JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN
THM TECHNISCHE HOCHSCHULE MITTELHESSEN | CAMPUS GIESSEN | MNI Mathematik, Naturwissenschaften und Informatik

Department 08 / Bioinformatics

Transformer-Based Wasserstein Autoencoder for De Novo Antimicrobial Peptide Design

Master's Thesis

submitted in partial fulfillment of the requirements for the degree of

Master of Science

by

Reda Mabrouki

Supervisor: Prof. Dr. Franz Cemic

Co-Supervisor: M. Sc. Fabian Tann

External-Supervisor: M. Sc. Dominik March

January 10, 2026

Abstract

The escalating crisis of antimicrobial resistance necessitates the development of novel therapeutic agents, yet the discovery of antimicrobial peptides through traditional experimental methods remains constrained by the vastness of the sequence space and the resource-intensive nature of synthesis and testing. This thesis presents the design, implementation, and evaluation of a Transformer-based Wasserstein Autoencoder with Maximum Mean Discrepancy (TWAE-MMD), a deep generative framework for the *de novo* generation of antimicrobial peptide candidates. The framework integrates the representational capacity of Transformer architecture with the distributional regularization properties of Wasserstein Autoencoders, employing multi-head attention mechanisms to capture both local and long-range dependencies in peptide sequences. The encoded representations are projected into a continuous latent space regularized through Maximum Mean Discrepancy and Wasserstein distance metrics, facilitating principled sampling and optimization. The model was trained on curated peptide sequences from five public databases (DBAASP, DRAMP, CAMP, APD, and additional sources) using a multi-objective loss function combining reconstruction, classification, and regularization objectives. Generated peptides were computationally validated through established prediction servers for activity and structural analysis, demonstrating that the framework produces diverse, chemically plausible candidates with properties characteristic of antimicrobial peptides. This work establishes a methodological foundation for AI-driven peptide design, with potential generalizability to other classes of bioactive molecules.

Acknowledgments

TBA

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

The 21st century has seen great progress in medicine. However, the rise of antimicrobial resistance (AMR) is a serious problem [1] [2]. It could cancel out all of these gains [1]. The World Health Organization (WHO) says antibiotic resistance (AMR) is a global health crisis that could undermine many important aspects of modern medicine [1] [2]. Routine surgical procedures, cancer chemotherapy, and organ transplants become much more dangerous without effective antibiotics [1]. Projections indicate that by 2050, AMR could claim 10 million lives per year [1] [2]. That is more than the current number of deaths from cancer and diabetes combined [1]. The usual process of creating new antibiotics has problems [2]. These problems include high costs, long development times, and lower returns on investment [2]. This is a serious situation. We need to change how we find and make new medicines to fight infections [2].

Antimicrobial peptides (AMPs) are a good alternative to conventional antibiotics [3] [4]. AMPs are important for the immune system of most living things [3]. They can fight many different types of germs in different ways [3]. Some ways include breaking down cell membranes [3]. It is less likely that germs can develop resistance to AMP [3]. However, there are so many possible peptide sequences that it would be impossible to test them all [4]. Nature has provided a template, but the challenge is in navigating this vast chemical space to efficiently identify new, strong, and safe AMP candidates [4] [17].

1.2 Current Approaches to AMP Discovery

The search for new AMPs has evolved from traditional biochemical methods to sophisticated computational approaches. This section gives a general idea of the current situation. It highlights the strengths and weaknesses of current methods and provides background for the work in this thesis.

1.2.1 Traditional and High-Throughput Screening

Historically, AMPs were discovered by purifying and studying them from natural sources (in vivo) such as plants, insects, and animal tissues [3] [4]. This approach has created many basic AMP families, but it is slow, expensive, and limited to what can be easily separated [4]. The ability to do this quickly and in large quantities has made it easier [4]. This is called high-throughput screening (HTS) and combinatorial chemistry [4]. These methods have enabled us to quickly synthesize and test a large number of peptides [4]. However, these libraries often consist of random sequences or simple variations of known AMPs, and their exploration lacks strategic direction [4] [5]. This often leads to a low hit rate and a failure to explore a significant portion of the chemical space, making it an inefficient method for discovering truly new scaffolds [4] [5].

1.2.2 Computational Approaches and Machine Learning

Experimental screening has clear limitations, which is why computational methods gained popularity as an alternative [5]. The first generation of these approaches centered on quantitative structure-activity relationship (QSAR) models [5]. These models applied machine learning algorithms—support vector machines and

random forests being common examples—to predict a peptide's activity [5]. The predictions relied on a set of physicochemical descriptors (charge, hydrophobicity, helicity, and so on) that had to be defined in advance [5]. QSAR models worked well enough for filtering through virtual libraries, but they had a fundamental problem [5]. They could only assess what was already there [5]. Since they operated entirely on predefined features, they had no mechanism for proposing new sequences [5] [17].

1.2.3 Deep Generative Models for De Novo Design

Deep generative models offered a new direction for de novo molecular design [6] [7] [8] [17]. The main idea was that these models could learn the underlying patterns from known AMPs and then generate new sequences [6] [17]. They work by creating a low-dimensional, continuous "latent" map of the complex sequence space [6] [8]. By sampling from this map, it becomes possible to generate novel candidates (AMPs) [6] [8]. Researchers have tried several types of generative models for this, and each comes with its own trade-offs [6] [7] [8].

Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) were a popular early choice [6]. A VAE has two main parts: an encoder that maps a sequence to a latent space, and a decoder that tries to reconstruct the original sequence from that latent vector [6]. The model is trained to do two tasks at once: accurately reconstruct the input, and organize the latent space according to a simple prior distribution (usually a standard normal) [6]. The problem is that while VAEs can generate a variety of sequences, they often encounter a problem called "posterior collapse" [6] [46]. This is where the decoder essentially gives up on using the latent vector and just produces generic, uninteresting sequences [46]. When that happens, exploring the latent space becomes a pointless target [46].

Generative Adversarial Networks (GANs)

GANs (Generative Adversarial Networks) took a completely different route [7]. The basic idea is to pit two networks against each other in a kind of game [7]. One network generator creates synthetic sequences [7]. The other network discriminator aims to distinguish real sequences from fake ones [7]. Over time, the generator learns to make sequences convincing enough to fool the discriminator [7]. GANs have demonstrated their ability to produce high-quality samples [7]. But there is a catch [7]. Training them is famously difficult [7]. The adversarial setup makes the whole process unstable, and GANs often fall into a trap called "mode collapse" [7] [66]. This is when the generator stops exploring and just churns out a few repetitive sequence types [66]. There is another issue: working with discrete data, such as peptide sequences, adds additional technical complications that make GANs even harder to use in practice [7].

Wasserstein Autoencoders (WAEs)

Wasserstein Autoencoders (WAEs) emerged as a solution to the problems with VAEs and GANs [8]. Instead of the usual objectives, a WAE optimizes the Wasserstein distance between the distribution of encoded data and the prior distribution [8] [28]. This makes training much more stable than it is with GANs [8]. It also helps avoid the posterior collapse problem that often plagues VAEs [8] [46]. When combined with a regularization metric like Maximum Mean Discrepancy (MMD), WAEs can produce a smooth, continuous latent space that is ideal for generation tasks [8] [23].

Diffusion Models

Diffusion models represent a more recent approach to generative modeling [60] [61] [62] [63]. These models work by gradually adding noise to data until it becomes pure random noise, and then learning to reverse this process [60]. During generation, the model starts with random noise and iteratively removes it, step by step, until a valid sequence emerges [60]. This iterative process can produce high-quality samples and avoid the

instability issues seen with GANs [60] [63]. However, diffusion models are computationally expensive [60] [61]. Generating a single sequence requires many denoising steps, which makes it slower [60]. Despite this cost, recent work has shown that diffusion models can generate antimicrobial peptides with high predicted activity, and some have been experimentally validated [63].

1.2.4 Transformer Architectures in Biological Sequence Modeling

The Transformer architecture, which was first a breakthrough in natural language processing, has also been successfully applied to biological sequence modeling [9] [10]. Models like recurrent neural networks (RNN) and Long Short-Term Memory (LSTM) had to process sequences stepwise, whereas the Transformer processes the entire sequence at once (in parallel) [9]. It does this using a "attention" mechanism, which lets the model decide how much importance to assign to each other amino acid when representing a specific one [9]. This is how it captures long-range dependencies and complex patterns—like the periodic arrangement of residues in an α-helix—that are so important for how a peptide functions [9]. The impressive results from protein structure prediction models like ESMFold demonstrate just how well Transformers learn the "grammar" of biological sequences [10].

1.2.5 Bayesian Optimization for Guided Search

Even with a generative model, just picking candidates randomly from the latent space is not very efficient [38]. This is where Bayesian Optimization (BO) comes in [38] [39]. BO offers an efficient way to search through complex, high-dimensional spaces without wasting effort [38] [39]. It works by building a probabilistic surrogate model, often a Gaussian Process to approximate the objective function, such as predicted antimicrobial activity [39]. It then uses an "acquisition function" to decide where to sample next [38] [40]. This function is designed to strike a balance between exploring uncertain regions and exploiting the best target regions [38]. In this way, BO can efficiently navigate the latent space and identify high-quality candidates with a minimal number of evaluations [38].

1.3 Problem Statement

The literature review reveals several key limitations in the current state of *de novo* AMP design:

1. **Limitations of Existing Generative Models:** The standard generative models have serious drawbacks. Variational Autoencoders (VAEs) often suffer from a "posterior collapse" problem, which significantly degrades the quality of the sequences they produce [19]. Generative Adversarial Networks (GANs) are another case. They are famously difficult to train. They frequently suffer from instability and "mode collapse," in which they produce only a handful of sequence types [21]. Even newer approaches, like Diffusion Models, are known to be computationally expensive. Sampling from them is very slow, which creates a major bottleneck for high-throughput screening [57][59]. They can also struggle to generate truly diverse and novel structures. Sometimes they fall short of what is actually needed for genuine discovery [58].

2. **The Problem of Undirected Sampling:** Most generative models are used in a passive way. The sampling process is randomly drawn from the latent space; this is inefficient. It provides no strategy for actively generating candidates with specific, desirable qualities. A more robust active search process is needed to really unlock the full potential of these generative latent spaces [38].

3. **Lack of Integration of Advanced Architectures:** Advanced architectures like the Transformer have proven a remarkable understanding of the grammar of biological sequences [9][10]. But they have not been properly integrated into a stable framework. There is a clear need for a model that combines

the solid representational ability of the Transformer with the generative stability of a framework like WAE, which has not been fully explored for the specific task of de novo AMP design.

This thesis aims to address this gap by developing and evaluating a novel generative framework, the TWAE-MMD [8] [9]. It combines a Transformer-based autoencoder with a Wasserstein Autoencoder regularized by MMD [8] [9] [23]. In addition, the work explores the use of Latent Space Bayesian Optimization (LSBO) as an active discovery strategy [38]. The goal is to perform a sophisticated search in the latent space generated by the TWAE-MMD model [8] [38].

1.4 Research Objectives and Hypotheses

The goal of this thesis is to address the research gap identified above. A generative framework was developed and tested specifically for the design of high-quality antimicrobial peptides from scratch [8] [9]. The essence of this work is the TWAE-MMD (Transformer-based Wasserstein Autoencoder with Maximum Mean Discrepancy), a custom model designed to produce a stable, meaningful latent representation of peptide sequence space [8] [9] [23].

Based on this foundation, this work will investigate two primary sampling strategies:

1. **EMA Baseline Strategy:** An approach based on a baseline [36] [37]. It uses stochastic sampling from the prior distribution, with an Exponential Moving Average (EMA) applied to the model weights [36] [37]. The idea was to see how well the TWAE-MMD model generates sequences on its own, without any guidance (no sampling strategy) [8].

2. **LSBO-Guided Strategy:** An active discovery approach [38]. It uses Latent Space Bayesian Optimization (LSBO) to intelligently search the latent space for regions that should correspond to high-activity AMPs [38] [39].

The central hypotheses of this thesis are as follows:

H1 The TWAE-MMD framework will produce a high-quality, continuous, and well-regularized latent space that captures the chemical grammar of antimicrobial peptides [8] [9] [23]. The quality of this latent space will be assessed and compared to that of traditional VAE and GAN architectures and the latest Diffusion models in terms of stability and sample quality [6] [7] [8] [60].

H2 The novel LSBO-guided sampling strategy will be evaluated against the passive EMA baseline to determine its effectiveness in identifying high-activity AMPs [38]. This comparison will demonstrate whether an active search strategy provides advantages over passive sampling in de novo sequence discovery [38].

By systematically evaluating these hypotheses, this thesis will not only present a new tool for AMP discovery but also provide evidence for a more efficient future for AI-driven molecular design.

Chapter 2

Background and Theoretical Foundations

2.1 The Challenge of Antimicrobial Resistance and the Role of AMPs

Antimicrobial resistance (AMR) has become a major threat to public health worldwide during the 21st century. Conventional antibiotics have been overused and misused, leading to the rapid emergence of multidrug-resistant (MDR) pathogens. Many first-line treatments no longer work against these resistant strains [12] [13]. This situation has created an urgent need for new therapeutic agents that operate through different mechanisms and can bypass the resistance pathways that bacteria have developed [14].

2.1.1 Antimicrobial Peptides: A Biological Overview

Antimicrobial peptides (AMPs), sometimes called host defense peptides (HDPs), form part of the innate immune system. Researchers have been studying them as potential therapeutic candidates to address AMR [15]. These molecules have persisted throughout evolution and are found in nearly all living organisms, from bacteria to mammals. They serve as a rapid, broad-spectrum defense mechanism against invading pathogens [15] [16].

Molecular Characteristics and Classification

AMPs are short peptides encoded by genes, typically containing 10-50 amino acids. A key feature of these peptides is their amphipathic structure, which means they contain both hydrophobic and hydrophilic regions. At physiological pH, AMPs carry a net positive charge (cationic nature). This combination of properties determines their biological function [17]. The positive charge attracts the peptide electrostatically to negatively charged components of microbial cell membranes. These components include lipopolysaccharides (LPS) in Gram-negative bacteria and teichoic acids in Gram-positive bacteria. The hydrophobic regions then enable the peptide to insert into the lipid bilayer [17].

AMPs show considerable variation in structure. They can be grouped according to their secondary structures:

- **α-helical peptides:** This group represents the largest class. These peptides adopt an amphipathic helical conformation. LL-37, the only cathelicidin found in humans, belongs to this class [18].
- **β-sheet peptides:** These peptides contain two or more disulfide bonds that stabilize a sheet-like structure. Defensins, a major class of mammalian AMPs, fall into this category [19].
- **Extended/Loop peptides:** This class lacks a regular secondary structure. These peptides often contain a high proportion of specific amino acids such as proline or glycine. Indolicidin, a tryptophan-rich extended peptide, is an example [20].

Mechanism of Action

Most AMPs act primarily by disrupting microbial cell membranes. This differs from conventional antibiotics, which typically target specific intracellular enzymes or metabolic pathways. AMPs physically damage the integrity of the membrane barrier. Several models have been proposed to describe this process:

- **Barrel-stave model:** Peptides insert into the membrane and aggregate to form a transmembrane pore. The hydrophobic surfaces of the peptides face the lipid tails, while the hydrophilic surfaces line the channel.
- **Toroidal pore model:** This model resembles the barrel-stave model, but the peptides cause the lipid monolayers to bend continuously through the pore. Both peptides and lipid head groups line the pore.
- **Carpet model:** Peptides accumulate on the membrane surface, forming a layer that disrupts the membrane curvature. At a critical concentration, this leads to micellization and bilayer disintegration.

These membrane disruption models have been described extensively in the AMP literature [16].

The membrane-centric mechanism has an important consequence. It is more difficult for pathogens to develop resistance by altering the fundamental biophysical properties of their membranes than by mutating a single target protein [21]. Some AMPs can also cross the plasma membrane and interfere with intracellular processes, including DNA replication, protein synthesis, and enzymatic activity [22].

Therapeutic Potential and Design Challenges

AMPs have broad-spectrum activity, a unique mechanism of action, and a low propensity for inducing resistance. These properties have led researchers to consider them as next-generation antibiotics. Many AMPs also have immunomodulatory functions. They can regulate inflammation, promote wound healing, and act as signaling molecules in the immune system [23].

However, translating natural AMPs into clinical therapeutics presents several challenges [17]. These include potential toxicity to host cells (hemolytic activity), poor protease stability, and higher manufacturing costs [17]. These limitations have motivated the use of computational methods for the *de novo* design of synthetic AMPs [17]. Generative models can learn the complex relationships between sequence, structure, and function from known peptides [17]. Computer models can learn patterns from known peptides. They can then generate new sequences that might have better properties [17]. The goal is high antimicrobial activity combined with low toxicity and greater stability. This computational approach serves as the basis for the present work.

2.2 Generative Models for *De Novo* Peptide Design

Traditional methods for discovering antimicrobial peptides face significant limitations. In response, computational methods using artificial intelligence have become central to modern peptide design. Generative models, a class of unsupervised machine learning algorithms, are at the forefront of this shift [17]. These models learn the underlying probability distribution of a given dataset. They can then generate new data points that are statistically consistent with the training data. In peptide design, this process is called *in silico* design. The Latin phrase *in silico* (literally "in silicon") refers to computational experiments conducted through computer simulation. This term draws an analogy to the biological terms *in vivo* (in living organisms) and *in vitro* (in laboratory glassware). Computational approaches have become foundational to modern research fields, including computational biology, bioinformatics, and bioengineering. They enable the modeling and analysis of complex biological systems that would be impractical or impossible to study through purely experimental means. A generative model can learn the sequence and structural principles of a set of known active AMPs. The model can then generate novel peptide sequences with a high probability of possessing similar biological

activity [17]. This approach can accelerate the discovery of new therapeutic peptide candidates by exploring vast regions of sequence space that are inaccessible to traditional high-throughput screening methods.

2.3 Optimal Transport Theory

Optimal Transport Theory: From Monge to Kantorovich

The Wasserstein distance is the theoretical foundation for the WAE. This distance was first studied in the classical problem of optimal transport, which was first defined by the French mathematician Gaspard Monge in 1781 [24]. Monge's problem was to find a way to move mass from one area to another while keeping the total cost as low as possible. The Monge problem is written formally as follows: given two probability measures μ and ν on a space $\mathcal{X}$, find a transport map $T : \mathcal{X} \to \mathcal{X}$ that minimizes the total cost of transporting mass from μ to ν. The Monge problem is formally written as:

$$\inf_{T:T_{\#}\mu=\nu} \int_{\mathcal{X}} c(x, T(x)) d\mu(x) \tag{2.1}$$

Here, $c(x, y)$ is the cost of transporting a unit of mass from x to y, and $T_{\#}\mu$ denotes the pushforward measure, defined by $(T_{\#}\mu)(A) = \mu(T^{-1}(A))$ for any measurable set A [24]. The constraint $T_{\#}\mu = \nu$ ensures that the transport map T transforms the measure μ into the measure ν [24].

The Monge problem is easy to understand, but hard to solve. The problem requires finding a deterministic transport map, which may not exist in all cases. For example, if μ is a Dirac measure (a point mass) and ν is a more diffuse measure, no single-valued function can transport all the mass from the point to the entire support of ν. To address this limitation, the Soviet mathematician and economist Leonid Kantorovich reformulated the problem in a more general framework in 1942 [25]. Instead of looking for a deterministic transport map, Kantorovich studied probabilistic transport plans. These are represented by joint probability measures γ on $\mathcal{X} \times \mathcal{X}$ whose marginals are μ and ν. The set of all such transport plans is denoted $\Pi(\mu, \nu)$:

$$\Pi(\mu, \nu) = \{\gamma \in \mathcal{P}(\mathcal{X} \times \mathcal{X}) : \pi_1 \# \gamma = \mu, \pi_2 \# \gamma = \nu\} \tag{2.2}$$

Here, π_1 and π_2 are the projections onto the first and second coordinates, respectively. The Kantorovich problem is then:

$$W_c(\mu, \nu) = \inf_{\gamma \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} c(x, y) d\gamma(x, y) \tag{2.3}$$

This formulation is more flexible than Monge's formulation, as it always admits a solution. This relaxation was an important development. It reformulated the problem so it could be done step by step, making it possible to use modern computer methods. The study of optimal transport maps and their properties remains an active area of research, with connections to analysis and partial differential equations [26]. When the cost function is the p-th power of a distance, $c(x, y) = d(x, y)^p$, the resulting optimal transport cost defines the p-Wasserstein distance:

$$W_p(\mu, \nu) = \left(\inf_{\gamma \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} d(x, y)^p d\gamma(x, y) \right)^{1/p} \tag{2.4}$$

The 1-Wasserstein distance, also known as the Earth Mover's Distance, is of particular interest in machine learning applications.

The Kantorovich-Rubinstein Duality

The Kantorovich formulation provides a solution to the optimal transport problem. However, it is still computationally challenging to solve directly, as it involves an optimization over the space of all joint distributions. A more practical form is provided by the Kantorovich-Rubinstein duality theorem [27]. This

theorem states that for the 1-Wasserstein distance on a metric space, the primal formulation can be equivalently expressed as a dual problem:

$$W_1(\mu,\nu) = \sup_{\|f\|_L \leq 1} \left(\int_{\mathcal{X}} f(x) d\mu(x) - \int_{\mathcal{X}} f(y) d\nu(y) \right) \tag{2.5}$$

The supremum is taken over all 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$. A function f is 1-Lipschitz if it satisfies:

$$|f(x) - f(y)| \leq d(x,y) \quad \forall x, y \in \mathcal{X} \tag{2.6}$$

This dual formulation is the foundation of Wasserstein GANs (WGANs) [28]. In WGANs, a neural network is trained to approximate the optimal 1-Lipschitz function. The Lipschitz constraint is enforced through techniques such as weight clipping or gradient penalty. The Kantorovich-Rubinstein duality provides a well-defined and stable loss metric even when the two distributions do not overlap. This property differs from other divergence measures like KL divergence, which can become infinite or undefined when distributions have disjoint supports.

In the context of WAEs, the Wasserstein distance measures the difference between the aggregated posterior distribution of the encoded data and the prior distribution. This makes sure that the latent space is organized by taking samples from the prior, which will give us latent codes that can be translated into data that is close to the original distribution.

2.3.1 The Wasserstein Autoencoder (WAE)

The Wasserstein Autoencoder (WAE) was introduced by Tolstikhin et al. [8]. It provides an alternative to VAEs by using tools from Optimal Transport theory to regularize the latent space. A WAE is an autoencoder trained to minimize a combined objective function:

$$\mathcal{L}_{WAE} = \mathbb{E}_{p_X}[\mathbb{E}_{q_\phi(z|x)}[c(x, g_\theta(z))]] + \lambda \cdot D(Q_Z, P_Z) \tag{2.7}$$

This objective consists of two terms [8]. The first term is a reconstruction cost $c(x, g_\theta(z))$ that ensures the decoder can reconstruct the input data [8]. The second term is a regularization penalty $D(Q_Z, P_Z)$ that enforces the aggregated posterior distribution of the encoded data, $Q_Z = \int q_\phi(z|x)p_X(x)dx$, to match a predefined prior distribution P_Z (typically $\mathcal{N}(0, I)$) [8]. The hyperparameter λ controls the strength of this regularization [8].

A key characteristic of the WAE framework is its flexibility in the choice of the divergence measure $D(Q_Z, P_Z)$. The original work proposes two primary variants for this penalty.

Core Architectural Components

Before detailing the regularization variants, it is necessary to understand the core components of the autoencoder architecture itself [8]. These components are shown explicitly in the architecture diagram (Figure 2.1). The WAE, like any autoencoder, consists of three fundamental parts: an encoder, a latent space (or bottleneck), and a decoder [8].

