



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Latent Space Exploration for Racing Track Design using Novelty Search

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING - INGEGNERIA IN-
FORMATICA

Author: **Milo Brontesi**

Student ID: 232259

Advisor: Prof. Daniele Loiacono

Co-advisors: Prof. Pierluca Lanzi

Academic Year: 2025-26

Abstract

Quality Diversity (QD) algorithms are evolutionary techniques that aim to discover a collection of solutions that are both high-performing and behaviourally diverse. In this thesis, we investigate the application of QD algorithms for the procedural generation of racing tracks. Specifically, we propose an autonomous, data-driven pipeline based on Novelty Search with Local Competition (NSLC). Rather than relying on hand-crafted behavioural descriptors, our methodology utilises a Variational Autoencoder (VAE) to learn a low-dimensional behavioural space directly from high-dimensional driving telemetry, drawing inspiration from the AURORA framework for autonomous skill discovery. The track geometries are synthesised using a procedural representation based on Voronoi diagrams and Catmull-Rom spline interpolation. We evaluate the impact of dynamically fine-tuning the VAE’s latent space during the evolutionary search, showing that this adaptive representation leads to a more robust exploration of the design space and a higher-quality, less redundant set of generated circuits compared to a static learned representation.

Keywords: Quality Diversity, Novelty Search, Procedural Content Generation, Racing Tracks, TORCS, Evolutionary Algorithms, AURORA

Abstract in lingua italiana

Gli algoritmi di Quality Diversity (QD) sono tecniche evolutive che mirano a scoprire una collezione di soluzioni allo stesso tempo performanti e diverse tra loro. In questa tesi, esaminiamo l'applicazione di questi algoritmi per la generazione procedurale di tracciati automobilistici. Nello specifico, proponiamo una pipeline basata sull'algoritmo di Novelty Search with Local Competition (NSLC). Anziché ricorrere a descrittori definiti manualmente, la nostra metodologia utilizza un Variational Autoencoder (VAE) per apprendere uno spazio comportamentale a bassa dimensione direttamente dalla telemetria di guida, traendo ispirazione dal framework AURORA tradizionalmente applicato all'esplorazione di comportamenti nel campo della robotica. Le geometrie dei tracciati sono generate mediante una rappresentazione procedurale basata su diagrammi di Voronoi e interpolazione con spline di Catmull-Rom. Lo studio valuta l'impatto del fine-tuning dinamico dello spazio latente del VAE durante la ricerca evolutiva, evidenziando che l'aggiornamento adattivo della rappresentazione conduce a un'esplorazione più robusta dello spazio di progettazione e a un insieme di circuiti generati di qualità superiore e meno ridondante rispetto all'impiego statico della medesima rappresentazione appresa.

Parole chiave: Quality Diversity, Novelty Search, Generazione Procedurale, tracciati automobilistici, TORCS, Algoritmi Evolutivi, AURORA

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
1 Introduction	1
2 State of the Art	3
2.1 Procedural Content Generation (PCG)	3
2.1.1 The Roots	3
2.1.2 Taxonomy and Classification of PCG	4
2.2 Evolutionary Algorithms and Quality Diversity	7
2.3 AURORA: AUtonomous RObots that Realize their Abilities	9
2.3.1 Core Mechanism	10
2.3.2 Dimensionality Reduction Algorithms	11
2.4 Variational Autoencoder (VAE)	11
2.4.1 The Autoencoder	11
2.4.2 From Deterministic to Probabilistic: The VAE	13
3 Simulation Environment	15
3.1 TORCS	15
3.1.1 Telemetry and Vehicle dynamics	16
3.1.2 Vehicles and AI drivers	18
3.1.3 Headless execution and Dockerisation	20
3.2 Track generation and solution	20
3.2.1 The TORCS representation	20
3.2.2 Voronoi spline generation	21
3.2.3 Conversion to TORCS XML	26

4	Evolutionary Pipeline	29
4.1	The Novelty Search algorithm	29
4.1.1	The Pyribs Framework	29
4.1.2	Pipeline overview	30
4.1.3	Generation & Variation (Ask)	30
4.1.4	Evaluation and Behavioural Descriptor	32
4.1.5	The Archive (Tell)	33
4.1.6	Dynamic Representation Learning	33
4.2	The Autoencoder	35
4.2.1	Input preprocessing and normalisation	36
4.2.2	Model Architecture	38
4.2.3	Training Setup and Optimisation	43
5	Experiments and Results	45
5.1	VAE Pre-training and Analysis	45
5.1.1	Experimental Setup	45
5.1.2	Experimental Results	46
5.1.3	Discussion	47
5.2	Novelty Search Setup and Analysis	51
5.2.1	Experimental Setup	51
5.2.2	Experimental Results	53
5.2.3	Discussion	57
6	Conclusions and Future work	65
	Bibliography	67
A	Phenotype Metrics Analysis	71
B	Latent Space Analysis	75
C	Resulting Racing Tracks	79
	List of Figures	85
	List of Tables	91

1 | Introduction

A core challenge in game development is the resource-intensive nature of asset design and content production. Depending on the context, creating new content can become a major bottleneck in development cycles. Modern games demand varied, replayable content without incurring prohibitive production costs and time constraints. Procedural Content Generation (PCG) addresses this by automatically creating digital assets through algorithmic methods. This automation acts not merely as a substitute for human effort, but allows developers to rapidly prototype, test, and design at a scale beyond traditional production limits. Furthermore, the ability of PCG methods to algorithmically generate high-quality content enables the exploration of novel game mechanics and innovative design paradigms.

A key challenge in PCG is generating content that is both engaging and meaningfully varied. Traditional Search-Based PCG and standard optimisation algorithms typically aim to converge on a single, globally optimal solution. However, in game and level design, discovering a single global optimum is rarely the goal. To accommodate diverse player preferences and varying design needs, developers require a broad portfolio of high-quality solutions. Quality-Diversity (QD) optimisation techniques are specifically designed to address this requirement. Rather than searching for a singular optimum, QD algorithms apply evolutionary techniques to explore the design space by discovering a vast, structured set of solutions that are simultaneously high-performing (quality) and fundamentally distinct from one another across a defined behavioural space (diversity).

This thesis investigates the application of Quality-Diversity algorithms to the **procedural generation of racing tracks**. Specifically, this research proposes a data-driven pipeline based on Novelty Search with Local Competition (NSLC). Instead of relying on hand-crafted heuristics, the system operates within a behavioural space that is autonomously learned and dynamically maintained through unsupervised representation learning, utilising the deep neural architecture of a Variational Autoencoder (VAE).

The motivation for adopting an unsupervised behavioural descriptor stems from the inherent limitations of hand-crafted heuristics. Manually defining what makes a track "di-

verse" requires significant domain expertise and has to rely heavily on designer intuition, this risks constraining the algorithm's exploratory potential by inadvertently introducing human bias. By contrast, an autonomously learned descriptor extracts the underlying factors of variation directly from the raw driving telemetry, allowing the system to recognise complex, unexpected behavioural patterns such as speed profiles, steering patterns, and emergent driving lines. The Variational Autoencoder is specifically chosen for this task due to its regularised latent space. Unlike standard autoencoders, which often yield fragmented and disjointed embeddings, the VAE relies on Kullback-Leibler (KL) divergence regularisation to enforce a dense, continuous latent manifold. This structural density guarantees that the Euclidean distances between embedded track representations meaningfully reflect their true behavioural similarities, which is an essential property for calculating novelty and guiding the evolutionary search effectively.

The dynamic representation learning techniques employed in this work draw significant inspiration from the AURORA framework introduced by Antoine Cully [3]. By adapting these methodologies that were originally developed for autonomous skill discovery in robotics, this research directly transfers advanced behavioural exploration techniques to the procedural generation of track geometries for game design.

To evaluate the quality and to extract behavioural characteristics of the generated circuits, this work utilises The Open Racing Car Simulator (TORCS) as the core evaluation environment. By running automated, single and multi-agent race simulations, the system extracts rich dynamic telemetry, which directly informs both the behavioural descriptors and the Fitness objective. The underlying track layouts are synthesised using a procedural approach based on Voronoi diagrams and spline interpolation. This geometric representation provides a highly flexible foundation for the genetic operators to mutate and recombine.

The remainder of this thesis is structured as follows. Chapter 2 reviews the state of the art, providing the theoretical foundations of Procedural Content Generation, Quality-Diversity optimisation, and Variational Autoencoders. Chapter 3 details the simulation environment, outlining the integration of TORCS, the telemetry extraction process, and the Voronoi-based track generation pipeline. Chapter 4 presents the core evolutionary architecture, detailing the Novelty Search algorithm, the custom VAE design, and the dynamic representation learning routines. Chapter 5 discusses the experimental setup and provides a comparative analysis of the results, specifically evaluating the impact of dynamically fine-tuning the latent space during the evolutionary search. Finally, Chapter 6 summarises the conclusions drawn from this research and proposes potential avenues for future work.

2 | State of the Art

This chapter establishes the theoretical foundation and reviews the existing literature that underpins the methodology developed in this thesis. It first explores the evolution of Procedural Content Generation (PCG) and the principles of Quality-Diversity (QD) optimisation. Subsequently, it introduces Variational Autoencoders (VAEs), which provide the essential representation learning capabilities required to autonomously map and explore complex behavioural spaces.

2.1. Procedural Content Generation (PCG)

Procedural Content Generation (PCG) refers to the algorithmic creation of data or structures based on a set of predefined rules and parameters, standing in contrast to manual asset creation. By shifting the burden of content generation from designers to algorithms, PCG enables the creation of vast, varied, dynamic environments and content.

2.1.1. The Roots

In the 1960s, Aristid Lindenmayer, a Hungarian biologist, was studying the growth patterns of yeast, fungi, and algae. He realised that nature didn't grow randomly, it followed recursive mathematical rules. In 1968 Aristid Lindenmayer developed L-Systems [27], a formal grammar/logical system used to model and reproduce the patterns he found. This method is often recognised as the precursor to modern PCG and in some forms is still in use today for the generation of assets and art in several projects and games.

The first game to use PCG is widely attributed to Beneath Apple Manor (1978) by Don Worth [25], it is the first dungeon crawler to employ randomised level generation. However, the paradigm-shifting moment occurred with the release of Elite (1984) by David Braben and Ian Bell [26]. Faced with the limitation of fitting a "universe" onto a machine with 32KB of RAM, the developers utilised pseudo-random number generators (PRNGs) to seed the creation of an entire galaxy. By utilising the same seed, the game could reconstruct the same star systems without storing them, effectively turning the game's

code into a sophisticated map-making engine. This era established the foundation of PCG concepts and philosophy.

Following the success of *Elite*, PCG became a cornerstone of game development. As hardware capabilities expanded, the motivation for PCG shifted from a necessity of memory optimisation to a tool for artistic expression and scale. Games started to implement PCG as a fundamental part of the gameplay loop, introducing the concept of "replayability" by generating unique layouts, enemy placements, and item distributions for every session.

In recent years, PCG has entered a new era defined by the integration of Machine Learning (ML) and Generative AI. While traditional PCG relies on developer-defined heuristics and explicit rule sets, modern AI-driven approaches allow systems to learn complex patterns, structures, and stylistic nuances directly from existing datasets, representing a shift from rule-based to data-driven generation.

2.1.2. Taxonomy and Classification of PCG

Because Procedural Content Generation encompasses a massive variety of techniques, it is best understood by classifying it across two distinct axes: the **design dimensions** (how the system integrates with the game and player) and the **algorithmic methods** (the underlying computational techniques used to generate the content).

Design Dimensions

To categorise the design and architectural traits of PCG, we adopt the taxonomy proposed by Togelius et al. [23]. The authors identify several dimensions that distinguish how PCG systems are implemented.

A first distinction is between **Online and Offline** methods. *Online* methods generate content dynamically at runtime, responding to player actions or real-time game states. In contrast, *Offline* methods generate content prior to gameplay, ensuring stability and thorough testing before deployment.

It's also possible to distinguish between **Stochastic and Deterministic Generation**. *Stochastic* algorithms yield varying outcomes across different runs with the same parameters. *Deterministic* algorithms always produce the exact same content given the same parameters, often functioning as a form of data compression.

The impact of the content on gameplay separates between **Necessary and Optional Content**. *Necessary* content is critical for player progress (e.g., levels, dungeons, or puzzle rules), whereas *optional* content (e.g., decorative foliage or side-quests) can be

excluded or discarded if it fails to meet quality standards without breaking the game.

To represent the algorithm’s control regime it is possible to distinguish between **Random Seeds vs. Parameter Vectors**. At one extreme, an algorithm is driven by a single random number seed; at the other, it is guided by a multidimensional vector of specific parameters.

Based on the algorithmic architecture Togelius et al. oppose **Constructive to Generate-and-Test** methods. *Constructive* algorithms generate content in a single pass, ensuring validity by design (e.g., fractals). In contrast, *generate-and-test* algorithms create candidate content and subject it to an evaluation function, iterating until the content meets specific quality criteria.

Algorithmic Methods

While the previous dimensions describe the traits that define a PCG system, modern PCG relies on a wide variety of underlying mathematical and computational models. To address this diversity, we utilise the survey by Farrokhi Maleki and Zhao [6], which organises PCG algorithms into five distinct categories.

This algorithmic taxonomy naturally extends from Togelius’s *algorithmic* dimension, beginning with Search-Based methods:

Search-Based Methods (SBPCG) Operating primarily on the “generate-and-test” paradigm, these methods go beyond simple random generation by incorporating an iterative optimisation process. A search-based PCG system typically consists of two primary components: a *generator* that creates content variations and a *Fitness function* that quantifies the quality or “acceptability” of that content to guide the search. Specific techniques include:

- *Evolutionary Algorithms (EAs)*: The most established SBPCG approach, EAs manage a population of candidate content instances. Through successive generations, the system ranks candidates based on Fitness, discarding inferior ones while mutating or recombining superior candidates.
- *Planning Algorithms*: These algorithms focus on problem-solving by defining an initial state and a goal state. By producing a sequence of actions that transition the game from start to finish without causal threats, they are highly effective for procedural narrative and quest design.
- *Wave Function Collapse (WFC)*: A constraint-satisfaction algorithm that ensures

output adheres strictly to local patterns found in a set of input examples. It reduces possible cell states until a valid configuration is reached, making it an industry standard for level generation.

- *Monte Carlo Tree Search (MCTS)*: Originally prominent in game AI for move selection, MCTS can be used to search through vast combinatorial spaces (e.g., level layouts) by exploring trees where nodes represent states and edges represent transitions.
- *Simulated Annealing and Particle Swarm Optimisation (PSO)*: Heuristic optimisation techniques. Simulated Annealing tweaks content and occasionally accepts “worse” results to escape local optima. PSO draws inspiration from biological swarms, where individual agents converge on an optimal solution based on collective search history.

Machine Learning-Based Methods (MLPCG) These techniques represent a shift from explicit rule creation to data-driven generation, leveraging the capabilities of neural networks to model content from large datasets. Key approaches include:

- *Neural Architectures*: Including simple neural networks, Recurrent Neural Networks (RNNs) for sequential data, and Autoencoders (e.g., VAEs) for latent space representations.
- *Generative Models*: Primarily Generative Adversarial Networks (GANs), which use competing generator and discriminator networks to produce high-fidelity visual and spatial content.
- *Probabilistic Models*: Such as Markov Models and their multidimensional variants (MdMCs), which generate content based on conditional probabilities of state transitions.
- *Reinforcement Learning (RL)*: Where generation tasks are framed as Markov Decision Processes (MDPs), allowing agents to learn strategies that maximise content quality through rewards.

Large Language Models (LLMs) A recent disruption in the PCG field, LLMs utilise transformer architectures to process and generate content as sequences of text. Beyond narrative and dialogue generation, they are increasingly applied to level design and as interactive game masters.

Combined Methods This category encompasses hybrid approaches that integrate multiple algorithms to overcome the limitations of individual methods. A prominent example is the Latent Variable approach, which pairs ML models (GANs or VAEs) with evolutionary search (SBPCG) to explore the latent space. Similarly, RL is frequently combined with search-based methods to assess and repair generated structures.

Other Methods Foundational techniques that often act as the *constructive* building blocks for larger PCG systems:

- *Pseudo-Random Number Generators (PRNGs)*: The foundational technique for creating variance, often implemented through noise functions (e.g., Perlin noise).
- *Generative Grammars*: Rule-based systems for string replacement, including L-systems and shape grammars.
- *Generative Graphs*: Advanced structures used to move beyond the inherent linearity of traditional grammars.
- *Fractals*: Used for simulating natural phenomena through self-similarity and recursive subdivision.

2.2. Evolutionary Algorithms and Quality Diversity

Inspired by the ability of natural evolution to generate species that are well adapted to their environments, evolutionary computation has a long history in the domain of optimisation, particularly in stochastic optimisation. To understand how these methods operate, it is essential to define the standard terminology that forms the foundation of most Evolutionary Algorithms (EAs).

In an EA, the optimisation process mimics biological evolution through several core concepts. The algorithm directly operates on a **genotype** (or genome), which serves as the underlying representation or data structure of a candidate solution. When evaluated in the task environment, this genotype expresses a **phenotype**, which encompasses the observable characteristics, traits, or behaviours of the solution. During the execution of the algorithm, particularly in the context of Quality Diversity, a collection of these candidate solutions is continuously generated, maintained, and updated, forming what is known as a **population**.

To effectively explore the search space, EAs apply **genetic operators** to generate new offspring from existing solutions. These mechanisms primarily consist of **mutation**, which applies random perturbations to a genotype, and **crossover** (or recombination), which

combines segments of two or more parent genotypes. The algorithm then utilises a **selection** process to determine which individuals from the current population will survive or reproduce.

In traditional EAs, these mechanisms are typically driven by a singular objective: converging toward one globally optimal solution. However, from a more general perspective and in contrast with traditional artificial evolution, natural evolution does not produce one single effective solution, but rather a large, diverse set of different organisms that are all well adapted to their respective environments.

Quality-Diversity (QD) optimisation is a subfield of evolutionary algorithms that distinguish themselves from previous EAs by not focusing on finding a single best solution to a problem, but aiming for a large and diverse collection of solutions that are both High-Performing (Quality) and different from each other (Diversity).

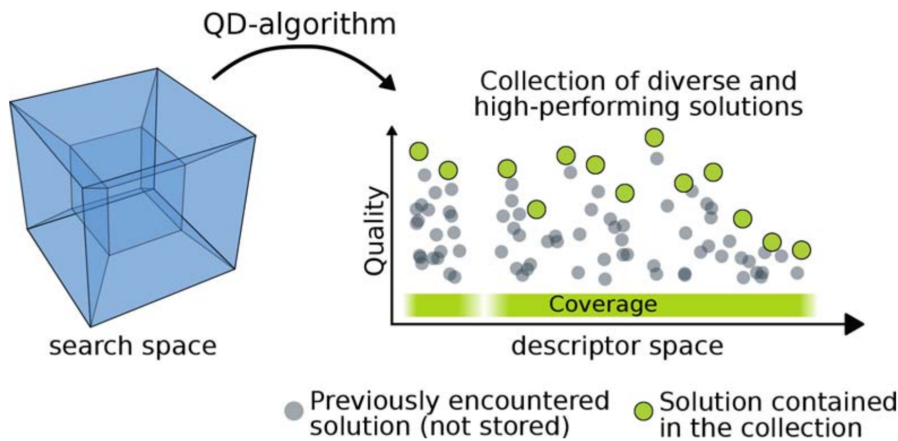


Figure 2.1: Overview of the Quality-Diversity components. Reproduced from [4]

To achieve this, QD algorithms rely on three main components (see Fig. 2.1):

Fitness Function (Quality) The metric measuring how well a solution solves the problem.

The Behavioural Descriptor(BD) (Diversity) A metric that captures how a solution behaves, functions or looks, it depends on the task and on what type of diversity the algorithm is trying to explore.

The Archive The algorithm maintains a map of the search space. Whenever a new solution is generated, the algorithm checks its behaviour. If it exhibits a new behaviour,

it is added to the archive. If it exhibits a behaviour already in the archive, it competes with the existing solution, and the one with the higher Fitness value is kept.

The two main algorithms form the foundations of QD are **MAP-Elites (Multi-dimensional Archive of Phenotypic Elites)** and **Novelty Search with Local Competition (NSLC)**: **MAP-Elites** divides the behavioural space into a discrete grid. It searches randomly, and if a solution lands in a grid cell, it is kept if that cell is empty or if it outperforms the current occupant. Instead of using a rigid grid **NSLC** uses an unstructured archive. It rewards solutions for being far away from other solutions (Novelty) while simultaneously competing locally with their nearest neighbours for Fitness.

Recent advancements include **Hybrid Algorithms** that combine QD with powerful traditional optimisers, such as CMA-ME (Covariance Matrix Adaptation MAP-Elites) or integrate QD with Deep Learning techniques.

2.3. AURORA: AUtonomous RObots that Realize their Abilities

While Quality-Diversity algorithms have proven highly effective at generating diverse and high-performing collections of solutions, they suffer from a significant bottleneck: they traditionally require a human expert to manually define the Behavioural Descriptor (BD). The choice of the BD is crucial, as it fundamentally dictates what the algorithm considers novel or diverse, thereby driving the entire exploration process. Defining this descriptor requires significant domain expertise and becomes a major bottleneck when exploring complex systems or novel task environments where the full spectrum of potential outcomes is not known in advance.

To address this limitation, Antoine Cully introduced **AURORA** (AUtonomous RObots that Realize their Abilities) [3], a novel framework that automates the definition of the BD in the field of robotics. AURORA achieves this by coupling Quality-Diversity optimisation with unsupervised **Dimensionality Reduction (DR)** algorithms, allowing robots to autonomously discover the range of their capabilities through self-exploration without requiring prior domain knowledge. A summary Fig. 2.2 is provided below.

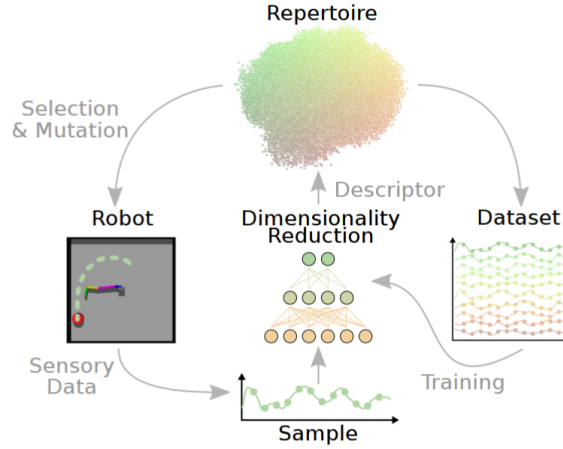


Figure 2.2: A summary image of the AURORA pipeline. Reproduced from [3]

2.3.1. Core Mechanism

The fundamental concept behind AURORA is to leverage the vast collection of solutions generated by the QD algorithm as a training dataset for a Dimensionality Reduction model [3]. In turn, the latent (low-dimensional) representation learned by the DR model serves as the BD for the QD algorithm.

