component-based PCA and UMAP*
Note: Samples close together deemed pairs are based on a threshold maximum distance of 2.5 and shown below the matrices with their sample IDs. These relate to samples selected and presented in Figure 55 and Figure 56.

```
PCA Projection - Spatial Based: Beta 0.3
Pairwise distances:
IDs:      46710,   235270,   83192,   193546,   214744,   222164,   125196,   18577,   224389,   270583,   131331,   86971,   244764,   100293
tensor([[ 0.0000,  1.6167,  1.5860,  8.6288,  8.4568,  1.2967,  3.3015,  4.7019, 10.2202,  2.5395,  1.9826,  6.1421,  2.4087,  3.0615],
        [ 1.6167,  0.0000,  3.1686,  7.0183,  6.8997,  2.7346,  3.6407,  4.9034, 11.8069,  2.2708,  3.3313,  6.5416,  3.9325,  4.5912],
        [ 1.5860,  3.1686,  0.0000, 10.1844, 10.0413,  0.6595,  3.2618,  5.4366,  8.8612,  3.2543,  0.9448,  5.6887,  0.8712,  1.4966],
        [ 8.6288,  7.0183, 10.1844,  0.0000,  1.4283,  9.7069,  9.4068,  9.3025, 18.6566,  7.9482, 10.2234, 11.5570, 10.9161, 11.5698],
        [ 8.4568,  6.8997, 10.0413,  1.4283,  0.0000,  9.6342,  9.7298,  8.4190, 18.2039,  8.2052, 10.2121, 12.1137, 10.8303, 11.4891],
        [ 1.2967,  2.7346,  0.6595,  9.7069,  9.6342,  0.0000,  2.6773,  5.6684,  9.4870,  2.5958,  0.6957,  5.2559,  1.2092,  1.8641],
        [ 3.3015,  3.6407,  3.2618,  9.4068,  9.7298,  2.6773,  0.0000,  7.9988, 11.4396,  1.5526,  2.4173,  2.9177,  3.2076,  3.5686],
        [ 4.7019,  4.9034,  5.4366,  9.3025,  8.4190,  5.6684,  7.9988,  0.0000, 10.5336,  7.0025,  6.3040, 10.8356,  6.2006,  6.6353],
        [10.2202, 11.8069,  8.8612, 18.6566, 18.2039,  9.4870, 11.4396, 10.5336,  0.0000, 12.0200,  9.2791, 12.3568,  8.4243,  7.9170],
        [ 2.5395,  2.2708,  3.2543,  7.9482,  8.2052,  2.5958,  1.5526,  7.0025, 12.0200,  0.0000,  2.7472,  4.3334,  3.5975,  4.1485],
        [ 1.9826,  3.3313,  0.9448, 10.2234, 10.2121,  0.6957,  2.4173,  6.3040,  9.2791,  2.7472,  0.0000,  4.7440,  0.8565,  1.4162],
        [ 6.1421,  6.5416,  5.6887, 11.5570, 12.1137,  5.2559,  2.9177, 10.8356, 12.3568,  4.3334,  4.7440,  0.0000,  5.2661,  5.2990],
        [ 2.4087,  3.9325,  0.8712, 10.9161, 10.8303,  1.2092,  3.2076,  6.2006,  8.4243,  3.5975,  0.8565,  5.2661,  0.0000,  0.6588],
        [ 3.0615,  4.5912,  1.4966, 11.5698, 11.4891,  1.8641,  3.5686,  6.6353,  7.9170,  4.1485,  1.4162,  5.2990,  0.6588,  0.0000]])

UMAP Projection - Spatial Based: Beta 0.3
Pairwise distances:
IDs:      46710,   235270,   83192,   193546,   214744,   222164,   125196,   18577,   224389,   270583,   131331,   86971,   244764,   100293
tensor([[ 0.0000,  0.2905,  5.3414,  4.2784,  4.2353,  4.8432,  3.5376,  2.1759,  3.1322,  9.4275,  7.2496,  2.9661,  6.1840,  7.6618],
        [ 0.2905,  0.0000,  5.3617,  4.0328,  3.9926,  4.8391,  3.6884,  1.8899,  3.1358,  9.6481,  7.4929,  3.1498,  6.3990,  7.8107],
        [ 5.3414,  5.3617,  0.0000,  8.4961,  8.4983,  0.6667,  8.5545,  6.1669,  8.4735, 13.4287, 11.0123,  7.8583, 10.4793, 12.5037],
        [ 4.2784,  4.0328,  8.4961,  0.0000,  0.0866,  7.8522,  5.4591,  2.3996,  3.6675, 11.4735,  9.8558,  5.4502,  8.4158,  8.6912],
        [ 4.2353,  3.9926,  8.4983,  0.0866,  0.0000,  7.8563,  5.3775,  2.3852,  3.5835, 11.3878,  9.7734,  5.3731,  8.3320,  8.6047],
        [ 4.8432,  4.8391,  0.6667,  7.8522,  7.8563,  0.0000,  8.1588,  5.5408,  7.9696, 13.2429, 10.8413,  7.4773, 10.2242, 12.1674],
        [ 3.5376,  3.6884,  8.5545,  5.4591,  5.3775,  8.1588,  0.0000,  4.7430,  1.8382,  6.2085,  4.4004,  0.7221,  3.0078,  4.1246],
        [ 2.1759,  1.8899,  6.1669,  2.3996,  2.3852,  5.5408,  4.7430,  0.0000,  3.4452, 10.9411,  8.9627,  4.4203,  7.7046,  8.6689],
        [ 3.1322,  3.1358,  8.4735,  3.6675,  3.5835,  7.9696,  1.8382,  3.4452,  0.0000,  7.8345,  6.2014,  2.0446,  4.7486,  5.2688],
        [ 9.4275,  9.6481, 13.4287, 11.4735, 11.3878, 13.2429,  6.2085, 10.9411,  7.8345,  0.0000,  2.4258,  6.5517,  3.2544,  3.3068],
        [ 7.2496,  7.4929, 11.0123,  9.8558,  9.7734, 10.8413,  4.4004,  8.9627,  6.2014,  2.4258,  0.0000,  4.5535,  1.5299,  3.4217],
        [ 2.9661,  3.1498,  7.8583,  5.4502,  5.3731,  7.4773,  0.7221,  4.4203,  2.0446,  6.5517,  4.5535,  0.0000,  3.2995,  4.7334],
        [ 6.1840,  6.3990, 10.4793,  8.4158,  8.3320, 10.2242,  3.0078,  7.7046,  4.7486,  3.2544,  1.5299,  3.2995,  0.0000,  2.5091],
        [ 7.6618,  7.8107, 12.5037,  8.6912,  8.6047, 12.1674,  4.1246,  8.6689,  5.2688,  3.3068,  3.4217,  4.7334,  2.5091,  0.0000]])

Pair: 46710 and 235270  |  PCA: 1.62, UMAP: 0.29
Pair: 83192 and 222164  |  PCA: 0.66, UMAP: 0.67
Pair: 193546 and 214744  |  PCA: 1.43, UMAP: 0.09
Pair: 131331 and 244764  |  PCA: 0.86, UMAP: 1.53
```

Figure L.83. *Pairwise matrices for selected samples for spatial-based PCA and UMAP*
Note: Samples close together deemed pairs are based on a threshold maximum distance of 2.5 and shown below the matrices with their sample IDs. These relate to samples selected and presented in Figure 57 and Figure 58.