The Encoder (E_θ) The encoder is a neural network parameterized by weights θ [8]. It performs dimensionality reduction [8]. Its function is to take a high-dimensional input data point, $x \in \mathcal{X}$, and map it to a low-dimensional representation, z, in the latent space, $\mathcal{Z} \subset \mathbb{R}^{d_z}$ [8]. This mapping is deterministic and can be written as [8]:

$$z = E_\theta(x) \tag{2.8}$$

The encoder is trained to capture relevant features of the input data while discarding noise and redundancy [8]. The output of the encoder for the entire dataset forms the aggregated posterior distribution, Q_Z

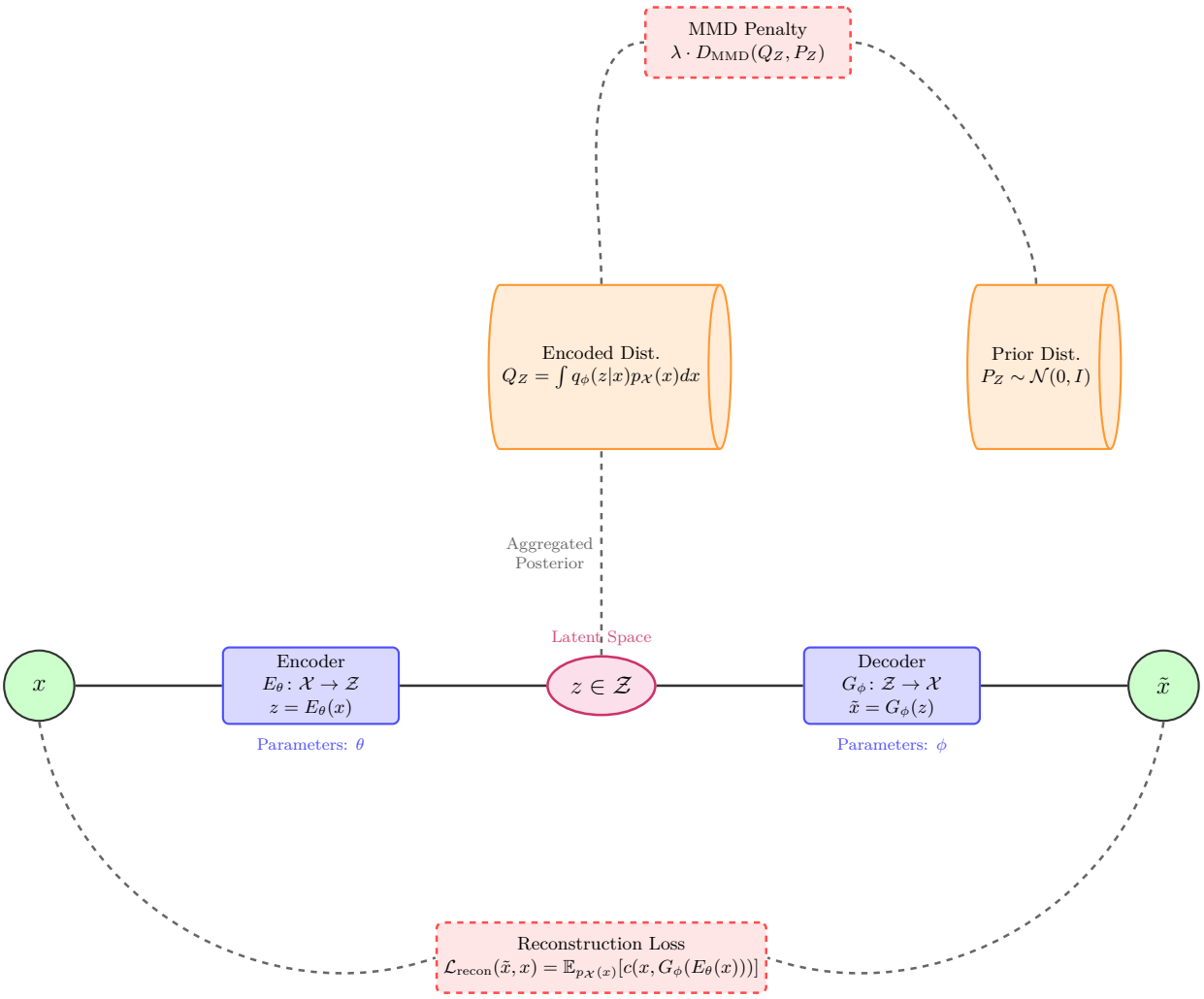

Figure 2.1: The Wasserstein Autoencoder (WAE) architecture with MMD regularization. Input data x is mapped to a latent representation z by encoder E_θ, then reconstructed as $\tilde{x}$ by decoder G_ϕ. The model is trained on two objectives: (1) a reconstruction loss $\mathcal{L}_{\text{recon}}$ to ensure fidelity, and (2) a regularization penalty that minimizes the Maximum Mean Discrepancy (MMD) between the aggregated posterior distribution of encoded data, Q_Z, and a prior distribution, P_Z.

[8].

The Latent Space ($\mathcal{Z}$) The latent space is the low-dimensional, compressed representation of the input data [8]. It serves as an **information bottleneck** [8]. This forces the model to learn a compact encoding of the data [8]. The structure of this latent space is critical [8]. A structured latent space is one where similar data points in the input space are mapped to nearby points in the latent space [8]. Smooth paths between points in the latent space correspond to continuous transitions in the data space [8]. The regularization penalty, $D(Q_Z, P_Z)$, is applied directly to this space to enforce a desired structure (e.g., a Gaussian distribution) [8]. This is necessary for generation [6] [8].

The Decoder (G_ϕ) The decoder is another neural network parameterized by weights ϕ. It performs the reverse operation of the encoder. It takes a point from the latent space, $z \in \mathcal{Z}$, and maps it back to the high-dimensional data space, producing a reconstruction of the original input, $\tilde{x}$. This is written as [6]:

$$\tilde{x} = G_\phi(z) \tag{2.9}$$

The decoder is trained to generate data that approximates the original distribution from the latent

representation [8]. The reconstruction cost term in the WAE objective, $\mathbb{E}[c(x, G_\phi(E_\theta(x)))]$, measures how well the decoder can reconstruct the original data after it has been passed through the encoder-decoder pipeline [8]. This ensures that the latent space retains sufficient information to recover the original data [8].

The interplay between the encoder compressing information into the latent bottleneck and the decoder reconstructing it, all while the latent space is being regularized to match a prior distribution, is the central mechanism of the Wasserstein Autoencoder [8].

WAE-MMD: Statistical Regularization with Maximum Mean Discrepancy

The second variant, WAE-MMD, uses a statistical metric called the Maximum Mean Discrepancy (MMD) to measure the distance between the two distributions [8] [23]. MMD is defined as the squared norm of the difference between the mean embeddings of the distributions in a Reproducing Kernel Hilbert Space (RKHS) [23]. This approach avoids adversarial training, which leads to a stable and deterministic objective [8].

The MMD penalty is given by:

$$D(Q_Z, P_Z) = \text{MMD}^2(Q_Z, P_Z) \tag{2.10}$$

The squared MMD is computed using the kernel trick:

$$\text{MMD}^2(Q_Z, P_Z) = \mathbb{E}_{z,z' \sim Q_Z}[k(z, z')] - 2\mathbb{E}_{z \sim Q_Z, z' \sim P_Z}[k(z, z')] + \mathbb{E}_{z,z' \sim P_Z}[k(z', z')] \tag{2.11}$$

Here, $k(\cdot, \cdot)$ is a positive definite kernel, such as the Gaussian (RBF) kernel [23]. The use of MMD provides a differentiable, stable, and non-adversarial objective for regularizing the latent space [8] [23].

Model Selection for this Thesis

For the work presented in this thesis, the **WAE-MMD** variant was selected as the foundational architecture for integration with the Transformer model. This choice is motivated by several characteristics of the MMD-based approach:

- **Training Stability:** The MMD penalty is deterministic and does not involve adversarial training. This avoids common issues like mode collapse and the need to carefully balance multiple competing networks. This leads to a stable and reproducible training process.
- **Differentiable Objective:** The MMD loss is fully differentiable with respect to the encoder's parameters. This provides smooth gradients that are suitable for training deep, high-capacity models like Transformers.
- **Flexibility:** The kernel-based nature of MMD allows for the use of various kernels (e.g., multi-scale RBF) to capture differences between distributions at multiple scales. This provides a flexible regularization mechanism.

These properties make WAE-MMD a suitable framework for learning a structured latent space of complex sequential data. This is a prerequisite for de novo design of peptides. The remainder of this section will therefore focus on the theoretical details of MMD.

Mathematical Deep Dive: Maximum Mean Discrepancy (MMD)

Maximum Mean Discrepancy (MMD) is a statistical metric used to measure the distance between two probability distributions based on samples drawn from them [23]. It is defined as the squared norm of the difference between the mean embeddings of the distributions in a Reproducing Kernel Hilbert Space (RKHS). An RKHS is a Hilbert space of functions in which all evaluation functionals are bounded linear functionals. This property ensures that if two functions are close in the norm of the space, they are also

close pointwise. The reproducing property states that for any function $f \in \mathcal{H}$ and any point x in the input space, $f(x) = \langle f, k(x, \cdot)\rangle_{\mathcal{H}}$, where k is the reproducing kernel.

For two distributions P and Q, the squared MMD is given by:

$$\text{MMD}^2(P, Q) = \|\mathbb{E}_{x\sim P}[\phi(x)] - \mathbb{E}_{y\sim Q}[\phi(y)]\|^2_{\mathcal{H}} \tag{2.12}$$

Here, $\phi(\cdot)$ is a feature map to an RKHS $\mathcal{H}$ [23]. The RKHS is a Hilbert space of functions where the inner product can be computed using a kernel function [23]. Using the kernel trick, the MMD can be computed without explicitly defining the feature map [23]. By expanding the squared norm and applying the reproducing property, we obtain [23]:

$$\begin{aligned}
\text{MMD}^2(P, Q) &= \langle \mathbb{E}_{x\sim P}[\phi(x)] - \mathbb{E}_{y\sim Q}[\phi(y)], \mathbb{E}_{x'\sim P}[\phi(x')] - \mathbb{E}_{y'\sim Q}[\phi(y')]\rangle_{\mathcal{H}} && (2.13)\\
&= \mathbb{E}_{x,x'\sim P}[\langle\phi(x), \phi(x')\rangle_{\mathcal{H}}] - 2\mathbb{E}_{x\sim P, y\sim Q}[\langle\phi(x), \phi(y)\rangle_{\mathcal{H}}] && (2.14)\\
&\quad + \mathbb{E}_{y,y'\sim Q}[\langle\phi(y), \phi(y')\rangle_{\mathcal{H}}] && (2.15)\\
&= \mathbb{E}_{x,x'\sim P}[k(x, x')] - 2\mathbb{E}_{x\sim P, y\sim Q}[k(x, y)] + \mathbb{E}_{y,y'\sim Q}[k(y, y')] && (2.16)
\end{aligned}$$

Here, $k(\cdot, \cdot)$ is a positive definite kernel, such as the Gaussian (RBF) kernel [23]:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \tag{2.17}$$

The choice of kernel is critical [23]. The Gaussian kernel is universal, which means that the RKHS it generates is dense in the space of continuous functions. This universality property ensures that MMD can distinguish between any two different distributions, provided sufficient samples are available [23]. In practice, a multi-scale RBF kernel is often used. This is a sum of multiple RBF kernels with different bandwidths (σ):

$$k_{multi}(x, y) = \sum_{i=1}^{n} \alpha_i \exp\left(-\frac{\|x - y\|^2}{2\sigma_i^2}\right) \tag{2.18}$$

Here, α_i are weighting coefficients (often set to $1/n$), and σ_i are different bandwidth parameters. This allows the MMD to be sensitive to differences in distributions at multiple scales, providing a flexible regularization penalty for the WAE model [8] [23]. The use of MMD provides a stable, differentiable objective that avoids the adversarial training of GANs and the issues of posterior collapse in VAEs. The MMD loss directly penalizes the discrepancy between the distribution of the encoded data and the prior, without requiring the encoder to produce a distribution with a specific parametric form (like a Gaussian in VAEs).

Unbiased MMD Estimation with U-Statistics

While Equation 2.16 provides the theoretical definition of MMD, in practice, it must be estimated from finite samples [23]. A common approach is to use an empirical estimator [23]. Given two sets of samples, $X = \{x_1, ..., x_m\}$ drawn from distribution P and $Y = \{y_1, ..., y_n\}$ drawn from distribution Q, a biased empirical estimate of the squared MMD is given by [23]:

$$\text{MMD}_b^2(X, Y) = \frac{1}{m^2}\sum_{i,j=1}^{m} k(x_i, x_j) - \frac{2}{mn}\sum_{i,j=1}^{m,n} k(x_i, y_j) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(y_i, y_j) \tag{2.19}$$

However, a more robust, unbiased estimator can be constructed using U-statistics, as detailed by Gretton et al. [23]. This formulation avoids the positive bias of the empirical estimate by excluding diagonal terms where $i = j$. The unbiased U-statistic estimator for squared MMD is:

$$\mathrm{MMD}_u^2(X,Y) = \frac{1}{m(m-1)}\sum_{i\neq j}^{m} k(x_i,x_j) + \frac{1}{n(n-1)}\sum_{i\neq j}^{n} k(y_i,y_j) - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} k(x_i,y_j) \tag{2.20}$$

This is the estimator used in the WAE-MMD algorithm proposed by Tolstikhin et al. [8]. Specifically, Algorithm 1 in their paper outlines the training procedure for the WAE-MMD, where this U-statistic is used to compute the MMD penalty between the aggregated posterior of the latent space, Q_Z, and the prior distribution, P_Z. This provides a stable way to regularize the latent space, encouraging it to match the desired prior distribution without the need for adversarial training.

The WAE-MMD Training Algorithm

The training procedure for the WAE-MMD, as outlined in Algorithm 1, provides a step-by-step process for optimizing the model [8]. The algorithm iteratively updates the encoder and decoder networks by minimizing a composite objective function [8]. This function consists of two main terms [8]:

1. **Reconstruction Cost:** The term $c(x_i, G_\theta(z_i))$ represents the reconstruction cost. This measures the dissimilarity between the original input data x_i and its reconstruction $G_\theta(z_i)$ after being encoded into the latent space. The choice of cost function c depends on the data modality. For sequential data like peptides, this is typically a cross-entropy loss.

2. **MMD Regularization Penalty:** The second term, scaled by the hyperparameter λ, is the MMD penalty. This term enforces the regularization by minimizing the discrepancy between the aggregated posterior of the encoded data, Q_Z, and the prior distribution, P_Z. The algorithm uses the unbiased U-statistic estimator of MMD, which is composed of three parts:

   - The first part, $\frac{1}{n(n-1)}\sum_{i\neq j} k(z_i, z_j)$, measures the similarity between pairs of encoded samples from the training data.
   - The second part, $\frac{2}{n^2}\sum_{i,j} k(z_i, \tilde{z}_j)$, measures the cross-similarity between the encoded data samples and samples drawn from the prior distribution.
   - The third part, $\frac{1}{n(n-1)}\sum_{i\neq j} k(\tilde{z}_i, \tilde{z}_j)$, measures the similarity between pairs of samples drawn from the prior distribution.

By descending the gradient of this combined objective, the algorithm simultaneously learns to reconstruct the input data accurately while ensuring that the latent space is well-regularized [8]. This dual objective allows the WAE-MMD to generate novel samples from a smooth and continuous latent space [8].

Algorithm 1 Wasserstein Auto-Encoder with MMD-based penalty (WAE-MMD) [8]

Require: Regularization coefficient $\lambda > 0$, characteristic positive-definite kernel k.

1: Initialize the parameters of the encoder Q_ϕ, decoder G_θ.
2: **while** (ϕ, θ) not converged **do**
3: Sample mini-batch $\{x_1, ..., x_n\}$ from the training set.
4: Sample mini-batch $\{\tilde{z}_1, ..., \tilde{z}_n\}$ from the prior P_Z.
5: For $i = 1, ..., n$, compute encoded samples $z_i = Q_\phi(x_i)$.
6: Update Q_ϕ and G_θ by descending the gradient of the objective:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} c(x_i, G_\theta(z_i)) + \lambda\left(\frac{1}{n(n-1)}\sum_{i\neq j} k(z_i,z_j) - \frac{2}{n^2}\sum_{i,j} k(z_i,\tilde{z}_j) + \frac{1}{n(n-1)}\sum_{i\neq j} k(\tilde{z}_i,\tilde{z}_j)\right)$$

7: **end while**

2.4 Sequence Modeling with the Transformer Architecture

While the Wasserstein Autoencoder provides a framework for learning a regularized latent space, the performance of the model on sequential data, such as antimicrobial peptides, is critically dependent on the architectural choices for the encoder and decoder. Historically, Recurrent Neural Networks (RNNs) and their variants, particularly Long Short-Term Memory (LSTM) networks [70], were the standard for sequence modeling. However, their sequential nature makes them inefficient for long sequences and can lead to difficulties in capturing long-range dependencies. The **Transformer architecture**[29], represents a paradigm shift in sequence processing. By dispensing with recurrence and relying entirely on a **attention mechanism**, the Transformer can process all tokens in a sequence in parallel. This mechanism allows the model to weigh the importance of all other tokens in the sequence when producing a representation for a given token, enabling it to capture complex, long-range contextual relationships. These capabilities make it applicable to modeling the intricate grammatical and syntactic rules of biological sequences like peptides. This section will provide a detailed theoretical examination of the Transformer architecture, beginning with its mathematical foundations, then deconstructing the distinct roles and mechanisms of its encoder and decoder components (which are explicitly shown in the architecture diagram, Figure 2.2), and finally discussing key implementation details such as activation functions.

2.4.1 Mathematical Foundations of the Transformer

The core component of the Transformer is the attention mechanism [29]. The specific implementation used is **Scaled Dot-Product Attention**, defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.21}$$

Where Q (Query), K (Key), and V (Value) are matrices derived from the input embeddings through learned linear transformations. Specifically, if $X \in \mathbb{R}^{n \times d_{model}}$ is the input matrix (where n is the sequence length and d_{model} is the model dimension), then:

$$Q = XW^Q, \quad W^Q \in \mathbb{R}^{d_{model} \times d_k} \tag{2.22}$$

$$K = XW^K, \quad W^K \in \mathbb{R}^{d_{model} \times d_k} \tag{2.23}$$

$$V = XW^V, \quad W^V \in \mathbb{R}^{d_{model} \times d_v} \tag{2.24}$$

The dot product $QK^T \in \mathbb{R}^{n \times n}$ computes the similarity between all pairs of positions in the sequence. The (i, j)-th entry of this matrix represents the affinity between the query at position i and the key at position j. The scaling factor $\frac{1}{\sqrt{d_k}}$ is crucial for preventing the *softmax* function from saturating when d_k is large [29]. To understand why, consider that if the elements of Q and K are independent random variables with mean 0 and variance 1, then the dot product $q_i^T k_j$ has variance d_k. Without scaling, the magnitude of the dot products grows with d_k, pushing the *softmax* into regions where it has extremely small gradients, making training difficult.

The *softmax* function is applied row-wise to the scaled attention scores, producing a probability distribution over the keys for each query:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.25}$$

These probabilities are then used to compute a weighted sum of the values, yielding the final attention output. The attention mechanism can be interpreted as a differentiable key-value lookup, where the query determines which keys (and thus which values) are most relevant [29].

Multi-Head Attention extends this by running the attention mechanism multiple times in parallel with different, learned linear projections of Q, K, and V. This allows the model to jointly attend to information from different representation subspaces at different positions. Formally:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.26}$$

Where:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.27}$$

And $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ are learned parameter matrices. The use of multiple heads allows the model to capture different types of relationships between positions [29]. For example, one head might focus on syntactic relationships, while another captures semantic similarities.

The **Feed-Forward Network** is a simple, fully connected network applied to each position separately and identically [9] [29]. It consists of two linear transformations with an activation function in between [9]:

$$\text{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2 \tag{2.28}$$

Where σ is the activation function (such as ReLU or GELU), $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$. Typically, $d_{ff} = 4 \cdot d_{model}$, which means the feed-forward network expands the representation to a higher-dimensional space and then projects it back down [29]. This expansion allows the network to learn more complex, non-linear transformations of the input.

Positional Encodings are added to the input embeddings to give the model information about the relative or absolute position of the tokens in the sequence. Since the attention mechanism is permutation-invariant, without positional encodings, the model would have no way to distinguish between different orderings of the input. The sine and cosine functions of different frequencies are used:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.29}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.30}$$

Where pos is the position and i is the dimension. This encoding has several properties [29]. First, it allows the model to learn to attend by relative positions, as for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}. Second, it can generalize to sequence lengths longer than those seen during training, as the sinusoidal functions are defined for all positions.

The **computational complexity** of the attention mechanism is $O(n^2 \cdot d)$, where n is the sequence length and d is the model dimension [29]. This quadratic dependence on sequence length can be a bottleneck for very long sequences. However, for the relatively short peptide sequences considered in this thesis (3–36 amino acids, corresponding to up to 37 tokens when including [CLS] and [SEP] special tokens), this is not a significant concern.

The Transformer architecture consists of two main components: an encoder stack and a decoder stack [9]. The encoder stack uses attention mechanism to learn deep representations of an entire input sequence, making it suitable for analysis tasks like classification or sequence tagging [9]. The decoder can perform generation through various mechanisms. The TWAE-MMD model in this thesis leverages a Transformer encoder to create a latent representation and a decoder to generate novel sequences from that representation through parallel generation.

2.4.2 The Transformer Encoder: Contextual Representation

The encoder stack of a Transformer is designed to generate deep contextual representations of an entire input sequence. Its primary function is to map an input sequence of symbol embeddings $X = (x_1, ..., x_n)$ to a sequence of context-aware hidden states $H = (h_1, ..., h_n)$.

Each layer in the encoder stack consists of two main sub-layers: a multi-head attention mechanism and a position-wise fully connected feed-forward network [9]. A key characteristic of the encoder's attention mechanism is that it allows **attention** across the entire sequence [9]. For any given position i, the attention mechanism can attend to all other positions in the sequence (both to the left and right) [9]. This allows the model to build a context-dependent representation for each token by incorporating information from the entire sequence [9]. This is mathematically achieved by allowing the Query, Key, and Value matrices in the attention calculation (Equation 2.15) to be derived from the complete input sequence without any masking [9].

Tokenization and Special Tokens for Peptide Sequences

In the TWAE-MMD framework, peptide sequences are converted into numerical representations through a tokenization process that maps each amino acid to a unique integer identifier [9]. The vocabulary consists of the 20 standard amino acids plus five special tokens, yielding a total vocabulary size of 25 tokens [9].

The **special tokens** serve distinct functional roles in sequence processing [9]:

- **[CLS] (Classification token):** Prepended to the beginning of every sequence [9]. In the encoder, the final hidden state corresponding to this token serves as an aggregate representation of the entire sequence for downstream tasks such as classification [9].
- **[SEP] (Separator token):** Appended to the end of every sequence to mark the sequence boundary [9]. This token signals to the model where the meaningful sequence content terminates [9].
- **[PAD] (Padding token):** Used to extend shorter sequences to a uniform length [9]. Since neural network models process batches of sequences simultaneously, all sequences within a batch must have identical length [9]. Padding tokens fill the remaining positions after [SEP] up to the maximum sequence length [9]. The attention mechanism is designed to ignore these positions through the use of attention masks [9].
- **[UNK] (Unknown token):** Serves as a replacement for any character not present in the standard 20-amino-acid vocabulary. This ensures the model can handle unexpected or non-standard residues without failure, though such cases are rare in curated AMP datasets.
- **[MASK] (Mask token):** Reserved for masked language modeling pre-training strategies, where random tokens in the input are masked and the model learns to predict them. This token is defined in the vocabulary but is not utilized in the supervised training approach employed in this thesis.

The tokenization process transforms a peptide sequence of length ℓ (where $3 \leq \ell \leq 36$ amino acids) into a fixed-length token sequence of 37 positions as follows:

$$\text{Tokenized sequence} = [\text{CLS}] \oplus \{a_1, a_2, ..., a_\ell\} \oplus [\text{SEP}] \oplus [\text{PAD}]^{(37-\ell-2)} \tag{2.31}$$

Where $\oplus$ denotes concatenation, $\{a_1, ..., a_\ell\}$ are the amino acid tokens, and $[\text{PAD}]^{(37-\ell-2)}$ represents $(37 - \ell - 2)$ padding tokens to reach the maximum length of 37 tokens.

For sequences at the maximum length of 36 amino acids, the tokenization would initially produce 38 tokens ([CLS] + 36 amino acids + [SEP]). To maintain the fixed maximum length of 37 tokens, truncation is applied: the sequence is trimmed to 35 amino acids, with [CLS] at position 1 and [SEP] at position 37. This

ensures that all sequences, regardless of their original length, are represented as 37-token vectors suitable for batch processing in the Transformer architecture.

The attention mask, a binary vector of length 37, is generated in parallel with the token sequence. It contains 1 for positions corresponding to [CLS], amino acids, and [SEP], and 0 for [PAD] positions. This mask is used in the attention mechanism to prevent the model from attending to padding tokens, ensuring that only meaningful sequence content influences the learned representations.

Sequence-Level Representation with [CLS] Token

For tasks that require a single vector representation of the entire sequence, such as sequence classification, the special classification token [CLS] (described above) is prepended to the input sequence. The final hidden state corresponding to this token, h_{CLS}, is then used as the aggregate sequence representation. This vector can be passed to a simple classifier (e.g., a single linear layer followed by a *softmax*) to predict properties of the entire sequence. The loss function for such a task is typically the **Cross-Entropy Loss**:

$$\mathcal{L}_{CE} = -\sum_{c=1}^{M} y_c \log(p_c) \tag{2.32}$$

Where M is the number of classes, y_c is a binary indicator (1 if the observation belongs to class c, 0 otherwise), and p_c is the predicted probability for class c. This loss penalizes the model for being confident and wrong.

2.4.3 The Decoder: Parallel Generation

The decoder is designed for parallel sequence generation. Its goal is to generate an output sequence $Y = (y_1, ..., y_m)$ where all tokens are produced simultaneously in a single forward pass:

$$P(Y|X) = \prod_{t=1}^{m} P(y_t|X) \tag{2.33}$$

The decoder processes the entire sequence at once through dense layers that project from the latent representation to all output positions simultaneously. This parallel approach generates all tokens in a single forward pass, making it computationally efficient for fixed-length sequences.