The process unfolds in the following phases:

- **Initialisation:** The QD algorithm is initialised with a set of randomly generated controllers. As these controllers are evaluated in the environment, the robot collects high-dimensional sensory data (e.g. complete time-series of trajectories).
- **Latent Space Creation:** A DR algorithm is trained on this initial dataset of sensory data to learn a latent, low-dimensional representation. The high-dimensional sensory data of each controller is projected into this latent space, and this projection is used as the BD.
- **Exploration (QD Iteration):** The algorithm proceeds with a standard QD loop (selection, mutation, evaluation). When attempting to place a new solution into the archive, the sensory data produced by the new controller is fed into the DR algorithm to determine its coordinates in the latent space (its BD).
- **Latent Space Refinement:** As the algorithm discovers new controllers and generates novel sensory data, the DR algorithm is periodically re-trained. Following a DR update, the latent space changes, so all controllers previously stored in the archive are reassigned new BDs and re-sorted into the container. Because the robot dis-

covers new behaviours more frequently at the beginning of the process, the update frequency decays exponentially over time [3].

2.3.2. Dimensionality Reduction Algorithms

AURORA is agnostic to the specific DR algorithm used, so long as it can act as an embedder from the original sensory space to a latent space. The original study [3] evaluates two primary DR techniques:

- **Principal Component Analysis (PCA):** A traditional machine learning approach that finds a linear projection into an orthogonal base. It captures the variables that represent the highest variance in the data [12].
- **Autoencoders (AE):** Deep neural networks consisting of an encoder and a decoder. The encoder compresses the high-dimensional data into a low-dimensional latent space, while the decoder attempts to reconstruct the original data. Unlike PCA, Autoencoders can capture non-linear relationships in complex sensory data.

When applied to simulated robotic tasks, AURORA has demonstrated the ability to operate effectively in complex, high-dimensional environments where traditional QD approaches without prior knowledge typically struggle [3]. Relying entirely on unsupervised sensory data, the framework generates behavioural repertoires that are just as diverse and comprehensive as those produced using manually hand-crafted descriptors. Ultimately, AURORA represents a significant step forward for autonomous skill discovery; by removing the dependency on human-engineered heuristics and integrating representation learning directly into the evolutionary loop, it allows automated systems to independently learn, map, and master their own operational boundaries.

2.4. Variational Autoencoder (VAE)

2.4.1. The Autoencoder

An autoencoder (AE) is a type of artificial neural network that applies unsupervised learning to compress (encode) input data down to its essential features, then reconstruct (decode) the original input from this compressed representation. [13]

Autoencoders are trained to discover latent variables of the input data: hidden or random variables that, despite not being directly observable, fundamentally inform the way data is distributed. Collectively, the latent variables of a given set of input data are referred to as **latent space**. During training, the autoencoder learns which latent variables can be

used to most accurately reconstruct the original data: this latent space representation thus represents only the most essential information contained within the original input. Most types of autoencoders are used for artificial intelligence tasks related to feature extraction, like data compression, image de-noising, anomaly detection and facial recognition. [13]

An autoencoder has two main parts an **encoder** and a **decoder** (see Fig. 2.3):

- **The Encoder:** A neural network that ingests the high-dimensional input data $\mathbf{x}$ and maps it to a lower-dimensional latent representation $\mathbf{z}$. Mathematically, this can be represented as a deterministic function $\mathbf{z} = f_\phi(\mathbf{x})$, where ϕ represents the weights and biases of the encoder network.
- **The Decoder:** A subsequent neural network that takes the compressed latent vector $\mathbf{z}$ and attempts to reconstruct the original input, outputting $\hat{\mathbf{x}}$. This is represented as $\hat{\mathbf{x}} = g_\theta(\mathbf{z})$, where θ represents the parameters of the decoder.

A standard autoencoder is trained by minimising a reconstruction loss, such as Mean Squared Error (MSE) or Cross-Entropy, which measures the difference between $\mathbf{x}$ and $\hat{\mathbf{x}}$. While highly effective for data compression and de-noising, standard AEs are inherently deterministic. Consequently, they tend to over-fit the training data and produce a fragmented, discontinuous latent space. If one were to sample a random point from the latent space of a standard AE and pass it through the decoder, the output would likely be nonsensical. This lack of structure makes standard AEs unsuitable for generative tasks, where the goal is to create entirely new, realistic data samples.

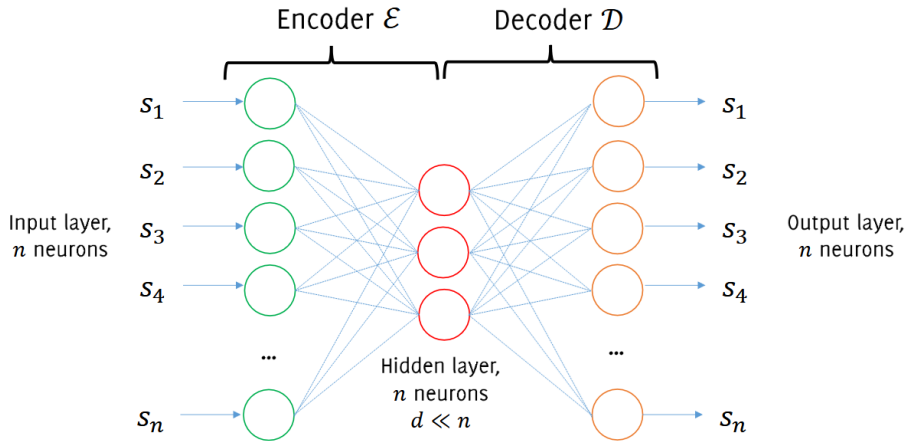


Figure 2.3: Typical Structure of an autoencoder [1]

2.4.2. From Deterministic to Probabilistic: The VAE

Standard autoencoders rely solely on minimising reconstruction loss, meaning the network has no incentive to organise the latent space systematically. The encoder maps training samples to isolated, discrete points in the latent space, often grouping similar data into tight clusters that are separated by vast, unmapped regions. If one were to sample a random latent vector from one of these "gaps" between clusters and pass it through the decoder, the network would fail to output a coherent reconstruction. Because the decoder was never trained on latent vectors originating from these empty regions, it has no meaningful mapping for them, resulting in nonsensical or highly distorted outputs.

In summary, a standard autoencoder does not learn a continuous manifold; it acts more like a complex lookup table. To perform generative tasks, the network requires a mechanism to actively regularise the latent space, forcing it to be dense, contiguous, and bounded by a known probability distribution from which new samples can be drawn.

To overcome these limitations, Diederik P. Kingma and Max Welling introduced the **Variational Autoencoder (VAE)** [15]. The fundamental innovation of the VAE is that it shifts the architecture from a deterministic framework to a probabilistic one, specifically to enforce regularisation upon the latent space.

Instead of encoding an input $\mathbf{x}$ into a single, fixed vector $\mathbf{z}$, the VAE encoder outputs parameters that describe a probability distribution for each latent attribute. Typically, this is assumed to be a multivariate Gaussian distribution, meaning the encoder outputs a mean vector ($\boldsymbol{\mu}$) and a variance vector ($\boldsymbol{\sigma}^2$). During training, the actual latent vector $\mathbf{z}$ passed to the decoder is randomly sampled from this distribution.

Encoding to a distribution rather than a point solves the fragmentation problem through two distinct mechanisms [15]. First, it introduces **local smoothing via stochasticity**. Because the encoded vector is drawn from a distribution, this is done by adding noise to the mean $\boldsymbol{\mu}$, the exact latent vector $\mathbf{z}$ changes slightly every time a specific data point is passed through the network. To achieve a low reconstruction loss, the decoder must learn to successfully reconstruct the original input not just from a single point, but from an entire local "cloud" or neighbourhood of points surrounding the mean. This forces local continuity, ensuring that points close to each other in the latent space naturally decode to similar outputs.

Second, the VAE relies on **KL divergence regularisation**. If left unchecked, the network could theoretically "cheat" the stochasticity by shrinking the variance of the posterior ($\boldsymbol{\sigma}^2 \rightarrow 0$) and placing the means infinitely far apart, thereby reverting to a standard,

fragmented autoencoder. To prevent this, VAEs introduce a **regularisation term** to the loss function using the Kullback-Leibler (KL) divergence [16]. This term forces the learned **approximate posterior** $q_\phi(\mathbf{z}|\mathbf{x})$ to closely match a predefined **prior distribution** $p(\mathbf{z})$, which is usually a standard normal distribution, $\mathcal{N}(\mathbf{0}, \mathbf{I})$. By pulling the distributions toward this prior, the network forces all individual latent distributions to centre around the origin and significantly overlap with one another.

Ultimately, by forcing these distributions to overlap, the VAE eliminates the "gaps" in the latent space, transforming it into a dense, continuous, and structured manifold. Consequently, one can sample any random point from the standard normal prior distribution, and the decoder—having been trained on heavily overlapping regions—will seamlessly interpolate the features and generate a novel, highly realistic data sample.

3 | Simulation Environment

This chapter presents the technical infrastructure used to generate and evaluate candidate racing tracks. It first details the integration of The Open Racing Car Simulator (TORCS) for extracting dynamic driving telemetry. Following this, it outlines the geometric pipeline developed to synthesise drivable, closed-loop circuits for the simulator.

3.1. TORCS

In this study we selected the open-source project TORCS [28] as our simulation platform. TORCS is a 3D racing platform with an established history of use in both entertainment and academic research. We chose it for its modularity, portability and ease of integration for our research purposes. It provides a suitable environment for procedurally generating and running simulations on racing tracks.



Figure 3.1: A gameplay screenshot from The Open Racing Car Simulator (TORCS).

First developed in 1997, TORCS has become a foundational platform for AI research, notably featured in competitions at the IEEE Conference on Computational Intelligence and Games [17]. It features a highly detailed physics engine that simulates comprehensive

vehicle dynamics at a discrete 0.002s time step. Through its low-level API, researchers can access the simulation state to develop complex driving agents, or “robots.” These multi-agent capabilities allow simulated cars to independently process track information to calculate optimal racing lines, vehicle setups, and pit strategies. This combination of features makes TORCS an ideal framework for our research. Its open-source architecture permitted the core modifications necessary to integrate custom telemetry modules for our analysis. Furthermore, TORCS natively supports the addition of custom tracks, providing a robust environment for procedural generation. While forks of TORCS exist, they are less suited to our methodology. Speed Dreams, for instance, focuses on enhanced graphics and refined gameplay for human players, while pyTORCS ports the platform to Python but alters core modules. Consequently, the original TORCS platform remains the most robust choice for this research.

3.1.1. Telemetry and Vehicle dynamics

When running simulations, we collect data using custom telemetry tools. These tools rely on modifications of the TORCS 1.3.7 C++ engine and the Python API originally developed by Sirianni [21], which provide the ability to capture raw race metrics and compute statistics. To suit our specific use case, we further modified both the engine and the API, allowing us to solve domain-specific issues, such as the closure problem (see section 3.2.3), and to extract the precise data needed for our research.

While these tools can capture a wide array of telemetry data, we detail below only the metrics most pertinent to our research, organised by their extraction context.

Track Geometry Metrics

Metric	Description
Track length	The length of the track perimeter in m
Closure error (ΔX , ΔY , $\Delta\theta$)	Deviation of the track endpoints from perfect closure (see section 3.2.3)
Bend counts	Number of left bends, right bends, and straight sections
Curvature statistics	Mean and variance of corner radii

Table 3.1: Track geometry metrics extracted from the generator.

The structural metrics presented in table 3.1 are directly extracted from the TORCS track

generator (`trackgen`).

Benchmark Lap Metrics

For each race track, before the actual race a benchmark simulation of 3 laps is executed using a single vehicle controlled by the `tita` bot (described in section 3.1.2). Telemetry is recorded during the second lap, providing a complete lap under stable driving conditions. The six telemetry channels summarised in table 3.2 are sampled every 200 ms.

Benchmark Lap Metric	Description
Speed	Car speed in m/s.
Steering angle	Value of the steering command between -1 and 1.
Throttle input	Value of the acceleration input between 0 and 1.
Brake input	Value of the brake input between 0 and 1.
Gear	Integer value of the engaged Gear: -1 (reverse), 0 (neutral) up to the number of gears of the car.
Distance from track boundary	Distance computed from the centre of gravity of the car to the right edge of the main road segment.

Table 3.2: Telemetry channels recorded during the benchmark lap.

Furthermore, the benchmark simulation is used to determine the **number of laps** for each race, given a fixed **target race duration** of 360 seconds. This choice of duration was motivated by the observation that the track lengths and inter-bot interactions are more representative of rallycross-style or arcade racing games, rather than highly realistic simulators.

The number of laps is then computed as:

$$lapCount = \left\lceil \frac{\text{target duration}}{\text{second benchmark lap time}} \right\rceil$$

Having a fixed **target race duration** allows **Emergent Race Metrics** to be directly comparable between different simulations without having to account for different track lengths.

Emergent Race Metrics

The metrics detailed in table 3.3 are computed from simulations involving four heterogeneous bots: *lliauw*, *berniw*, *tita*, and *inferno*. Each simulation is run over a number of laps determined by the time-normalised procedure described in the previous section.

To mitigate bias introduced by the initial grid ordering, each configuration is executed four times with different starting positions. The reported values correspond to the average across these runs.

Emergent Race Metric	Description
Speed Entropy	$H_{speed} = - \sum_{i=1}^{30} p_i \log_2(p_i)$
Acceleration Entropy	$H_{accel} = - \sum_{i=1}^{30} p_i \log_2(p_i)$
Braking Entropy	$H_{brake} = - \sum_{i=1}^{30} p_i \log_2(p_i)$
Total Overtakes	The total number of successful passing manoeuvres between all vehicles during the race
Mean and Variance of Inter-Bot Gaps	Average time separation between consecutive cars on track.
Mean and Variance of Race Position	How much drivers reshuffled from their starting grid positions.

Table 3.3: Emergent metrics computed from the multi-agent race simulations.

Several of the metrics proposed in table 3.3 rely on **Shannon entropy** [20] to quantify the variability and complexity of different aspects of gameplay. To compute this for continuous features, the observed values are first discretised into a 30-bin histogram. The probability p_i is defined as the fraction of the respective feature’s samples—such as speed, throttle input, or brake input—falling within bin i .

3.1.2. Vehicles and AI drivers

For our research we chose the AI bots from an array of combinations of built-in drivers and cars, we chose the drivers based on a visual inspection of various race simulations and chose the most dynamic and competitive combination. Following we present a description of each driver (table 3.4) and car.

AI Driver	Description	Car
berniw	The AI brakes earlier and carries less speed through corners than the other bots would in the same car, it struggles with overtaking and is generally slower and more cautious.	car1-trb1
lliaw, tita, inferno	These bots share the same AI. They have an aggressive driving style that commits to corners fast and brakes late. It runs very close to the track edge, and pursues overtakes whenever it has space ahead. Behaviour differences between these bots are warranted by the use of different cars.	<i>car1-trb1</i> , <i>car2-trb1</i> , <i>car4-trb1</i> respectively.

Table 3.4: The AI driver and car combinations selected for the race simulations, with a qualitative description of each driver’s behaviour.

Car Detail:

- *car1-trb1*: This vehicle represents the most balanced and accessible driving character. It responds predictably to driver inputs across a wide range of cornering conditions and rewards consistent, methodical driving over aggressive or reactive control strategies. Its behaviour remains stable under braking and through corner entry, with no strong tendency to favour one type of circuit over another.
- *car2-trb1*: This vehicle is characterised by a natural tendency to rotate, placing greater demands on the timing and precision of throttle application. Power delivery is subject to a pronounced delay relative to driver input, such that the vehicle’s response to acceleration commands is both deferred and sudden, a property that penalises reactive driving strategies and rewards those capable of anticipating corner conditions well in advance. Its agility in changing direction makes it potentially faster through technical sections, but at the cost of increased instability when inputs are poorly timed.
- *car4-trb1*: This vehicle is the most physically imposing to steer through changes of direction, exhibiting a reluctance to alter its path that requires earlier and more committed steering decisions. In compensation, its power delivery is the most immediate of the three: the engine responds to throttle inputs without delay, offering a straightforward and transparent relationship between driver action and vehicle acceleration. Its character favours circuits with long open sections over densely

technical layouts, and tends to reward driving lines that minimise the need for abrupt directional change.

3.1.3. Headless execution and Dockerisation

To accelerate simulation speeds and enable horizontal scaling, TORCS was operated exclusively in its non-graphical (headless) execution mode.

Parallelising TORCS inherently presents challenges regarding legacy dependency management and file log collisions. We resolved these issues by employing Docker [5] to isolate the simulation environment. Configured via a custom `Dockerfile`, the environment is built on an Ubuntu base and includes the X Virtual Framebuffer (Xvfb) to support headless rendering, as well as our telemetry modules.

System orchestration is handled by a Node.js server utilising an ephemeral container strategy. Rather than reusing a persistent pool of environments, the server provides a fresh, single-use container for each track evaluation and destroys it upon completion. This isolation ensures architectural resilience; a frozen or crashed simulation can be safely timed out and purged without compromising the stability of other concurrent or subsequent evaluations.

3.2. Track generation and solution

3.2.1. The TORCS representation

To be successfully rendered by the simulator's engine, a racing track must adhere to a specific XML schema. This XML file is what the engine uses to generate and render the 3D environment and is divided into two main components:

- **Global Parameters:** Sections at the beginning of the file establish general track settings, including road width, barrier heights, and the physical friction and resistance properties of the track surfaces.
- **Track Segments:** This section defines the sequential geometry of the layout. Each piece is defined by a type attribute (`str` for straight, `lft` for left, or `rgt` for right). Straight segments are parametrised only by their length, whereas curved segments require both a radius and an arc angle.

It is important to note that the XML schema relies entirely on relative sequential generation rather than spatial coordinate data. The track is built segment-by-segment in the

exact order they are listed. As a result, the algorithmic generation of the track must ensure geometric consistency and proper circuit closure upstream (see section 3.2.3) before the data is written to the XML file.

3.2.2. Voronoi spline generation

Working directly with the intrinsic XML Representation would be difficult, so we opted for a new representation for the purpose of generation that allows us to easily manipulate the global shape of the layout while mathematically guaranteeing a continuous, closed-loop structure. Specifically, we adopted a procedural approach based on Voronoi diagrams and spline interpolation.

The advantages of Voronoi diagrams in this context are numerous. Most notably, extracting boundaries from a contiguous subset of Voronoi cells inherently produces closed-loop paths, entirely bypassing the need for additional closure logic in the generation phase. Furthermore, the flexibility in manipulating seed point distributions opens to the exploration of diverse track geometries, spanning from simple layouts to intricate patterns. These benefits are further amplified by the computational efficiency of standard Voronoi generation algorithms.

Site Generation

Sites are distributed within a bounding domain according to one of two strategies, controlled by a generation mode parameter:

Uniform Sites are drawn independently and uniformly at random from the rectangular domain using a seeded pseudo-random number generator.

Simplex noise based A more complex generator that allows us to obtain spatially non-uniform site distributions which translate into regions of varying track complexity. Candidate positions are accepted or rejected through a density field derived from two-dimensional **Simplex Noise** [18]. A candidate at position (x, y) is accepted with probability $p(x, y)$:

$$p(x, y) = \max(0, n(x, y) - \tau)^\delta \quad (3.1)$$

- where $n(x, y) \in [0, 1]$ is the noise value rescaled from its native $[-1, 1]$ range to $[0, 1]$
- τ is a density threshold that controls the overall sparsity
- δ is an exponent sharpening the contrast between high and low-density regions

A grid-based occupancy constraint prevents any two accepted sites from falling within the same cell, ensuring a degree of spatial regularity. The noise frequency governs the spatial scale of the resulting clusters.

Voronoi Diagram Computation

Given the set of sites $S = \{s_1, \dots, s_N\}$, a planar Voronoi diagram is computed using the Fortune's sweep-line algorithm [7]. Each cell C_i is defined as the region of the plane closer to s_i than to any other site:

$$C_i = \{p \in \mathbb{R}^2 \mid \|p - s_i\| \leq \|p - s_j\| \ \forall j \neq i\} \quad (3.2)$$

The resulting diagram (Fig. 3.2) partitions the domain into convex polygons whose shared edges form the graph used in subsequent stages.

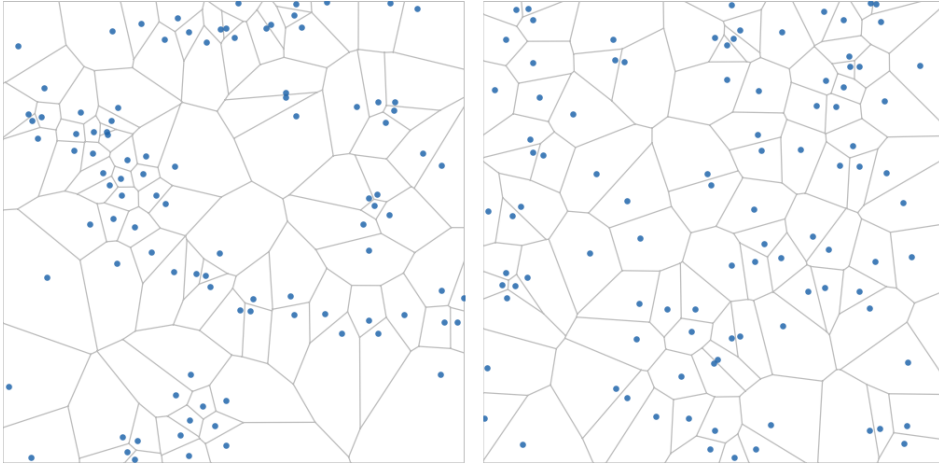


Figure 3.2: Simplex noise (left) yields uneven Voronoi cells, small in clusters, large across voids. Uniform random (right) produces more evenly sized cells.

Cell Selection

A connected subset of cells is chosen to define the interior of the track. The number of selected cells controls the overall track length and complexity.

This cell selection can be done with two different approaches: A **Random walk (in generative mode)** and by **Targeted Selection**. The **Random walk** starts selection from the cell closest to the domain centre, a random walk proceeds on the cell adjacency graph (where two cells are adjacent if they share a Voronoi edge), appending unvisited neighbours until the desired number of cells has been collected.

Alternatively, the track layout can be defined by an explicit list of **target coordinates** provided by an external process. Each coordinate pair is mapped to its corresponding Voronoi cell. Because an arbitrary set of target points may resolve to structurally disconnected regions (which would fail to produce a single, continuous track), a connectivity repair step is strictly enforced to guarantee a closed-loop track. First, a union-find data structure is utilised to group the initially selected cells into contiguous connected components. If multiple disjoint components are detected, a breadth-first search (BFS) algorithm computes the shortest adjacency path between them. The intermediate bridging cells found by the BFS are then appended to the overall selection, ensuring a single, uninterrupted geometric structure.

The final **track boundary** corresponds to the outer perimeter of the selected cell region. An edge belongs to this boundary if and only if it is shared by exactly one selected cell that is not interior to the selected region. These *external edges* are collected and traversed sequentially by following the vertex adjacency chain, yielding an ordered, closed polygonal outline of the selected cells.

Spline Smoothing

The raw boundary polygon has sharp corners at Voronoi vertices and is unsuitable as a racing circuit. A multi-pass smoothing procedure is applied. In each refinement iteration, three operations are performed in sequence.

1. First, Catmull-Rom spline interpolation [2] subdivides each segment, producing a locally smooth curve through the control points according to

$$\mathbf{p}(t) = \frac{1}{2} \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad (3.3)$$

where $\mathbf{p}(t)$ is the interpolated point at curve position $t \in [0, 1]$, tracing the segment between control points $\mathbf{p}_1$ and $\mathbf{p}_2$, while the neighbouring points $\mathbf{p}_0$ and $\mathbf{p}_3$ set the tangents.

2. Second, a "push-apart" step merges any two points that fall within a minimum distance of one another, preventing degenerate micro-segments.
3. Third, an angle-correction step adjusts the position of any point that forms too sharp a turn with its neighbours, preventing hairpin corners that would be undrivable.

After the iterative passes, a final high-resolution Catmull-Rom pass produces a dense output polyline, examples are provided in Figure 3.3.



Figure 3.3: Two generated track lines after cell selection and smoothing steps, simplex noise (left) uniform (right); the simplex noise based generation displays its potential of generating large straight sections in sparse areas, whilst cluster enforce a winding topology.

Spline Canonicalisation

Once the spline has been generated, a sequence of normalisation steps is applied to ensure geometric validity and a consistent representation. This canonicalisation is essential for the efficacy of the subsequent Quality Diversity (QD) algorithm.

Winding order. The orientation of the point sequence is standardised to remove redundancy in the search space. From the perspective of vehicle dynamics, two tracks that are mirror images of one another often do not represent a meaningful difference in driving complexity.

To consolidate these mirrored shapes into a single canonical form, we calculate the signed area of the closed polygon using the shoelace formula:

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \quad (3.4)$$

If the area A is positive, indicating a counter-clockwise winding in Cartesian coordinates, the sequence is mirrored. This ensures all tracks are represented with a uniform clockwise orientation, preventing the QD algorithm from treating mirrored versions of the same geometry as distinct individuals (Fig. 3.4).

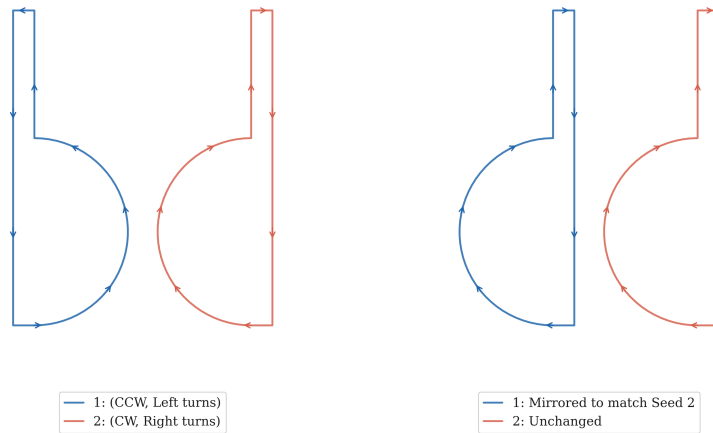


Figure 3.4: Winding canonicalisation on mirrored tracks

It is important to note, however, that this specific method of canonicalisation is designed only to resolve geometric mirroring. It intentionally does not consolidate "reverse" tracks (Fig. 3.5). Because reversing the traversal of a track changes the sequence of corners and apexes, it represents a distinctly different driving behaviour.

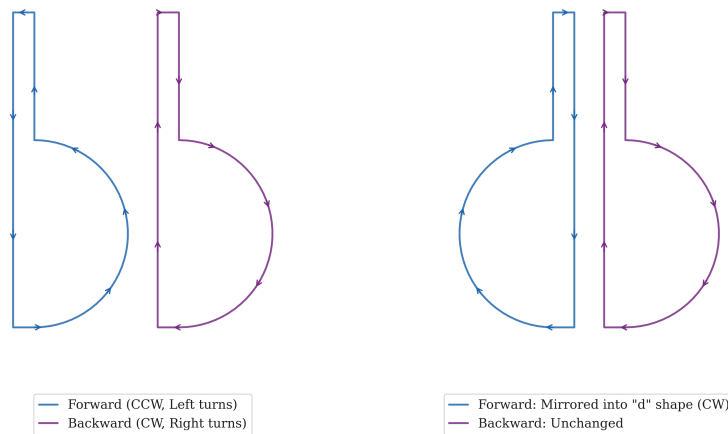


Figure 3.5: Winding canonicalisation on reversed track

Start position. Because the track is a closed loop with no inherent starting point, a canonical entry point must be chosen algorithmically. The polyline is scanned for its longest low-curvature segment—the section closest to a straight line in terms of angular deviation between consecutive points. The array sequence is then cyclically shifted so

that this segment serves as the start and finish of the circuit, corresponding to a natural placement of the starting line on a straight.

Self-intersection check. We implemented a segment-intersection test on all non-adjacent edge pairs to prevent the inclusion of self-intersecting geometries. Any track exhibiting an intersection is discarded. This constraint is necessary due to the limitations of the TORCS simulator, which cannot natively process overlapping track segments. Such geometries lead to erratic simulation behaviour and corrupted telemetry, rendering the results unusable for our purposes. Furthermore, this also avoids an edge case of the shoelace formula (eq. (3.4)): for self-intersecting polygons, sub-regions wound in opposite directions cancel out in the algebraic area sum, making the sign of the result an unreliable indicator of the overall winding order.

3.2.3. Conversion to TORCS XML

As established in section 3.2.1, TORCS represents racing circuits as an ordered sequence of geometric primitives rather than a raw point array. Translating the discrete spline-generated polyline into this format requires a two-stage process: segmenting the point sequence into discrete geometric types, and ensuring the final primitive list forms a seamless loop.

Spline segmentation

The spline is traversed point by point, and each local neighbourhood is classified as either straight or curved based on a curvature threshold. Contiguous low-curvature points are merged into a single straight segment. Where curvature exceeds the threshold, a circular arc is fitted to three consecutive points.

The Closure Error

A limitation of this segmentation approach is that it introduces a series of geometric errors, which accumulate into a **closure error** at the end of the track.

Converting low-curvature parts of the track into straight sections will inherently misalign the two representations, but more importantly, it introduces a **tangent discontinuity** at every straight-curve junction. Because TORCS integrates the track forward, starting each primitive at the heading θ_s inherited from the previous segment, it does not account for the fitted arc's actual entry tangent θ_c .

The disparity between the inherited heading and the fitted geometry ($\theta_s \neq \theta_c$) causes a

rotational shift that displaces the segment’s exit coordinates. Because these errors are compounded at every junction in the sequence, the final primitive fails to align with the starting point, leaving a residual position error $(\Delta x, \Delta y)$ and a heading error $\Delta\theta$.

Closure Correction

To handle the closure error, we devised two complementary solutions: a primary approach that modifies the track segments directly, and a fallback solution relying on a patch to the TORCS engine.

The segment-based approach operates in two sequential steps: a heading correction followed by a position correction.

Heading correction. The track is integrated forward from a fixed starting pose to obtain the net heading error $\Delta\theta$ at the point of closure. This angular discrepancy is distributed across all curve sections, with each curve absorbing a share proportional to its arc angle relative to the total absolute arc of the circuit. The correction is applied with the appropriate sign depending on turn direction, ensuring that left and right curves contribute in the correct rotational sense. After this step, the track closes with zero heading error, though a residual position error $(\Delta x, \Delta y)$ typically remains.

Position correction. The position error is corrected by adjusting the lengths of the straight sections. Each straight, traversed at a fixed heading determined by the sections before it, contributes a displacement vector to the total path. The problem of finding length adjustments that cancel $(\Delta x, \Delta y)$ is therefore a linear system, and the solution that minimises the total correction in the least-squares sense is computed analytically. A non-negativity constraint is enforced throughout, as TORCS silently discards segments with negative length.

If the primary solver fails, for instance, when all straights are parallel and the system is degenerate, a fallback method is applied. This distributes the correction proportionally across straights according to each segment’s length and how strongly its direction aligns with the error vector, clamped to prevent negative lengths. This fallback may be applied iteratively until the residual error falls within an acceptable tolerance or a maximum number of iterations is reached.

TORCS engine patch. While the segment-based methods are able to completely remove the gap on the vast majority of tracks, there are still rare occasions where a small gap remains visible at the end of the circuit. To resolve this, we applied a direct patch to the C++ TORCS track generator. This patch forces the mesh to form a closed loop by

directly modifying the final geometry to bridge any remaining gap.

4 | Evolutionary Pipeline

Our purpose is to develop an automated, data-driven system capable of autonomously discovering a diverse and high-performing collection of racing track designs. To achieve this we chose the following pipeline, it largely takes inspiration from Cully’s work for robotics in AURORA [3] but expands on it by implementing ad hoc features and solutions. The proposed system implements a Quality-Diversity (QD) optimisation pipeline, specifically Novelty Search with Local Competition, integrated with deep representation learning. Its objective is to autonomously discover a diverse, high-performing collection of solutions (in our case track designs) by continuously exploring a dynamically learned behavioural space.

Based on the PCG taxonomy established in section 2.1.2, our system can be formally classified across both its design dimensions and algorithmic methods. In terms of design, it operates as an **Offline, Stochastic, Generate-and-Test** architecture that produces **Necessary** game content. While the initial track generation relies on **Constructive** mathematical building blocks (Voronoi diagrams and splines), the overall optimisation loop relies heavily on test-driven evaluation.

4.1. The Novelty Search algorithm

4.1.1. The Pyribs Framework

Pyribs [22] implements a modular conceptual framework called RIBS, which unifies methods such as Novelty Search, Map-elites and other variations. The framework is built on three main components that work in concert:

1. **The Archive:** Acts as the memory of the system, responsible for storing the highest-performing and most diverse solutions (elites) discovered during the search (following the standard nomenclature of the QD field, popularised by the MAP-Elites algorithm, these archived solutions will be referred to as "elites" throughout this work). It defines the structure of the behavioural space and dictates the criteria for admission.

2. **The Emitter:** Serves as the engine of exploration and variation. It is responsible for generating new candidate solutions, typically by sampling existing elites from the archive and applying search operations to explore new regions of the design space.
3. **The Scheduler:** The Scheduler is the high-level orchestrator that coordinates the interaction between the Archive and the Emitters. It follows an "ask-tell" interface: it "asks" the emitters to generate a new batch of solutions. After these solutions have been evaluated externally by the user's own system, the Scheduler is "told" the results, which it then passes to the emitters and the archive to update their respective states.

4.1.2. Pipeline overview

Our pipeline (Fig. 4.1) operates through a continuous cycle for which each iteration is a sequence of generation, evaluation, archiving, and representation updating.

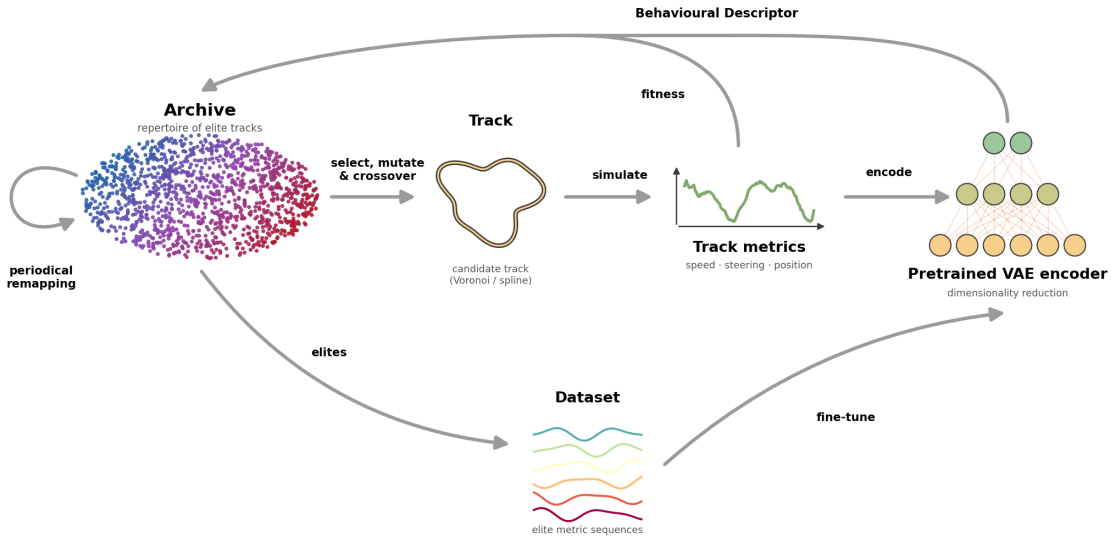


Figure 4.1: Overview of our Quality Diversity pipeline.

4.1.3. Generation & Variation (Ask)

At each iteration the Emitter generates a batch of candidate solutions, defined by the `BATCH_SIZE` parameter. To create an initial population the first iterations are generated using our Voronoi Spline Generator (see section 3.2.2) alternating between uniform and

Simplex noise based generation (a 50/50 split). After a set `POPULATION_ITERATIONS` we switch to applying genetic operators to the individuals currently in the archive.

Genetic Operators: Genetic Operators used in our work are adopted from the methodology presented in [19], where they are described and justified in detail. Following we present a qualitative description:

Mutation: The algorithm selects an existing "Elite" track and applies a small "nudge" to its geometry. A Voronoi site is randomly selected and displaced by a random offset. Because of the mechanisms of the Voronoi-Spline pipeline (section 3.2.2), even a small movement of a site can result in a smooth, drivable change in a corner's radius or a straight's length, maintaining the structural integrity of the circuit. A mutation example is provided in the following Fig. 4.2.

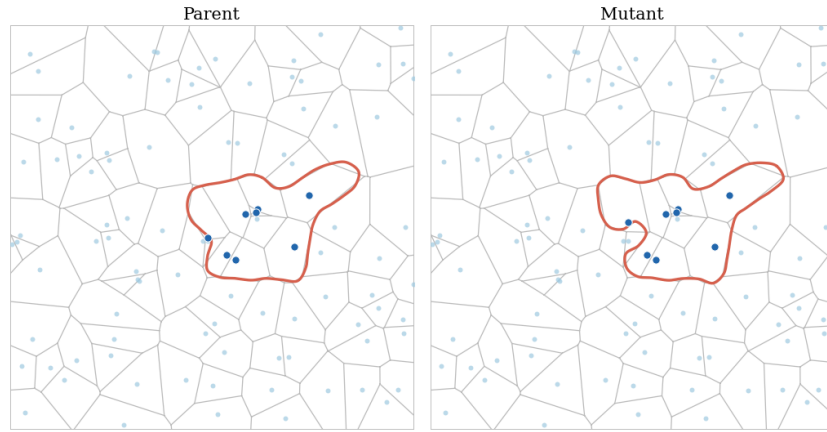


Figure 4.2: Mutated track and its parent: a small change in a site's position heavily accentuates a previously soft curve.

Crossover: Two parent tracks are aligned using a reference point on each (the first Voronoi Site), and one parent's track layout is shifted so that this reference point matches the other's. The shifted layout is then merged with the first parent's layout, discarding any sites that would end up too close together. A connectivity repair step is then performed to ensure the resulting hybrid forms a single, continuous, and closed loop. A crossover example is provided in the following Fig. 4.3.

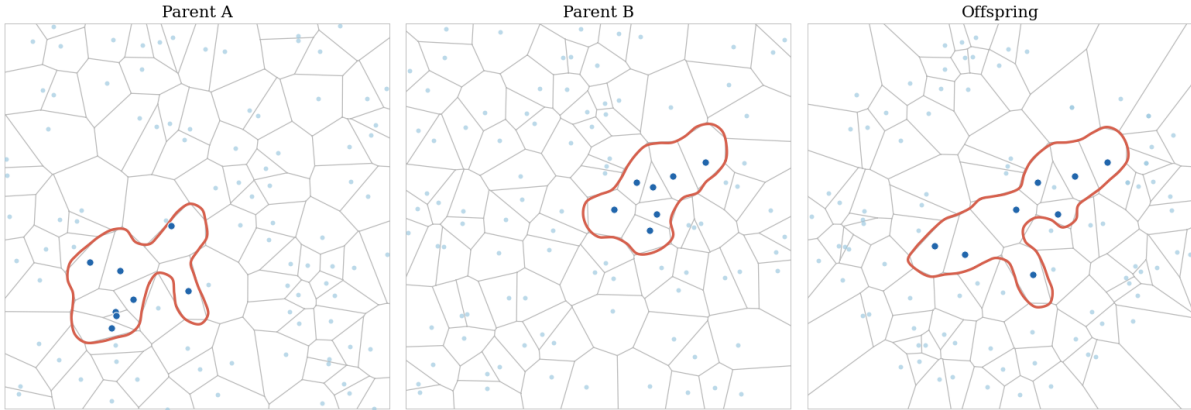


Figure 4.3: Crossover between two parent tracks: the offspring track inherits distinctive features from both parents.

4.1.4. Evaluation and Behavioural Descriptor

Each candidate track is then passed into a simulation environment that runs and returns the telemetry described in section 3.1.1:

Behavioural Descriptor (BD)

A crucial architectural decision in our pipeline is how to define a track's "behaviour". Rather than relying on the track's static geometry, we chose to define diversity entirely through raw driving telemetry derived from a benchmark lap (see table 3.2). This choice is motivated by the fact that a racing circuit is fundamentally defined by its experiential vehicle dynamics rather than its simple top-down shape. By utilising telemetry sequences, we capture physics-based contextual dependencies, such as how momentum and driving styles adapt to varying track topologies. To allow the algorithm to explore these complex behaviours, we need to compress this high-dimensional data into a lower-dimensional representation. To achieve this, we employ a Variational Autoencoder (described in section 4.2), utilising its latent representation as our BD.

Objective Fitness

To drive the Local Competition toward high-quality solutions, we selected the **total overtakes** metric from the multi-agent simulations (see table 3.3) as our Fitness objective. Because our target domain leans toward arcade-style and rallycross racing, a successful track must facilitate dynamic gameplay and close vehicle interactions, rather than simply acting as a complex time-trial route. By optimising for the number of successful passing manoeuvres between cars, the evolutionary search naturally selects for highly competitive track topologies.

4.1.5. The Archive (Tell)

To implement our Novelty Search algorithm, we adopted the `ProximityArchive` from the `pyribs` library. Unlike grid-based archives, this unstructured archive stores solutions based on their distance in the behaviour space, enabling a flexible and adaptive representation of diversity. Insertions into the archive are processed in batches, corresponding to the `BATCH_SIZE` defined during the generation phase. When a batch of newly evaluated solutions is submitted, each candidate is considered for admission. A solution is added to the archive if its novelty score exceeds a defined threshold (Thr). Because local competition is enabled, a solution that fails to meet this novelty threshold is not simply discarded; instead, it competes directly with its k -Nearest Neighbours in the behaviour space. If the candidate solution yields a higher objective value than its nearest neighbour, it replaces it in the archive. This batched update mechanism ensures efficient processing while retaining high-performing solutions within each local niche and maintaining overall diversity across the archive.

4.1.6. Dynamic Representation Learning

A defining characteristic of our pipeline is its ability to remain adaptive over time. As the algorithm explores and discovers novel solutions, the underlying latent space must evolve to accurately categorise and distinguish them. This adaptability is maintained through the following core dynamic operators.

Model Fine-Tuning: To ensure the latent manifold remains a relevant and high-fidelity descriptor of the discovered behaviours, the VAE is periodically updated. By fine-tuning the model on the data of newly discovered elites, the representation is refined to capture the specific features and nuances of the current population. The technical details of the tuning pipeline are further expanded in section 5.2.1.

Archive Remapping: This operation populates an empty archive with a set of elites in a single pass. Since `pyribs` does not provide a native mechanism for this procedure, and sequential batch insertions would introduce ordering bias, the population is processed simultaneously. With the same mechanism as in the Proximity Archive, a thinning step based on k -Nearest Neighbours (k -NN) novelty is applied. The operation then partitions elites into two functional groups:

- *Novel Elites:* Solutions whose mean distance to their k nearest neighbours exceeds the current threshold are preserved as unique representatives of the new manifold.

- *Competitive Survivors*: In regions where multiple solutions are clustered (non-novel), a “winner-takes-all” local competition is performed. By grouping solutions by their absolute nearest neighbour and preserving only the highest-performing individuals, the system effectively manages archive density while preserving the highest-quality solutions found in each region.

Adaptive Threshold Scaling: To regulate the archive’s growth and maintain a stable population size across shifts in representation density, the novelty threshold is adjusted via a feedback-driven mechanism. Utilising a multiplicative proportional controller as proposed by Grillotti and Cully [9], the threshold is scaled based on the deviation from a target capacity:

$$Thr_{new} = \max(\epsilon, Thr_{old} \times (1 + K_{CSC}(N_{archive} - N_{target}))) \quad (4.1)$$

where:

- Thr_{new} and Thr_{old} represent the novelty threshold before and after the adaptation.
- K_{CSC} is the proportional controller gain.
- $N_{archive}$ and N_{target} represent the current and desired number of elites, respectively.
- ϵ is a small constant lower bound ensuring that the threshold remains functional even if the archive size is significantly below the target.

Integration in the Search Loop

Rather than being applied at every step, these dynamic operators are integrated into the main evolutionary loop at predefined intervals to balance search stability with adaptability. The pipeline orchestrates these operators through two distinct cyclical routines: a comprehensive representation update and a periodic structural maintenance.

Once a set number of global iterations is reached, the **Representation Update Routine** is triggered to realign the latent space with the most recent discoveries. This process unfolds systematically:

1. **Data Extraction:** The system retrieves the high-dimensional phenotype data (the raw simulated telemetry) associated with all the elites currently residing in the archive.
2. **Latent Space Update:** Triggering the *Model Fine-Tuning* operator, this dataset is used to train the VAE. The model shifts its representational focus toward the

high-performing, feasible regions of the solution space discovered so far.

3. **Re-encoding of the Archive:** Because the fine-tuning process inevitably alters the topology of the latent space, the previous BDs of all archived solutions become instantly obsolete. To reconcile this, the updated encoder is used to re-evaluate the raw phenotype data of every elite, assigning them new coordinates in the updated measure space.
4. **State Reconstruction:** The current archive is completely emptied. The pipeline then executes the *Archive Remapping* operation, filtering and re-inserting the elites based on their newly established geometric relationships.
5. **System Synchronisation:** Finally, *Adaptive Threshold Scaling* is applied to recalculate the novelty threshold, accounting for the new spatial density of the updated manifold.

Furthermore, to mitigate the ordering bias inherent in sequential archive insertions during standard Ask-Evaluate-Tell cycles, a secondary **Archive Maintenance Routine** is triggered more frequently (e.g., once every few epochs). During this phase, the VAE remains frozen, but the *Archive Remapping* and *Adaptive Threshold Scaling* operators are executed. This ensures that the archive density remains optimised and free of historical insertion bias between major representation updates.