The decoder receives the latent vector z as input and projects it through multiple dense layers to produce logits for all sequence positions simultaneously. This allows the decoder to generate the complete output sequence in one step, conditioned on the latent representation.

The loss function for training a generative decoder is typically the **Reconstruction Loss**, which is the negative log-likelihood of the target sequence given the model's predictions. For a vocabulary of size V, this is also a Cross-Entropy Loss, calculated at each position and summed over the sequence:

$$\mathcal{L}_{Recon} = -\sum_{t=1}^{m} \log P(y_t|z) \tag{2.34}$$

This loss function trains the decoder to predict all tokens in the sequence accurately, given the conditioning context (the latent vector z).

2.4.4 Activation Functions in Feed-Forward Networks

The choice of activation function within the position-wise feed-forward networks (FFN) is critical for the model's performance. While the original Transformer paper used the Rectified Linear Unit (ReLU), modern Transformer architectures have adopted alternative functions.

ReLU (Rectified Linear Unit)

The ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{2.35}$$

It is computationally inexpensive and helps to mitigate the vanishing gradient problem. However, it has the drawback of the "dying ReLU" problem, where neurons can become inactive and only output zero for any input, effectively preventing them from learning.

GELU (Gaussian Error Linear Unit)

The GELU activation function is defined as:

$$\text{GELU}(x) = x \cdot \Phi(x) \tag{2.36}$$

Where $\Phi(x)$ is the cumulative distribution function of the standard Gaussian distribution. GELU can be approximated as:

$$\text{GELU}(x) \approx 0.5x \left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right]\right) \tag{2.37}$$

GELU provides a smooth, non-monotonic activation that has been shown to improve performance in modern Transformer architectures [71]. Unlike ReLU, GELU allows small negative values to pass through with a non-zero gradient, which helps prevent the dying ReLU problem [71]. This makes it particularly effective for deep networks. In the TWAE-MMD implementation employed in this thesis, GELU is used as the default activation function in all feed-forward networks within both the encoder and decoder stacks, as well as in the latent space projection layers.

2.5 Architectural Integration of Transformers and WAE with Maximum Mean Discrepancy Penalties

The Transformer-based Wasserstein Autoencoder with Maximum Mean Discrepancy (TWAE-MMD) architecture is constructed from the theoretical integration of its constituent parts [8] [9]. This architecture synergizes the sequence modeling power of Transformers with the generative framework of a WAE, using MMD as a principled regularizer [8] [9] [23]. This section provides a theoretical synthesis of how these components work in concert to learn a structured latent space for generating novel peptide sequences.

At its core, the TWAE-MMD is a function that learns a mapping from the space of peptide sequences, $\mathcal{X}$, to a low-dimensional latent space, $\mathcal{Z} \subset \mathbb{R}^{d_z}$, and back. This is achieved through the interplay of a Transformer-based encoder, E_θ, and a Transformer-based decoder (generator), G_ϕ.

2.5.1 Core Operations: Encoding and Generation

The entire TWAE-MMD framework is built upon two fundamental transformations, which are explicitly shown in the architecture diagram (Figure 2.3):

1. **Encoding:** The process of mapping a high-dimensional input peptide sequence, x, to a low-dimensional, continuous latent vector, z. This is performed by the encoder network and is mathematically represented as:

$$z = E_\theta(x) \tag{2.38}$$

Here, E_θ is the Transformer-based encoder parameterized by weights θ. This operation can be viewed as a form of information compression, where the complex syntactic and semantic features of a peptide sequence are distilled into a dense vector representation in the latent space $\mathcal{Z}$.

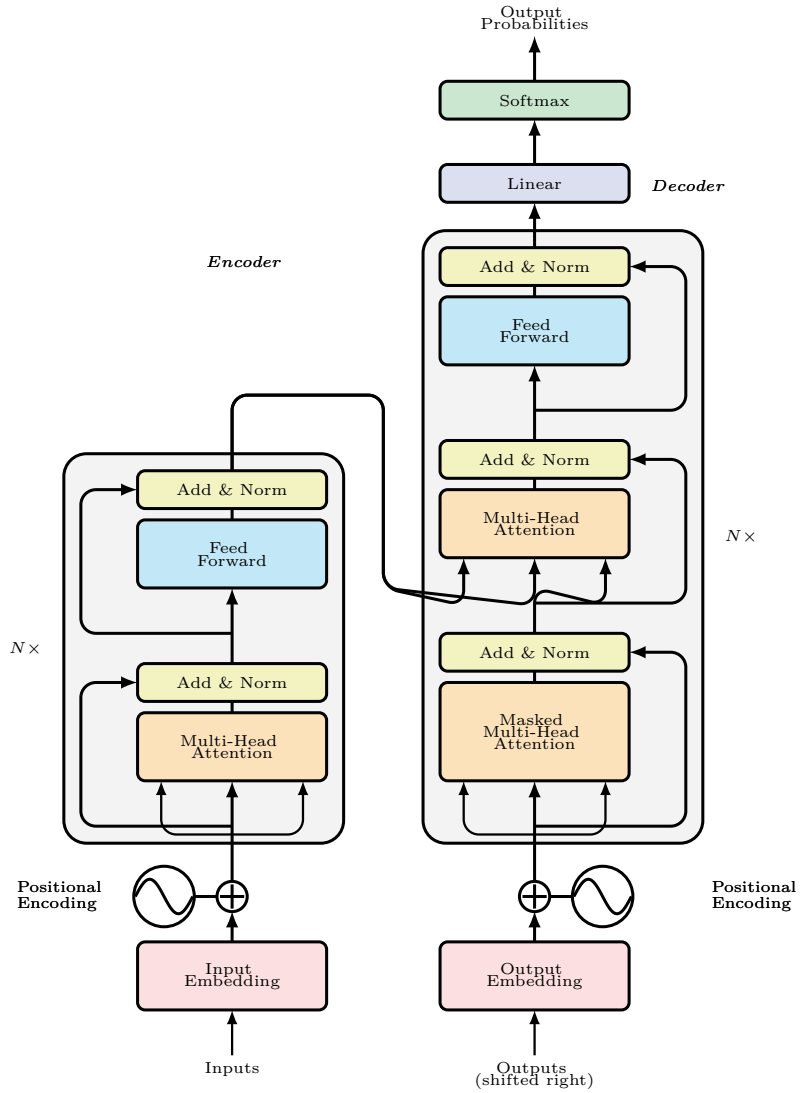

Figure 2.2: The Transformer Architecture, illustrating the multi-head attention mechanism in both the encoder and decoder stacks. The encoder (left) processes the input sequence with attention mechanism to create contextualized representations.

2. **Generation (Decoding):** The process of mapping a point, z, from the latent space back to the high-dimensional space of peptide sequences to produce a sequence, $\tilde{x}$. This is performed by the generator (or decoder) network and is represented as:

$$\tilde{x} = G_\phi(z) \tag{2.39}$$

Here, G_ϕ is the Transformer-based generator parameterized by weights ϕ. This operation has a dual purpose. If the latent vector z is the output of the encoder ($z = E_\theta(x)$), then $\tilde{x}$ is a **reconstruction** of the original input sequence. If z is sampled from the prior distribution ($z \sim P_Z$), then $\tilde{x}$ is a **novel, de novo generated** sequence that shares the learned characteristics of the training data.

The interplay between these two core operations, governed by the joint optimization of the reconstruction loss and the MMD regularization penalty, is what allows the model to learn a structured representation of the peptide sequence space.

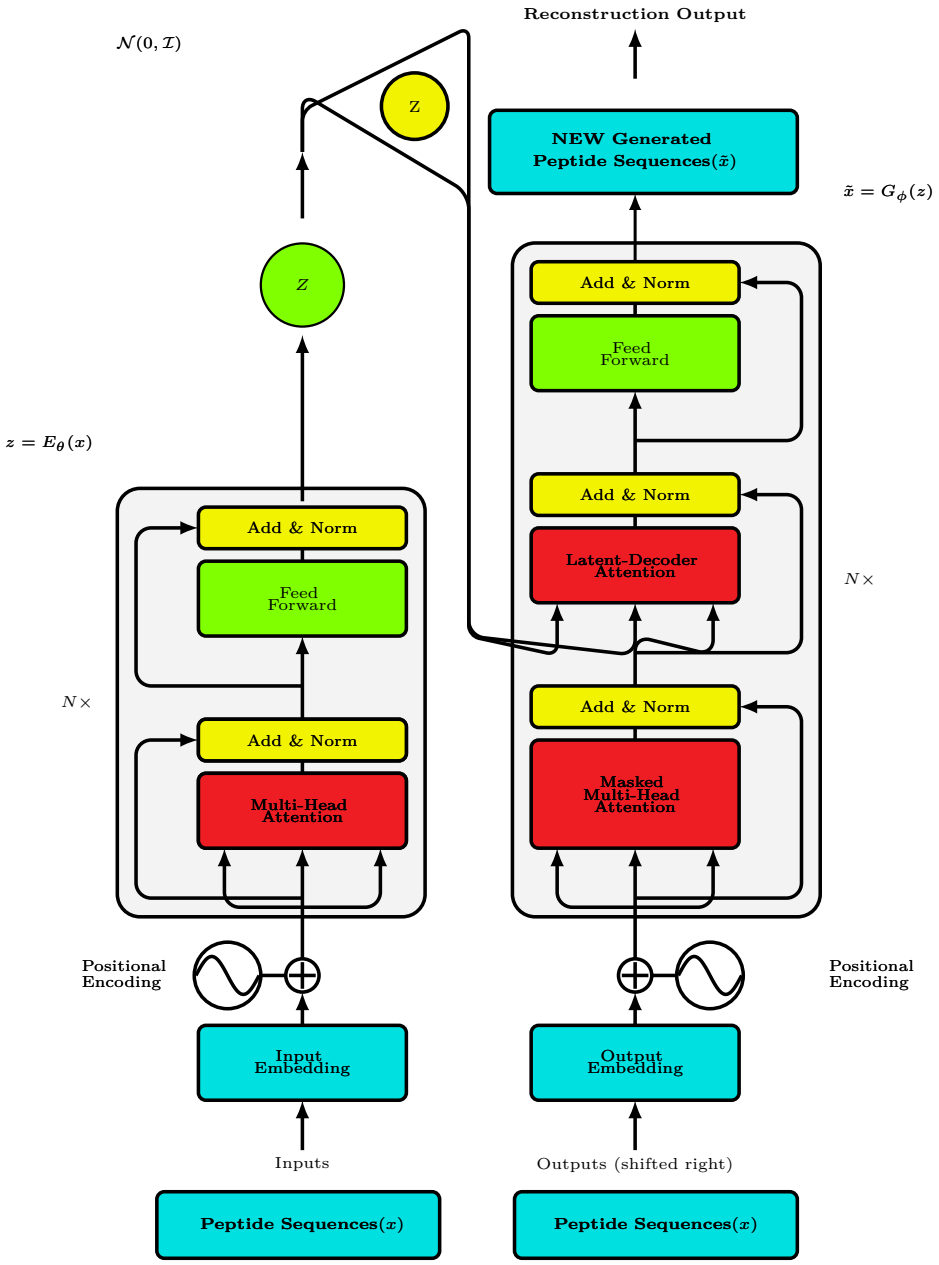

Figure 2.3: The architecture of the Transformer-based Wasserstein Autoencoder (TWAE-MMD). Input peptide sequences are encoded into a latent vector z, which is then regularized to match a prior distribution $\mathcal{N}(0, I)$ using an MMD loss. The latent vector is then decoded to reconstruct the original sequence, and can also be sampled to generate novel sequences.

2.5.2 The Encoder: Learning a Structured Latent Manifold

The encoder, E_θ, is a Transformer encoder stack that maps an input sequence $x \in \mathcal{X}$ to a latent vector $z \in \mathcal{Z}$. The attention mechanism is crucial. By processing the entire input sequence simultaneously, the attention mechanism allows the encoder to build a context-aware representation of each token. The final output of the encoder is typically a pooled representation of the token-level outputs, such as the embedding of a special '[CLS]' token, which is then projected to the latent space dimension d_z. This process defines the encoding distribution $q_\phi(z|x)$.

The key theoretical challenge is to ensure that the aggregated posterior distribution, $Q_Z = \int q_\phi(z|x)p_X(x)dx$, is regularized. Without regularization, the encoder could map different input sequences to disjoint regions of the latent space, leading to a fractured and non-smooth latent manifold. This is where the MMD penalty becomes critical.

2.5.3 MMD Regularization: Enforcing a Smooth Generative Latent Space

The MMD penalty, as defined in Equation 2.16, forces the aggregated posterior Q_Z to match a predefined prior distribution, P_Z, which is typically a standard normal distribution, $\mathcal{N}(0, I)$. The total objective function for the TWAE-MMD is:

$$\mathcal{L}_{TWAE-MMD} = \mathbb{E}_{p_X}[\mathcal{L}_{recon}(x, G_\phi(E_\theta(x)))] + \lambda \cdot \text{MMD}^2(P_Z, Q_Z) \tag{2.40}$$

Where $\mathcal{L}_{recon}$ is the reconstruction loss (e.g., cross-entropy between the input and output sequences), and λ is a hyperparameter that balances reconstruction fidelity with regularization strength. By minimizing the MMD, we are minimizing the distance between the mean embeddings of the encoded data and the prior in an RKHS. This has several theoretical implications:

1. **Continuity and Smoothness:** Forcing Q_Z to be close to a continuous distribution like $\mathcal{N}(0, I)$ encourages the encoder to map similar input sequences to nearby points in the latent space. This creates a smooth and continuous latent manifold, which is essential for generating novel sequences via interpolation.

2. **Avoiding Posterior Collapse:** Unlike the KL divergence in VAEs, the MMD penalty does not require the posterior to have a specific parametric form. This makes it less prone to posterior collapse, especially when paired with a high-capacity decoder. The MMD penalty is a non-parametric measure of distributional similarity, which provides a flexible regularization signal.

3. **Enabling Latent Space Sampling:** A regularized latent space ensures that points sampled from the prior distribution P_Z will be decoded into plausible and diverse sequences. The MMD penalty forces the encoder to utilize the entire latent space, preventing the model from collapsing to a small number of modes.

2.5.4 The Decoder: Parallel Generation from the Latent Space

The decoder, G_ϕ, takes a latent vector z as input and generates an output sequence x' through parallel generation. The decoder processes the latent vector through dense layers to produce all sequence positions simultaneously:

$$p(x'|z) = \prod_{t=1}^{T} p(x'_t|z) \tag{2.41}$$

The decoder architecture consists of dense layers that project the latent vector through progressively larger hidden dimensions, ultimately producing logits for all sequence positions in parallel. This approach generates the complete sequence in a single forward pass, ensuring that all positions are conditioned on the same latent representation. The parallel generation strategy is computationally efficient and well-suited for fixed-length sequences such as antimicrobial peptides.

In summary, the TWAE-MMD architecture represents a synthesis of modern deep learning techniques [8] [9]. The Transformer encoder learns a contextualized representation of the input data [9]. The WAE framework, with its MMD penalty, provides a principled way to regularize the latent space, ensuring that it is smooth, continuous, and suitable for generation [8] [23]. Finally, the decoder provides a parallel generation mechanism for producing novel sequences from the latent space. This integrated architecture is theoretically founded and has applicable to a wide range of generative tasks, including the de novo design of functional peptides.

2.6 Sampling Strategies for Latent Space Exploration

Once a generative model is trained, the process of creating new samples involves drawing points from the latent space and decoding them. The strategy used for this sampling process can have a significant impact on the quality and novelty of the generated candidates. This section describes two distinct sampling strategies employed in this work, with a focus on their theoretical foundations.

2.6.1 Stochastic Prior Sampling with Exponential Moving Average (EMA)

The most common approach is to sample directly from the prior distribution (typically $\mathcal{N}(0, I)$) that the model was trained to match [8]. This method can be enhanced with techniques like **Exponential Moving Average** (EMA) of the model weights [36] [37]. EMA maintains a shadow copy of the model's weights that is a moving average of the weights over time [36]. The EMA parameters, $\theta_{EMA,t}$, are updated at each training step t as follows [36]:

$$\theta_{EMA,t} = \delta \cdot \theta_{EMA,t-1} + (1 - \delta) \cdot \theta_t \tag{2.42}$$

This averaging process has several theoretical justifications [37]. EMA acts as a form of implicit regularization, smoothing out the training trajectory and reducing the impact of noisy gradient updates. Additionally, empirical evidence suggests that EMA leads to better generalization, as the averaged parameters often lie in flatter regions of the loss landscape, which are associated with better out-of-sample performance [37].

The generation process with EMA is as follows:

1. A latent vector z is sampled from the standard normal prior distribution:

$$z \sim \mathcal{N}(0, I) \tag{2.43}$$

2. This vector z is then passed through the decoder network using the EMA parameters, $g_{\theta_{EMA}}(z)$, which maps the point from the latent space back to the original data space (in this case, the space of peptide sequences):

$$x' = g_{\theta_{EMA}}(z) \tag{2.44}$$

While simple and direct, this is a passive and undirected exploration of the chemical space. All points in the latent space are equally likely to be sampled, regardless of their potential to yield candidates with desired properties.

2.6.2 Guided Search with Latent Space Bayesian Optimization (LSBO)

An alternative approach is to actively search the latent space for regions that are most likely to yield candidates with desired properties. **Bayesian Optimization** is a global optimization strategy that is well-suited for this task, especially when the objective function (e.g., predicting the antimicrobial activity of a decoded peptide) is expensive to evaluate [38]. Recent work has demonstrated the effectiveness of semi-supervised latent Bayesian optimization for designing antimicrobial peptides [55]. Bayesian Optimization is a sequential design strategy for the global optimization of **black-box functions**. In this context, a function is considered a "black-box" if its internal workings are unknown or opaque, and we can only observe its output for a given input.

For this thesis, the objective function being optimized is a representative example of a black-box [38] [39]. It represents the entire pipeline from a latent vector to a final property score [38]:

$$\text{score} = f(z) = \text{PropertyPredictor}(G_\phi(z)) \tag{2.45}$$

Here, we can provide a latent vector z and receive a score, but we do not have a simple analytical form

for $f(z)$ [38]. We cannot compute its gradient (i.e., $\nabla_z f(z)$) because the process involves a forward pass through the trained decoder G_ϕ and an external, potentially non-differentiable property prediction model [38]. Bayesian Optimization is designed for precisely this scenario, as it does not require any knowledge of the function's form or its derivatives [38] [39].

Gaussian Processes as Surrogate Models

The core of Bayesian Optimization is a probabilistic surrogate model that approximates the true objective function. A **Gaussian Process** (GP) is commonly used for this purpose [39]. A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution [39]. A GP is fully specified by its mean function $m(x)$ and covariance function (kernel) $k(x, x')$:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{2.46}$$

The mean function is often set to zero, $m(x) = 0$, and the covariance function encodes assumptions about the smoothness and structure of the function [39]. A common choice is the squared exponential (RBF) kernel:

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right) \tag{2.47}$$

Where σ_f^2 is the signal variance and ℓ is the length scale, which controls how quickly the correlation between function values decreases with distance [39].

Given a set of observations $D = \{(x_i, y_i)\}_{i=1}^t$, where $y_i = f(x_i) + \epsilon_i$ and $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$ is Gaussian noise, the GP provides a posterior distribution for the function value $f(x_*)$ at a new point x_* [39]. This posterior is also Gaussian [39]:

$$p(f(x_*)|D, x_*) = \mathcal{N}(\mu_t(x_*), \sigma_t^2(x_*)) \tag{2.48}$$

Where the posterior mean and variance are given by:

$$\mu_t(x_*) = k(x_*)^T (K + \sigma_n^2 I)^{-1} y \tag{2.49}$$

$$\sigma_t^2(x_*) = k(x_*, x_*) - k(x_*)^T (K + \sigma_n^2 I)^{-1} k(x_*) \tag{2.50}$$

Here, $k(x_*)$ is the vector of covariances between x_* and the observed points, K is the Gram matrix with entries $K_{ij} = k(x_i, x_j)$, and y is the vector of observed function values. The posterior mean $\mu_t(x_*)$ provides a prediction of the function value, while the posterior variance $\sigma_t^2(x_*)$ quantifies the uncertainty in this prediction. This uncertainty is crucial for Bayesian Optimization, as it allows the algorithm to balance exploration (sampling where uncertainty is high) and exploitation (sampling where the predicted function value is high) [38] [39].

Acquisition Functions

An **acquisition function** guides the search for the next point to evaluate. It balances **exploitation** (sampling where the surrogate model predicts a high objective value) and **exploration** (sampling where the prediction uncertainty is high). A common choice is **Expected Improvement** (EI) [40]:

$$\mathrm{EI}(x) = \mathbb{E}[\max(0, f(x) - f(x^+))] \tag{2.51}$$

Where $f(x^+)$ is the best value observed so far, $f(x^+) = \max_{i=1,\ldots,t} y_i$. Under the GP posterior, this expectation has a closed-form expression:

$$\mathrm{EI}(x) = \begin{cases} (\mu_t(x) - f(x^+))\Phi(Z) + \sigma_t(x)\phi(Z) & \text{if } \sigma_t(x) > 0 \\ 0 & \text{if } \sigma_t(x) = 0 \end{cases} \tag{2.52}$$

Where:

$$Z = \frac{\mu_t(x) - f(x^+)}{\sigma_t(x)} \tag{2.53}$$

And $\Phi(\cdot)$ and $\phi(\cdot)$ are the cumulative distribution function and probability density function of the standard normal distribution, respectively. The EI acquisition function has an intuitive interpretation: it is high when either the predicted function value is high (exploitation) or the uncertainty is high (exploration), or both [40].

The LSBO Algorithm

The formal LSBO procedure is presented in Algorithm 2. The algorithm begins by initializing a small set of random samples from the latent space, evaluating them with the objective function, and then iteratively refining the search by fitting a Gaussian Process surrogate model and selecting the next candidate to evaluate based on the acquisition function.

Algorithm 2 Latent Space Bayesian Optimization (LSBO) [11] [41]

Require: Trained generative model decoder g_θ, objective function f, acquisition function α (e.g., Expected Improvement), initial sample size n, maximum iterations T.

1: Initialize by sampling n random latent vectors $\{z_1, ..., z_n\}$ from the prior distribution P_Z.
2: Decode each latent vector to a peptide: $x_i = g_\theta(z_i)$ for $i = 1, ..., n$.
3: Evaluate each peptide with the objective function: $y_i = f(x_i)$ for $i = 1, ..., n$.
4: **for** $t = n + 1$ to T **do**
5: Fit a Gaussian Process surrogate model to all observations: $\{(z_i, y_i)\}_{i=1}^{t-1}$.
6: Find the next point to sample by maximizing the acquisition function:

$$z_t = \arg\max_z \alpha(z)$$

7: Decode the latent vector: $x_t = g_\theta(z_t)$.
8: Evaluate the peptide: $y_t = f(x_t)$.
9: Augment the observation set: $\{(z_i, y_i)\}_{i=1}^{t} \leftarrow \{(z_i, y_i)\}_{i=1}^{t-1} \cup \{(z_t, y_t)\}$.
10: **end for**
11: **return** Best observed peptide: $x^* = \arg\max_{i=1,...,T} y_i$.

The theoretical foundation of Bayesian Optimization provides guarantees on its performance [38] [41]. In particular, it can be shown that under certain conditions, Bayesian Optimization achieves **sublinear regret** [41], meaning that the cumulative difference between the optimal function value and the values obtained by the algorithm grows sublinearly with the number of iterations [41]. This property characterizes the algorithm's convergence behavior towards near-optimal solutions [41]. The sample efficiency of this approach, relative to random sampling, stems from the probabilistic surrogate model's ability to guide the search by balancing exploration of uncertain regions with exploitation of high-scoring regions in the latent space [38] [39] [41].

—

Chapter 3

Materials and Methods

This chapter presents the methods used to develop and test the Transformer-based Wasserstein Autoencoder with Maximum Mean Discrepancy (TWAE-MMD) for antimicrobial peptide generation. The work includes data collection, computer setup, data handling, model design, training, and two different ways to generate new sequences: Exponential Moving Average (EMA) sampling and Latent Space Bayesian Optimization (LSBO) with biological constraints.

3.1 Data Acquisition and Preprocessing

3.1.1 Data Sources

The training and validation datasets were built by collecting antimicrobial peptide sequences from five public databases. These databases were: the Database of Antimicrobial Activity and Structure of Peptides (DBAASP) [12], the Data Repository of Antimicrobial Peptides (DRAMP) [72], the Collection of Anti-Microbial Peptides (CAMP) [73], the Antimicrobial Peptide Database (APD) [74], and additional sources. All these databases contain experimentally tested antimicrobial peptides with information about their biological activity.

The sequences from each database were extracted and combined into one file called `DB-AMP.csv`. This initial collection contained 74,241 peptide sequences. The dataset included both antimicrobial peptides (AMPs) and non-antimicrobial peptides (non-AMPs) as negative examples. Sequences with antimicrobial activity were labeled as class 1 (AMP), and sequences without activity were labeled as class 0 (non-AMP).