4.2. The Autoencoder

We chose to use a VAE as our Dimensionality Reduction algorithm. This allows us to obtain an expressive, non-linear encoding of our high-dimensional phenotype data (the benchmark lap metrics, see section 3.1.1) and to use the resulting latent space directly as the behavioural space in which Novelty Search operates.

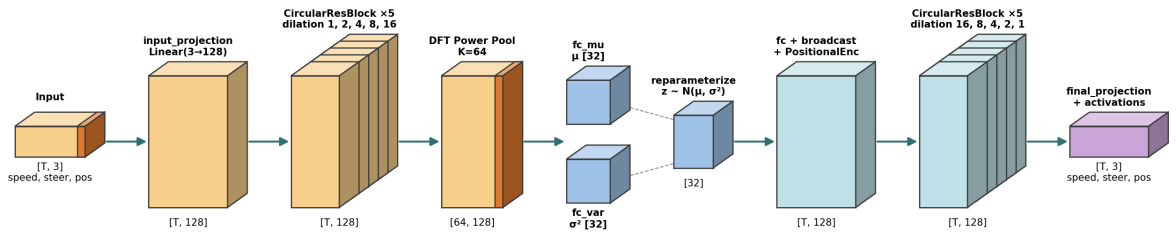


Figure 4.4: Overview of the VAE's architecture

At every iteration, a batch of newly generated tracks must be assigned coordinates within

the existing latent space before insertion into the archive. With a VAE this is a single deterministic forward pass through the encoder, whereas non-parametric methods such as t-SNE offer no native transform for unseen data, and refitting the embedding per batch would silently change the space underneath the archive, invalidating all previously assigned descriptors.

The limitations of these embedding methods, however, extend beyond the lack of a parametric mapping. Our data is inherently a variable-length, multichannel cyclical time series and for t-SNE and UMAP it would need to be padded and flattened into fixed-size vectors before embedding. Furthermore these methods cannot compute shift invariant representations of the data, two nearly identical laps whose sampling is shifted by a few steps (in our case a shifted race start position) can appear distant in latent space. A learned encoder sidesteps this problem entirely, as the notion of similarity is itself learned from the data and the relevant invariances can be enforced architecturally, as detailed in the following subsections.

Furthermore, while any dimensionality reduction necessarily discards information, a VAE is trained under an explicit reconstruction objective, so the amount of behavioural information preserved by the latent representation is directly measurable

The choice of using a VAE instead of a traditional Autoencoder is closely related to how NS operates, the novelty score at the core of our algorithm is computed as the mean Euclidean distance of a candidate to the k -Nearest Neighbours. This works if **distances are comparable across the entire latent space**. The KL-divergence regularisation of the VAE (see section 2.4.2) produces a dense, approximately homogeneous latent geometry without the unmapped “gaps” that fragment the latent space of a standard Autoencoder.

Finally, the VAE integrates naturally with the *Dynamic Representation Learning* routine described in the previous section: the encoder can be periodically fine-tuned on a buffer of newly discovered elites and the archive remapped accordingly, following the approach of [9]. The decoder, although unused during the search itself, provides a measurable audit of descriptor performance: reconstruction fidelity directly quantifies how much behavioural information the latent representation preserves, a validation that pure embedding methods cannot offer. An overview of the VAEs architecture is provided in Fig. 4.4.

4.2.1. Input preprocessing and normalisation

Our input data is the raw benchmark telemetry data described in section 3.1.1, while the computed data contains various channels we chose to discard some of those based on their signal form and the usefulness of the information they contained.

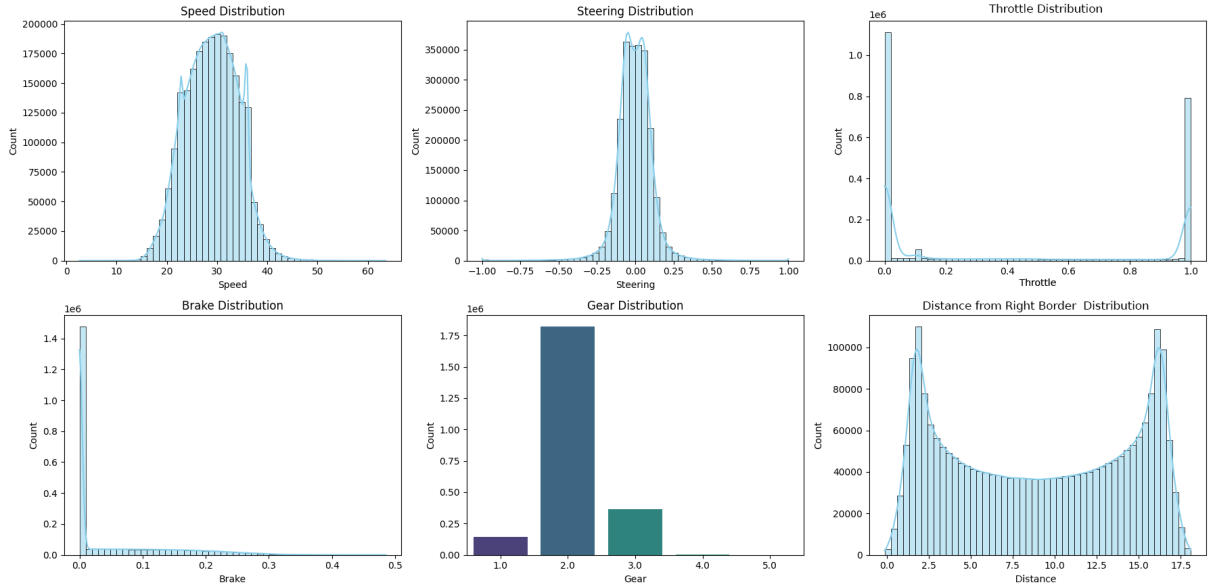


Figure 4.5: Distribution of race metrics values in the pre-training dataset

Chosen Metrics and Normalisation

Of the six telemetry channels outputted by our simulator we chose to keep only three: **speed**, **steering** and **distance from track boundary**.

Throttle, **brake**, and **gear** are discarded. The discarded channels are command values whose effects are already captured by the retained ones: acceleration and braking are recoverable from the speed derivative, and gear selection is essentially a quantisation of speed. Moreover, the discarded signals are ill-suited to the model’s reconstruction objective. **Throttle** and **brake** behave nearly as on/off signals, spending most of the lap pinned at their extremes, and gear is a discrete step function (see the graph in Fig. 4.5 and additional analysis in Appendix A).

Validity Checks

Each sample is subjected to two validity checks before entering the rest of the preprocessing pipeline. First, the raw telemetry is verified to contain only finite values, rejecting recordings corrupted by numerical artifacts (*NaN* or ∞). Second, every retained channel must exhibit a minimum standard deviation: a near-constant channel indicates a degenerate simulation—typically a vehicle that is stuck or a recording that failed to capture an actual lap—rather than a genuine driving behaviour. Samples failing either check are discarded. This filtering is essential in our setting, because degenerate telemetry admitted into the pipeline would not merely degrade reconstruction quality: being unlike any valid

lap, it would likely appear highly novel, polluting the archive with broken solutions that the novelty mechanism actively rewards.

Normalisation

To stabilise the training, the remaining channels are normalised using fixed physical constants of the simulation environment; not relying on dataset statistics keeps the input space stationary across fine-tuning, even as the algorithm finds out-of-distribution data.

- **speed**: its value in $[m/s]$ is divided by a fixed maximum (the top speed of the benchmark car) and clipped to $[0, 1]$.
- **steering**: its value, already between $[-1, 1]$ is mapped to $[-0.5, 0.5]$ to have an unitary range.
- **distance from track boundary**: originally measured in $[m]$ it's divided by the track width, clipped to $[0, 1]$ and centred around 0 to be between $[-0.5, 0.5]$.

The choice of ranges reflects the algebraic structure of each quantity: **steering** and **distance from track boundary** are signed, symmetric signals with a physically meaningful neutral point (a centred wheel; the track midline). While speed is a non-negative magnitude with a meaningful zero.

4.2.2. Model Architecture

The core challenge in embedding our telemetry data lies in its structure: a lap is a variable-length, multi-channel time series representing a continuous physical loop. Standard Convolutional or Recurrent Neural Networks rely on zero-padding and fixed anchoring, making them highly sensitive to the starting point of the lap. This can be a big issue for our architecture case since a small mutation of the track can create a big shift in the starting point.

To ensure that two identical laps starting from different points on the track map to the exact same point in the latent space, we designed a VAE architecture that enforces shift-invariance structurally .

The Shift-Equivariant Encoder

The encoder processes the normalised telemetry sequences through a learned linear projection followed by a stack of 1D Convolutional **Residual Blocks** [10]. To respect the closed-loop topology of a lap, we replace standard zero-padded convolutions with Circu-

lar Convolutions. By wrapping the sequence around its boundaries, the network avoids boundary artifacts and processes the track as a continuous cycle.

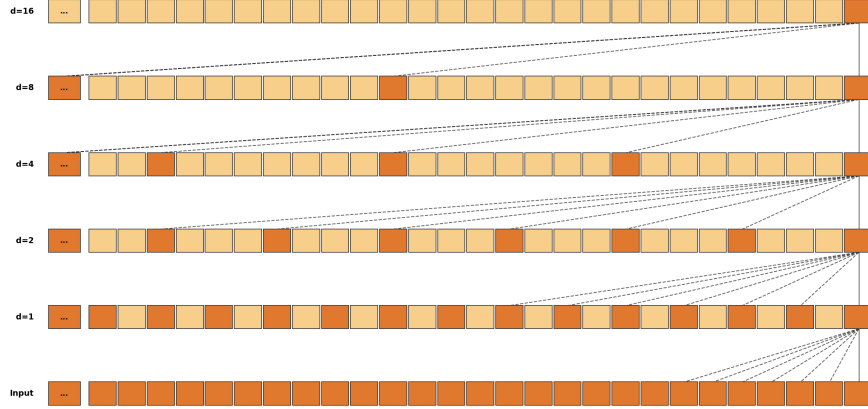


Figure 4.6: Receptive field of a single output position after dilations $[1, 2, 4, 8, 16]$, $kernelsize = 7$; Every active cell has the same fan-out pattern; "...": earlier positions, off-screen; for clarity, the dilated-tap edges are drawn for only one representative cell per layer, since every active cell shares the same fan-out pattern.

Each **Residual Block** is comprised of two circular convolutional layers. To preserve the shift-equivariance, we employ a custom Channel-Wise Layer Normalisation rather than standard temporal or batch normalisation. This is followed by a GELU activation function [11]. Finally, the processed features are combined with the block's input via an additive identity skip connection.

We use 5 of these stacked **Residual Blocks** with a kernel size of 7 and exponentially increasing dilation rates (1, 2, 4, 8, 16). This coarse-to-fine hierarchy rapidly expands the receptive field (to a total of 373), allowing the network to capture both high-frequency local behaviours (e.g., small steering corrections) and low-frequency global trends (e.g., long straights or complex corner combinations) while remaining strictly shift-equivariant, increasing the number of Residual Blocks any further didn't affect performance for our dataset. An overview of the dilation pattern is shown in Fig. 4.6.

Shift-Invariant Pooling via Length-Normalised DFT

While the convolutional backbone is shift-equivariant, the latent space requires a fully shift-invariant representation. Standard approaches like global average pooling and global max pooling are often used, however these operations discard the vast majority of the spatial structure, collapsing complex waveform variations into a single scalar per channel.

To avoid this loss of information, while guaranteeing shift invariance, we implement a

custom **Discrete Fourier Transform (DFT) Power Pooling** layer. Our DFT formulation dynamically adapts to the variable length of each lap. For a feature map sequence of actual length L , we compute the Fourier coefficients evaluated precisely at $K = 64$ normalised frequencies.

By explicitly dividing by L in the phase calculation, the frequency bins do not represent absolute temporal frequencies (which would fluctuate depending on the tracks length and the driver’s speed), but rather **cycles per lap**.

The real and imaginary components of these coefficients are divided by L to normalise the amplitude against sequence length variations. We then compute the power spectrum ($Real^2 + Imag^2$), deliberately **discarding the phase information**. Because temporal shifts in the time domain manifest as phase shifts in the frequency domain, discarding the phase yields a representation that is mathematically invariant to the starting point of the lap.

This operation is performed independently across all 128 hidden channels, producing an unrolled 128×64 vector of length-invariant structural frequencies, which is subsequently mapped via fully connected layers to the parameters of the 32-dimensional latent distribution (μ and Σ).

The Latent Bottleneck

The DFT-pooled features are flattened into an 8192-dimensional vector (128 channels $\times$ 64 bins) and passed through two parallel fully connected layers. These form the core information bottleneck, mapping the sequence to the parameters of a 32-dimensional Gaussian latent distribution: the mean μ and log-variance $\log \sigma^2$. The final behavioural embedding z is then sampled via the standard reparametrisation trick ($z = \mu + \sigma \odot \epsilon$), allowing deterministic backpropagation.

To prevent the Kullback-Leibler (KL) divergence from exploding in early epochs, we apply strict initialisation heuristics to the bottleneck. The variance projection head is orthogonally initialised with a tightly constrained gain (0.01) and a negative bias (-2.0). This forces the network to begin with small, conservative variances, allowing the reconstruction loss to properly arrange the latent space before KL regularisation takes over. Additionally, the predicted log-variance is strictly clamped between $[-20.0, 10.0]$ to guarantee numerical stability throughout training.

The Decoder and Output Bounding

The decoder’s task is to reconstruct the original time series from the sampled 32-dimensional latent vector z and a target sequence length. Since we discarded the phase, z contains no temporal information so it is projected, broadcasted across the target sequence length, and injected with a Sinusoidal Positional Encoding [24]. This positional signal provides the necessary temporal scaffolding for the convolutions to unfold the latent features back into a sequential structure.

The reconstruction proceeds through a reverse-dilation stack of the Circular Residual Blocks described previously (16, 8, 4, 2, 1). Finally, to ensure the network’s predictions respect the physical bounds established during normalisation, the output layer applies specific bounding activation functions: a σ (sigmoid) activation to bound **speed** within $[0, 1]$, and a $0.5 \cdot \tanh$ activation to strictly bound **steering** and **distance from track boundary** within $[-0.5, 0.5]$.

Shift-Invariant Loss and Regularisation

Because the encoder discards phase information to achieve shift-invariance, the decoder generates a reconstructed lap that is structurally accurate but arbitrarily aligned in time relative to the target input. A standard Sum of Squared Errors (SSE) reconstruction loss would heavily penalise this temporal misalignment, forcing the network to memorise arbitrary starting positions.

We explicitly avoid computing the loss directly on the phase-discarded DFT, because such an approach would allow the decoder to learn reconstructions that don’t take into account the temporal structure or inter-channel relationships of the sequence, allowing for an unrealistic reconstruction with no real penalty.

Instead, we evaluate the loss in the time domain. To do this, we need to align the reconstructed sequence and the input before computing the error. This could be done by calculating the SSE for every possible discrete circular shift between the input and the reconstruction and selecting the minimum, but the computational cost doesn’t scale well for longer sequences ($\mathcal{O}(L^2)$). To ensure computational speed remains acceptable for long

sequences, we implemented a **Weighted Cross-Correlation Aligned SSE** loss:

Let

$$\mathbf{x} \in \mathbb{R}^{L \times C}, \quad \hat{\mathbf{x}} \in \mathbb{R}^{L \times C}$$

denote, respectively, the ground-truth metric sequence and the sequence reconstructed by the decoder, with $x_{t,c}$ and $\hat{x}_{t,c}$ for time index $t \in \{0, \dots, L-1\}$ and channel $c \in \{0, \dots, C-1\}$. Additionally, let w_c represent the channel weight assigned to channel c to scale each metric's contribution to the error.

We define the reconstruction error as the minimum weighted squared error over all circular shifts $\tau \in \{0, \dots, L-1\}$ of the reconstruction:

$$E(\tau) = \sum_{t=0}^{L-1} \sum_{c=0}^{C-1} w_c (x_{t,c} - \hat{x}_{(t-\tau) \bmod L, c})^2,$$

and the per-track reconstruction loss as:

$$\mathcal{L}_{\text{rec}} = \min_{\tau} E(\tau).$$

Expanding the square decouples $E(\tau)$ into shift-independent energy terms and a single shift-dependent cross-correlation term:

$$E(\tau) = \underbrace{\|\mathbf{x}\|_{\mathbf{w}}^2 + \|\hat{\mathbf{x}}\|_{\mathbf{w}}^2}_{\text{constant in } \tau} - 2 \text{CC}(\tau),$$

where $\|\cdot\|_{\mathbf{w}}^2$ denotes the weighted sum of squares over all $L \times C$ entries:

($\|\mathbf{x}\|_{\mathbf{w}}^2 = \sum_{t=0}^{L-1} \sum_{c=0}^{C-1} w_c x_{t,c}^2$), and

$$\text{CC}(\tau) = \sum_{c=0}^{C-1} w_c \sum_{t=0}^{L-1} x_{t,c} \hat{x}_{(t-\tau) \bmod L, c}$$

is the weighted channel-summed circular cross-correlation. Since only $\text{CC}(\tau)$ depends on τ , minimising $E(\tau)$ is equivalent to maximising $\text{CC}(\tau)$:

$$\mathcal{L}_{\text{rec}} = \|\mathbf{x}\|_{\mathbf{w}}^2 + \|\hat{\mathbf{x}}\|_{\mathbf{w}}^2 - 2 \max_{\tau} \text{CC}(\tau).$$

The circular cross-correlation is evaluated for all shifts simultaneously in $\mathcal{O}(L \log L)$ time using the Fast Fourier Transform. Denoting by $\mathcal{F}$ the discrete Fourier transform along

the time axis and by $\overline{(\cdot)}$ the complex conjugate, we have:

$$\text{CC}(\tau) = \sum_{c=0}^{C-1} w_c \mathcal{F}^{-1} \left\{ \mathcal{F}\{x_{\cdot,c}\} \cdot \overline{\mathcal{F}\{\hat{x}_{\cdot,c}\}} \right\}(\tau),$$

by the circular correlation theorem.

This reconstruction objective is balanced with the standard Kullback-Leibler (KL) divergence regularisation (see section 2.4.2), making the final loss:

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \beta \mathcal{L}_{\text{KL}}, \quad (4.2)$$

where the hyperparameter β controls the trade-off between reconstruction fidelity and the regularity of the learned latent space.

4.2.3. Training Setup and Optimisation

The variational autoencoder (VAE) is trained on variable-length sequences of per-step driving metrics, this section details the optimisation procedure, the loss formulation, and the regularisation and stabilisation techniques employed during both pre-training and fine-tuning.

The model is optimised with Adam [14]. Sequences within a batch are zero-padded to a common length and a boolean padding mask is propagated and handled by the network and the loss so that padded positions never contribute to either the reconstruction error or the gradients.

Training runs for a set amount of epochs at most, but its duration is handled in practice by **early stopping**: the validation reconstruction loss (detailed in section 4.2.2) is monitored and the epochs with the lowest value are cached, if no improvement larger than a small delta is observed for a set amount of consecutive epochs, training halts and the best-scoring weights are restored.

We chose to monitor the reconstruction quality and not the total loss since our interest is mostly how good the embedding captures the information of the sequence, while we monitored and tuned other parameters manually to have satisfying KLD values.

To improve convergence we dynamically control the learning rate by using a **Reduce on Plateau** schedule, which reduces the rate after a set amount of epochs without improvement in the validation reconstruction term.

Validation is performed deterministically: the latent mean $\boldsymbol{\mu}$ is used directly instead of drawing a sample from the posterior, removing stochastic noise from the reported metrics.

Cyclical KL Annealing To mitigate posterior collapse, the KLD weight β in Equation (4.2) is not fixed but follows a cyclical annealing schedule [8]. Within each cycle, β ramps linearly from 0 to its peak value $\beta_{\max}$ over the first fraction ρ of the cycle and then remains at $\beta_{\max}$ for the remainder:

$$\beta(e) = \begin{cases} \beta_{\max} \frac{e \bmod C}{\rho C}, & e \bmod C < \rho C, \\ \beta_{\max}, & \text{otherwise,} \end{cases} \quad (4.3)$$

where $C = \lfloor E/n_{\text{cycles}} \rfloor$ is the cycle length in epochs.

Channel Weights Even though the input telemetry channels (speed, steering, and distance to boundary) are normalised and bounded during preprocessing, their intrinsic variances and behavioural significance can differ substantially. To prevent the reconstruction error from being dominated by a single metric, the loss function incorporates explicit channel-wise weights.

5 | Experiments and Results

This chapter details the experimental work conducted to achieve our final algorithm, we will first detail the pre-training procedure and results for our VAE, following we will provide a description of the results and experiments regarding the Novelty Search algorithm in its two different modes (with or without latent space fine-tuning). To ensure the robustness of our findings, all experiments are fully reproducible and were consistently verified across multiple trials using various random seeds.

5.1. VAE Pre-training and Analysis

A key part of our architecture is the pre-trained autoencoder, which serves as the foundational behaviour extractor for our Novelty search algorithm. This section outlines the configuration and provides an in depth analysis of the resulting latent space, its structural properties and the model’s reconstruction quality.

5.1.1. Experimental Setup

To learn a robust representation the VAE was trained on a dataset consisting of approximately 20,000 track metrics, these tracks were generated using the Voronoi method described in section 3.2.2 with a 50/50 split between the Uniform and Simplex Noise techniques, we chose generation seeds to ensure no overlap with either the **Test Dataset** or the NS algorithm.

After passing through the validity checks and preprocessing described in section 4.2.1, we apply a random 80/20 split between training and validation data. Before training to prevent channels with higher variances from dominating the reconstruction loss, we compute the channel weights (see section 4.2.3) based on the dataset’s statistics, the weights are set to the inverse of the variance of each channel $[1/\sigma_{c_1}^2, 1/\sigma_{c_2}^2, 1/\sigma_{c_3}^2]$, then they are normalised so their sum equalled the number of dimensions. The training was conducted with a batch size of 128.

Based on our preliminary tuning the final architecture is instantiated with the following

setup:

- **Model Dimensions:** 128 hidden channels, 5 circular residual blocks with a kernel size of 7, and a latent bottleneck dimension of 32.
- **Learning rate & scheduling:** Initial learning rate of 5×10^{-4} with a *ReduceLROnPlateau* scheduler with a factor of 0.5 and a patience of 25 epochs, monitoring validation reconstruction loss.
- **Training duration & Early Stopping:** The maximum amount of training epochs was set to 700, we employed Early Stopping with a patience of 50 epochs, also this mechanism monitored validation reconstruction loss.
- **KL-divergence Regularisation:** For our annealing cycle we observed our best results by using a single cycle peaking at $\beta_{\max} = 0.02$, the ramp-up spanned the first 30% of the maximum training epochs.

5.1.2. Experimental Results

During our final experiment training went on for 233 epochs, as observable in Fig. 5.1 the values of both the reconstruction and KLD losses exhibit a sharp transient phase at the very beginning of training. To better analyse the subsequent training dynamics, Fig. 5.2 isolates the loss curves starting from epoch 5, where the magnitude changes are at observable levels.

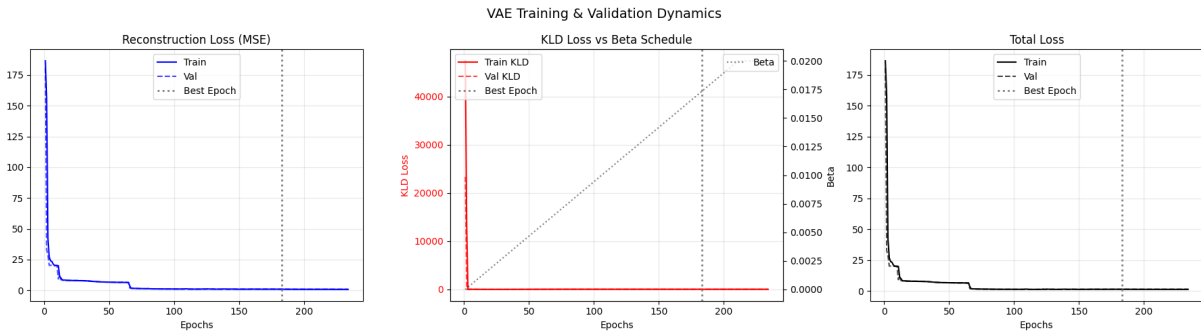


Figure 5.1: Plots of reconstruction (left), KLD (centre) and total (right) losses spanning the entire training.

The reconstruction loss (Fig. 5.2, left) displays a distinctive step-like behaviour. After plateauing early in the run, it experiences a sharp, sudden drop around epoch 65, then it gradually drops until reaching a final plateau of around 0.88, which represents the total shift-aligned error across the entire trajectory sequence.

The KLD Loss plot exhibits its expected behaviour, while it starts low as the network stabilises it then ramps up and gradually lowers influenced by the increasing β pressure given by the cycle. The reconstruction loss value on the other hand has a sequence of big drops and as the network stabilises it slowly converges to its final state.

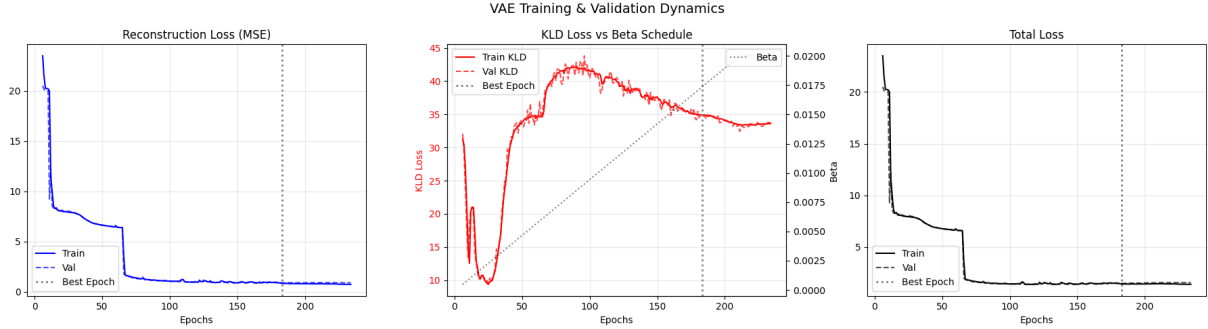


Figure 5.2: Plots of reconstruction (left), KLD (centre) and total (right) losses starting from epoch 5.

Overall, the training dynamics are healthy and highly stable. The validation curves tightly track the training curves across all plot, indicating good generalisation with no visible over-fitting. The early stopping mechanism functioned exactly as intended: it identified the optimal model weights at epoch 183 (marked by the vertical dotted line) where the validation reconstruction loss reached its absolute minimum. Following this optimal point, the validation loss plateaued, prompting the training to correctly halt after its 50-epoch patience was exhausted.

5.1.3. Discussion

In this section we'll provide a detailed analysis of the learned latent space. To ensure the robustness of our findings, the active dimensions are analysed over the training dataset while all the subsequent evaluations are conducted on a test dataset comprising 20,000 unseen tracks, generated and seeded separately from the training set. This dataset was generated using the same pipeline as the training set, but initialised with distinct random seeds to guarantee no overlap between the two sets.

Active Dimensions and Posterior Collapse

We evaluated the "health" of the 32-dimensional bottleneck by computing the mean KL-divergence for each latent dimension over the training set. We defined an "active" dimension as one maintaining a KL-divergence above a threshold of 0.1 nats. As illustrated in Fig. 5.3 our results showed a balanced distribution of information: all dimensions

remained active, with mean KL values smoothly distributed in the acceptable range. This indicates that the VAE is utilising the entirety of its bottleneck capacity to capture the complex, high-dimensional structural variations of the laps, rather than concentrating the information into a small subset of dimensions.

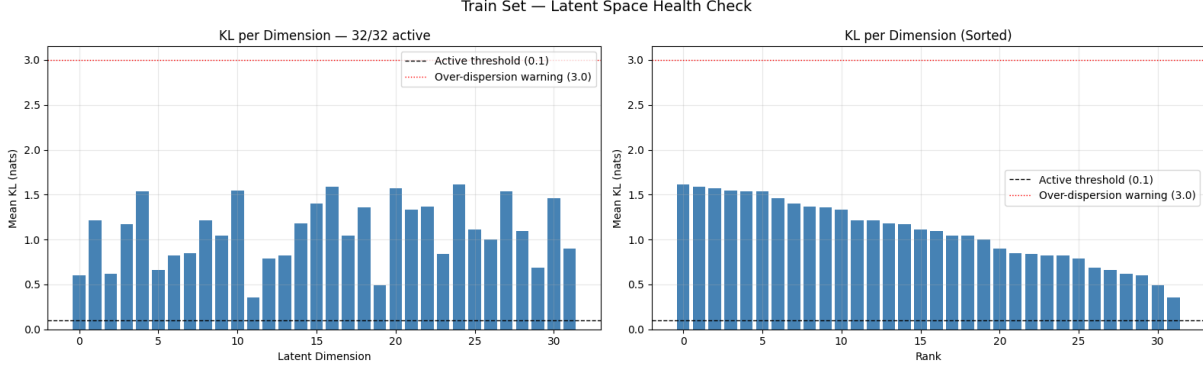


Figure 5.3: Active dimension analysis on training set.

Furthermore no dimension approaches the overdispersion threshold, suggesting that our β schedule and our variance initialisation constraints (detailed in section 4.2) successfully regularised the latent space. Additional analysis plots are provided in Appendix B.

Reconstruction Quality

To evaluate how well our latent space compresses information, we examine the reconstructed waveforms of unseen data, Fig. 5.4 plots five randomly sampled sequences from the test dataset against their corresponding reconstructions. Due to the shift-invariant property of the architecture, the reconstructed laps are phase-aligned to the original tracks.

As observable in Fig. 5.4, the decoder demonstrates an overall ability to reconstruct the original telemetry across varying track lengths and different behavioural patterns. Some high frequency details are lost, this is most noticeable in the steering channel as it is naturally the most prone to micro-adjustments across all track geometries. Overall the track reconstruction quality is satisfactory for our purposes. It is important to emphasise that the latent space is heavily compressed (32 dimensions), and the primary focus of our algorithm is not the generative capability of the decoder itself, but rather how good the encoder’s ability to embed the complex telemetry sequences. Nonetheless the ability to recover the global topology of the laps, suggests that this compressed representation successfully captures the essential behavioural fingerprint of each track.

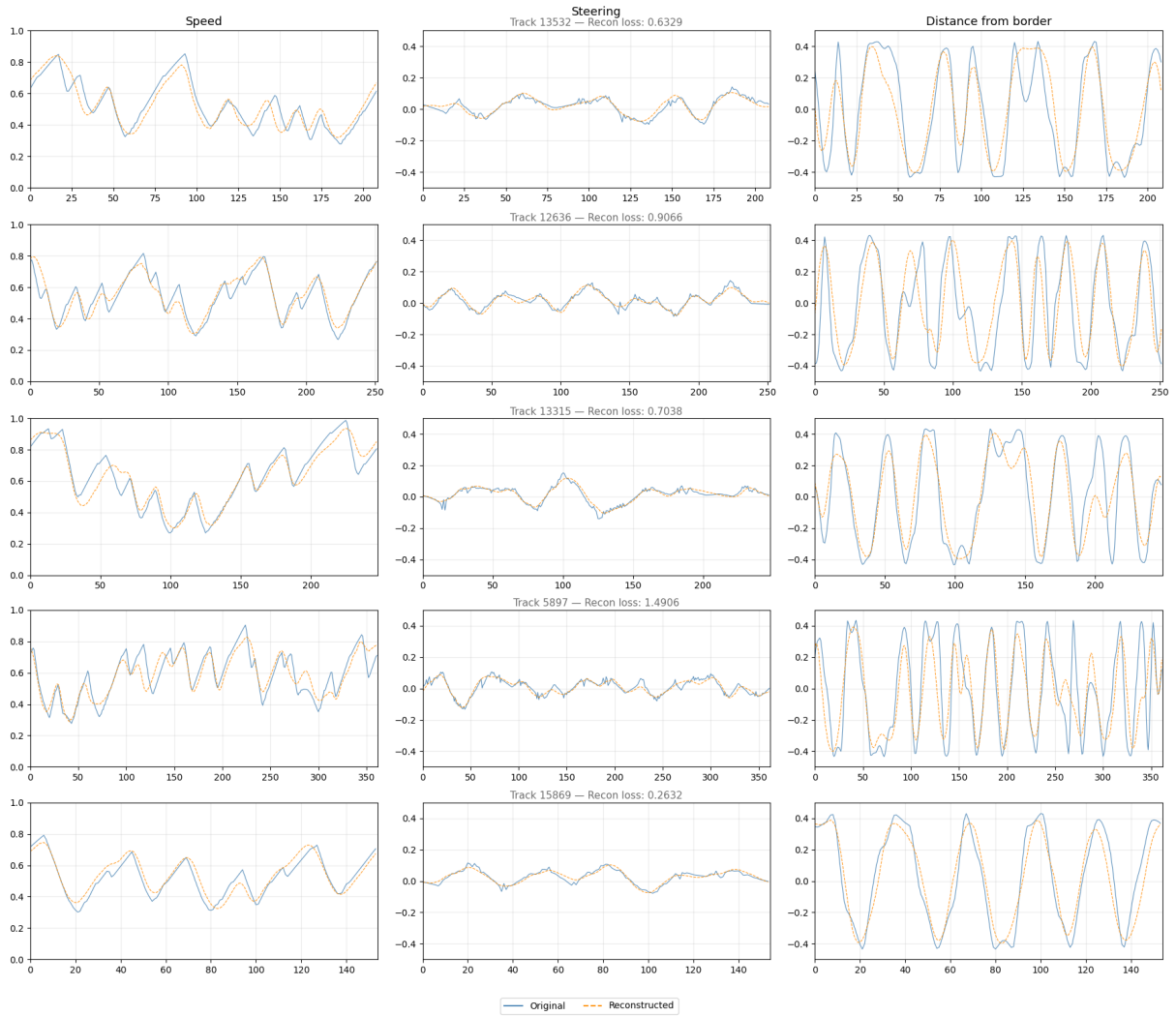


Figure 5.4: Phase aligned time-domain reconstruction of five randomly sampled tracks from the test set. The original telemetry (solid blue) and the VAE's reconstruction (dashed orange) are compared across the three behavioural channels: speed, steering, and distance from track boundary.

Nearest Neighbours in Latent Space

The core mechanism of Novelty Search relies on computing the Euclidean distance between individuals in the behavioural space. For this to be effective, spatial proximity in the latent space must accurately and consistently reflect behavioural similarity in the physical simulation. To verify this property, we calculated and displayed the k -Nearest Neighbours of randomly sampled tracks. To enable a 1:1 visual comparison, each neighbouring sequence is first temporally resampled to match the exact step length of the query track, and subsequently phase-aligned. This allows us to visually inspect whether tracks that cluster together in the compressed 32-dimensional space actually exhibit the same behavioural patterns across the channels.

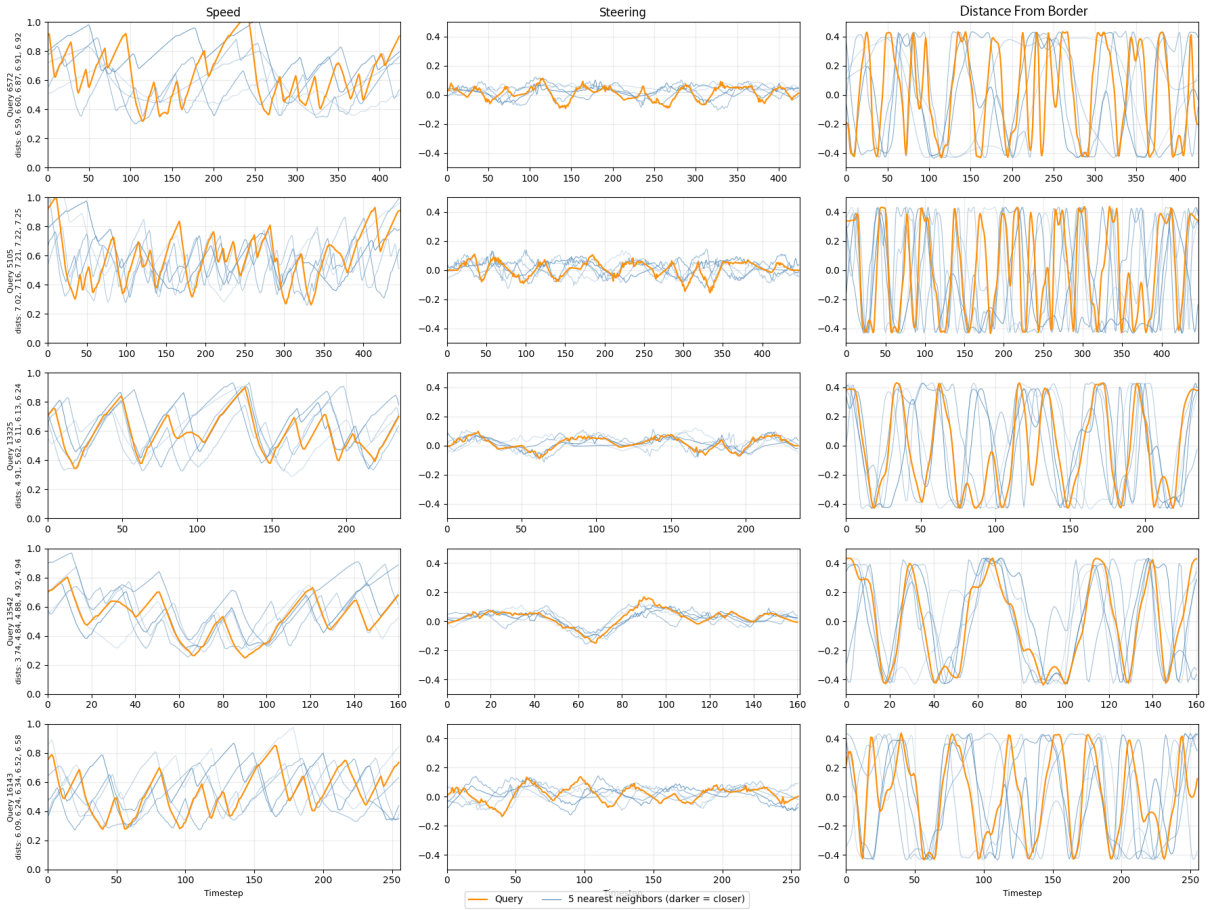


Figure 5.5: Time-domain visualisation of five random query tracks (solid orange) and their 5 nearest Neighbours in the latent space (blue, with darker shades indicating closer proximity). Neighbours are resampled to match the query length and phase-aligned to account for the VAE’s shift-invariance.

As observable in Fig. 5.5, the proximity in latent space directly correlates with physical

and structural similarity in the metric sequences. This is a critical validation for our purposes, suggesting that the behavioural metric is both meaningful and consistent.

Overall, these results provide strong empirical evidence that the 32-dimensional latent bottleneck has learned a dense, structurally cohesive representation of the telemetry. Consequently, computing Novelty Search distances within this space should reward genuine behavioural divergence.

5.2. Novelty Search Setup and Analysis

This section outlines the setup and result analysis of our Quality Diversity pipeline. We conducted two separate experiments one with fine-tuning and one without, this allows us to directly compare the impact of dynamically adapting the embedding space.

5.2.1. Experimental Setup

From now on we will refer to the two pipelines as **base-NS** and **finetune-NS**. Both setups rely on the same fundamental Quality-Diversity infrastructure but differ in how the behavioural embedding space is treated over time.

Common Configuration

Evolutionary Algorithm and Operators Both pipelines are executed for a total of 1,100 iterations, with a `BATCH_SIZE` of 20 evaluations per iteration. The search begins with a warm-up phase consisting of 200 iterations (4,000 evaluations), where random track topologies are generated (as outlined in section 4.1.3) and evaluated (see section 4.1.4) to populate the initial archive.

Following the warm-up, the Emitter switches to genetic operations (see section 4.1.3) for generation. It alternates evenly between mutation and crossover operations at each iteration to explore new regions of the behavioural space.

Archive and Novelty Calculation As outlined in section 4.1.5, the system relies on the **Proximity Archive** with Local Competition enabled. The BDs are calculated through the encoder of the pre-trained model described in section 5.1 and then inserted in the archive. Based on preliminary tuning, the starting acceptance threshold (Thr) is set to 6.3. In both pipelines **Adaptive Threshold Scaling** (see section 4.1.6) is applied with a target archive size (N_{target}) of 500 and a controller gain of $K_{CSC} = 10^{-5}$. The archive k -Nearest Neighbours calculation is performed with $k = 15$. Additionally,

both pipelines apply **Archive Remapping** to prevent ordering bias caused by sequential batch insertions. Both of these operations are applied within an **Archive Maintenance Routine** every 10 iterations after the initial population phase.

The base-NS Configuration

In the **base-NS** experiment, the representation learning model acts as a static feature extractor. The VAE is initialised from its pre-trained state, and its weights remain completely frozen for the entire evolutionary run.

The finetune-NS Configuration

The **finetune-NS** experiment fully activates the *Dynamic Representation Learning* capabilities of our pipeline. By continuously updating the VAE, the system forces the latent space to structurally adapt to the out-of-distribution behaviours discovered by the evolving population.

Following the initial 200-iteration warm-up, the *Representation Update Routine* is triggered periodically (specifically, every 100 iterations). As explained in detail in section 4.1.6 this routine unfolds as follows: elite extraction, latent space fine-tuning, re-encoding and remapping of the archive, and finally, threshold readaptation.

Fine-tuning setup The model fine-tuning is done with a much smaller dataset than the pre-training (around 500 samples compared to 20,000), furthermore we aim to adapt the model’s structure to learn the out-of-distribution behaviours without degrading our previous acquired representations. Thus, we employed different hyper-parameters and techniques:

To preserve the generic local features learned during pre-training, the first two **Circular Residual Blocks** of the encoder are frozen. The deeper encoder blocks, the latent bottleneck and the full decoder are fine-tuned using the same loss formulation as in pre-training, but with the following variations: Since a robust latent structure is already established there is no need for the KL cycle ramp-up, instead we employ a fixed β of 0.05. Although this is slightly larger than the $\max_ \beta$ used in pre-training, this stronger penalty is necessary to prevent a *KL* divergence from exploding when training on the much smaller dataset. We also decreased the batch size to 32 to compensate for the smaller dataset size and to maintain a sufficient number of optimisation steps per epoch.

We kept the learning rate the same as in the pre-training (5×10^{-4}). The maximum number of epochs is set to 50 with a 20 epoch early stopping. Because of the very short

training cycles we avoid using the `ReduceLROnPlateau` scheduler, so effectively we never reduce the learning rate.

Finally, for the loss channel weights, we reused the values computed during pre-training on the full 20,000-sample dataset. These values provide a more robust approximation of the true signal variances than weights recalculated solely on the smaller fine-tuning subset.

5.2.2. Experimental Results

Archive Dynamics and Population Growth

In both configurations, our adaptive threshold works as expected, effectively enforcing a target elite population of around 500 individuals. Furthermore, in Fig. 5.6, we can observe distinct phases, clearly delineating the initial random generation phase from the subsequent mutation and crossover phase. At iteration 200, the first archive remap is triggered in both setups. This results in a massive drop in the population, underscoring the archive’s strong dependency on insertion order. Following this drop, the archive continues to grow in a saw-tooth pattern, influenced by periodic remapping and the continuous adaptation of the threshold.

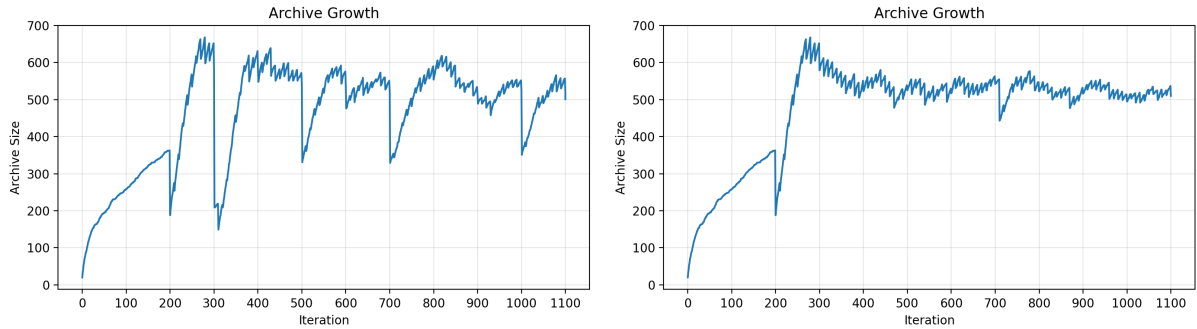


Figure 5.6: Archive population growths, **finetune-NS** configuration on the left and **base-NS** configuration on the right.

The fine-tuning configuration, compared to the base setup, grows in a much more jagged manner characterised by large spikes and deep drops. Each drop corresponds to an archive remap following a fine-tuning routine, while the peaks show the archive threshold relaxing to refill the archive. Most 100 iteration cycles between fine-tunings are clearly divided in two phases: one where the archive rapidly refills due to lowered threshold pressure, and another where this behaviour stabilises. This trend is clearly observable in the new insertion and substitution graphs in Fig. 5.7.

In both Fig. 5.7 and Fig. 5.8, we can observe how the new insertion diversity in the initial population quickly decays. However, once the genetic operators are applied, there is a substantial spike in new insertions. While some of this is facilitated by the adaptive threshold, it primarily indicates the efficacy of the mutation and crossover operators in discovering novel solutions.