After collecting the sequences, several filters were applied to ensure quality. Four main criteria were used: (1) sequence length between 3 and 36 amino acids, which matches the typical length of natural antimicrobial peptides and keeps the computational work manageable; (2) only the 20 standard amino acids were allowed, excluding modified or unusual residues and unclear characters; (3) each sequence could appear only once to avoid duplicates and training problems; (4) only two classes (AMP and non-AMP) were kept for supervised learning.

These filters produced a clean dataset saved as `final_amp.csv`. This dataset was then split into training and validation sets using stratified random sampling to keep the same proportion of AMPs and non-AMPs in both sets [13]. The split used an 80:20 ratio, with 80% of sequences for training and 20% for validation. Stratified sampling made sure both classes were represented fairly in each set, which prevents problems during training and testing [13].

3.1.2 Dataset Characteristics

The final dataset had 74,241 peptide sequences before handling. The training file (`train_3_36.csv`) had 64,801 sequences, and the validation file (`validation_3_36.csv`) had 10,500 sequences. After the handling steps described in Section 3.3.3, the training set was reduced to 40,051 valid sequences (38.1% were removed), while the validation set kept all 10,500 sequences (0% were removed). More sequences were removed from the training set because it contained more duplicates and sequences that failed the quality checks.

Sequence lengths ranged from 3 to 36 amino acids, and length statistics were calculated during handling. Each sequence was written as a string of single-letter amino acid codes using the standard 20-letter alphabet

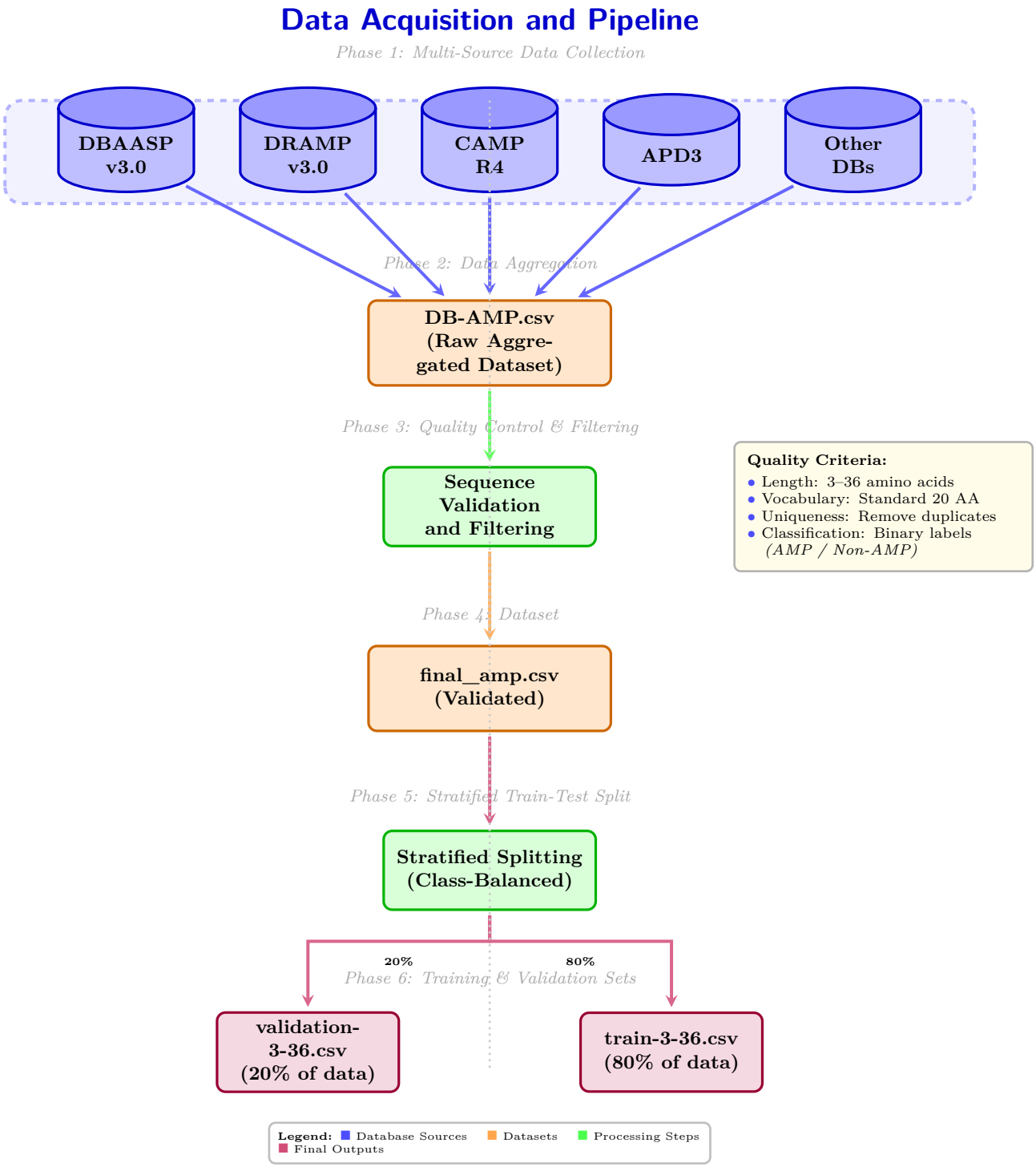

Figure 2.1: Comprehensive data acquisition and preprocessing pipeline for antimicrobial peptide dataset. The workflow consists of six sequential phases: (1) Multi-source data collection from five major AMP databases; (2) Aggregation into a unified raw dataset; (3) Quality control with stringent filtering criteria including sequence length validation, standard vocabulary enforcement (20 amino acids), duplicate removal, and binary classification labeling; (4) Generation of a validated dataset; (5) Stratified splitting to maintain class balance; and (6) Creation of training (80%) and validation (20%) sets for model development and evaluation.

(A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V). The dataset showed natural variation in sequence composition, length, hydrophobicity, and net charge, which reflects the diversity of antimicrobial peptide structures and how they work.

3.2 Computational Infrastructure

3.2.1 Development Environment Configuration

The TWAE-MMD model was developed and trained in a containerized environment to ensure reproducibility, keep dependencies separate, and use resources efficiently. Docker containerization was used to provide a consistent environment across different computers.

The computer used for this work had an NVIDIA GeForce RTX 2060 graphics card with 6 GB of video memory (VRAM) and CUDA compute capability 7.5. The operating system was Ubuntu 20.04 LTS, with NVIDIA driver version 470.x for GPU support. Docker Engine was set up with custom settings to improve container performance and resource use.

The container was built using a Docker image based on the official TensorFlow 2.13.0 GPU image, which includes CUDA 11.8 and cuDNN libraries for GPU-accelerated deep learning. The container specification (`Dockerfile`) installed Python 3.8.10 as the main programming environment, along with scientific calculating and bioinformatics libraries.

The machine learning software included TensorFlow 2.13.0 (with Keras API), NumPy 1.24.3 for numerical operations, SciPy 1.10.1 for scientific functions, and scikit-learn 1.3.0 for machine learning tools. Bioinformatics packages included BioPython 1.81 for sequence work and Peptides 0.3.0 for peptide property calculations. Hyperparameter optimization was done with Optuna 3.2.0, Bayesian-Optimization 1.4.3, and Hyperopt 0.2.7. Visualization tools included Matplotlib 3.7.1, Seaborn 0.12.2, Plotly 5.14.1, and TensorBoard 2.13.0 for monitoring training.

The container was connected to Visual Studio Code (VS Code) Server to allow interactive development, debugging, and code editing through a web interface on port 8080. TensorBoard ran on port 6006 for real-time training visualization, and Jupyter Notebook ran on port 8888 for data exploration. The development environment had Python extensions and Jupyter kernel support in VS Code.

Data storage used five Docker volumes with 7 GB capacity: `vscode_data` for IDE settings, `twae_data` for datasets, `twae_models` for trained models, `twae_results` for outputs, and `twae_logs` for training logs. The Docker image, with all software and settings, took about 12 GB of disk space, for a of 19 GB for the development environment.

GPU memory was set to grow dynamically with a limit of 5.5 GB to prevent memory errors while leaving room for system operations. Calculations used float32 (32-bit floating-point) to balance accuracy and memory use on the RTX 2060.

3.3 Data Preprocessing and Loading Pipeline

3.3.1 Preprocessing Module Architecture

The data preprocessing pipeline is in the `./src/data/` directory. This module handles sequence tokenization, validation, cleaning, and property calculation. The preprocessing part works independently of TensorFlow, which allows flexible data handling before the model sees it.

The tokenization part (`PeptideTokenizer`) uses character-level encoding with 25 tokens. The vocabulary has the 20 standard amino acids and 5 special tokens: `[PAD]` (padding, ID=0), `[UNK]` (unknown, ID=1), `[CLS]` (classification, ID=2), `[SEP]` (separator, ID=3), and `[MASK]` (masking, ID=4). Each amino acid gets a unique number from 5 to 24, after the special tokens.

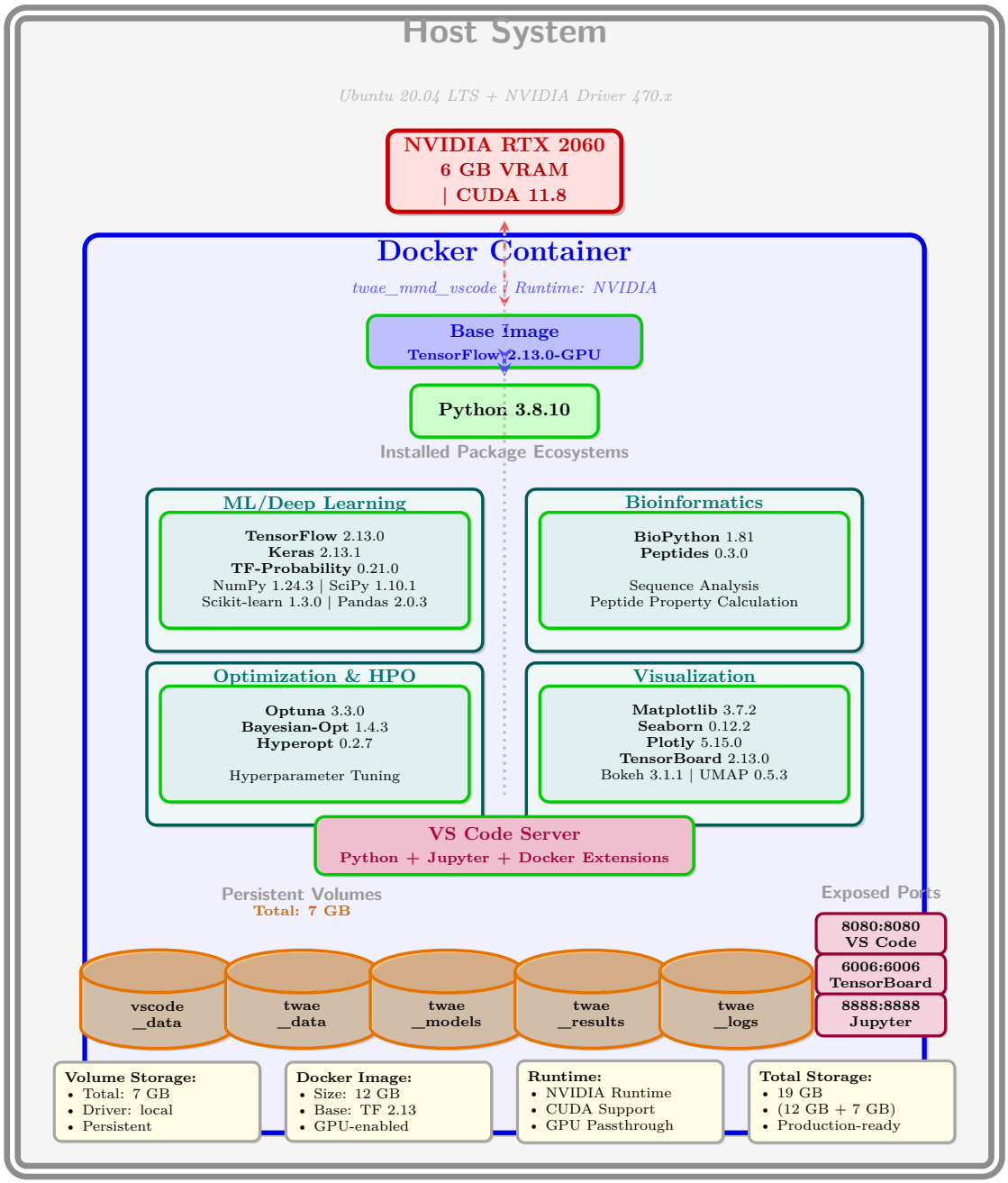

Figure 2.2: Docker containerized development environment for TWAE-MMD implementation. The architecture comprises a GPU-enabled container running TensorFlow 2.13.0-GPU on Python 3.8.10, integrated with VS Code Server for interactive remote development. The environment encompasses four comprehensive package ecosystems: (1) ML/Deep Learning stack; (2) Bioinformatics tools; (3) Optimization frameworks; and (4) Visualization libraries. Persistent storage is managed through five Docker volumes ensuring data persistence across container lifecycles. The container exposes three ports: 8080 (VS Code Server), 6006 (TensorBoard), and 8888 (Jupyter), enabling interactive development, real-time training monitoring, and notebook-based experimentation. Total environment footprint: 19 GB, configured with NVIDIA runtime for GPU passthrough and CUDA support.

The tokenization process converts raw peptide sequences into fixed-length integer arrays for the neural network. For each sequence, the tokenizer adds a `[CLS]` token at the start and a `[SEP]` token at the end. The token sequence is then padded or cut to 36 positions. Padding uses `[PAD]` tokens (ID=0) added to the right of shorter sequences. An attention mask is created at the same time, with 1 for real tokens and 0 for padding.

The preprocessing part (`PeptidePreprocessor`) has multiple validation and cleaning steps. The cleaning stage does three things: (1) removes leading and trailing spaces, (2) converts all letters to uppercase for consistency, and (3) removes non-amino acid characters using pattern matching. The validation stage checks three things: (1) sequences must not be empty, (2) length must be between 3 and 36 amino acids, and (3) all characters must be from the standard 20-letter alphabet when strict validation is on.

Duplicate removal uses exact sequence matching and keeps only the first occurrence of each unique sequence. After validation and duplicate removal, three physicochemical properties are calculated for each sequence: (1) sequence length as the number of amino acids, (2) average hydrophobicity using the Kyte-Doolittle scale with values from -4.5 (arginine, most hydrophilic) to 4.5 (isoleucine, most hydrophobic), and (3) net charge by adding the charges of basic residues (R=+1, K=+1, H=+0.1) and acidic residues (D=-1, E=-1).

3.3.2 TensorFlow Data Loading Module

The TensorFlow data loading module (`./src/twae_data_loader/`) connects the preprocessed data to the model training by creating optimized `tf.data.Dataset` objects. This module uses pre-tokenization to avoid `tf.py_function`, which ensures reliable dataset iteration and allows caching.

The data loading starts with CSV file validation. This checks if the file exists, verifies the CSV format, reads sample rows to check structure, confirms required columns (`sequence` and `label`) are present, and validates data types (string for sequences, numeric for labels). After validation, the CSV file is loaded into a pandas DataFrame, and the preprocessing steps from Section 3.3.1 are applied if validation is turned on in the settings.

After creating the dataset, several operations are applied to improve training performance. For the training dataset, shuffling uses a buffer of 10,000 sequences to randomize sample order across epochs while keeping memory use reasonable. Batching groups sequences into fixed sizes: 64 for training and 128 for validation. A padding operation is used to each batch using TensorFlow's `tf.pad()` function to make all sequences in a batch the same length (36 positions), padding shorter sequences with zeros and cutting longer sequences if needed.

Performance is improved through two methods: (1) dataset caching using `dataset.cache()`, which stores the processed dataset in memory after the first epoch, so preprocessing does not need to be repeated in later epochs, and (2) prefetching using `dataset.prefetch(tf.data.AUTOTUNE)`, which overlaps data loading and preprocessing with model training by preparing the next batch while the current batch is being processed. The `AUTOTUNE` parameter lets TensorFlow automatically choose the best prefetch buffer size based on available resources and training speed.

3.3.3 Complete Pipeline Execution

The data preprocessing and loading pipeline, shown in Figure 2.3, runs in two phases. Phase 1 (`./src/data/`) does preprocessing: cleaning sequences (removing extra spaces, converting to uppercase, removing invalid characters), validating sequences (length 3–36 amino acids, valid amino acid composition, strict validation), removing duplicates, and calculating physicochemical properties (hydrophobicity using Kyte-Doolittle scale, net charge, sequence length). Phase 2 (`./src/twae_data_loader/`) does TensorFlow data loading: CSV validation (file format, column presence, data types, quality control), pre-tokenization (converting all sequences to token IDs and attention masks using the 25-token vocabulary), TensorFlow dataset creation (wrapping

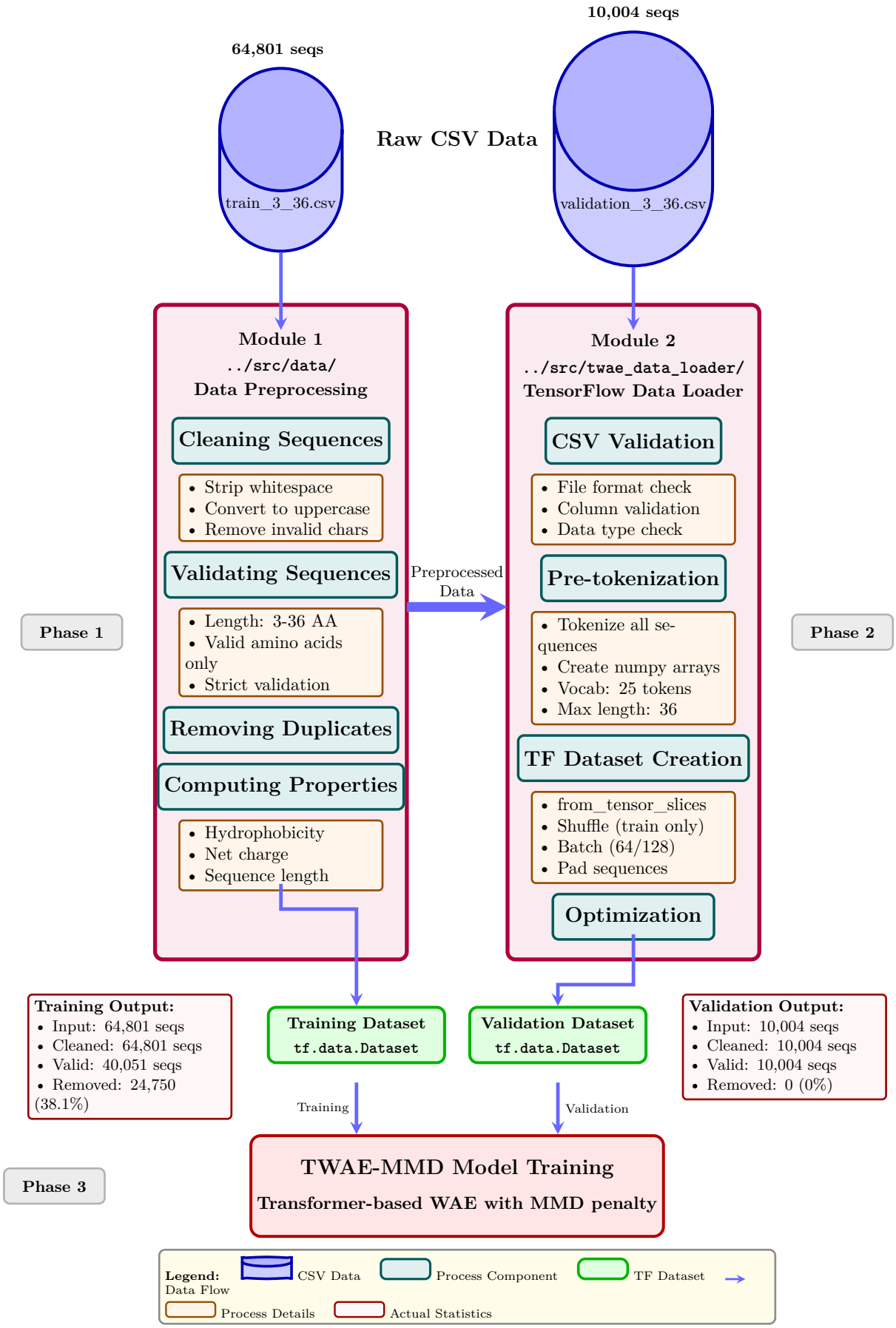

Figure 2.3: Data peptides_preprocessing and loading pipeline for TWAE-MMD model training with actual statistics from production runs. The pipeline consists of two sequential modules: **Module 1** performs comprehensive data preprocessing. **Module 2** creates optimized TensorFlow datasets through CSV validation, pre-tokenization with a 25-token vocabulary, batching, and optimization strategies. The pipeline produces `tf.data.Dataset` objects containing 626 training batches and 78 validation batches, ready for TWAE-MMD model training. This shared pipeline is used for both EMA and LSBO sampling strategies.

pre-tokenized arrays into `tf.data.Dataset`), and optimization (shuffling with buffer 10,000 for training, batching with size 64 for training and 128 for validation, caching in memory, prefetching with `AUTOTUNE`).

The pipeline processes the training dataset (`train_3_36.csv`) with 64,801 input sequences and produces 40,051 valid sequences after preprocessing (38.1% removed due to duplicates and validation failures), organized into 626 batches of size 64. The validation dataset (`validation_3_36.csv`) with 10,004 input sequences produces 10,004 valid sequences (0% removed), organized into 78 batches of size 128. Each batch has three parts: `input_ids` with shape (batch_size, 36) for tokenized sequences, `attention_mask` with shape (batch_size, 36) showing real tokens versus padding, and `labels` with shape (batch_size,) containing binary class labels (0 for non-AMP, 1 for AMP).

The pipeline is the same for both EMA-based and LSBO-based training, which gives consistent data preprocessing and loading for different sampling strategies. The modular design allows independent development and testing of preprocessing, data loading, and model training, which makes the code easier to maintain and extend.

3.4 TWAE-MMD Model Architecture with EMA Sampling

3.4.1 Architectural Overview

The Transformer-based Wasserstein Autoencoder with Maximum Mean Discrepancy (TWAE-MMD) uses a generative model structure combining transformer-based sequence encoding and decoding with distribution matching regularization.

The architecture, which is explicitly shown in the architecture diagrams (Figure 2.4, Figure 2.5, Figure 2.6, Figure 2.7 and Figure 2.8), has three primary components: (1) a transformer encoder that maps input sequences to fixed-dimensional representations, (2) a latent space manager that projects encoder outputs to a 128-dimensional latent space and reconstructs sequences from latent codes, and (3) a transformer decoder that generates output sequences from latent representations.

The model uses Exponential Moving Average (EMA) tracking of latent space statistics to enable stable sampling during generation.

The architecture has 4,127,137 trainable parameters distributed across encoder (1,892,608 parameters), latent space manager (230,529 parameters), and decoder (2,004,000 parameters). The model processes input sequences of maximum length 37 tokens (including special tokens) with a vocabulary size of 25 tokens, producing both classification predictions (binary AMP/non-AMP) and reconstructed sequences. Training uses a composite loss function combining classification loss, reconstruction loss, Maximum Mean Discrepancy (MMD) loss, and Wasserstein loss to jointly optimize classification accuracy and generative quality.

Phase 1: Input Data Processing

The input step sets up the data interface between preprocessed peptide sequences and the transformer encoder, using batching, tokenization validation, and attention mask generation. Input sequences are organized into batches of size $B = 64$, each containing $L = 37$ token positions representing the maximum sequence length including special tokens. The tokenization scheme uses a vocabulary of 25 discrete tokens: 20 standard proteinogenic amino acids and 5 special tokens ([PAD] with ID 0, [UNK] with ID 1, [CLS] with ID 2, [SEP] with ID 3, [MASK] with ID 4). Each sequence is prefixed with the [CLS] token to mark sequence beginning and suffixed with the [SEP] token to mark sequence termination, with shorter sequences padded to length 37 using [PAD] tokens appended to the right.

The attention mask mechanism gives binary indicators distinguishing real tokens from padding positions, which lets the transformer encoder to selectively attend meaningful sequence content while ignoring padding artifacts. The attention mask tensor has shape $[B, L] = [64, 37]$ with values of 1 indicating real tokens (amino acids and special tokens [CLS], [SEP]) and values of 0 indicating padding tokens. This mask is broadcast

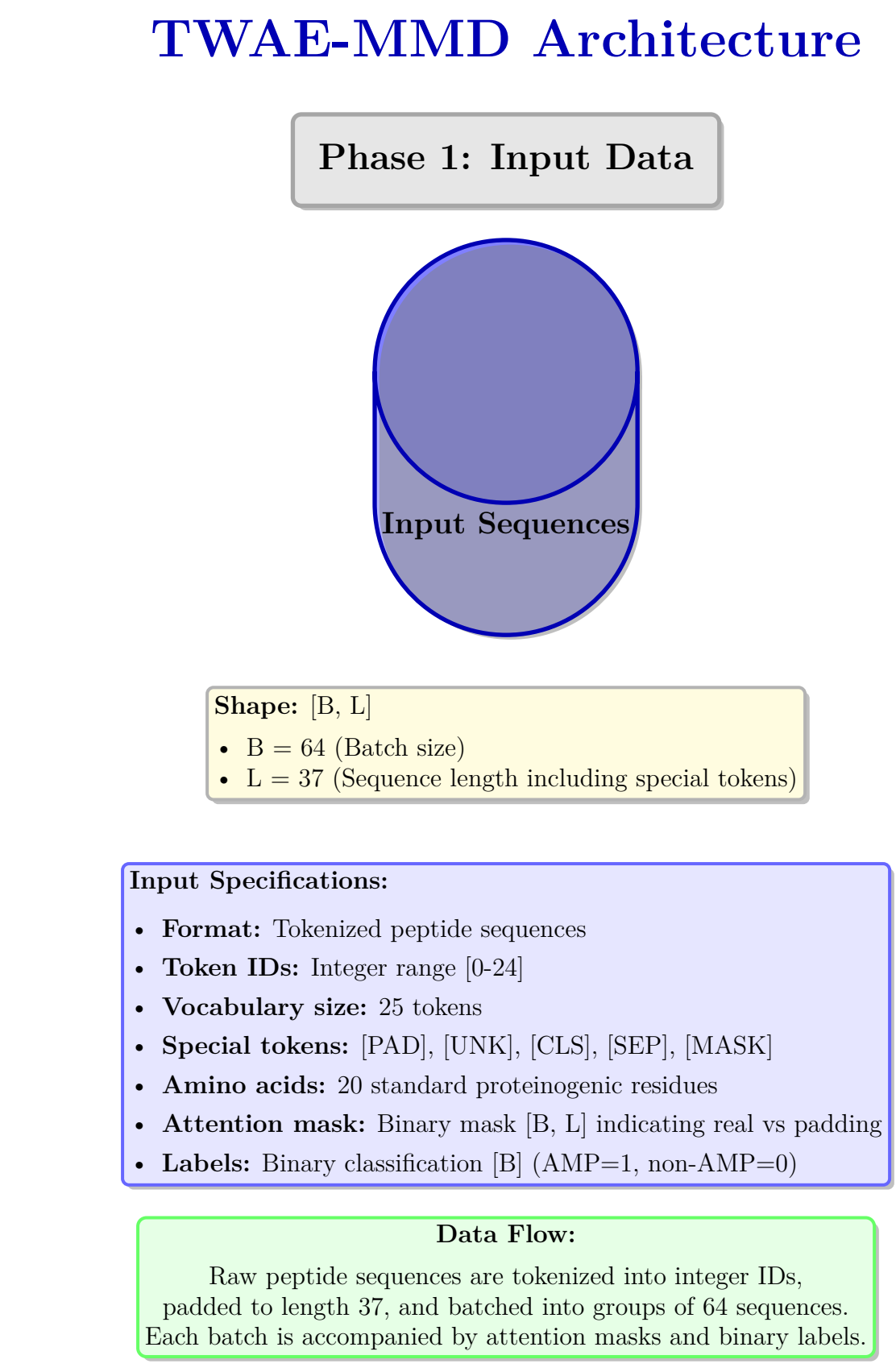

Figure 2.4: TWAE-MMD Architecture Phase 1: Input Data. The input phase processes batched peptide sequences with shape [B, L] where B=64 (batch size) and L=37 (sequence length including special tokens). Sequences are tokenized into integer IDs (range 0-24) from a vocabulary of 25 tokens comprising 20 standard amino acids and 5 special tokens ([PAD], [UNK], [CLS], [SEP], [MASK]). Each batch is accompanied by attention masks indicating real tokens versus padding, and binary labels for AMP/non-AMP classification.

across attention heads during multi-head attention computation, effectively setting attention weights to zero for padding positions. Each batch is accompanied by binary classification labels with shape $[B] = [64]$, where label 1 indicates antimicrobial peptides (AMP) and label 0 indicates non-antimicrobial peptides (non-AMP), which allows supervised learning of sequence-activity relationships. The input step diagram (Figure 2.4) illustrates the input data structure, batch organization, and tokenization scheme.

3.4.2 Transformer Encoder Architecture

The transformer encoder uses a multi-layer architecture consisting of token embedding, positional encoding, stacked transformer blocks, and attention pooling [9]. The token embedding layer maps each input token ID (0–24) to a 256-dimensional dense vector representation, creating an embedding matrix of shape (25, 256). The positional encoding component adds learned positional embeddings to the token embeddings, which lets the model capture sequential information in the input sequences. The combined embeddings are passed through a dropout layer with rate 0.25 to prevent overfitting.

The encoder consists of 4 transformer blocks, each using multi-head attention mechanism followed by a feed-forward network [9]. The multi-head attention mechanism uses 8 attention heads, each operating on 32-dimensional projections (256 / 8 = 32 dimensions per head) [9]. The attention mechanism computes scaled dot-product attention [9]: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$, where Q, K, and V represent query, key, and value matrices, and $d_k = 32$ is the dimension per head. The multi-head outputs are concatenated and projected back to 256 dimensions. Attention dropout with rate 0.15 is used to attention weights to improve generalization.

The feed-forward network in each transformer block uses a two-layer architecture with expansion: the first layer projects from 256 to 1024 dimensions using GELU activation [71], and the second layer projects back from 1024 to 256 dimensions. Each transformer block uses pre-normalization with LayerNormalization applied before attention and feed-forward operations, residual connections around both sub-layers, dropout with rate 0.25, stochastic depth (layer dropout) for training regularization, and layer scale parameters for stable gradient flow.

The encoder output is aggregated using attention pooling, which computes a weighted average of the sequence representations: $\mathbf{h} = \sum_{i=1}^{L} \alpha_i \mathbf{h}_i$, where $\alpha_i = \frac{\exp(w^T \mathbf{h}_i)}{\sum_j \exp(w^T \mathbf{h}_j)}$ are learned attention weights and $\mathbf{h}_i$ are the encoder outputs at position i. This produces a single 256-dimensional vector representation for each input sequence, which is input to the latent space manager.

Phase 2: Encoding Operations

The encoding step transforms variable-length token sequences into fixed-dimensional semantic representations through a hierarchical three-stage process. The first stage uses token embedding and positional encoding, mapping discrete token IDs to continuous 256-dimensional vectors through a learned embedding matrix of shape $(25, 256)$ and augmenting these embeddings with learned positional information to capture sequential dependencies. The embedding layer applies dropout with rate 0.25 to prevent overfitting, producing output tensors of shape $[B, L, d_{model}] = [64, 37, 256]$ where each token is represented by a 256-dimensional vector.

The second stage processes embedded sequences through the transformer encoder, consisting of 4 stacked transformer blocks with multi-head attention mechanism and position-wise feed-forward networks. Each transformer block computes attention across all sequence positions using 8 attention heads operating on 32-dimensional subspaces, which lets the model capture diverse linguistic patterns and long-range dependencies. The feed-forward networks expand representations to 1024 dimensions before projecting back to 256 dimensions, with GELU activation which gives non-linearity. Pre-normalization, residual connections, stochastic depth (linearly increasing from 0.0 to 0.1 across layers), and layer scale parameters (initialized to 10^{-4}) ensure stable gradient flow and effective training of deep architectures.

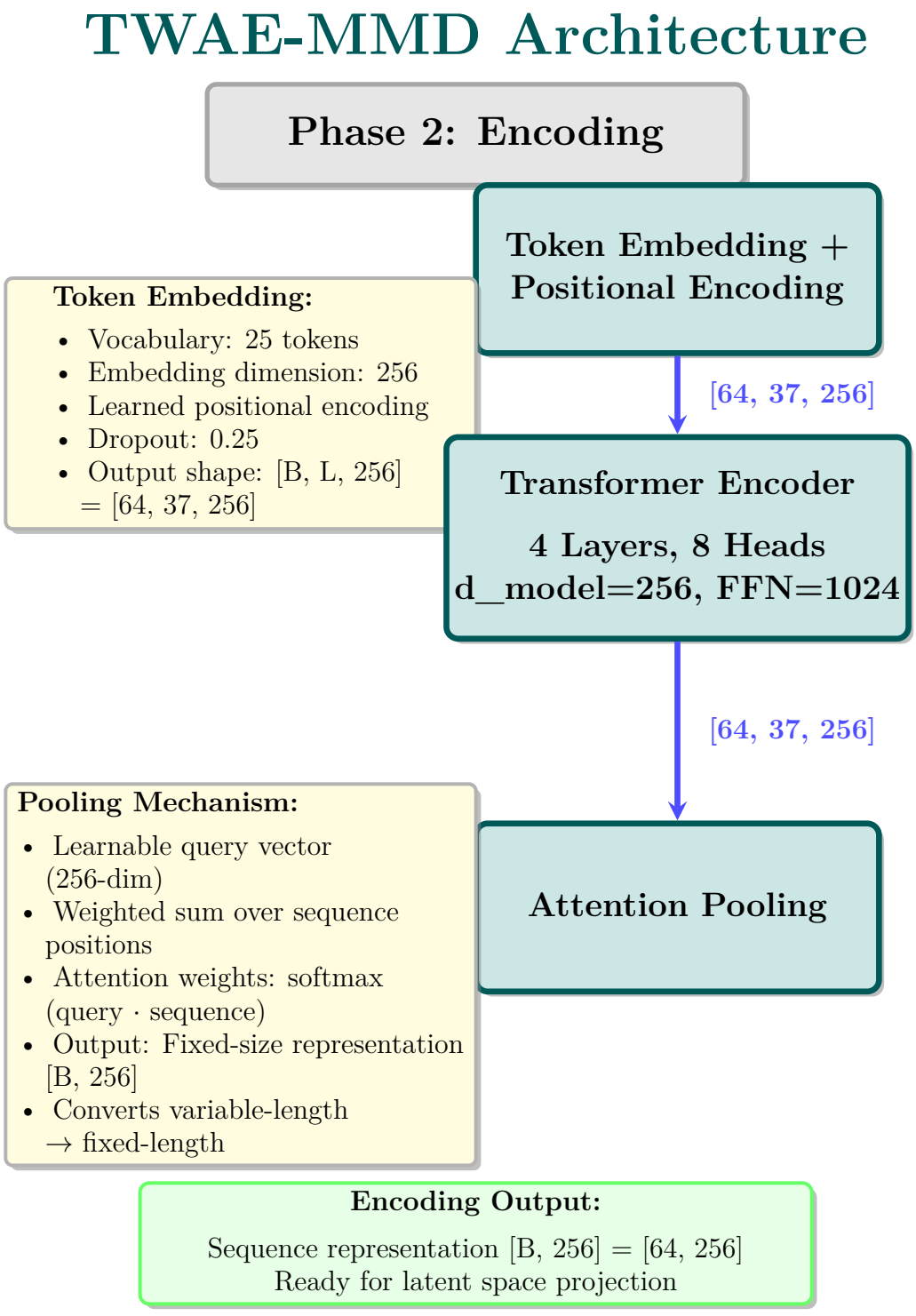

Figure 2.5: TWAE-MMD Architecture Phase 2: Encoding. The encoding phase transforms input token sequences into fixed-size representations through three sequential operations: (1) Token embedding and positional encoding map token IDs to 256-dimensional vectors with learned positional information, (2) Transformer encoder with 4 layers and 8 attention heads processes sequences using multi-head attention mechanism and feed-forward networks (256→1024→256) with pre-normalization, stochastic depth, and dropout regularization, (3) Attention pooling aggregates variable-length sequences into fixed-size 256-dimensional representations using a learnable query vector and weighted sum mechanism.

The third stage aggregates variable-length sequence representations into fixed-size vectors through attention pooling, calculating a weighted average of encoder outputs using learned attention weights. The pooling mechanism uses a learnable query vector to compute attention scores over all sequence positions, producing attention weights $\alpha_i = \text{softmax}(w^T \mathbf{h}_i)$ that emphasize informative positions while de-emphasizing padding and less relevant tokens. The resulting pooled representation has shape $[B, 256] = [64, 256]$, which gives a fixed-dimensional encoding suitable for latent space projection. The encoding step diagram (Figure 2.5) illustrates the encoding pipeline from token embeddings to pooled representations.

3.4.3 Latent Space Architecture

The latent space manager uses a bottleneck architecture that compresses the 256-dimensional encoder output to a 128-dimensional latent representation and subsequently expands it back to a format suitable for decoder input. The latent encoder component consists of three fully connected layers with dimensions 256 → 512 → 256 → 128, employing GELU activation functions, batch normalization after each layer, and dropout with rate 0.25 for regularization. The final layer produces the latent vector $\mathbf{z} \in \mathbb{R}^{128}$ with tanh activation, constraining latent codes to the range $[-1, 1]^{128}$.

The latent space is regularized to approximate a standard Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I}_{128})$ through the MMD and Wasserstein loss components, which allows sampling of new sequences during generation.

The Exponential Moving Average (EMA) tracker maintains running estimates of the latent space mean $\boldsymbol{\mu}_{\text{EMA}}$ and standard deviation $\boldsymbol{\sigma}_{\text{EMA}}$ across training batches. The EMA statistics are updated using decay factor $\beta = 0.999$: $\boldsymbol{\mu}_{\text{EMA}} \leftarrow \beta \boldsymbol{\mu}_{\text{EMA}} + (1 - \beta)\boldsymbol{\mu}_{\text{batch}}$ and $\boldsymbol{\sigma}_{\text{EMA}} \leftarrow \beta \boldsymbol{\sigma}_{\text{EMA}} + (1 - \beta)\boldsymbol{\sigma}_{\text{batch}}$, where $\boldsymbol{\mu}_{\text{batch}}$ and $\boldsymbol{\sigma}_{\text{batch}}$ are calculated from the current batch of latent vectors. During generation, new latent codes are sampled from the EMA distribution: $\mathbf{z}_{\text{new}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{EMA}}, \text{diag}(\boldsymbol{\sigma}^2_{\text{EMA}}))$.

The latent decoder component expands the 128-dimensional latent vector back to a format suitable for the transformer decoder. It consists of three fully connected layers with dimensions 128 → 256 → 512 → 925, employing GELU activation and batch normalization. The output is reshaped to (37, 25), representing 37 time steps with 25-dimensional feature vectors, which serve as input to the transformer decoder.

Phase 3: Latent Space Compression and EMA Tracking

The latent space phase uses a bottleneck architecture that compresses 256-dimensional encoder representations to 128-dimensional latent codes while maintaining sufficient information for sequence reconstruction and classification. The latent encoder consists of three fully connected layers using progressive dimensionality transformations: 256 → 512 (expansion), 512 → 256 (compression), and 256 → 128 (final bottleneck). Each layer uses GELU activation to introduce non-linearity, batch normalization to stabilize training dynamics, and dropout with rate 0.25 to prevent overfitting. The final layer applies tanh activation to bound latent codes to the range $[-1, 1]^{128}$, constraining the latent space to a compact hypercube and which helps stable optimization.

The latent vector $\mathbf{z} \in \mathbb{R}^{128}$ is the information bottleneck, forcing the model to learn compressed representations that capture essential sequence features while discarding noise and redundancy. The latent distribution is regularized to approximate a standard Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I}_{128})$ through Maximum Mean Discrepancy (MMD) and Wasserstein loss components, which allows sampling of new sequences during generation by drawing latent codes from the learned distribution.

The Exponential Moving Average (EMA) tracker [36] [37] maintains running estimates of latent space statistics across training batches, calculating mean $\boldsymbol{\mu}_{\text{EMA}} \in \mathbb{R}^{128}$ and standard deviation $\boldsymbol{\sigma}_{\text{EMA}} \in \mathbb{R}^{128}$ with decay factor $\beta = 0.999$ [36] [37]. The EMA update equations $\boldsymbol{\mu}_{\text{EMA}} \leftarrow 0.999 \cdot \boldsymbol{\mu}_{\text{EMA}} + 0.001 \cdot \boldsymbol{\mu}_{\text{batch}}$ and $\boldsymbol{\sigma}_{\text{EMA}} \leftarrow 0.999 \cdot \boldsymbol{\sigma}_{\text{EMA}} + 0.001 \cdot \boldsymbol{\sigma}_{\text{batch}}$ provide exponentially-weighted averages that adapt slowly to changing latent distributions while filtering out batch-level noise [36] [37]. During generation, new latent codes are sampled from the EMA distribution $\mathbf{z}_{\text{new}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{EMA}}, \text{diag}(\boldsymbol{\sigma}^2_{\text{EMA}}))$, which allows stable and high-quality sequence generation without requiring explicit constraint enforcement [36] [37].

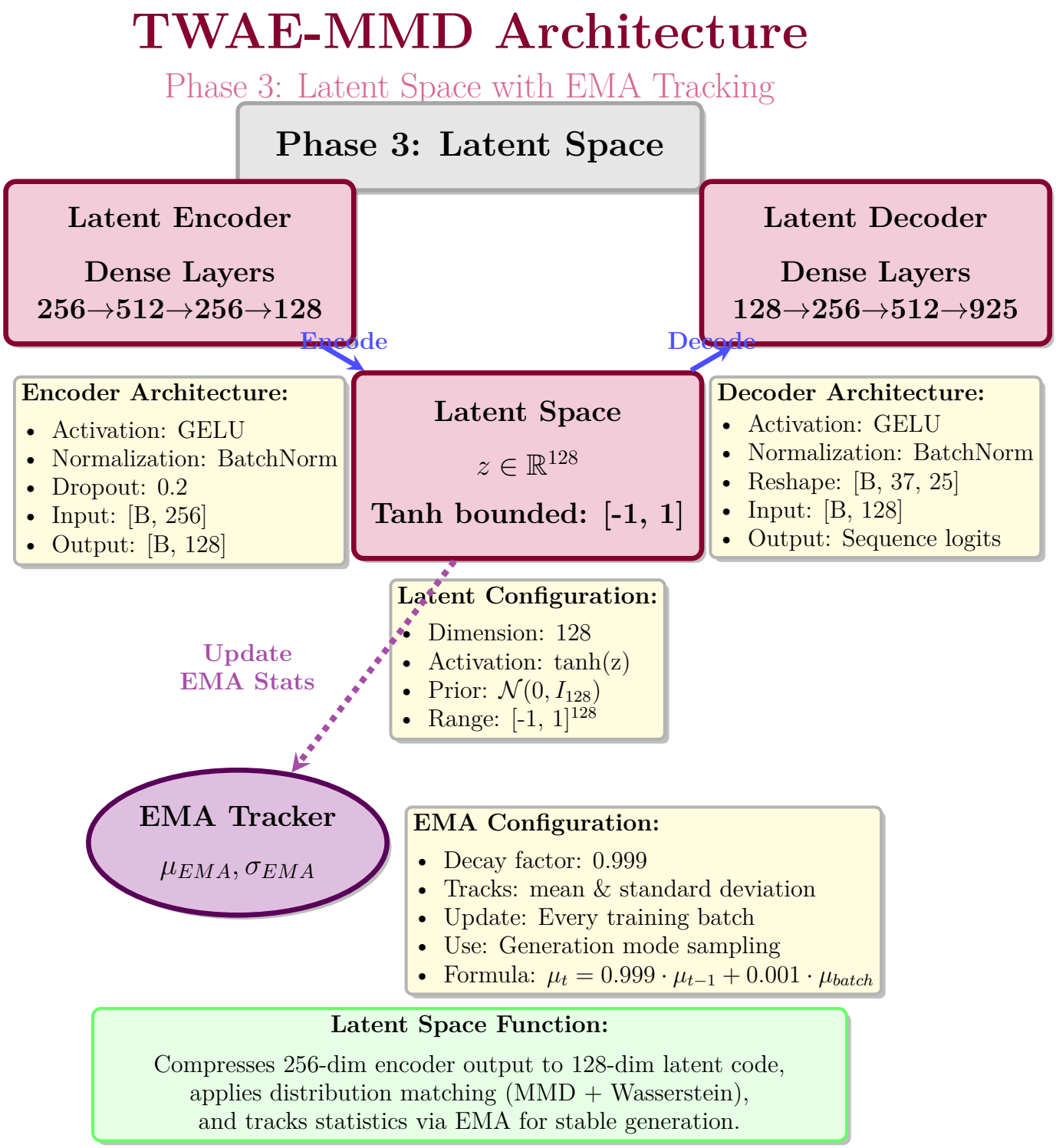

Figure 2.6: TWAE-MMD Architecture Phase 3: Latent Space with EMA Tracking. The latent space phase compresses encoder representations to a 128-dimensional bottleneck through three components: (1) Latent encoder with dense layers (256→512→256→128) using GELU activation and batch normalization, (2) Latent vector $z \in \mathbb{R}^{128}$ with tanh activation bounding values to [-1, 1], (3) Latent decoder with dense layers (128→256→512→925) reshaping to [B, 37, 25] for sequence reconstruction. The Exponential Moving Average (EMA) tracker maintains running statistics (μ_{EMA}, σ_{EMA}) with decay factor 0.999, updated every training batch and used for stable sampling during generation.

The latent decoder expands 128-dimensional latent codes back to a format suitable for transformer decoder input through three fully connected layers: $128 \to 256$, $256 \to 512$, and $512 \to 925$. The output is reshaped to $(37, 25)$, representing 37 time steps with 25-dimensional feature vectors corresponding to vocabulary logits at each position. This reshaping operation bridges the latent space and sequence space, which lets the transformer decoder generate token sequences conditioned on latent codes. The latent space phase diagram (Figure 2.6) illustrates the latent space architecture including encoder, latent vector, decoder, and EMA tracking mechanism.

3.4.4 Transformer Decoder Architecture

The transformer decoder uses an generation architecture consisting of 4 decoder blocks, each containing masked attention head, cross-attention to encoder outputs, and feed-forward networks [9]. The decoder processes the reshaped latent representation (37, 25) through learned projection to 256 dimensions, followed by positional encoding addition.

Each decoder block uses three sub-layers [9]: (1) multi-head attention [9], employing 8 attention heads and attention dropout rate 0.15; (2) multi-head cross-attention to the encoder output (the 256-dimensional pooled representation broadcasted across 36 positions) [9], which lets the decoder condition generation on the encoded input; (3) feed-forward network with expansion from 256 to 1024 dimensions using GELU activation, followed by projection back to 256 dimensions. Each sub-layer uses pre-normalization, residual connections, dropout with rate 0.25, and layer scale parameters.

The final decoder output is projected to vocabulary logits using a linear layer: $256 \to 25$ dimensions, producing unnormalized log-probabilities for each of the 25 tokens at each of the 36 positions. During training, these logits are used to compute reconstruction loss via sparse categorical cross-entropy. During generation, all tokens are produced simultaneously in a single forward pass through parallel generation.

3.4.5 Classification Head

A classification head is attached to the encoder output to enable supervised learning of AMP versus non-AMP discrimination. The classification head consists of a single fully connected layer projecting the 256-dimensional encoder output to 2 dimensions, representing logits for the two classes (non-AMP and AMP). During training, these logits are used to compute binary cross-entropy loss against the ground-truth labels. During inference, the logits are converted to probabilities using softmax, and the class with higher probability is selected as the prediction.

Phase 4: Decoding and Classification Pathways

The decoding step uses two parallel computational pathways that jointly optimize sequence reconstruction and classification objectives. The first pathway uses the transformer decoder to reconstruct input sequences from latent representations, using a parallel generation architecture with 4 decoder blocks. Each decoder block has three sub-layers: multi-head attention, multi-head cross-attention to the latent representation which allows conditioning on compressed sequence information, and position-wise feed-forward networks with expansion from 256 to 1024 dimensions. The decoder processes the reshaped latent representation $(37, 25)$ through learned projection to 256 dimensions, adds positional encodings, and applies the 4-layer decoder stack to produce output representations of shape $[B, L, d_{model}] = [64, 37, 256]$.