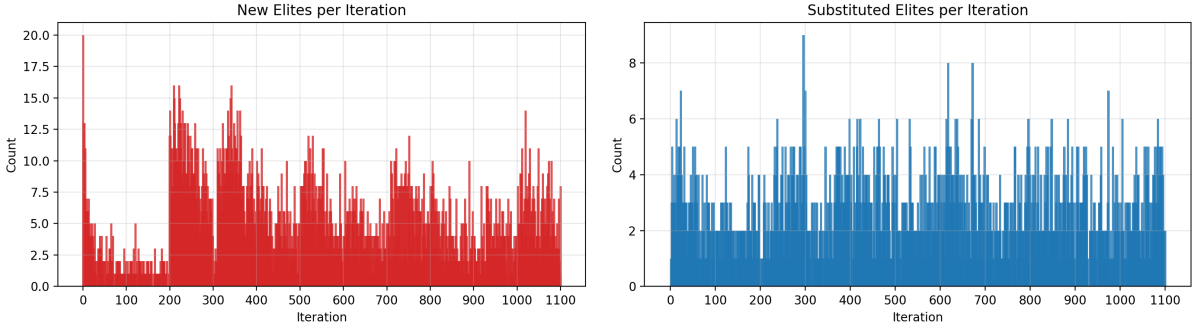


Figure 5.7: Archive insertions and substitutions dynamics for **finetune-NS** configuration.

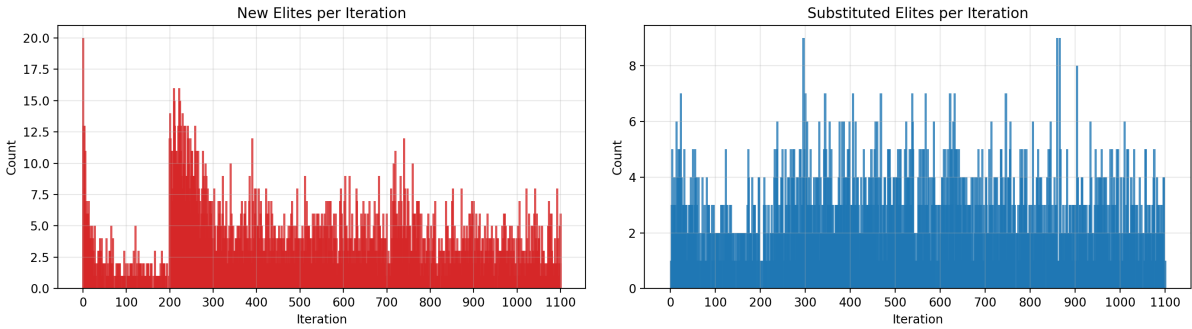


Figure 5.8: Archive insertions and substitutions dynamics for **base-NS** configuration.

Another clear dynamic observable (see Fig. 5.9) in both configurations is the progressive increase in overall Fitness. While the Fitness curves align for the initial population phase, the **finetune-NS** setup subsequently shows a steeper increase. This acceleration is a direct consequence of the fine-tuning cycles, since they generally reduce the population to a smaller, fitter subset. As the archive repopulates from this refined pool of elites, the overall Fitness of the newly generated individuals improves more rapidly.

Fine-tuning Loss Progression

Unlike the base setup where the VAE weights remain frozen, the **finetune-NS** configuration dynamically updates the model. Because this fine-tuning actively transforms the geometry of the latent space over time, its stability must be continuously monitored.

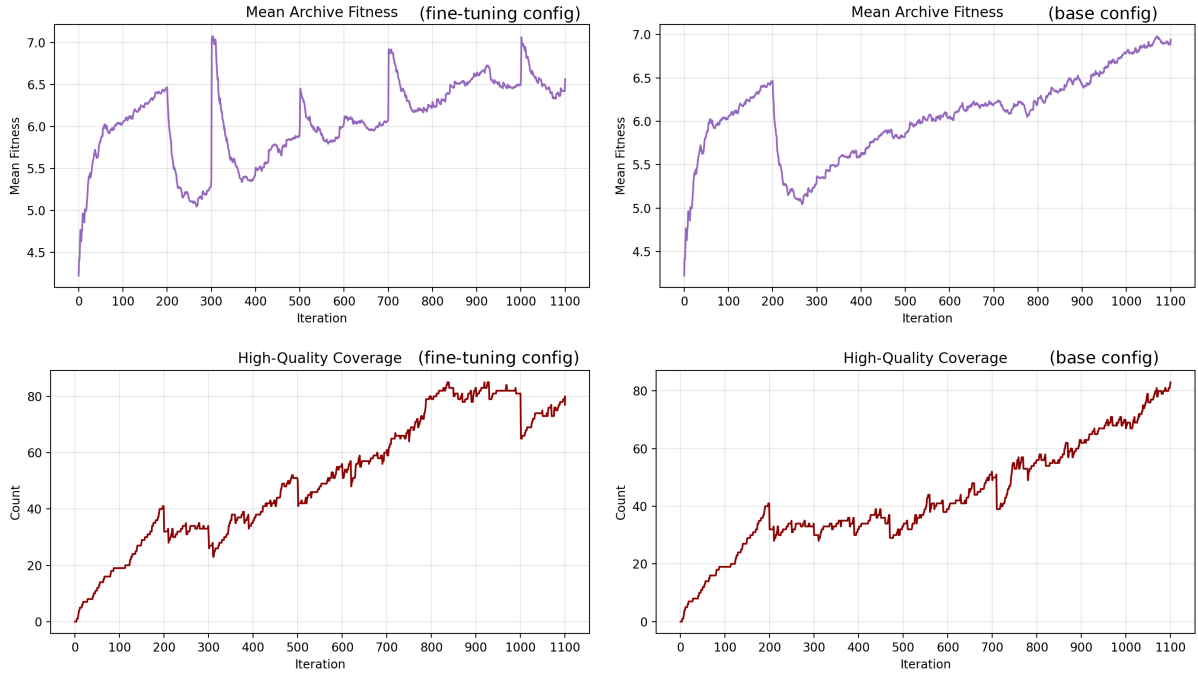


Figure 5.9: Fitness dynamics for both configurations, **finetune-NS** configuration on the left, and **base-NS** on the right. High-Quality Coverage counts the number of elites with Fitness ≥ 20 .

Fig. 5.10 illustrates the validation reconstruction loss across multiple fine-tuning cycles. Effective training lengths vary widely; some fine-tuning phases only span a couple of epochs, while others are significantly longer. When considering this plot alongside the archive growth in Fig. 5.6, we can observe that archive population drops coincide with fine-tuning cycles. Although the final reconstruction loss for each cycle is generally higher than the baseline pre-training loss, it remains within an acceptable threshold. This elevation is expected, as the algorithm continuously discovers highly novel, out-of-distribution individuals that are inherently more challenging to reconstruct than the static pre-training dataset. Overall, the reconstruction loss exhibits a downward trend in later iterations, indicating that the algorithm stabilises as it exhausts its ability to find new and diverse data.

As displayed in Fig. 5.11, the Kullback-Leibler Divergence (KLD) remains low and stable throughout the fine-tuning process. It maintains values closely aligned with those observed during pre-training, indicating that the β hyper-parameter was tuned correctly to prevent posterior collapse while allowing the latent space to adapt.

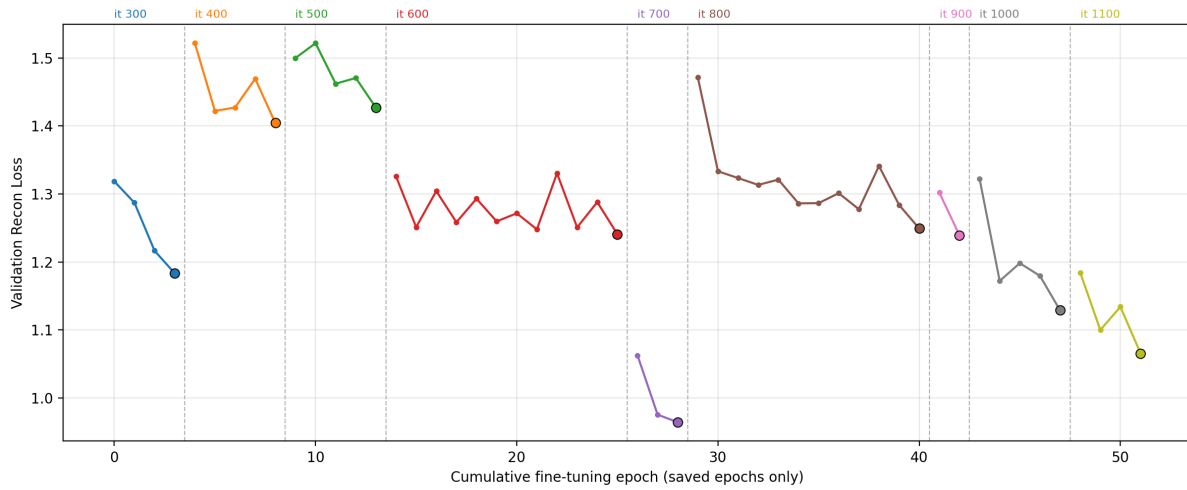


Figure 5.10: Validation Reconstruction loss recorded over cumulative fine-tuning epochs. Fine-tuning triggered at specific iterations, labelled above the plot. The final saved epoch for each cycle is marked by an enlarged circle.

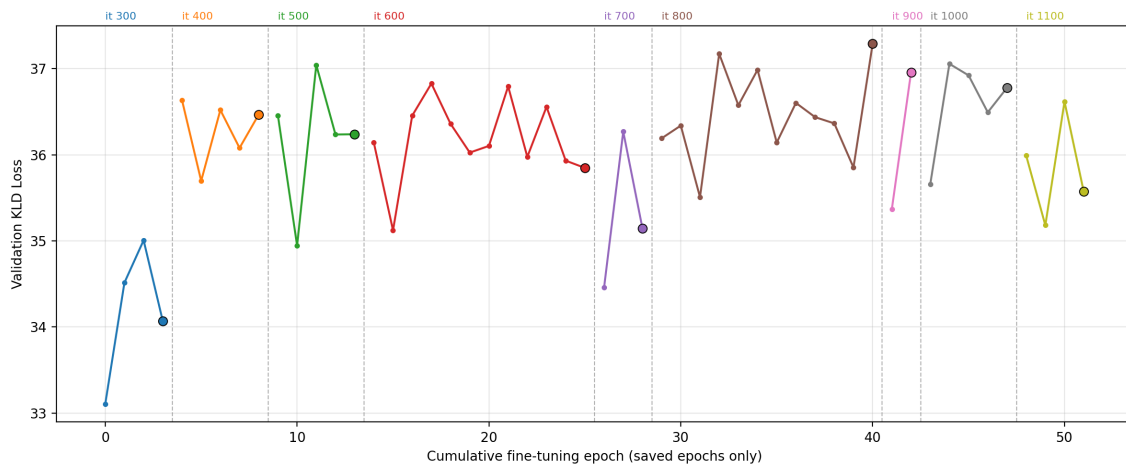


Figure 5.11: Validation Kullback-Leibler Divergence (KLD) loss recorded over cumulative fine-tuning epochs. Fine-tuning triggered at specific iterations, labelled above the plot. The final saved epoch for each cycle is marked by an enlarged circle.

5.2.3. Discussion

This section provides a detailed analysis of the final elite sets generated by both the **base-NS** and **finetune-NS** configurations. We'll provide visualisations and arguments to support the improvements of the latter. Through qualitative visualisations and comparative analysis, we demonstrate the structural improvements achieved by the fine-tuning approach. Additionally, we investigate the evolution of the learned representation by comparing the finalised **finetune-NS** latent space against the static, pre-trained baseline.

Latent Space Analysis & Comparison

As mentioned previously the KLD Loss of the **finetune-NS** setup maintains similar values as in the pre-trained model, this can be further confirmed by looking at the active dimensions in Fig. 5.12 these values are computed by encoding the elites with the final fine-tuned model at the end of the run.

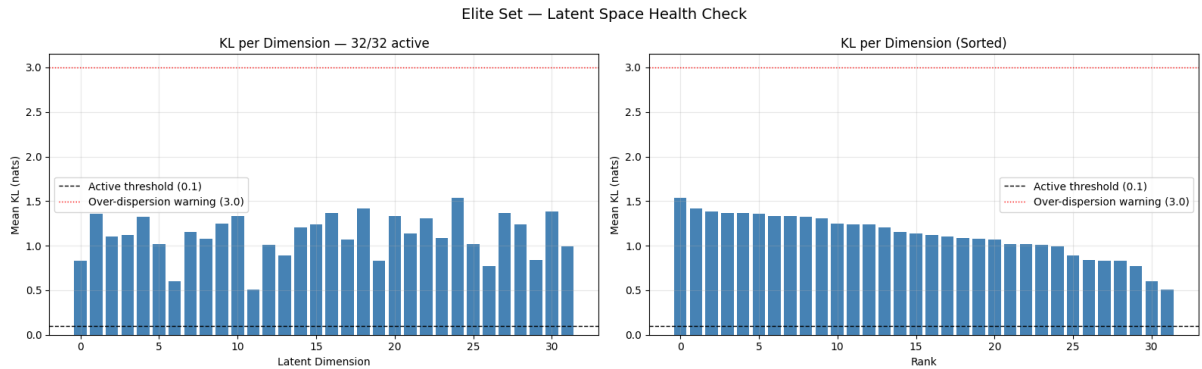


Figure 5.12: Active dimension analysis on the final elites from the **finetune-NS** setup.

To verify whether the latent space's ability to compress information has degraded, we compare the reconstruction quality of the final fine-tuned VAE on the elite set against how the original, non-fine-tuned model would have reconstructed them. As observed in Fig. 5.13, the reconstruction of diverse elites, which the original network struggles to represent, is greatly improved. This comparative improvement highlights the dynamic network's capacity to seamlessly integrate and encode newly discovered, out-of-distribution behavioural trajectories.

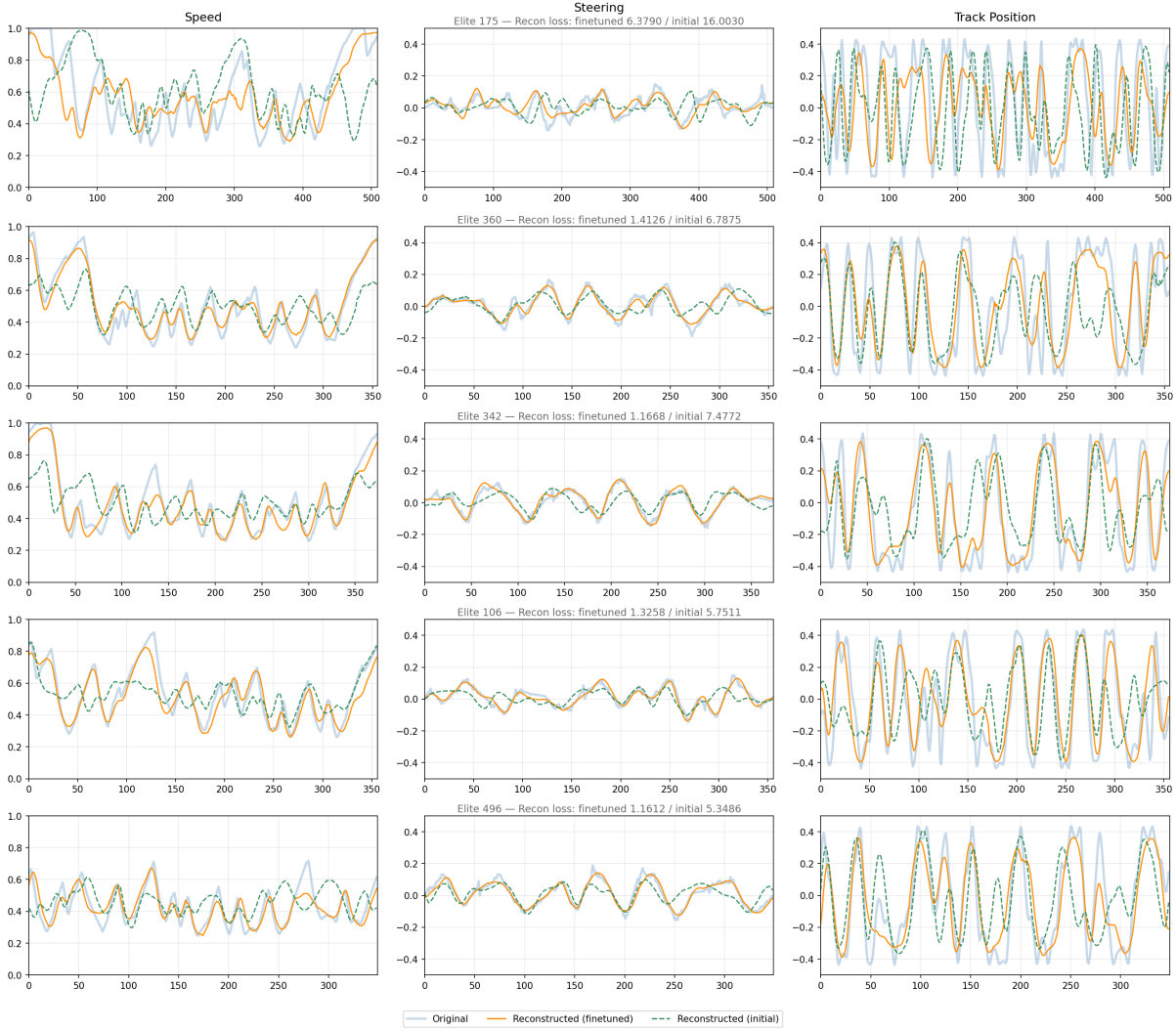


Figure 5.13: Phase aligned time-domain reconstruction of the top five tracks by reconstruction difference (fine-tuned model vs. original pre-trained) from the final elites of the **finetune-NS** setup. The original telemetry (solid light blue), the fine-tuned VAE’s reconstruction (orange), and the original pre-trained VAE’s reconstruction are compared across the three behavioural channels: speed, steering, and distance from track boundary.

Elites Comparison

While both configurations generate a satisfyingly diverse set of elites (the complete elite sets for both configurations are available in Appendix C), a clear issue emerges in the final elite set of the **base-NS** configuration. Beyond the performance advantages of the **finetune-NS** setup outlined in section 5.2.2, the static baseline frequently suffers from the occurrence of highly similar tracks, and occasionally, exact duplicates. While the latter exposes some inefficiencies with the mutation operators, a properly functioning latent

space should naturally prevent the survival of such duplicates.

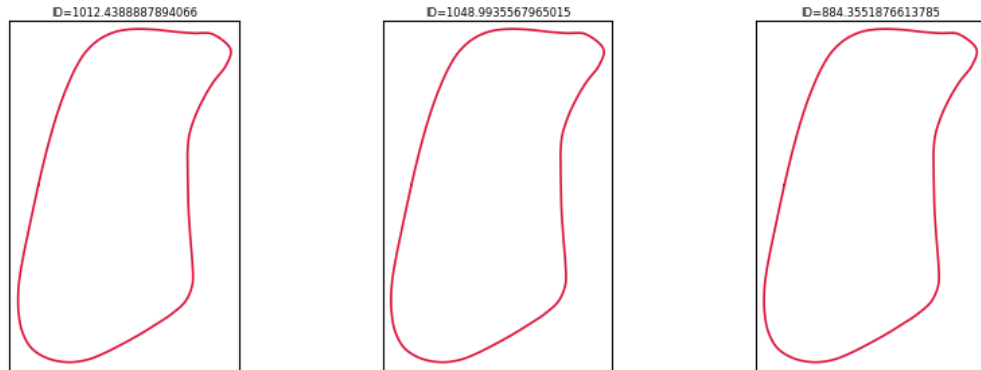


Figure 5.14: Example of three identical tracks in the final elite set of a run using the **base-NS** configuration.

This phenomenon can be explained by the mechanics of the novelty calculation. If the archive contains a highly novel elite that is deeply isolated in the latent space, a very light mutation will place the new individual in that exact same isolated area (see Fig. 5.15). Because both elites are far away from the rest of the population, the k-NN calculation will still output a high average distance score (as displayed in Fig. 5.16). Depending on the degree of isolation, this allows multiple redundant elites to form a cluster and survive in the archive.

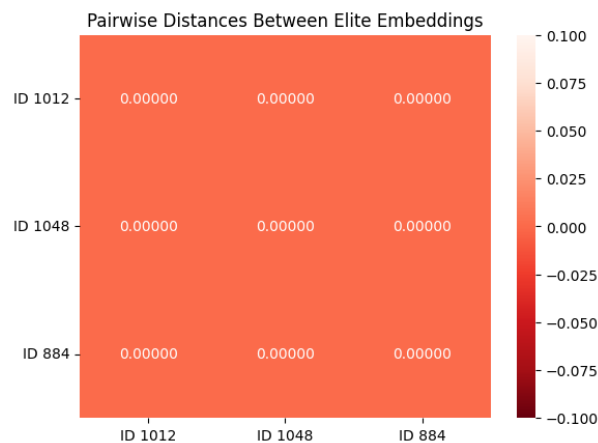


Figure 5.15: Pairwise distance in latent space of tracks plotted in Fig. 5.14

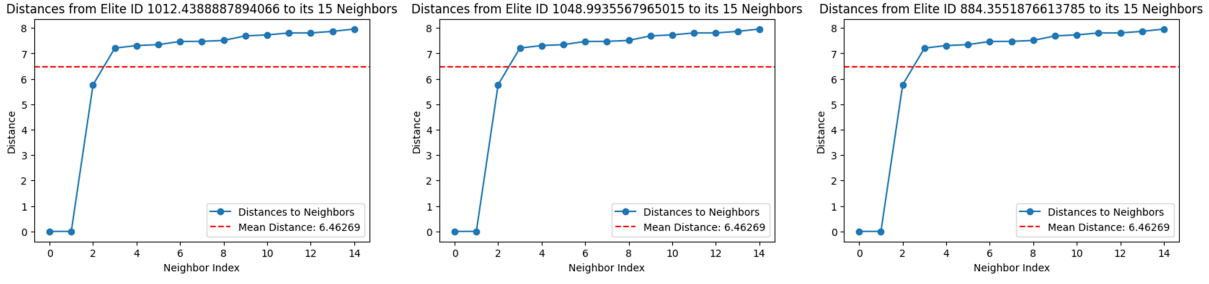


Figure 5.16: Distance from 15 nearest neighbours in the elite set of the tracks displayed in Fig. 5.14.

To further illustrate how frequently this issue occurs without latent space adaptation, Fig. 5.17 and Fig. 5.18 present additional examples of duplicate clusters found in a **base-NS** final elite set. In Fig. 5.17, a cluster of four identical tracks manages to co-exist, while Fig. 5.18 shows a pair of identical tracks. In all these instances, the pairwise distances between the tracks within each cluster evaluate to zero (Fig. 5.19), yet their extreme isolation in the static latent space allows them to maintain a high average k -NN distance score (Fig. 5.20), artificially inflating their novelty and ensuring their survival.



Figure 5.17: A second example showing a cluster of four identical tracks (IDs 960, 994, 1028, 677) in a **base-NS** elite set.