The decoder output is projected to vocabulary logits through a final linear layer: $256 \to 25$ dimensions, producing unnormalized log-probabilities for each of the 25 tokens at each of the 36 sequence positions. During training, teacher forcing is employed: the decoder receives ground-truth tokens as input and generates predictions for the next token at each position, with reconstruction loss computed via sparse categorical cross-entropy comparing predicted token distributions to ground-truth sequences. During generation, the

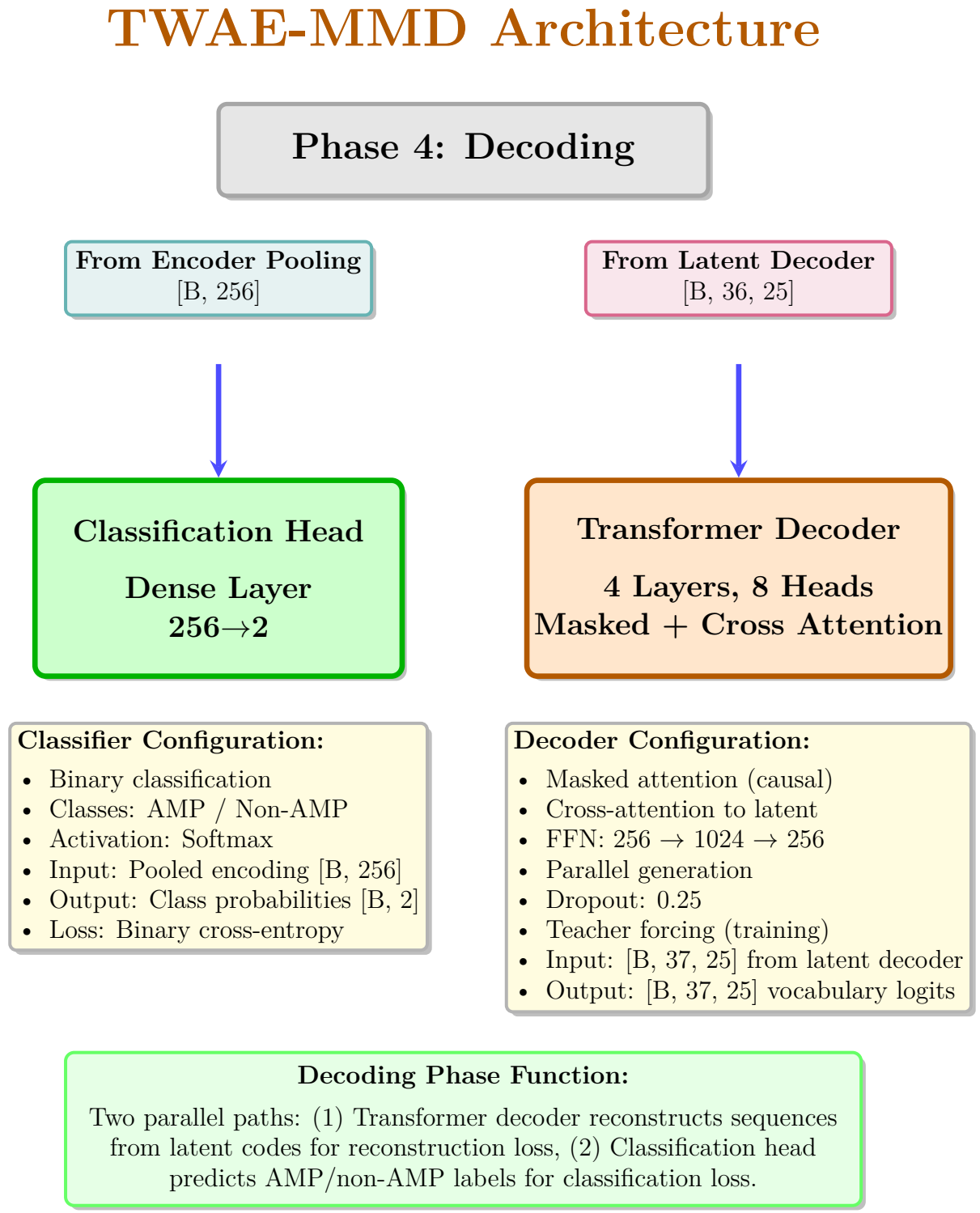

Figure 2.7: TWAE-MMD Architecture Phase 4: Decoding. The decoding phase implements two parallel pathways for sequence reconstruction and classification: (1) Transformer decoder receives latent representations [B, 37, 25] from the latent decoder and generates reconstructed sequences through 4 layers with masked attention, cross-attention to latent codes, and feed-forward networks (256→1024→256), employing teacher forcing during training and producing vocabulary logits [B, 37, 25] for reconstruction loss computation, (2) Classification head receives pooled encoder representations [B, 256] and projects to binary class logits [B, 2] via a dense layer (256→2) with *softmax* activation, enabling supervised learning of AMP versus non-AMP discrimination through binary cross-entropy loss.

decoder produces all tokens simultaneously in a single forward pass through parallel generation. The logits are converted to probabilities using *softmax*, and tokens are sampled using categorical sampling.

The second pathway uses binary classification through a classification head attached to the pooled encoder representation. The classification head consists of a single fully connected layer projecting the 256-dimensional encoder output to 2 dimensions, representing unnormalized logits for the two classes: non-AMP (class 0) and AMP (class 1). During training, these logits are used to compute binary cross-entropy loss against ground-truth labels, which allows supervised learning of sequence-activity relationships. During inference, the logits are converted to probabilities using *softmax*: $p(\text{AMP}) = \frac{\exp(\text{logit}_1)}{\exp(\text{logit}_0)+\exp(\text{logit}_1)}$, and the class with higher probability is selected as the prediction. This dual-pathway architecture lets the model simultaneously learn discriminative features for classification and generative features for sequence reconstruction, with shared encoder representations benefiting both tasks through multi-task learning. The decoding step diagram (Figure 2.7) illustrates the decoding architecture including both reconstruction and classification pathways.

3.4.6 Loss Function Formulation

The TWAE-MMD model uses a composite loss function combining four components with fixed weights:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{recon}} + 100.0 \cdot \mathcal{L}_{\text{MMD}} + 10.0 \cdot \mathcal{L}_{\text{Wasserstein}} \tag{2.1}$$

The classification loss $\mathcal{L}_{\text{cls}}$ uses binary cross-entropy to measure the discrepancy between predicted class probabilities and ground-truth labels:

$$\mathcal{L}_{\text{cls}} = -\frac{1}{B}\sum_{i=1}^{B}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \tag{2.2}$$

where B is the batch size, $y_i \in \{0,1\}$ is the ground-truth label, and $\hat{y}_i$ is the predicted probability for class 1 (AMP).

The reconstruction loss $\mathcal{L}_{\text{recon}}$ uses sparse categorical cross-entropy to measure the discrepancy between predicted token distributions and ground-truth sequences:

$$\mathcal{L}_{\text{recon}} = -\frac{1}{B \cdot L}\sum_{i=1}^{B}\sum_{t=1}^{L} \log p(x_{i,t}|\mathbf{z}_i) \tag{2.3}$$

where $L = 37$ is the sequence length, $x_{i,t}$ is the ground-truth token at position t in sequence i, and $p(x_{i,t}|\mathbf{z}_i)$ is the predicted probability for the ground-truth token given latent code $\mathbf{z}_i$.

The Maximum Mean Discrepancy (MMD) loss measures the distance between the empirical distribution of latent codes Q_Z and the prior distribution $P_Z = \mathcal{N}(\mathbf{0}, \mathbf{I})$ using the radial basis function (RBF) kernel:

$$\mathcal{L}_{\text{MMD}} = \frac{1}{B^2}\sum_{i,j} k(\mathbf{z}_i, \mathbf{z}_j) - \frac{2}{B \cdot M}\sum_{i,m} k(\mathbf{z}_i, \mathbf{z}'_m) + \frac{1}{M^2}\sum_{m,n} k(\mathbf{z}'_m, \mathbf{z}'_n) \tag{2.4}$$

where $k(\mathbf{z}_i, \mathbf{z}_j) = \exp(-\|\mathbf{z}_i - \mathbf{z}_j\|^2/(2\sigma^2))$ is the RBF kernel with bandwidth σ, $\mathbf{z}_i$ are empirical latent codes, and $\mathbf{z}'_m$ are samples from the prior P_Z.

The Wasserstein loss term, $\mathcal{L}_{\text{Wasserstein}}$, is computed using two different methods for distinct purposes within the TWAE-MMD framework. For The regularization part of the loss function (Equation 1.1), the Wasserstein distance is estimated using the **energy distance** method (‘method="energy"‘), which gives a computationally measure of the discrepancy between the encoded latent distribution Q_Z and the prior distribution P_Z. This encourages the latent distribution to match the prior while preserving its topological structure. For the latent quality evaluation metric (Section 1.4.7), the 1D Wasserstein Distance is computed independently for each latent dimension using the **Sinkhorn divergence** method (‘method="sinkhorn"‘), which gives a more precise measure of distribution matching for analysis and analysis.

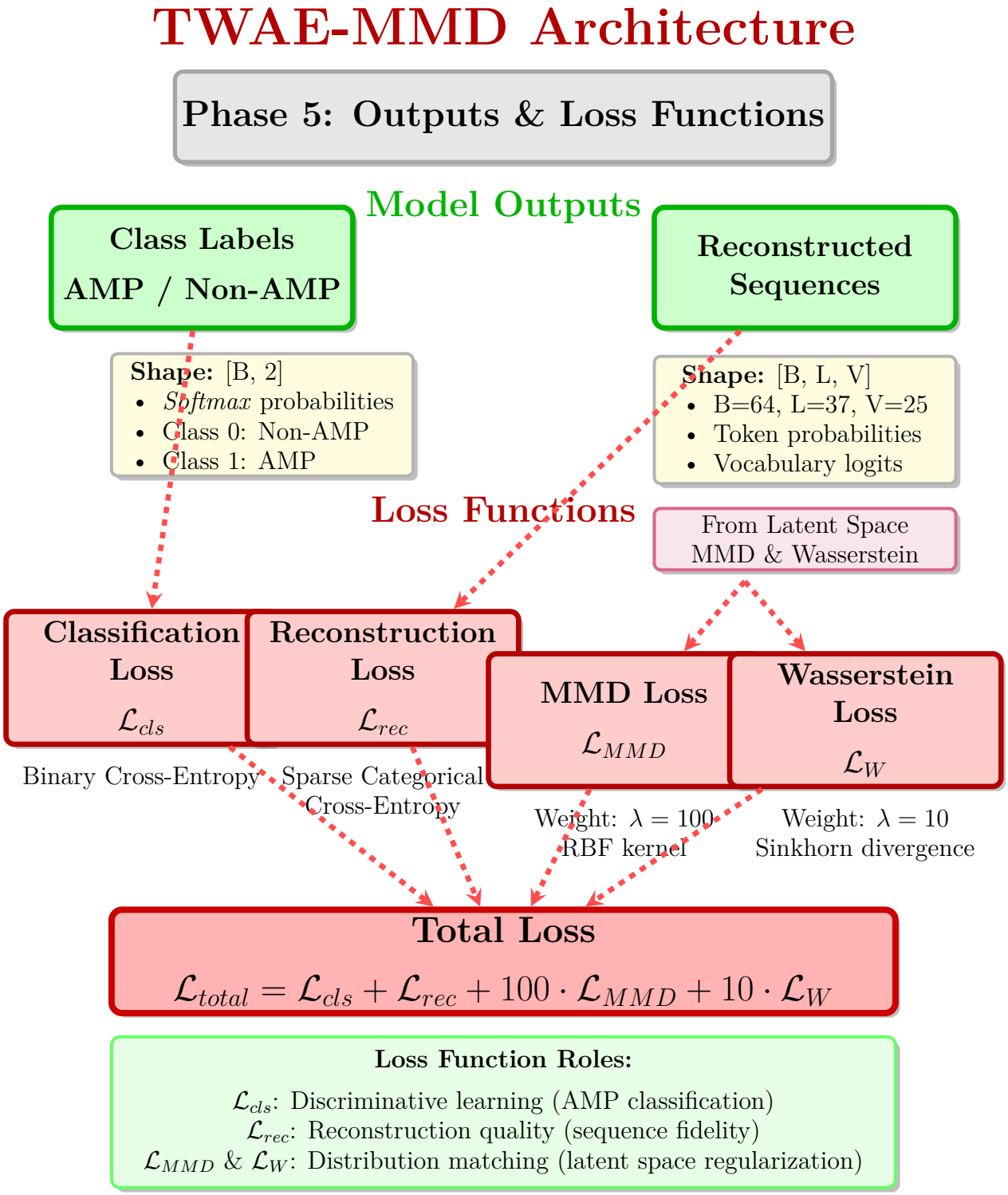

Figure 2.8: TWAE-MMD Architecture Phase 5: Outputs and Loss Functions. The output phase generates model predictions and computes composite loss for training: (1) Model outputs comprise class labels [B, 2] with *softmax* probabilities for AMP/non-AMP classification and reconstructed sequences [B, L, V] with token probabilities (B=64, L=37, V=25), (2) Four loss components are computed: classification loss $\mathcal{L}_{cls}$ using binary cross-entropy, reconstruction loss $\mathcal{L}_{rec}$ using sparse categorical cross-entropy, Maximum Mean Discrepancy loss $\mathcal{L}_{MMD}$ with weight $\lambda = 100$ measuring latent distribution divergence via RBF kernel, and Wasserstein loss $\mathcal{L}_W$ with weight $\lambda = 10$ measuring optimal transport cost via Sinkhorn divergence, (3) Total loss combines all components: $\mathcal{L}_{total} = \mathcal{L}_{cls} + \mathcal{L}_{rec} + 100 \cdot \mathcal{L}_{MMD} + 10 \cdot \mathcal{L}_W$, jointly optimizing discriminative classification and generative reconstruction with latent space regularization.

Phase 5: Model Outputs and Composite Loss Computation

The output step generates model predictions and computes a composite loss function that jointly optimizes discriminative classification and generative reconstruction objectives with latent space regularization. The model produces two primary outputs: class labels with shape $[B, 2] = [64, 2]$ containing *softmax* probabilities for AMP and non-AMP classes, and reconstructed sequences with shape $[B, L, V] = [64, 36, 25]$ containing token probability distributions at each sequence position. These outputs enable evaluation of both classification accuracy (measuring the model's ability to distinguish AMPs from non-AMPs) and reconstruction quality (measuring the model's ability to faithfully reconstruct input sequences through the latent bottleneck).

The composite loss function combines four components with fixed weights to balance multiple training objectives. The classification loss $\mathcal{L}_{\text{cls}}$ uses binary cross-entropy to measure discrepancy between predicted class probabilities and ground-truth labels, encouraging accurate discrimination of AMP versus non-AMP sequences. The reconstruction loss $\mathcal{L}_{\text{recon}}$ uses sparse categorical cross-entropy to measure discrepancy between predicted token distributions and ground-truth sequences, encouraging faithful reconstruction through the 128-dimensional latent bottleneck. The Maximum Mean Discrepancy (MMD) loss $\mathcal{L}_{\text{MMD}}$ with weight $\lambda = 100$ measures the distance between the empirical latent distribution and a standard Gaussian prior using the radial basis function (RBF) kernel, encouraging the latent space to approximate $\mathcal{N}(\mathbf{0}, \mathbf{I}_{128})$ and which allows sampling of new sequences. The Wasserstein loss $\mathcal{L}_{\text{W}}$ with weight $\lambda = 10$ measures the best transport cost between the empirical and prior distributions using Sinkhorn divergence approximation, which gives an alternative regularization signal that preserves topological structure in the latent space.

The loss $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{recon}} + 100 \cdot \mathcal{L}_{\text{MMD}} + 10 \cdot \mathcal{L}_{\text{W}}$ is minimized through gradient descent, with gradients backpropagated through all model components (encoder, latent manager, decoder, classifier) to update the 4,127,137 trainable parameters. The loss weights (1.0 for classification, 1.0 for reconstruction, 100.0 for MMD, 10.0 for Wasserstein) were determined through hyperparameter optimization to balance the four objectives: classification loss ensures discriminative accuracy, reconstruction loss ensures generative quality, MMD and Wasserstein losses ensure latent space regularity and enable stable sampling. This multi-objective optimization strategy lets the model simultaneously learn discriminative representations for classification and generative representations for sequence synthesis, with distribution matching regularization to make sure the latent space has desirable properties for new sequence generation. The output and loss phase diagram (Figure 2.8) illustrates the calculating loss architecture including all four loss components and their integration into the loss.

3.5 Training Procedure with EMA Sampling

3.5.1 Training Configuration

The TWAE-MMD model with EMA sampling is trained using the AdamW optimizer [56] with learning rate $\eta = 10^{-4}$ and weight decay $\lambda_{\text{wd}} = 0.01$. The training process, which is explicitly shown in the training process diagrams (Figure 2.9, Figure 2.10, Figure 2.11, Figure 2.12, and Figure 2.13), spans a maximum of 100 epochs with a target validation accuracy threshold of 96%. Training uses batch sizes of 64 for the training set (626 batches per epoch) and 128 for the validation set (78 batches per epoch).

Regularization techniques include dropout (rate 0.25 for feed-forward layers, rate 0.15 for attention weights), stochastic depth (layer dropout) for transformer blocks, and layer scale parameters for stable gradient flow. Mixed precision training is not employed due to numerical stability considerations on the RTX 2060 GPU; all computations use float32 precision.

GPU memory management is configured with dynamic memory growth and a 5.5 GB limit on the RTX 2060 (6 GB VRAM). Gradient clipping with maximum norm 1.0 is used to prevent exploding gradients. The model is checkpointed at the end of each epoch, with the best-doing checkpoint (highest validation accuracy)

retained for subsequent analysis and generation.

Phase 1: Training Environment Initialization

The training environment initialization phase sets up the computational infrastructure and data structures required for model optimization. The initialization sequence commences with dataset loading, wherein the preprocessed training dataset containing 40,051 peptide sequences is loaded from disk and organized into 626 batches of size 64, and the validation dataset containing 10,004 sequences is loaded and organized into 78 batches of size 128. The batch organization uses stratified sampling to maintain class balance across batches, with each batch containing approximately equal proportions of AMP (class 1) and non-AMP (class 0) sequences.

The TWAE-MMD model structure is instantiated with 4,127,137 trainable parameters distributed across three primary components: the transformer encoder (1,892,608 parameters using 4 transformer layers with 8 attention heads, 256-dimensional embeddings, and 1024-dimensional feed-forward networks), the latent space manager (230,529 parameters using encoder layers 256→512→256→128 and decoder layers 128→256→512→925), and the transformer decoder (2,004,000 parameters using 4 transformer layers with masked attention and cross-attention mechanisms). Model weights are initialized using Xavier uniform initialization for linear layers and orthogonal initialization for embedding layers, after established best practices for transformer architectures.

The Exponential Moving Average (EMA) tracker is initialized to maintain running estimates of latent space statistics across training batches. The EMA tracker stores two 128-dimensional vectors: mean $\boldsymbol{\mu}_{EMA} \in \mathbb{R}^{128}$ and standard deviation $\boldsymbol{\sigma}_{EMA} \in \mathbb{R}^{128}$, initialized to zero and one respectively. The decay factor is set to $\beta = 0.999$, which gives exponentially-weighted averaging that emphasizes recent batches while maintaining stability across training epochs. The EMA statistics are updated after handling each training batch, which lets continuous adaptation the evolving latent distribution throughout training. The initialization step diagram (Figure 2.9) illustrates the initialization sequence including dataset loading, model instantiation, and EMA tracker configuration.

3.5.2 Training Loop Architecture

The training loop uses a standard supervised learning procedure with five phases per epoch: initialization (loading datasets, initializing model and optimizer, initializing EMA tracker with decay 0.999), training (iterating over 626 training batches, calculating forward pass through encoder-latent-decoder-classifier, calculating composite loss with four components, doing backpropagation and weight updates, updating EMA statistics with current batch latent codes), validation (iterating over 78 validation batches, calculating forward pass without calculating gradients, calculating validation loss and accuracy, tracking best validation accuracy), latent space analysis (sampling 500 latent codes from training set, calculating quality metrics including mean, standard deviation, and distribution statistics), and per-epoch generation (generating 50 sequences using EMA sampling, calculating quality scores, saving sequences to file).

The EMA update mechanism works in parallel with the main training loop. After calculating latent codes $\mathbf{z}_i$ for each training batch, the batch statistics $\boldsymbol{\mu}_{\text{batch}} = \frac{1}{B}\sum_{i=1}^{B}\mathbf{z}_i$ and $\boldsymbol{\sigma}_{\text{batch}} = \sqrt{\frac{1}{B}\sum_{i=1}^{B}(\mathbf{z}_i - \boldsymbol{\mu}_{\text{batch}})^2}$ are calculated. The EMA statistics are then updated using decay factor $\beta = 0.999$: $\boldsymbol{\mu}_{\text{EMA}} \leftarrow \beta\boldsymbol{\mu}_{\text{EMA}} + (1 - \beta)\boldsymbol{\mu}_{\text{batch}}$ and $\boldsymbol{\sigma}_{\text{EMA}} \leftarrow \beta\boldsymbol{\sigma}_{\text{EMA}} + (1 - \beta)\boldsymbol{\sigma}_{\text{batch}}$. These statistics are saved with each model checkpoint to enable consistent generation across training runs.

Checkpointing occurs at the end of each epoch if the current validation accuracy exceeds the previous best. Each checkpoint consists of three files: model weights (`.h5` format) containing all 4.1M trainable parameters, EMA statistics (`.pkl` format) containing $\boldsymbol{\mu}_{\text{EMA}}$ and $\boldsymbol{\sigma}_{\text{EMA}}$, and training configuration (`.json` format) containing hyperparameters and training state. Training continues for up to 100 epochs, with the best-doing model (highest validation accuracy) selected for final generation. The target validation accuracy is 96%, though the model may achieve higher performance during training.

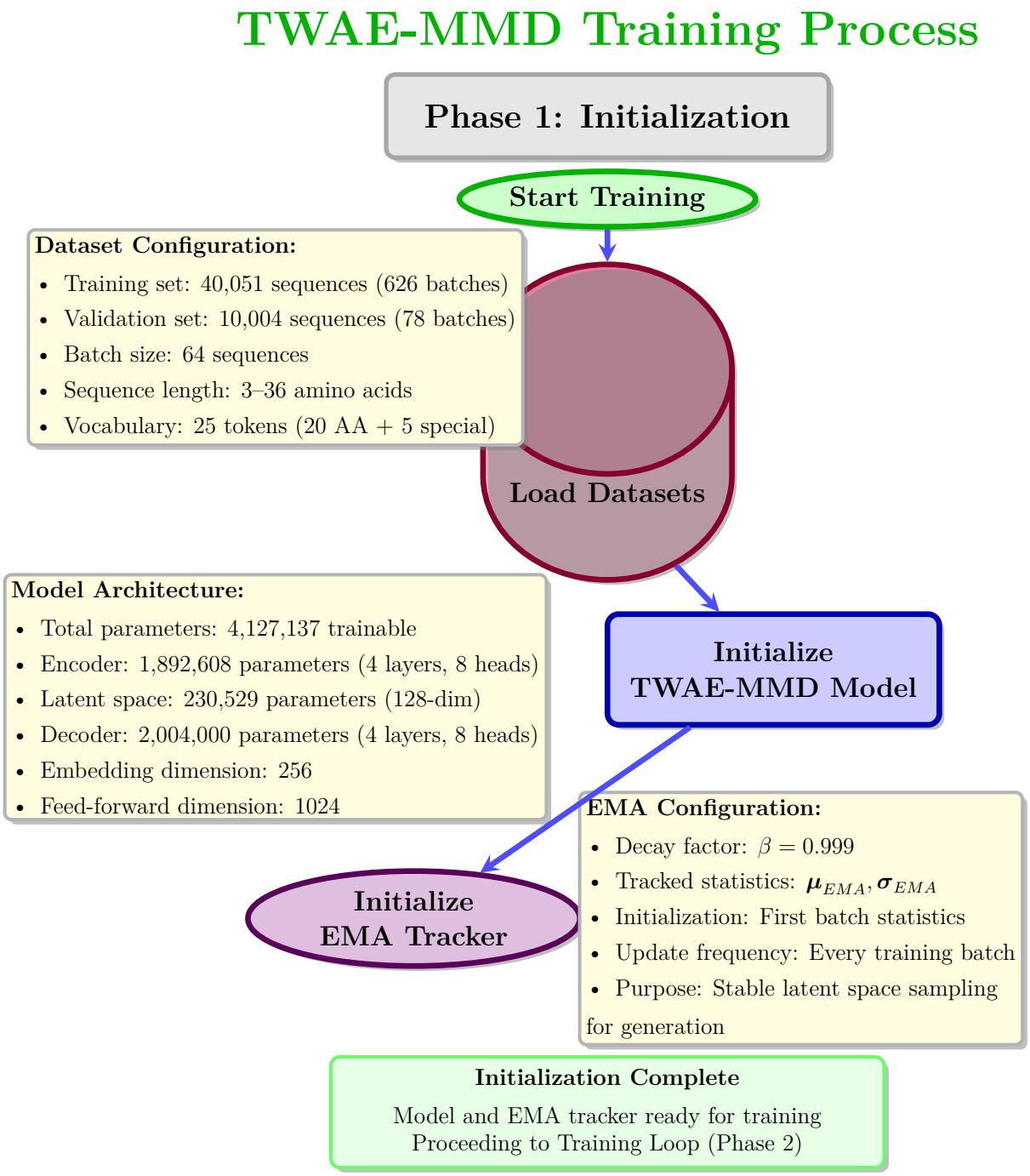

Figure 2.9: TWAE-MMD Training Process Phase 1: Initialization. The initialization phase establishes the training environment by loading preprocessed datasets (40,051 training sequences organized into 626 batches and 10,004 validation sequences organized into 78 batches with batch size 64), instantiating the TWAE-MMD model architecture with 4,127,137 trainable parameters distributed across encoder (1,892,608 parameters with 4 transformer layers and 8 attention heads), latent space manager (230,529 parameters implementing 128-dimensional bottleneck), and decoder (2,004,000 parameters with 4 transformer layers and 8 attention heads), and initializing the Exponential Moving Average (EMA) tracker with decay factor $\beta = 0.999$ to maintain running estimates of latent space mean $\boldsymbol{\mu}_{EMA}$ and standard deviation $\boldsymbol{\sigma}_{EMA}$ for stable sequence generation during and after training.