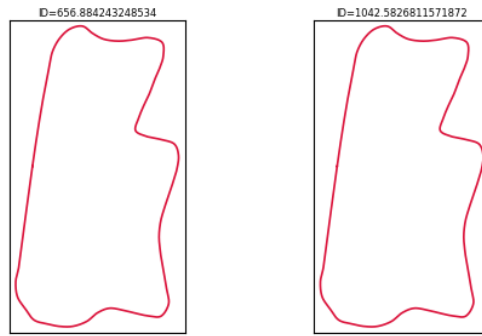
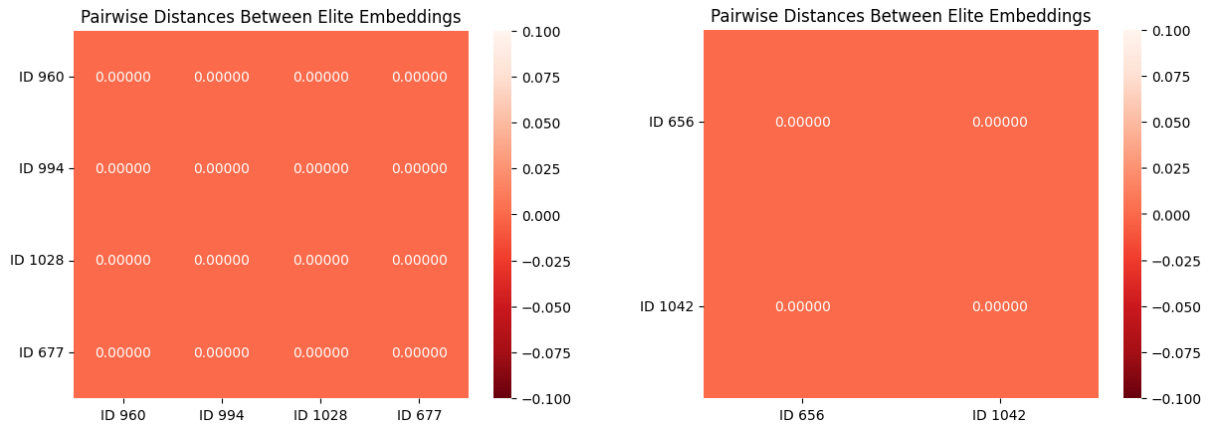


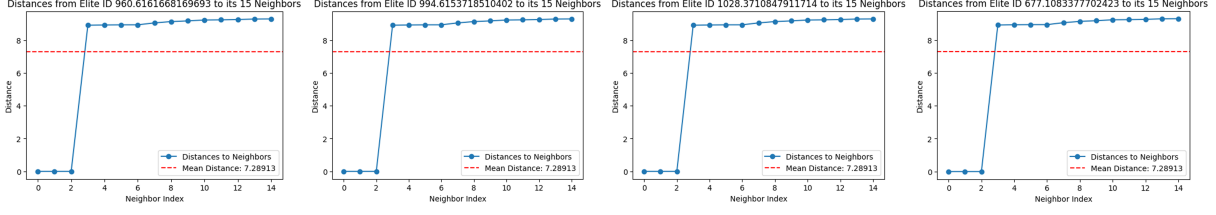
Figure 5.18: A third example showing a pair of identical tracks (IDs 656, 1042) in a base-NS elite set.



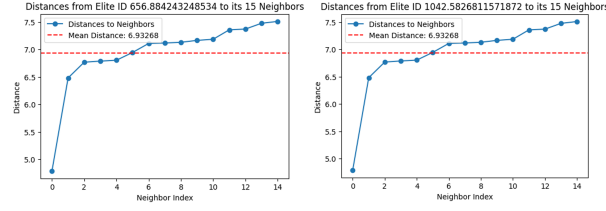
(a) Pairwise distances for the four tracks in Fig. 5.17

(b) Pairwise distances for the two tracks in Fig. 5.18

Figure 5.19: Pairwise distance heatmaps confirming the tracks within each additional cluster are mapped to the same latent coordinates.



(a) 15 nearest neighbours distances for the tracks in Fig. 5.17



(b) 15 nearest neighbours distances for the tracks in Fig. 5.18

Figure 5.20: Distance from the 15 nearest neighbours for the additional duplicate sets. Despite intra-cluster distances of zero, the extreme isolation inflates the mean distance score above the survival threshold.

In the **finetune-NS** configuration, we are able to greatly reduce this problem (as observed in the figures in Appendix C). Thanks to the periodic latent space restructuring triggered by the fine-tuning procedure, elites that were previously extreme outliers are mapped into a more coherent, less isolated position within the updated latent space, preventing duplicates from artificially scoring high novelty.

To illustrate the range of layouts generated by our pipeline, Fig. 5.21 presents a curated selection of twelve distinct track designs from the **finetune-NS** archive. The selected layouts reveal a wide variety of geometric structures. Some tracks combine long, flat straightaways with localised technical sections, as seen in the prominent straight baseline of ID=422.8 or the high-speed diagonal of ID=851.8. Others exhibit repetitive, highly winding structures, such as the vertical serpentine shape of ID=672.6 or the continuous, wave-like undulations of ID=509.0. Furthermore, comparing ID=1084.9 and ID=1096.9 reveals a clear genetic relationship; while they share a highly similar global silhouette, ID=1096.9 features a highly technical curve sequence in its top-left sector instead of the smoother turn of ID=1084.9. These variations are substantial enough to alter driving dynamics and justify their coexistence as separate elites, contrasting with the literal duplicates and structural redundancy observed in the baseline search.



Figure 5.21: A curated selection of twelve track layouts generated by the **finetune-NS** configuration, showcasing the range of geometric and topological variations preserved in the archive.

6 | Conclusions and Future work

Conclusions

The primary objective of this thesis was to develop an automated, data-driven pipeline capable of generating diverse and high-performing racing tracks. By implementing our Novelty Search pipeline, we successfully created a system that explores the vast design space of circuit layouts based on lap car metrics rather than purely geometric heuristics.

A core strength of our research lies in the successful adaptation of an AURORA-like architecture. Traditionally applied to autonomous skill discovery in robotics, the integration of unsupervised representation learning into the evolutionary loop proved highly effective for Procedural Content Generation. By automating the extraction of BDs directly from raw simulation data, the system obviates the need to manually define what makes a track "novel" or "diverse". This paradigm shift allows the algorithm to map the boundaries of the design space autonomously, bypassing the limitations and biases of hand-crafted heuristics to dynamically discover genuinely unique racing experiences.

Crucial to the success of this architecture was the choice of using car metrics as BDs and the design of the ad hoc VAE. As demonstrated in our experiments, the resulting embedding space accurately clustered structurally and behaviourally similar tracks, providing a robust metric space for Novelty Search to operate within.

Furthermore, the comparison between the static (`base-NS`) and dynamic (`finetune-NS`) configurations highlighted the benefits of active latent space restructuring. The dynamic pipeline successfully mitigated the formation of redundant tracks, indicating that continuous adaptation of the representation space yields a more genuinely diverse set of elite solutions.

Future Work

While the proposed pipeline achieved promising results, several areas present opportunities for refinement and expansion in future research.

Refining Dynamic Representation Learning Although the fine-tuning of the VAE successfully resolved the issue of highly isolated duplicate tracks, the dynamic updating of the latent space requires further stabilisation. Training a deep generative model on small, newly discovered batches of out-of-distribution data makes the network highly sensitive to hyper-parameter choices. Future iterations could benefit from more sophisticated Continual Learning techniques.

Advanced AI Drivers The quality and expressiveness of the BD are intrinsically tied to the telemetry generated during the evaluation phase. In this work, we relied on a selection of TORCS's built-in heuristic bots (*berniw*, *tita*, *inferno*). While they provided a reliable and computationally inexpensive baseline, their somewhat rigid driving behaviours inherently limit the depth of the emergent phenomena captured by the VAE. Replacing these traditional bots with more advanced agents could expose much more nuanced vehicle dynamics, thereby providing a much richer phenotypic dataset for the QD algorithm to explore. Furthermore, complex multi-agent interactions could also yield a richer Fitness landscape.

Simulation Environment Enhancements Finally, the choice of TORCS as our simulation engine brought both significant advantages and notable limitations. Its open-source nature, lightweight execution, and ability to be containerised into ephemeral Docker instances were instrumental in achieving the throughput necessary for evolutionary search. However, TORCS is built on a legacy codebase with a rigid XML schema that enforces sequential, segment-by-segment track generation and doesn't allow for elevation changes or intersecting tracks. The segment-by-segment generation characteristic was the root cause of the "closure problem" described in this work, necessitating complex geometric workarounds and engine patches.

Future extensions of this framework could migrate the evaluation pipeline to a more modern, flexible simulation engine.

Bibliography

- [1] P. G. Boracchi and P. M. Matteucci. Advanced neural networks and deep learning. Course slides, 2024. Politecnico di Milano.
- [2] E. Catmull and R. Rom. A class of local interpolating splines. In R. E. BARNHILL and R. F. RIESENFELD, editors, *Computer Aided Geometric Design*, pages 317–326. Academic Press, 1974. ISBN 978-0-12-079050-0. doi: <https://doi.org/10.1016/B978-0-12-079050-0.50020-5>. URL <https://www.sciencedirect.com/science/article/pii/B9780120790500500205>.
- [3] A. Cully. Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19*, page 81–89. ACM, July 2019. doi: 10.1145/3321707.3321804. URL <http://dx.doi.org/10.1145/3321707.3321804>.
- [4] A. Cully and Y. Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2018. doi: 10.1109/TEVC.2017.2704781.
- [5] Docker Inc. Docker: Accelerated container application development, 2026. URL <https://www.docker.com/>.
- [6] M. Farrokhi Maleki and R. Zhao. Procedural content generation in games: A survey with insights on emerging llm integration. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 20(1):167–178, Nov. 2024. ISSN 2326-909X. doi: 10.1609/aiide.v20i1.31877. URL <http://dx.doi.org/10.1609/aiide.v20i1.31877>.
- [7] S. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1):153–174, 1987. doi: 10.1007/BF01840357.
- [8] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin. Cyclical annealing schedule: A simple approach to mitigating KL vanishing. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*,

- Volume 1 (Long and Short Papers)*, pages 240–250, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1021. URL <https://aclanthology.org/N19-1021/>.
- [9] L. Grillotti and A. Cully. Unsupervised behaviour discovery with quality-diversity optimisation, 2022. URL <https://arxiv.org/abs/2106.05648>.
 - [10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
 - [11] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.
 - [12] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.
 - [13] IBM. What is an autoencoder?, 2026. URL <https://www.ibm.com/think/topics/autoencoder>.
 - [14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
 - [15] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
 - [16] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
 - [17] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lönneker, L. Cardamone, D. Perez, Y. Sáez, M. Preuss, and J. Quadflieg. The 2009 simulated car racing championship. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):131–147, 2010. doi: 10.1109/TCIAIG.2010.2050590.
 - [18] K. Perlin. Improving noise. *ACM Trans. Graph.*, 21(3):681–682, July 2002. ISSN 0730-0301. doi: 10.1145/566654.566636. URL <https://doi.org/10.1145/566654.566636>.
 - [19] M. Piaggi. Quality diversity for racing track design, 2024. URL <https://www.politesi.polimi.it/handle/10589/239983>. Supervisors: Pierluca Lanzi, Daniele Loiacono.
 - [20] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

- [21] J. Sirianni. Supporto alla progettazione e analisi di tracciati per un simulatore di guida, July 2016. URL <https://hdl.handle.net/10589/122987>. Supervisor: Daniele Loiacono.
- [22] B. Tjanaka, M. C. Fontaine, D. H. Lee, Y. Zhang, N. R. Balam, N. Dennler, S. S. Garlanka, N. D. Klapsis, and S. Nikolaidis. Pyribs: A bare-bones python library for quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '23*, page 220–229, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701191. doi: 10.1145/3583131.3590374. URL <https://doi.org/10.1145/3583131.3590374>.
- [23] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011. doi: 10.1109/TCIAIG.2011.2148116.
- [24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [25] Wikipedia contributors. Beneath apple manor, 2026. URL https://en.wikipedia.org/wiki/Beneath_Apple_Manor.
- [26] Wikipedia contributors. Elite (video game), 2026. URL [https://en.wikipedia.org/wiki/Elite_\(video_game\)](https://en.wikipedia.org/wiki/Elite_(video_game)).
- [27] Wikipedia contributors. L-system, 2026. URL <https://en.wikipedia.org/wiki/L-system>.
- [28] B. Wymann, E. Espié, C. Guionneau, et al. TORCS: The open racing car simulator. <https://sourceforge.net/projects/torcs/>, 2026.

A | Phenotype Metrics Analysis

This appendix provides a detailed breakdown of the telemetry data captured during the benchmark simulation phases. Included are the probability distributions of the raw signals, correlation matrices between different metrics, and example time-domain sequences that illustrate the input data processed by the VAE.

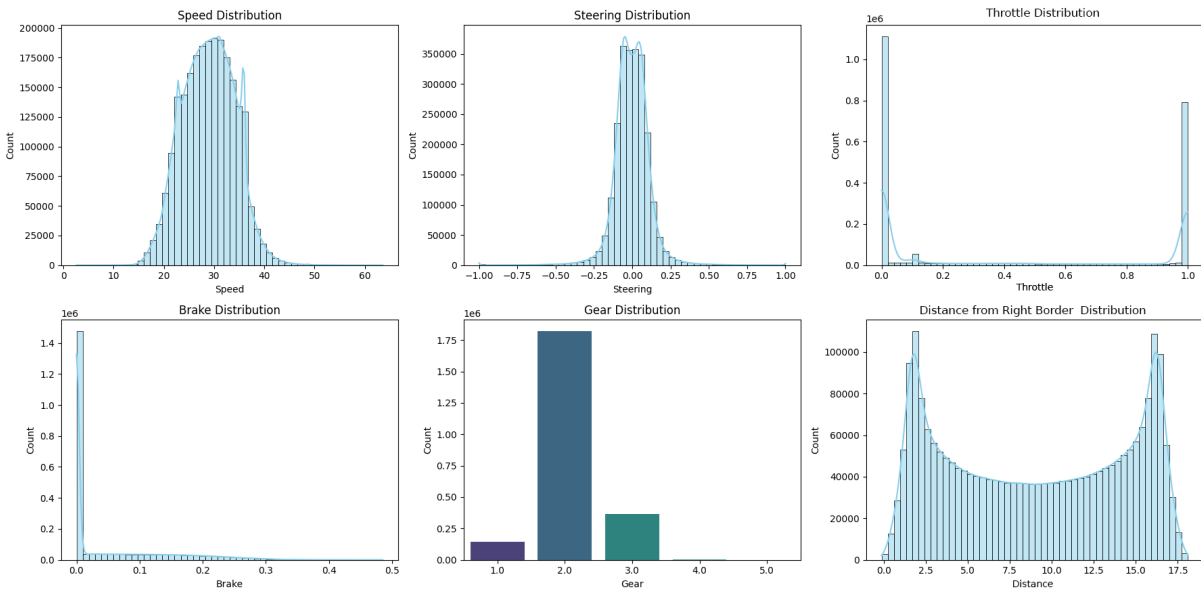


Figure A.1: Empirical distributions of the six raw telemetry metrics collected across the pre-training dataset during benchmark laps. The histograms highlight the distinct statistical behaviours that motivated the feature selection process. The retained channels (**speed**, **steering**, and **distance from track boundary**) exhibit continuous, information-rich distributions. In contrast, the discarded channels (**throttle**, **brake**, and **gear**) behave primarily as discrete or highly polarised on/off signals—spending the majority of the lap pinned at their extreme values—rendering them ill-suited for the continuous reconstruction objective of the VAE.

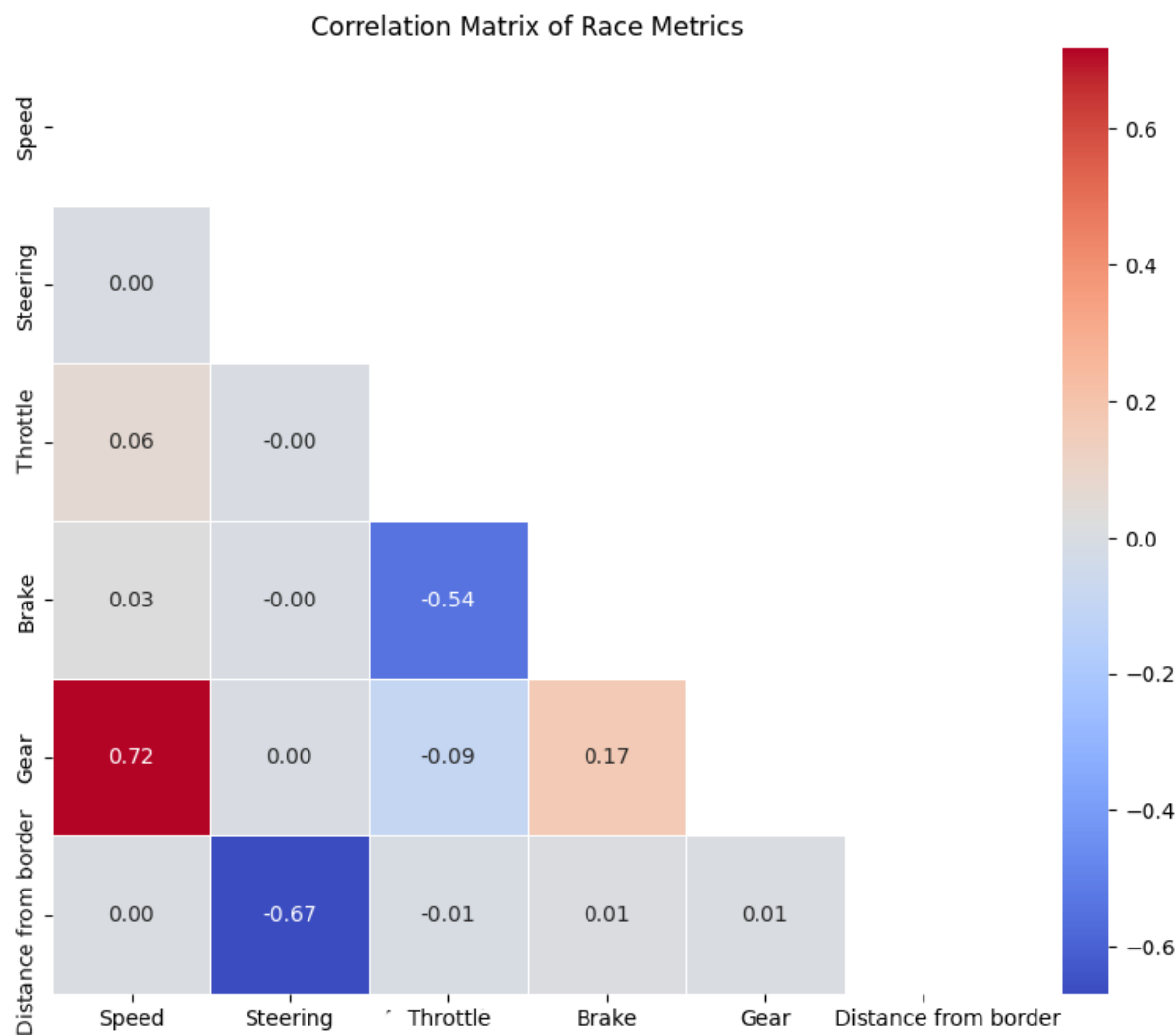


Figure A.2: Pearson correlation matrix of the raw telemetry metrics across the pre-training dataset. This heatmap provides empirical support for the feature reduction strategy discussed in Section 4.2.1. Notably, the strong positive correlation ($r = 0.72$) between **Gear** and **Speed** confirms that gear selection operates as a quantisation of the vehicle’s speed. Additionally, the inverse relationship between **Throttle** and **Brake** ($r = -0.54$) reflects the mutually exclusive nature of longitudinal inputs.

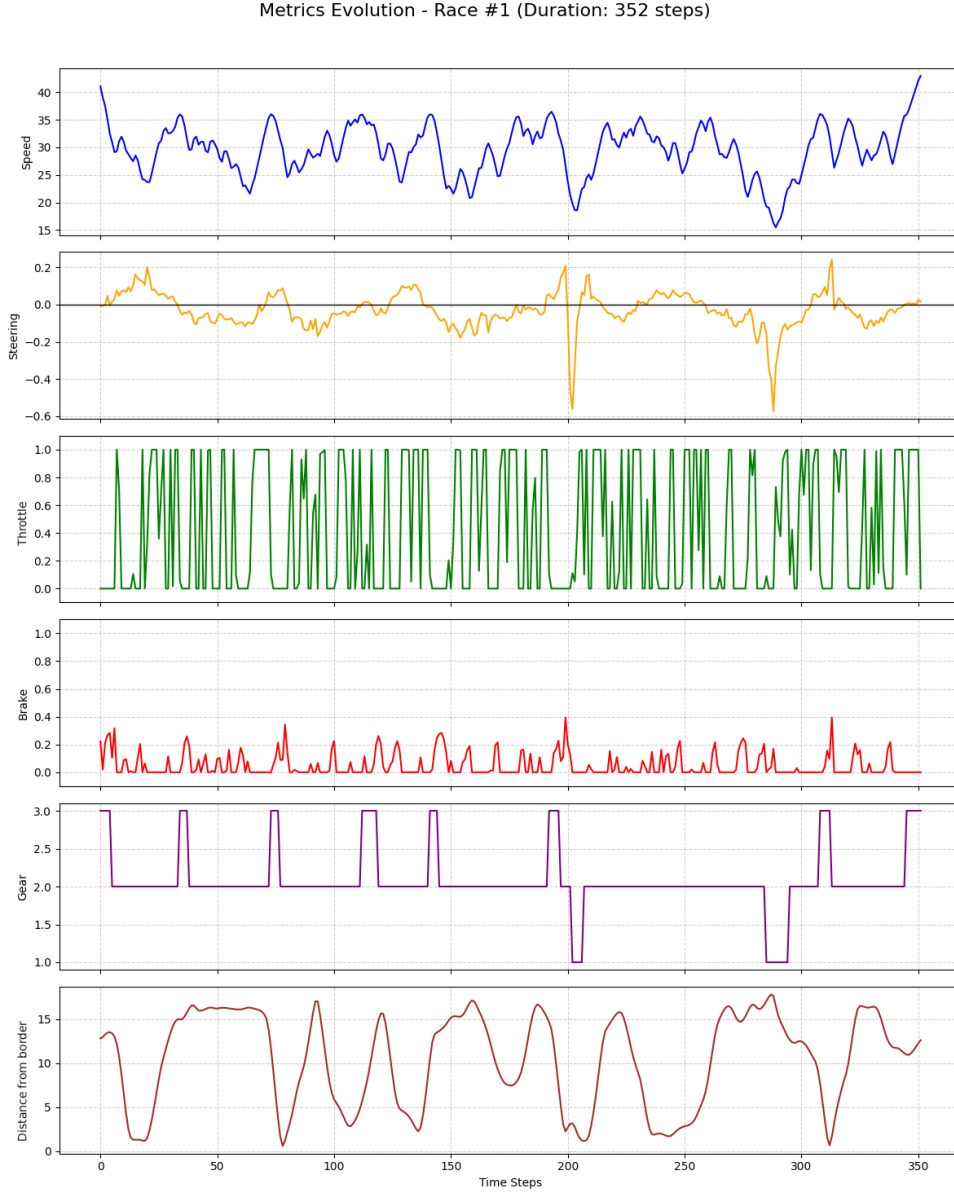


Figure A.3: Time-domain visualisation of the six raw telemetry channels recorded over a single benchmark lap on a randomly generated track. This sequential view provides a final visual confirmation of the feature selection rationale outlined in Section 4.2.1. The retained metrics (**Speed**, **Steering**, and **Distance from border**) form smooth, continuous waveforms that effectively capture the physical dynamics and spatial layout of the circuit, making them ideal inputs for the VAE’s convolutional architecture and Fourier-based pooling. Conversely, the discarded metrics exhibit discontinuous or erratic behaviours: **Throttle** heavily oscillates like a high-frequency binary "bang-bang" controller, **Brake** is characterised by sparse, sharp spikes, and **Gear** acts as a rigid step function. Such non-smooth, high-frequency signals would severely disrupt the model’s continuous reconstruction objective.