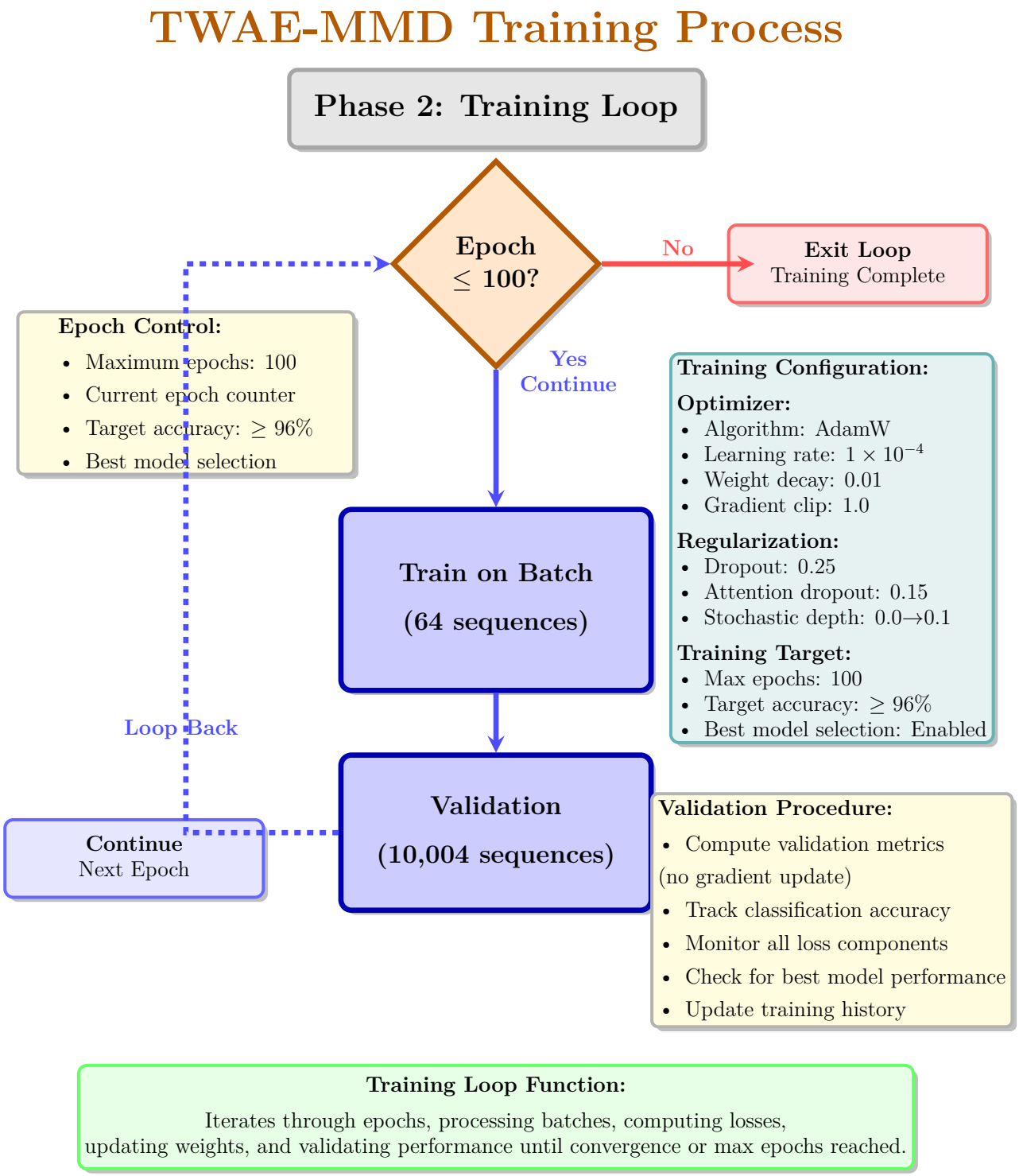

Figure 2.10: TWAE-MMD Training Process Phase 2: Training Loop. The training loop phase implements the iterative optimization procedure controlling epoch progression and batch processing. The loop executes for a maximum of 100 epochs target accuracy ($\geq 96\%$). Each iteration processes 64 training batches through forward pass (encoder $\rightarrow$ latent space $\rightarrow$ decoder), computes composite loss from four components (classification, reconstruction, MMD, Wasserstein), performs backward pass to compute gradients via backpropagation, and updates 4,127,137 model parameters using AdamW optimizer with learning rate 1×10^{-4}, weight decay 0.01, and gradient clipping with maximum norm 1.0. Following batch processing, validation is performed on 10,004 sequences without gradient updates to monitor classification accuracy and loss components. The loop continues until maximum epochs are reached or target accuracy is achieved, with training history tracked across all epochs.

Phase 2: Epoch-Level Training Loop Control

The epoch-level training loop uses the iterative training process that controls the progression of training across multiple passes through the dataset. The loop uses a decision structure that evaluates whether the current epoch number remains within the maximum epoch limit (100 epochs), which allows continuation of training or termination based on convergence criteria. Each epoch iteration includes four steps: batch-level training on 626 batches of 64 sequences each, validation on 78 batches of 128 sequences each, latent space quality assessment, and per-epoch sequence generation.

The training step within each epoch processes batches sequentially, running forward pass through the model structure (encoder → latent space → decoder and classifier), calculating the composite loss function $\mathcal{L}_{total} = \mathcal{L}_{cls} + \mathcal{L}_{rec} + 100 \cdot \mathcal{L}_{MMD} + 10 \cdot \mathcal{L}_{W}$ from four components, doing backward pass to compute gradients for all 4,127,137 parameters, and running weight updates via the AdamW optimizer with learning rate $\eta = 10^{-4}$ and weight decay $\lambda_{wd} = 0.01$. Gradient clipping with maximum norm 1.0 is used before weight updates to prevent gradient explosion, particularly during early training epochs when loss magnitudes may be large.

The validation step evaluates model performance on held-out data without calculating gradients, calculating classification accuracy and loss components to monitor generalization capability. The validation accuracy is the primary metric for model selection, with checkpoints saved whenever current validation accuracy exceeds the historical best. Initiating final sequence generation and training termination. The loop continues iterating until either the maximum epoch limit is reached or the target accuracy threshold is achieved, with training history logged at each epoch for subsequent analysis. The training loop control diagram (Figure 2.10) illustrates the epoch-level control flow including decision points, batch handling, validation, and loop continuation logic.

Phase 3: Batch-Level Forward-Backward Pass

The batch-level forward-backward pass uses the core optimization cycle that updates model parameters based on prediction errors. The forward pass processes input batches with shape $[B, L] = [64, 37]$ through the model structure, beginning with the transformer encoder that maps token sequences to 256-dimensional pooled representations, continuing through the latent space manager that compresses representations to 128-dimensional latent codes $\mathbf{z} \in \mathbb{R}^{128}$ with tanh activation bounding values to $[-1, 1]^{128}$, proceeding through the latent decoder that expands codes to reshaped representations $(37, 25)$ suitable for sequence reconstruction, and concluding with parallel pathways through the transformer decoder producing vocabulary logits $[64, 36, 25]$ and the classification head producing class logits $[64, 2]$.

Four loss components are calculated from the forward pass outputs to quantify prediction errors across multiple objectives. The classification loss $\mathcal{L}_{cls} = -\frac{1}{B}\sum_{i=1}^{B}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$ uses binary cross-entropy to measure discrepancy between predicted class probabilities $\hat{y}_i$ and ground-truth labels $y_i \in \{0, 1\}$, with weight 1.0 emphasizing classification accuracy. The reconstruction loss $\mathcal{L}_{rec} = -\frac{1}{B \cdot L}\sum_{i=1}^{B}\sum_{t=1}^{L} \log p(x_{i,t}|\mathbf{z}_i)$ uses sparse categorical cross-entropy to measure token-level prediction errors, with weight 1.0 encouraging faithful sequence reconstruction through the latent bottleneck. The Maximum Mean Discrepancy loss $\mathcal{L}_{MMD}$ uses the radial basis function kernel $k(\mathbf{z}_i, \mathbf{z}_j) = \exp(-\|\mathbf{z}_i - \mathbf{z}_j\|^2/(2\sigma^2))$ to measure the distance between the empirical latent distribution and the standard Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I}_{128})$, with weight 100.0 strongly regularizing the latent space to enable sampling. The Wasserstein loss $\mathcal{L}_W$ uses Sinkhorn divergence approximation to measure best transport cost between distributions, with weight 10.0 which gives complementary regularization that preserves topological structure.

The loss $\mathcal{L}_{total} = \mathcal{L}_{cls} + \mathcal{L}_{rec} + 100 \cdot \mathcal{L}_{MMD} + 10 \cdot \mathcal{L}_{W}$ aggregates all components with fixed weights determined through hyperparameter optimization to balance competing objectives. The backward pass computes gradients $\nabla_\theta \mathcal{L}_{total}$ for all model parameters θ via automatic differentiation, applying the chain rule to propagate error signals from the loss through all layers to the input embeddings. Gradient clipping with maximum norm 1.0 is used: $\nabla_\theta \leftarrow \nabla_\theta / \max(1, \|\nabla_\theta\|_2)$, rescaling gradients that exceed the threshold

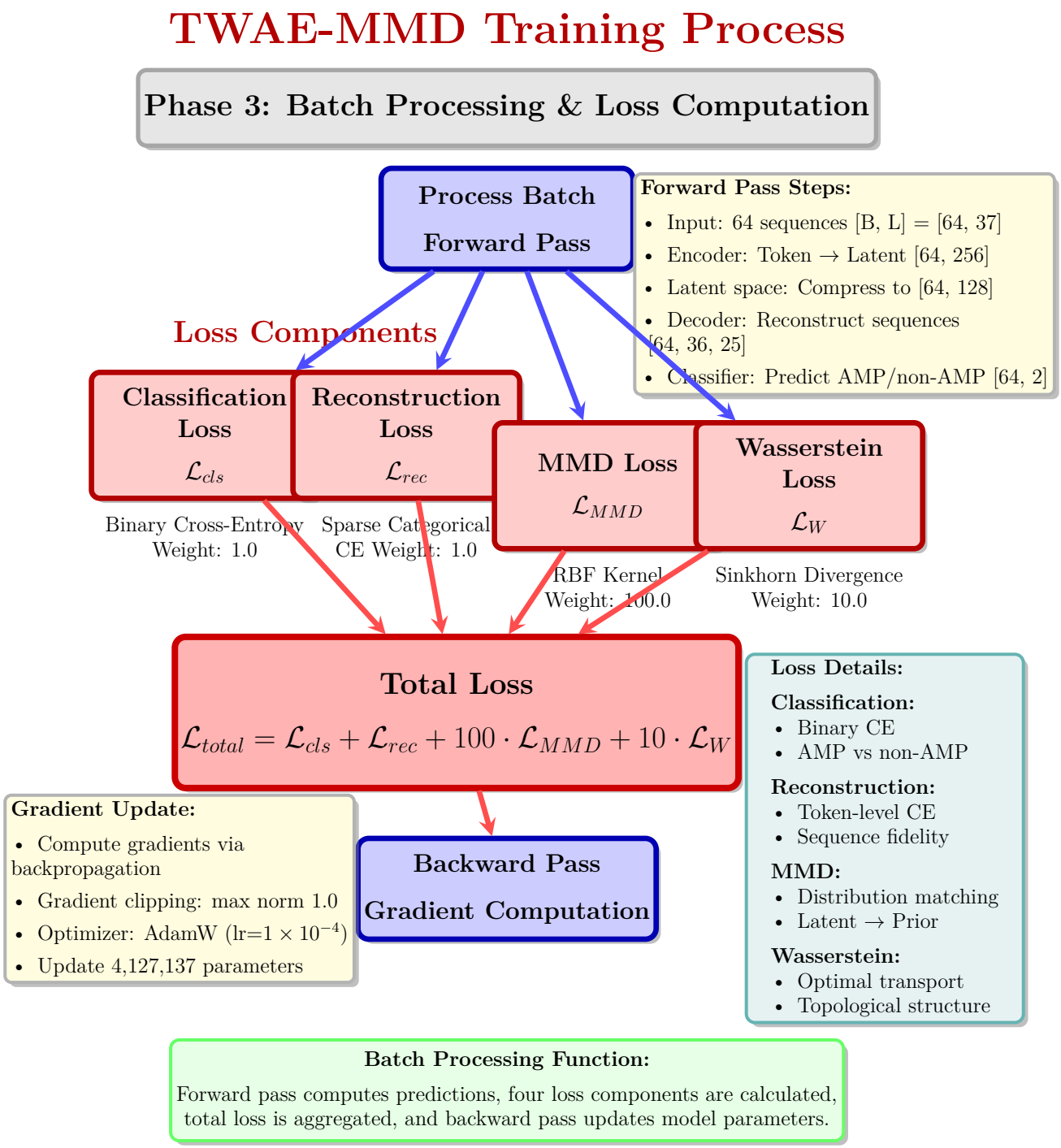

Figure 2.11: TWAE-MMD Training Process Phase 3: Batch Processing and Loss Computation. The batch processing phase implements the forward-backward pass cycle for parameter optimization. The forward pass processes batches of 64 sequences with shape [B, L] = [64, 36] through the complete model architecture: encoder transforms token sequences to 256-dimensional representations, latent space manager compresses to 128-dimensional codes, decoder reconstructs sequences to vocabulary logits [64, 36, 25], and classifier predicts binary labels [64, 2]. Four loss components are computed: classification loss $\mathcal{L}_{cls}$ using binary cross-entropy with weight 1.0, reconstruction loss $\mathcal{L}_{rec}$ using sparse categorical cross-entropy with weight 1.0, Maximum Mean Discrepancy loss $\mathcal{L}_{MMD}$ using RBF kernel with weight 100.0 measuring latent distribution divergence from standard Gaussian prior, and Wasserstein loss $\mathcal{L}_{W}$ using Sinkhorn divergence with weight 10.0 measuring optimal transport cost. The total loss $\mathcal{L}_{total} = \mathcal{L}_{cls} + \mathcal{L}_{rec} + 100 \cdot \mathcal{L}_{MMD} + 10 \cdot \mathcal{L}_{W}$ is minimized through backward pass computing gradients via backpropagation, followed by gradient clipping (max norm 1.0) and parameter update via AdamW optimizer.

to prevent numerical instability. The AdamW optimizer updates parameters using the clipped gradients: $\theta \leftarrow \theta - \eta(\mathbf{m}_t/(\sqrt{\mathbf{v}_t} + \epsilon) + \lambda_{wd}\theta)$, where $\mathbf{m}_t$ and $\mathbf{v}_t$ are the first and second moment estimates, $\eta = 10^{-4}$ is the learning rate, $\epsilon = 10^{-8}$ is a numerical stability constant, and $\lambda_{wd} = 0.01$ is the weight decay coefficient. The batch handling diagram (Figure 2.11) illustrates the forward-backward cycle including calculating loss and weight updates.

3.5.3 EMA-Based Sampling Strategy

The EMA-based sampling strategy allows generation of new antimicrobial peptide sequences by sampling latent codes from the learned EMA distribution and decoding them through the transformer decoder [36] [37]. The sampling procedure consists of four steps: (1) sample latent codes from the EMA distribution $\mathbf{z}_{\text{new}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{EMA}}, \text{diag}(\boldsymbol{\sigma}^2_{\text{EMA}}))$ using the saved EMA statistics [36] [37], (2) decode latent codes through the latent decoder ($128 \rightarrow 256 \rightarrow 512 \rightarrow 925$, reshape to (37, 25)) and transformer decoder (4 layers, parallel generation), (3) sample tokens from the decoder output logits using categorical sampling with optional temperature scaling, (4) convert token IDs to amino acid sequences using the tokenizer's decode function, stopping at `[PAD]` or `[SEP]` tokens.

During training, 50 sequences are generated per epoch using this procedure to monitor generation quality and diversity. At the end of training (when validation accuracy $\geq 96\%$), 100 final sequences are generated using the best checkpoint's EMA statistics [36] [37]. The EMA-based approach gives stable sampling without requiring explicit constraint enforcement or optimization, relying on the learned latent distribution to produce biologically plausible sequences [36] [37].

Phase 4: EMA Statistics Update and Sequence Generation

The EMA statistics update and sequence generation phase maintains running estimates of the latent space distribution and produces new antimicrobial peptide sequences throughout training. after each training batch's forward pass, latent codes with shape $[B, d] = [64, 128]$ are extracted from the latent space manager's encoder output, representing compressed sequence representations in the 128-dimensional latent space. Batch-level statistics are calculated as the mean $\boldsymbol{\mu}_{batch} = \frac{1}{B}\sum_{i=1}^{B}\mathbf{z}_i$ and standard deviation $\boldsymbol{\sigma}_{batch} = \sqrt{\frac{1}{B}\sum_{i=1}^{B}(\mathbf{z}_i - \boldsymbol{\mu}_{batch})^2}$, which gives empirical estimates of the current batch's latent distribution.

The EMA tracker updates running estimates using exponentially-weighted moving averages with decay factor $\beta = 0.999$, using the update equations $\boldsymbol{\mu}_{EMA} \leftarrow 0.999 \cdot \boldsymbol{\mu}_{EMA} + 0.001 \cdot \boldsymbol{\mu}_{batch}$ and $\boldsymbol{\sigma}_{EMA} \leftarrow 0.999 \cdot \boldsymbol{\sigma}_{EMA} + 0.001 \cdot \boldsymbol{\sigma}_{batch}$. The decay factor of 0.999 corresponds to an effective window size of approximately 1000 batches, which gives stable statistics that adapt slowly to changing distributions while filtering batch-level noise. The EMA statistics converge to stable values after handling approximately 3000–5000 batches (5–8 epochs), after which they provide consistent sampling distributions for generation.

Every epoch, 50 new AMP sequences are generated using the current EMA statistics to monitor generation quality and diversity throughout training. The generation procedure samples latent codes from the EMA distribution $\mathbf{z}_{new} \sim \mathcal{N}(\boldsymbol{\mu}_{EMA}, \text{diag}(\boldsymbol{\sigma}^2_{EMA}))$ using the multivariate Gaussian distribution with diagonal covariance matrix, decodes latent codes through the latent decoder ($128 \rightarrow 256 \rightarrow 512 \rightarrow 925$, reshape to $(37, 25)$) and transformer decoder (4 layers, parallel generation), samples tokens from the output logits using categorical sampling with temperature 1.0, and converts token IDs to amino acid sequences using the tokenizer's decode function. Generated sequences are analyzed for quality metrics including uniqueness ratio (proportion of distinct sequences), length distribution (mean, standard deviation, range), amino acid composition (frequency of each residue), and diversity score (average pairwise Levenshtein distance). The EMA update and generation diagram (Figure 2.12) illustrates the EMA update procedure and generation workflow.

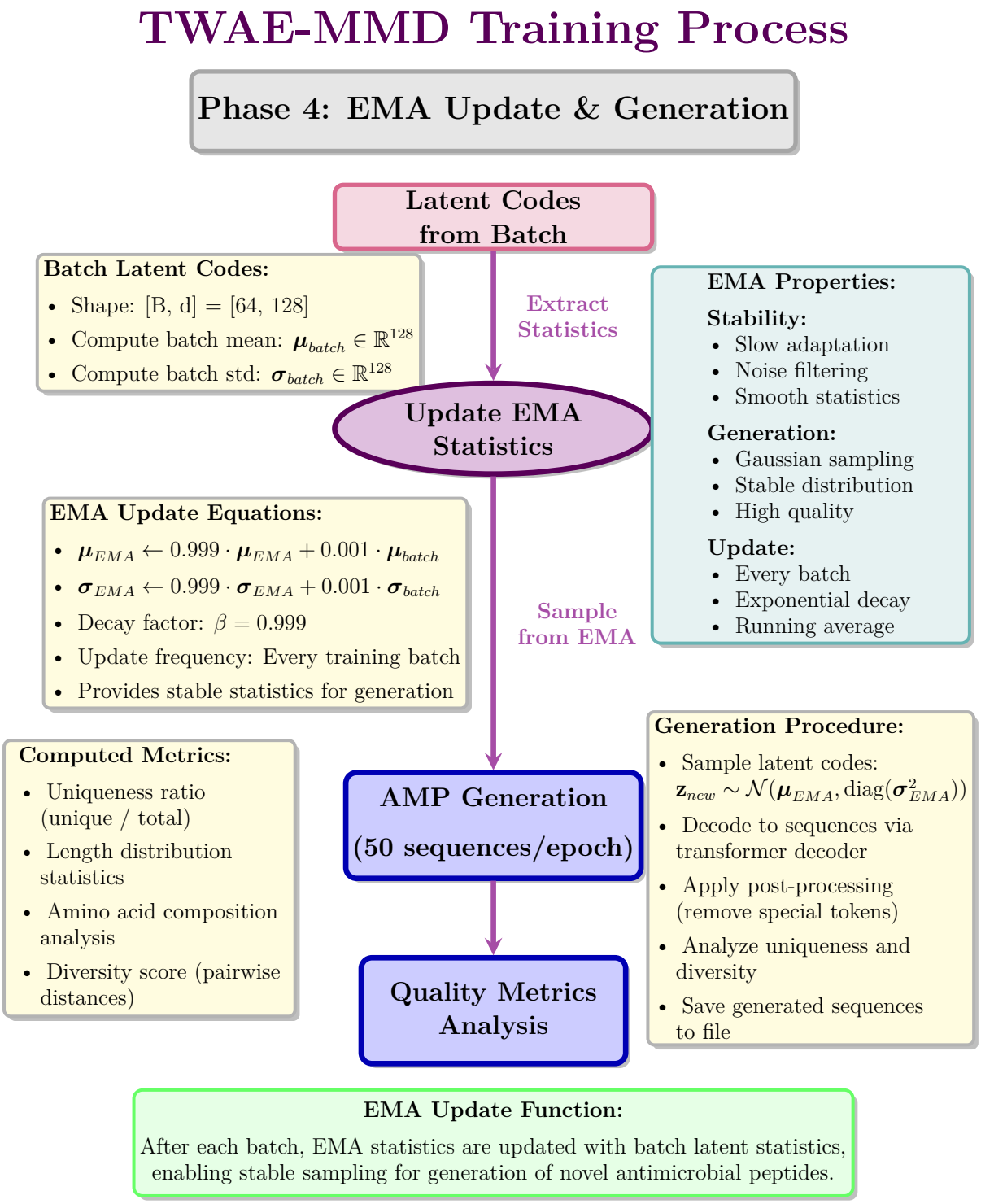

Figure 2.12: TWAE-MMD Training Process Phase 4: EMA Update and Generation. The EMA update phase maintains running statistics of the latent space distribution and generates novel antimicrobial peptide sequences. Following each training batch, latent codes with shape [B, d] = [64, 128] are extracted from the latent space manager, and batch-level statistics (mean $\boldsymbol{\mu}_{batch} \in \mathbb{R}^{128}$ and standard deviation $\boldsymbol{\sigma}_{batch} \in \mathbb{R}^{128}$) are computed. The EMA tracker updates running estimates using exponentially-weighted moving averages with decay factor $\beta = 0.999$: $\boldsymbol{\mu}_{EMA} \leftarrow 0.999 \cdot \boldsymbol{\mu}_{EMA} + 0.001 \cdot \boldsymbol{\mu}_{batch}$ and $\boldsymbol{\sigma}_{EMA} \leftarrow 0.999 \cdot \boldsymbol{\sigma}_{EMA} + 0.001 \cdot \boldsymbol{\sigma}_{batch}$, providing stable statistics that filter batch-level noise while adapting to evolving latent distributions. Every epoch, 50 novel AMP sequences are generated by sampling latent codes from the EMA distribution $\mathbf{z}_{new} \sim \mathcal{N}(\boldsymbol{\mu}_{EMA}, \text{diag}(\boldsymbol{\sigma}^2_{EMA}))$, decoding through the transformer decoder, and analyzing quality metrics including uniqueness ratio, length distribution, amino acid composition, and diversity score.