B | Latent Space Analysis

This appendix offers a deeper look into the structural health of the VAE's latent bottleneck. The plots demonstrate the distribution of latent means against the standard normal prior and show pairwise scatter plots of latent dimensions to confirm that the space is dense, isotropic, and free of significant feature correlations.

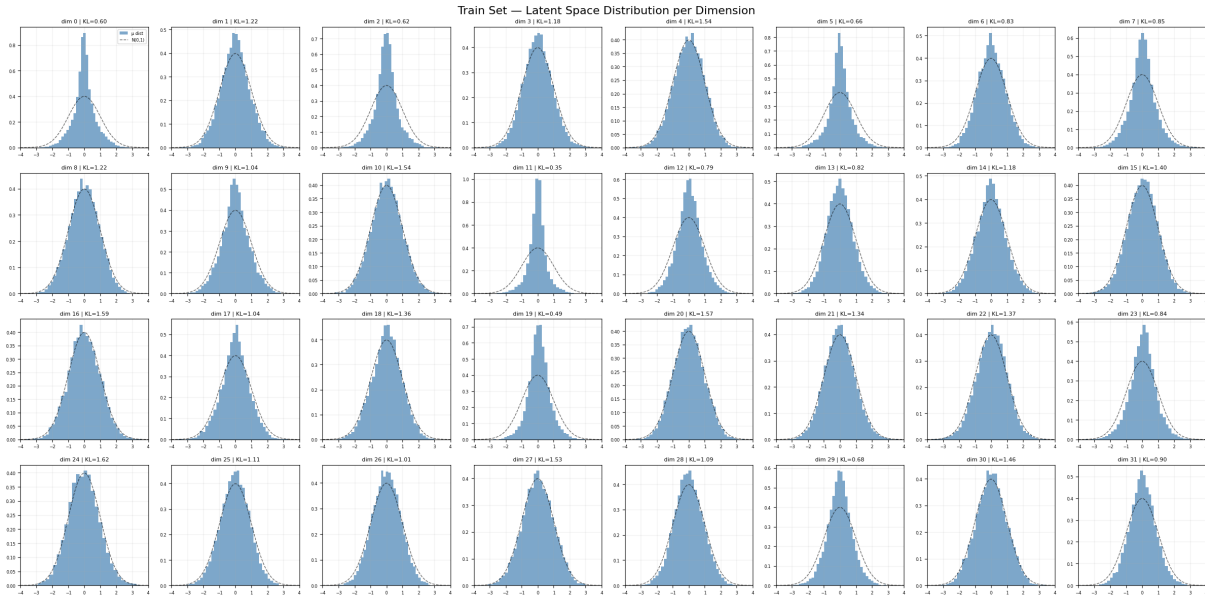


Figure B.1: Empirical distributions of the encoded latent means (μ) across all 32 dimensions for the pre-training dataset, overlaid with the standard normal prior $\mathcal{N}(0, 1)$ (dashed lines). This grid provides direct visual confirmation of the network’s structural health discussed in Section 5.1.3. None of the distributions have collapsed into zero-variance spikes at the origin, corroborating the quantitative finding in Fig. 5.3 that all 32 dimensions remain actively engaged in encoding track variations. Furthermore, the close adherence to the target prior without severe flattening (overdispersion) suggests that the cyclical β annealing schedule successfully regularised the latent space, forcing a dense, continuous manifold.

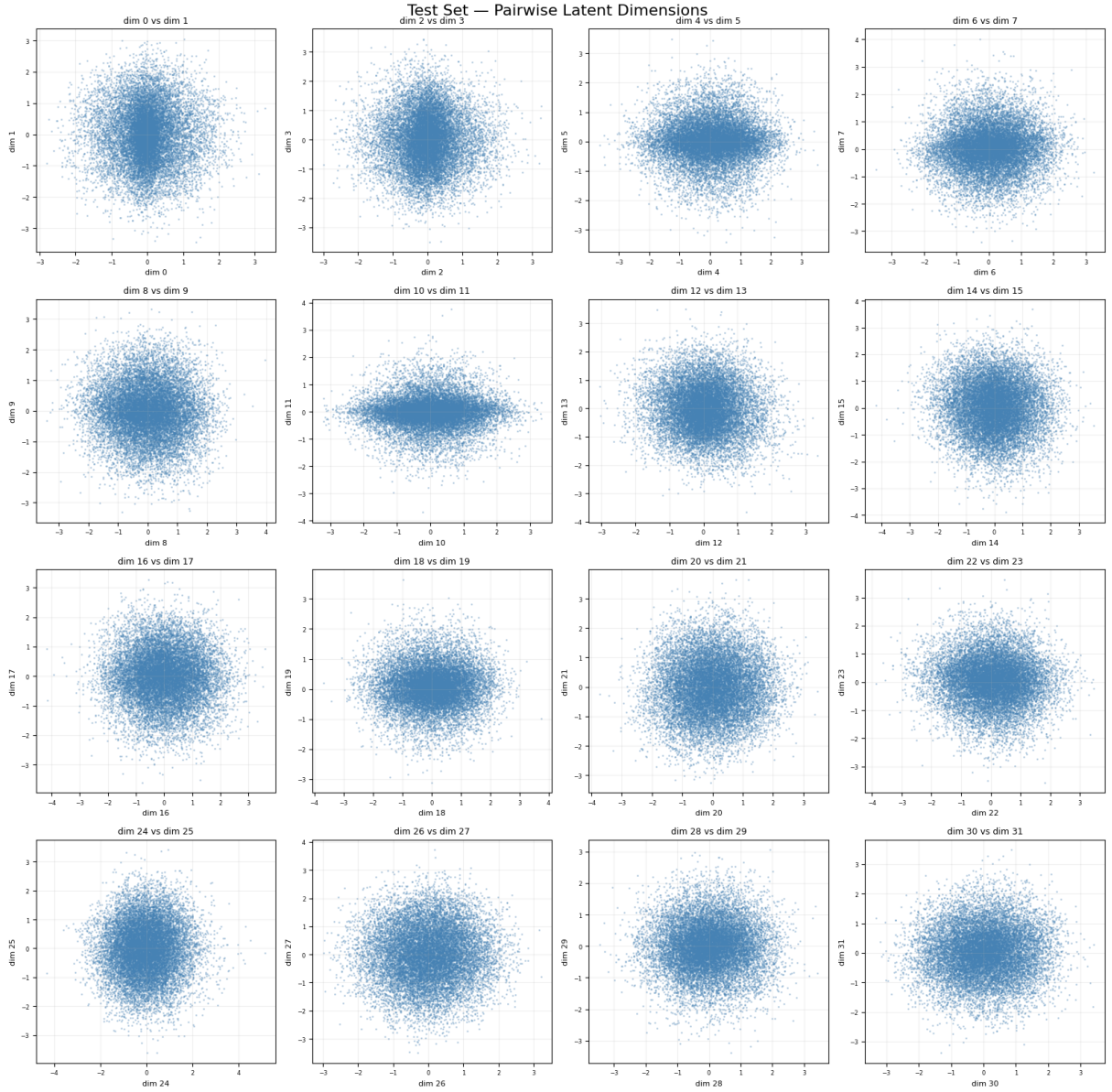


Figure B.2: Pairwise scatter plots of adjacent latent dimensions evaluated on the unseen test dataset. The embeddings form dense, roughly spherical clusters centered at the origin, confirming that the KL-divergence regularisation successfully pulled the approximate posterior toward the isotropic multivariate normal prior, $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Notably, the latent space is devoid of the fragmented gaps or isolated data islands typical of standard deterministic autoencoders, and the lack of strong diagonal structures indicates that the network has learned largely uncorrelated latent variables. This continuous, unfragmented geometry is critical for the system’s success, as it provides a reliable, well-behaved metric space where the Euclidean distance calculations used by the k -Nearest Neighbours (k -NN) Novelty Search algorithm accurately reflect genuine behavioural divergence.

C | Resulting Racing Tracks

This section visualises the final collections of racing tracks (the "elites") discovered by both the baseline and fine-tuned Novelty Search configurations. The t-SNE embeddings project the high-dimensional track geometries into a 2D plane, providing a qualitative overview of the structural diversity and clustering achieved by the evolutionary process.



Figure C.1: t-SNE embedding of track renders of elite archive from the `finetune-NS` configuration at the final iteration. The embedding is computed from a rotation-invariant descriptors (angularly averaged 2-D Fourier power spectrum), so proximity reflects similarity of track geometry independent of orientation.

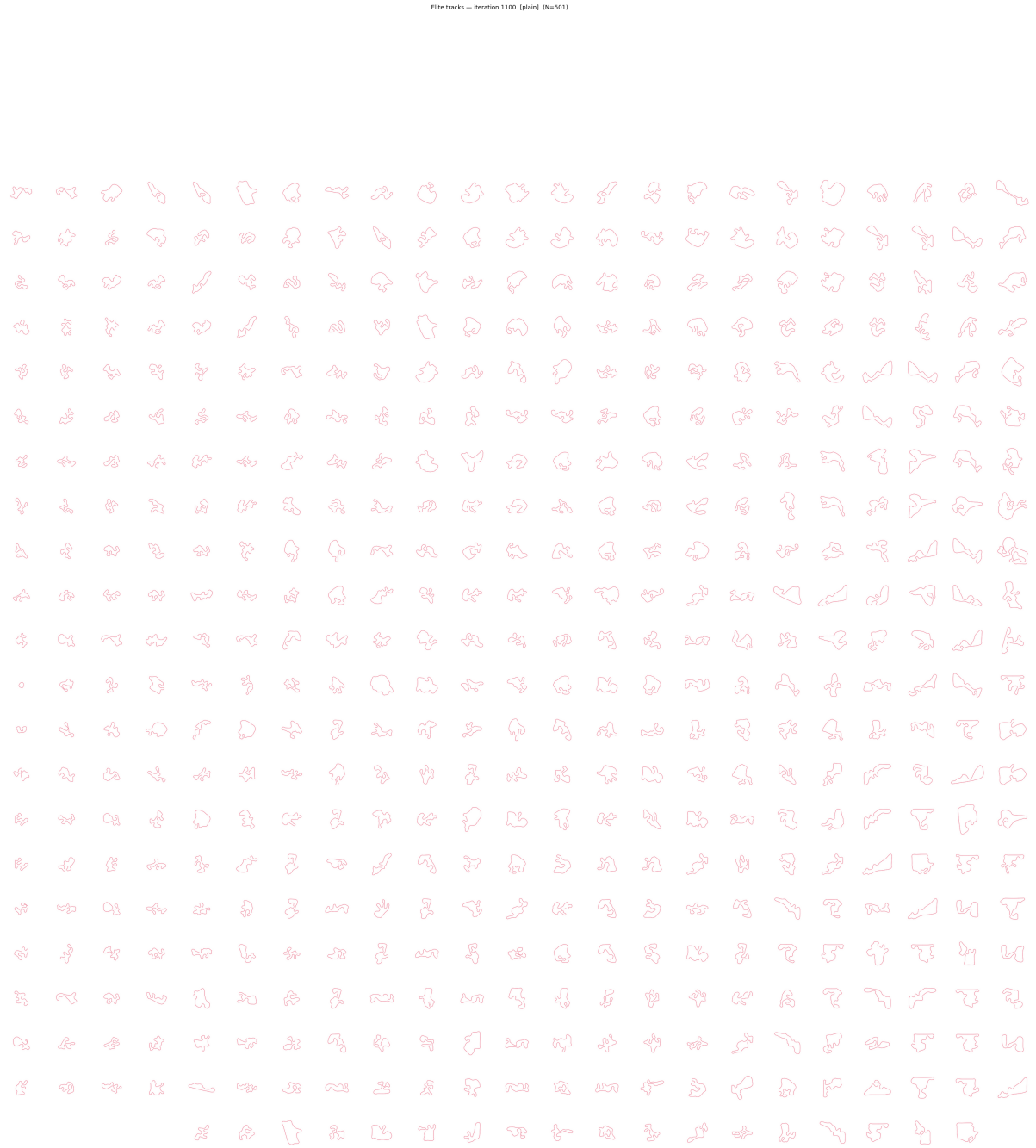


Figure C.2: Grid view of t-SNE embedding of elite archive from the **finetune-NS** configuration at the final iteration. The embedding is computed from a rotation-invariant descriptors (angularly averaged 2-D Fourier power spectrum), so proximity reflects similarity of track geometry independent of orientation.



Figure C.3: t-SNE embedding of track renders of elite archive from the **base-NS** configuration at the final iteration. The embedding is computed from a rotation-invariant descriptors (angularly averaged 2-D Fourier power spectrum), so proximity reflects similarity of track geometry independent of orientation.



Figure C.4: Grid view of t-SNE embedding of elite archive from the **base-NS** configuration at the final iteration. The embedding is computed from a rotation-invariant descriptors (angularly averaged 2-D Fourier power spectrum), so proximity reflects similarity of track geometry independent of orientation.

List of Figures

2.1	Overview of the Quality-Diversity components. Reproduced from [4]	8
2.2	A summary image of the AURORA pipeline. Reproduced from [3]	10
2.3	Typical Structure of an autoencoder [1]	12
3.1	A gameplay screenshot from The Open Racing Car Simulator (TORCS). .	15
3.2	Simplex noise (left) yields uneven Voronoi cells, small in clusters, large across voids. Uniform random (right) produces more evenly sized cells. . .	22
3.3	Two generated track lines after cell selection and smoothing steps, simplex noise (left) uniform (right); the simplex noise based generation displays its potential of generating large straight sections in sparse areas, whilst cluster enforce a winding topology.	24
3.4	Winding canonicalisation on mirrored tracks	25
3.5	Winding canonicalisation on reversed track	25
4.1	Overview of our Quality Diversity pipeline.	30
4.2	Mutated track and its parent: a small change in a site's position heavily accentuates a previously soft curve.	31
4.3	Crossover between two parent tracks: the offspring track inherits distinctive features from both parents.	32
4.4	Overview of the VAE's architecture	35
4.5	Distribution of race metrics values in the pre-training dataset	37
4.6	Receptive field of a single output position after dilations $[1, 2, 4, 8, 16]$, $kernelsize = 7$; Every active cell has the same fan-out pattern; "...": earlier positions, off-screen; for clarity, the dilated-tap edges are drawn for only one representative cell per layer, since every active cell shares the same fan-out pattern.	39
5.1	Plots of reconstruction (left), KLD (centre) and total (right) losses spanning the entire training.	46
5.2	Plots of reconstruction (left), KLD (centre) and total (right) losses starting from epoch 5.	47

5.3	Active dimension analysis on training set.	48
5.4	Phase aligned time-domain reconstruction of five randomly sampled tracks from the test set. The original telemetry (solid blue) and the VAE's reconstruction (dashed orange) are compared across the three behavioural channels: speed, steering, and distance from track boundary.	49
5.5	Time-domain visualisation of five random query tracks (solid orange) and their 5 nearest Neighbours in the latent space (blue, with darker shades indicating closer proximity). Neighbours are resampled to match the query length and phase-aligned to account for the VAE's shift-invariance.	50
5.6	Archive population growths, finetune -NS configuration on the left and base -NS configuration on the right.	53
5.7	Archive insertions and substitutions dynamics for finetune -NS configuration.	54
5.8	Archive insertions and substitutions dynamics for base -NS configuration.	54
5.9	Fitness dynamics for both configurations, finetune -NS configuration on the left, and base -NS on the right. High-Quality Coverage counts the number of elites with Fitness ≥ 20	55
5.10	Validation Reconstruction loss recorded over cumulative fine-tuning epochs. Fine-tuning triggered at specific iterations, labelled above the plot. The final saved epoch for each cycle is marked by an enlarged circle.	56
5.11	Validation Kullback-Leibler Divergence (KLD) loss recorded over cumulative fine-tuning epochs. Fine-tuning triggered at specific iterations, labelled above the plot. The final saved epoch for each cycle is marked by an enlarged circle.	56
5.12	Active dimension analysis on the final elites from the finetune -NS setup.	57
5.13	Phase aligned time-domain reconstruction of the top five tracks by reconstruction difference (fine-tuned model vs. original pre-trained) from the final elites of the finetune -NS setup. The original telemetry (solid light blue), the fine-tuned VAE's reconstruction (orange), and the original pre-trained VAE's reconstruction are compared across the three behavioural channels: speed, steering, and distance from track boundary.	58
5.14	Example of three identical tracks in the final elite set of a run using the base -NS configuration.	59
5.15	Pairwise distance in latent space of tracks plotted in Fig. 5.14	59
5.16	Distance from 15 nearest neighbours in the elite set of the tracks displayed in Fig. 5.14.	60
5.17	A second example showing a cluster of four identical tracks (IDs 960, 994, 1028, 677) in a base -NS elite set.	60

5.18	A third example showing a pair of identical tracks (IDs 656, 1042) in a base-NS elite set.	61
5.19	Pairwise distance heatmaps confirming the tracks within each additional cluster are mapped to the same latent coordinates.	61
5.20	Distance from the 15 nearest neighbours for the additional duplicate sets. Despite intra-cluster distances of zero, the extreme isolation inflates the mean distance score above the survival threshold.	62
5.21	A curated selection of twelve track layouts generated by the finetune-NS configuration, showcasing the range of geometric and topological variations preserved in the archive.	63
A.1	Empirical distributions of the six raw telemetry metrics collected across the pre-training dataset during benchmark laps. The histograms highlight the distinct statistical behaviours that motivated the feature selection process. The retained channels (speed , steering , and distance from track boundary) exhibit continuous, information-rich distributions. In contrast, the discarded channels (throttle , brake , and gear) behave primarily as discrete or highly polarised on/off signals—spending the majority of the lap pinned at their extreme values—rendering them ill-suited for the continuous reconstruction objective of the VAE.	72
A.2	Pearson correlation matrix of the raw telemetry metrics across the pre-training dataset. This heatmap provides empirical support for the feature reduction strategy discussed in Section 4.2.1. Notably, the strong positive correlation ($r = 0.72$) between Gear and Speed confirms that gear selection operates as a quantisation of the vehicle’s speed. Additionally, the inverse relationship between Throttle and Brake ($r = -0.54$) reflects the mutually exclusive nature of longitudinal inputs.	73

- A.3 Time-domain visualisation of the six raw telemetry channels recorded over a single benchmark lap on a randomly generated track. This sequential view provides a final visual confirmation of the feature selection rationale outlined in Section 4.2.1. The retained metrics (**Speed**, **Steering**, and **Distance from border**) form smooth, continuous waveforms that effectively capture the physical dynamics and spatial layout of the circuit, making them ideal inputs for the VAE's convolutional architecture and Fourier-based pooling. Conversely, the discarded metrics exhibit discontinuous or erratic behaviours: **Throttle** heavily oscillates like a high-frequency binary "bang-bang" controller, **Brake** is characterised by sparse, sharp spikes, and **Gear** acts as a rigid step function. Such non-smooth, high-frequency signals would severely disrupt the model's continuous reconstruction objective. . . . 74
- B.1 Empirical distributions of the encoded latent means (μ) across all 32 dimensions for the pre-training dataset, overlaid with the standard normal prior $\mathcal{N}(0, 1)$ (dashed lines). This grid provides direct visual confirmation of the network's structural health discussed in Section 5.1.3. None of the distributions have collapsed into zero-variance spikes at the origin, corroborating the quantitative finding in Fig. 5.3 that all 32 dimensions remain actively engaged in encoding track variations. Furthermore, the close adherence to the target prior without severe flattening (overdispersion) suggests that the cyclical β annealing schedule successfully regularised the latent space, forcing a dense, continuous manifold. 76
- B.2 Pairwise scatter plots of adjacent latent dimensions evaluated on the unseen test dataset. The embeddings form dense, roughly spherical clusters centered at the origin, confirming that the KL-divergence regularisation successfully pulled the approximate posterior toward the isotropic multivariate normal prior, $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Notably, the latent space is devoid of the fragmented gaps or isolated data islands typical of standard deterministic autoencoders, and the lack of strong diagonal structures indicates that the network has learned largely uncorrelated latent variables. This continuous, unfragmented geometry is critical for the system's success, as it provides a reliable, well-behaved metric space where the Euclidean distance calculations used by the k -Nearest Neighbours (k -NN) Novelty Search algorithm accurately reflect genuine behavioural divergence. 77

C.1	t-SNE embedding of track renders of elite archive from the finetune-NS configuration at the final iteration. The embedding is computed from a rotation-invariant descriptors (angularly averaged 2-D Fourier power spectrum), so proximity reflects similarity of track geometry independent of orientation.	80
C.2	Grid view of t-SNE embedding of elite archive from the finetune-NS configuration at the final iteration. The embedding is computed from a rotation-invariant descriptors (angularly averaged 2-D Fourier power spectrum), so proximity reflects similarity of track geometry independent of orientation.	81
C.3	t-SNE embedding of track renders of elite archive from the base-NS configuration at the final iteration. The embedding is computed from a rotation-invariant descriptors (angularly averaged 2-D Fourier power spectrum), so proximity reflects similarity of track geometry independent of orientation. .	82
C.4	Grid view of t-SNE embedding of elite archive from the base-NS configuration at the final iteration. The embedding is computed from a rotation-invariant descriptors (angularly averaged 2-D Fourier power spectrum), so proximity reflects similarity of track geometry independent of orientation. .	83

List of Tables

3.1	Track geometry metrics extracted from the generator.	16
3.2	Telemetry channels recorded during the benchmark lap.	17
3.3	Emergent metrics from multi-agent simulations	18
3.4	The AI driver and car combinations selected for the race simulations, with a qualitative description of each driver’s behaviour.	19

Acknowledgements

I would like to express my deepest gratitude to Prof. Daniele Loiacono and Prof. Pierluca Lanzi for granting me the opportunity to conclude my Master's degree with such interesting and compelling research. This project has allowed me to further explore my passion for video game development and my growing interest in deep learning. I am also grateful for their Videogame Design and Programming course, which remains one of the most inspiring and impactful experiences of my academic career. Finally, I would like to thank Martino Piaggi for providing the foundation of this project with his thesis and for suggesting that I expand upon his work.