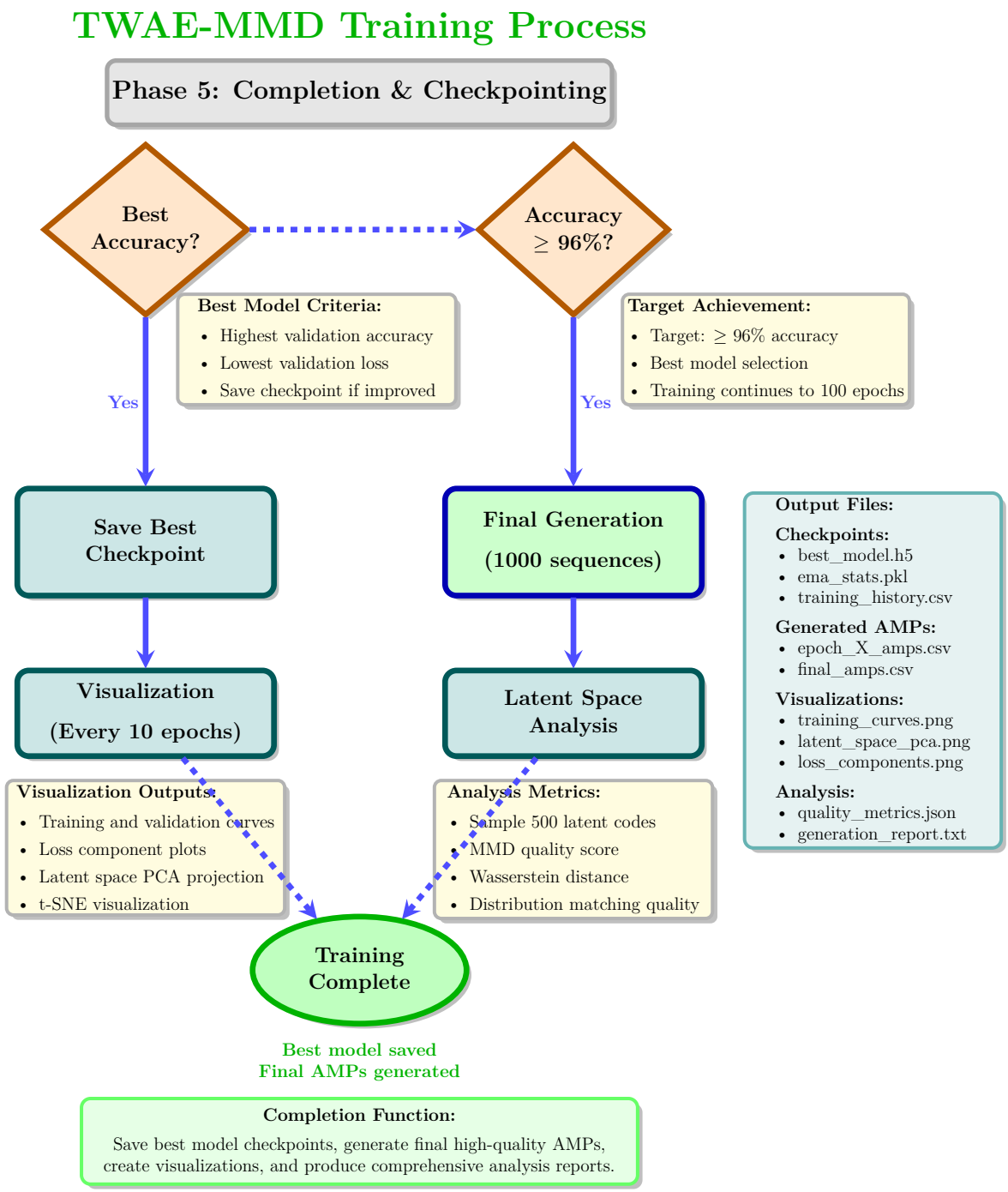

Figure 2.13: TWAE-MMD Training Process Phase 5: Completion and Checkpointing. The completion phase implements model checkpointing, final generation, and result visualization. Following each epoch's validation, the system evaluates whether the current model achieves the best validation accuracy observed during training; if so, a checkpoint is saved containing model weights (4,127,137 parameters), optimizer state (AdamW), EMA statistics ($\boldsymbol{\mu}_{EMA}, \boldsymbol{\sigma}_{EMA}$), training history, epoch number, and best accuracy. Simultaneously, the system checks whether target accuracy ($\geq 96\%$) has been achieved; upon reaching this threshold, final generation is triggered to produce 1000 high-quality novel AMP sequences using the best EMA statistics, followed by comprehensive quality analysis including physicochemical property computation. Every 10 epochs, visualization outputs are generated including training and validation curves, loss component plots, latent space PCA projections, and t-SNE visualizations. Additionally, latent space analysis is performed by sampling 500 latent codes and computing distribution matching quality metrics (MMD score, Wasserstein distance, energy distance). Training completes when either maximum epochs (100) are reached or target accuracy is achieved, producing output files including best model checkpoint (best_model.h5), EMA statistics (ema_stats.pkl), training history (training_history.csv), per-epoch generated sequences (epoch_X_amps.csv), final generated sequences (final_amps.csv), and visualization plots.

Phase 5: Training Completion and Model Checkpointing

The training completion and model checkpointing phase uses model persistence, final sequence generation, and result visualization. after each epoch's validation phase, the system evaluates whether the current validation accuracy exceeds the historical best accuracy observed during training; if this condition is satisfied, a checkpoint is saved to persistent storage. Each checkpoint has three components: model weights stored in HDF5 format (`best_model.h5`) containing all 4,127,137 trainable parameters including embedding matrices, transformer layer weights, latent space dense layers, and classification head weights; EMA statistics stored in pickle format (`ema_stats.pkl`) containing the 128-dimensional mean vector $\boldsymbol{\mu}_{EMA}$ and standard deviation vector $\boldsymbol{\sigma}_{EMA}$; and training configuration stored in JSON format (`training_config.json`) containing hyperparameters, optimizer state, epoch number, best validation accuracy, and training history.

Simultaneously with checkpoint evaluation, the system assesses whether the target validation accuracy threshold ($\geq 96\%$) has been achieved. Upon reaching this threshold, final generation is triggered to produce 1000 high-quality new AMP sequences using the best checkpoint's EMA statistics. The final generation procedure loads the best checkpoint from disk, extracts the saved EMA statistics $\boldsymbol{\mu}_{EMA}$ and $\boldsymbol{\sigma}_{EMA}$, samples 100 latent codes from the EMA distribution $\mathbf{z}_{new} \sim \mathcal{N}(\boldsymbol{\mu}_{EMA}, \text{diag}(\boldsymbol{\sigma}^2_{EMA}))$, decodes all latent codes in parallel through the transformer decoder, and saves the generated sequences to CSV format (`final_amps.csv`).

Every 10 epochs, visualization outputs are generated to monitor training progress and latent space evolution. The visualization pipeline produces four primary outputs: training and validation curves plotting loss components and classification accuracy across epochs, loss component plots showing the individual contributions of $\mathcal{L}_{cls}$, $\mathcal{L}_{rec}$, $\mathcal{L}_{MMD}$, and $\mathcal{L}_W$ over time, latent space PCA projections reducing the 128-dimensional latent codes to 2D representations colored by AMP/non-AMP labels, and t-SNE visualizations which gives nonlinear dimensionality reduction to reveal cluster structure in the latent space. Additionally, latent space analysis happens by sampling 500 latent codes from the training set, calculating distribution matching quality metrics including MMD score measuring divergence from the standard Gaussian prior, Wasserstein distance measuring best transport cost, and energy distance which gives an alternative divergence measure.

Training completes when the maximum epoch limit (100 epochs) is reached. Upon completion, the system produces a set of output files: best model checkpoint (`best_model.h5`), EMA statistics (`ema_stats.pkl`), training history (`.csv`) containing per-epoch metrics, per-epoch generated sequences (`epoch_X_amps.csv`) for X $\in$ {1, 2, .., N}, final generated sequences (`.csv`), visualization plots (`.png`), quality metrics (`.json`) containing uniqueness ratios and diversity scores, and generation report (`.txt`) summarizing training outcomes and generated sequence characteristics. The training completion diagram (Figure 2.13) illustrates the checkpointing, final generation, and visualization workflow.

3.6 TWAE-MMD Model Architecture with LSBO and Constraints

3.6.1 Architectural Overview

The TWAE-MMD architecture with Latent Space Bayesian Optimization (LSBO) [38] [39] and biological constraints extends the base architecture described in Section 3.4 with three additional components: (1) a Gaussian Process (GP) surrogate model for latent space quality prediction [39], (2) a biological constraint system enforcing six AMP-specific criteria, and (3) a neural property scorer using ten weighted quality factors. The core transformer encoder, latent space manager, and transformer decoder remain identical to the EMA version, maintaining the same 4,127,137 parameters and architectural specifications. The LSBO system, which is explicitly shown in the architecture diagrams (Figure 2.14, Figure 2.15, Figure 2.16, Figure 2.17, Figure 2.18, Figure 2.19, Figure 2.20), works during generation to identify high-quality regions in the latent space and sample from them with constraint enforcement.

The input data preparation phase for LSBO-guided generation maintains identical specifications to the

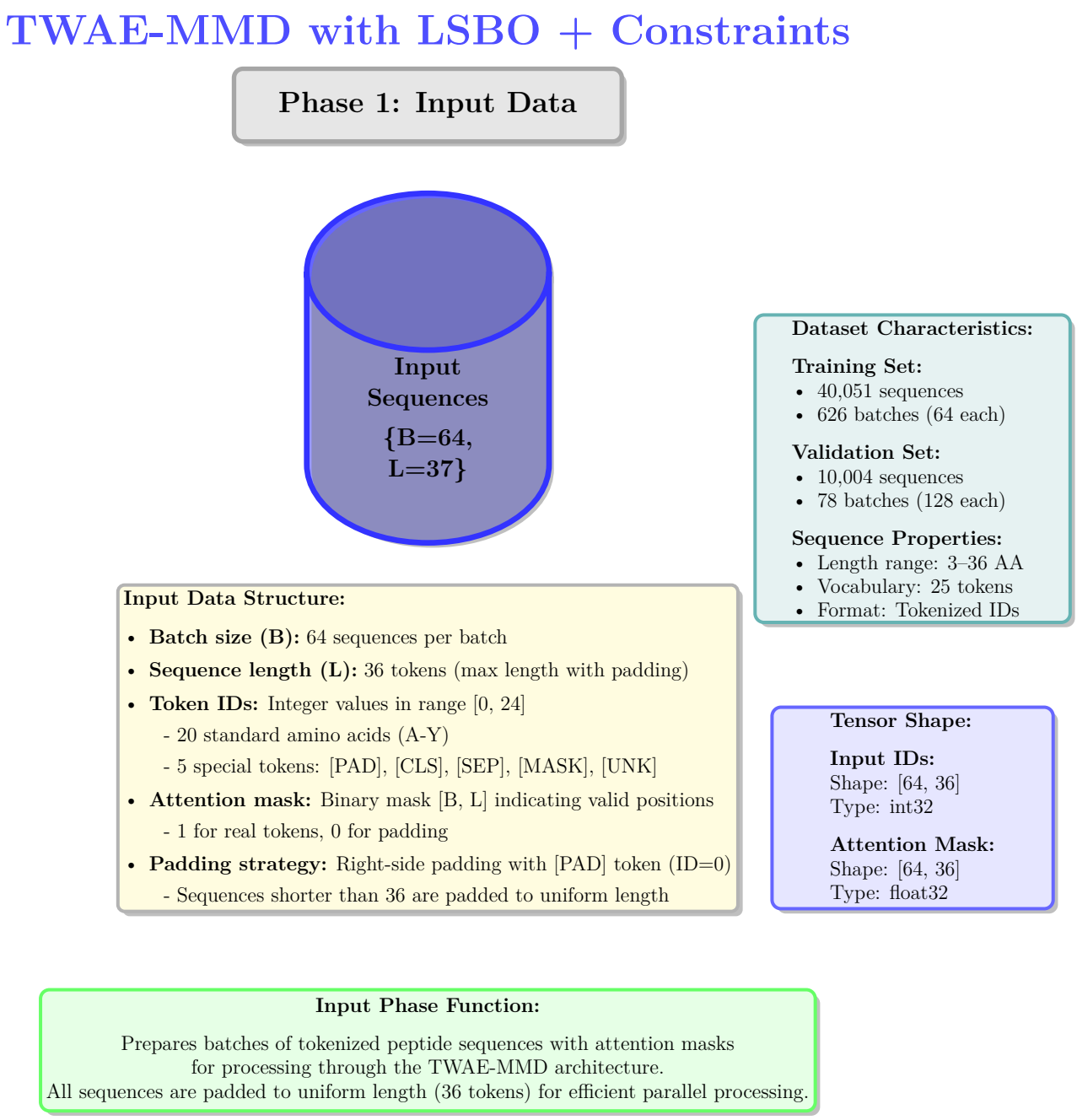

Figure 2.14: TWAE-MMD with LSBO Phase 1: Input Data Preparation. The input phase prepares batches of tokenized antimicrobial peptide sequences for processing through the TWAE-MMD architecture with LSBO optimization. Input sequences are organized into batches of size 64 with maximum sequence length 37 tokens, where each token is represented as an integer ID in the range [0, 24] corresponding to 20 standard amino acids and 5 special tokens ([PAD], [CLS], [SEP], [MASK], [UNK]). Attention masks with shape [64, 36] indicate valid token positions (1 for real tokens, 0 for padding), enabling the transformer architecture to distinguish between actual sequence content and padding. The training dataset comprises 40,051 sequences organized into 626 batches, while the validation dataset contains 10,004 sequences organized into 78 batches with batch size 128, both maintaining 50% class balance between AMP and non-AMP sequences. Sequences shorter than 36 amino acids are right-padded with [PAD] tokens to ensure uniform tensor dimensions for efficient parallel processing on GPU hardware.

EMA training pipeline described in Section 3.3. Input sequences are organized into batches of size 64 with maximum sequence length 37 tokens, where each token is represented as an integer identifier in the range [0, 24] corresponding to 20 standard amino acids and 5 special tokens ([PAD], [CLS], [SEP], [MASK], [UNK]). Attention masks with shape [64, 37] indicate valid token positions (1 for actual sequence content, 0 for padding), which lets the transformer encoder distinguish between informative positions and padding artifacts. The training dataset has 40,051 sequences organized into 626 batches, while the validation dataset has 10,004 sequences organized into 78 batches with batch size 128, both maintaining 50% class balance between AMP and non-AMP sequences. Sequences shorter than 36 amino acids are right-padded with [PAD] tokens (ID=0) to ensure uniform tensor dimensions [64, 37] for parallel handling on GPU hardware. The input step (Figure 2.14) illustrates the batch structure, tokenization scheme, attention masking strategy, and dataset organization that enable training and generation with the TWAE-MMD architecture.

The LSBO-guided approach represents a paradigm shift from random sampling to targeted exploration of high-quality latent regions. Rather than sampling latent codes from a fixed Gaussian distribution (as in EMA), the LSBO system maintains a database of discovered high-quality regions (HQR) with associated quality scores, uses a Gaussian Process to model the quality landscape of the latent space, uses acquisition functions (Expected Improvement, Upper Confidence Bound, Probability of Improvement) to guide exploration toward promising regions, and enforces biological constraints to ensure 100% validity of generated sequences. This approach achieves 10–20× efficiency improvement compared to random sampling while maintaining biological plausibility.

The TWAE-MMD architecture uses an end-to-end autoencoder with transformer-based sequence handling and dense bottleneck for continuous latent space representation, comprising 14 sequential layers organized into five functional components. The handling pipeline begins with input sequences organized as batches [B=64, L=37] containing tokenized amino acid sequences represented as integer identifiers in the vocabulary range [0, 24], which undergo token embedding (25×256 matrix, 6,400 parameters) and positional encoding ([36, 256] learned parameters, 9,472 parameters) before entering the transformer encoder. The transformer encoder uses 4 identical layers, each containing multi-head attention mechanisms with 8 heads operating on 32-dimensional subspaces (attention weights computed as $softmax(\mathrm{QK}^T / \sqrt{32})$), position-wise feed-forward networks with architecture 256→1024→256 employing GELU activation, layer normalization for stable gradient flow, and residual connections preserving information across layers, culminating in attention pooling that aggregates the sequence of token representations [64, 37, 256] into fixed-length sequence representations [64, 256] through learned weighted averaging, with the encoder containing 1,892,608 trainable parameters distributed across attention mechanisms (approximately 200,000 parameters per layer), feed-forward networks (approximately 270,000 parameters per layer), and normalization components. The latent space manager compresses these 256-dimensional representations through a 3-layer latent encoder with progressive dimensionality reduction (256→512→256→128), where the first two layers employ GELU activation, batch normalization, and dropout (rate 0.25) for regularization, and the final layer applies tanh activation to bound latent codes within the hypercube $[-1, 1]^{128}$, producing 128-dimensional latent vectors $\mathbf{z} \in \mathbb{R}^{128}$ that serve as compressed continuous representations with standard Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I}_{128})$ which allows sampling for generation. The latent space manager subsequently expands these 128-dimensional codes through a 3-layer latent decoder with progressive dimensionality expansion (128→256→512→925), where each hidden layer uses GELU activation and batch normalization, and the final 925-dimensional output undergoes reshape operation to (37, 25) representing 36 sequence positions with 25 vocabulary logits per position, which gives initial predictions for the transformer decoder with the latent space manager containing 230,529 trainable parameters distributed across encoder (131,072 + 131,072 + 32,768 parameters for weight matrices) and decoder (32,768 + 131,072 + 473,600 parameters) pathways. The transformer decoder uses 4 identical layers handling the reshaped latent representations through seattention, cross-attention layers with queries from decoder state and keys/values from encoder output which allows conditioning on input

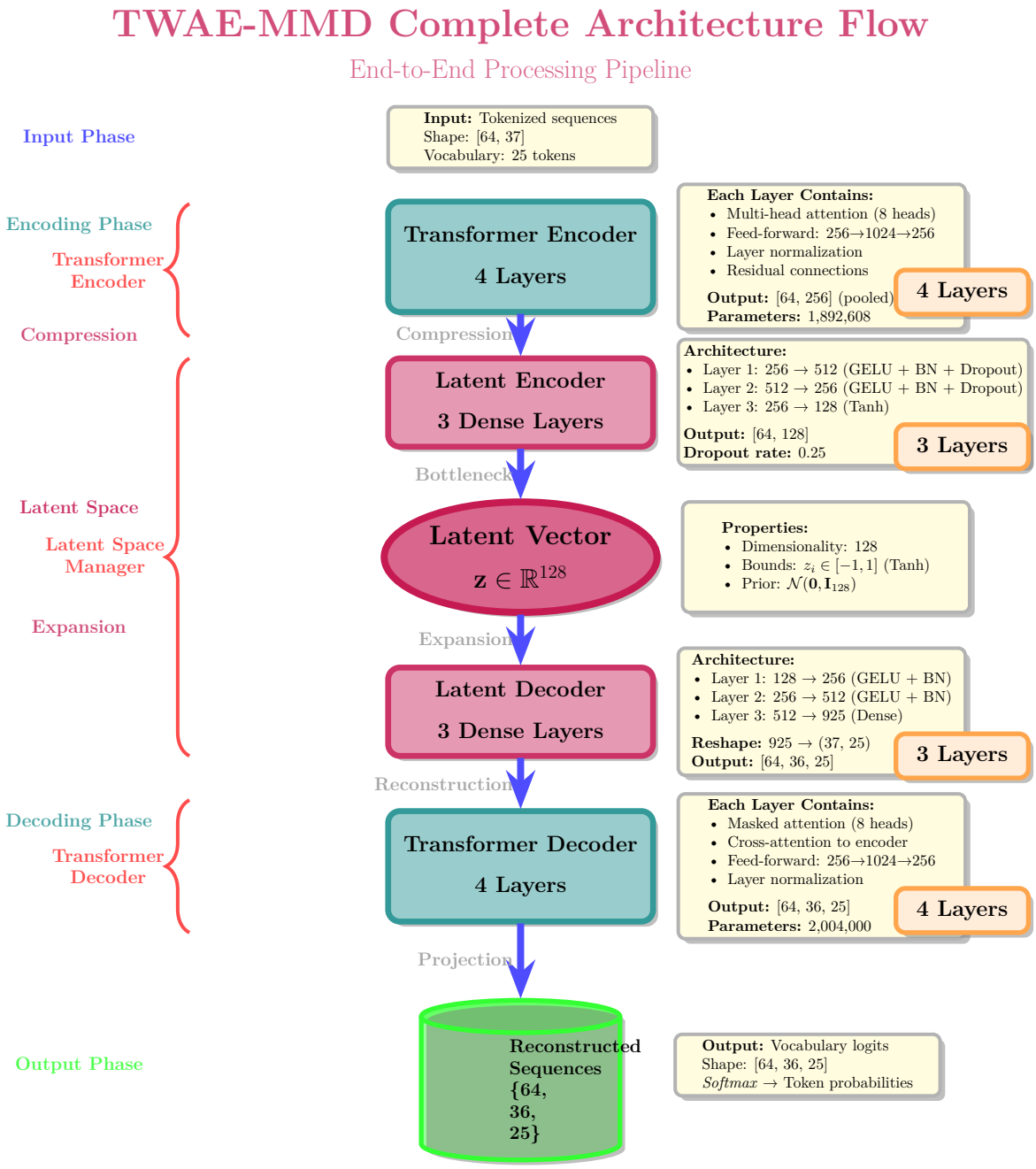

Figure 2.15: TWAE-MMD Complete Architecture Flow. The complete end-to-end processing pipeline illustrates the sequential transformation of input sequences through five major components. The transformer encoder processes tokenized input sequences [64, 37] through embedding and attention pooling to produce fixed-length representations [64, 256]. The latent encoder compresses the 256-dimensional representations into 128-dimensional latent codes through progressive dimensionality reduction (256→512→256→128) with final tanh activation bounding codes within $[-1, 1]^{128}$. The latent vector $\mathbf{z} \in \mathbb{R}^{128}$ serves as the information bottleneck with Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I}_{128})$ enabling both reconstruction and generation capabilities. The latent decoder expands the 128-dimensional codes back to decoder input format through progressive dimensionality expansion with final reshape operation converting the 925-dimensional vector to (37, 25) representing 36 positions with 25 vocabulary logits per position. The transformer decoder (4 layers with attention mechanism, cross-attention to encoder, and feed-forward networks) reconstructs sequences from the latent representation through parallel generation, producing output logits [64, 36, 25] representing probability distributions over the 25-token vocabulary for each of 36 positions. The complete architecture contains 14 layers (4+3+3+4) with 4,127,137 total parameters distributed across encoder (1,892,608), latent space manager (230,529), and decoder (2,004,000) components, implementing an end-to-end autoencoder with transformer-based sequence processing and dense bottleneck for continuous latent space representation.

context, position-wise feed-forward networks (256→1024→256, GELU), and output projection mapping 256-dimensional representations to 25-dimensional vocabulary logits, producing reconstruction logits [64, 36, 25] representing probability distributions over the vocabulary for each position, with the decoder containing 2,004,000 trainable parameters distributed across 4 layers (approximately 501,000 parameters per layer including masked attention, cross-attention, feed-forward networks, and layer normalization). The architecture flow (Figure 2.15) illustrates the sequential transformation of input sequences through transformer encoding (4 layers, 1.9M parameters), latent compression (3 layers, compression pathway of latent space manager), information bottleneck (128-dimensional latent vector with tanh bounding), latent expansion (3 layers, expansion pathway of latent space manager), and transformer decoding (4 layers, 2.0M parameters), using a 14-layer end-to-end architecture with 4,127,137 parameters that allows both sequence reconstruction through the forward pass and sequence generation through sampling from the learned latent space distribution.

3.6.2 Gaussian Process Surrogate Model

The Gaussian Process (GP) surrogate model [39] gives a probabilistic framework for predicting sequence quality scores from latent codes without requiring explicit decoding and scoring. The GP is defined by a mean function $m(\mathbf{z})$ (typically zero) and a covariance function (kernel) $k(\mathbf{z}_i, \mathbf{z}_j)$ that quantifies the similarity between latent codes. The Matérn kernel with smoothness parameter $\nu = 2.5$ is employed:

$$k(\mathbf{z}_i, \mathbf{z}_j) = \sigma^2 \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right) \tag{2.5}$$

where $r = \|\mathbf{z}_i - \mathbf{z}_j\|_2$ is the Euclidean distance, σ^2 is the signal variance, and ℓ is the length scale parameter. The Matérn kernel with $\nu = 2.5$ gives twice-differentiable sample paths, which allows smooth interpolation of quality scores across the latent space.

Given a set of observed latent codes $\{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ with corresponding quality scores $\{s_1, \ldots, s_N\}$, the GP posterior distribution at a new latent code $\mathbf{z}_*$ is Gaussian with mean $\mu(\mathbf{z}_*) = \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{s}$ and variance $\sigma^2(\mathbf{z}_*) = k(\mathbf{z}_*, \mathbf{z}_*) - \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*$, where $\mathbf{K}$ is the kernel matrix with entries $K_{ij} = k(\mathbf{z}_i, \mathbf{z}_j)$, $\mathbf{k}_*$ is the kernel vector with entries $k(\mathbf{z}_*, \mathbf{z}_i)$, $\mathbf{s} = [s_1, \ldots, s_N]^T$ is the vector of observed scores, and σ_n^2 is the noise variance.

The GP is fitted using 20 initial random samples from the latent space, decoded to sequences, scored using the neural property scorer, and added to the GP training set. The GP hyperparameters (σ^2, ℓ, σ_n^2) are optimized by maximizing the log marginal likelihood. The fitted GP is then used to guide exploration through acquisition functions.

The sequence encoding phase transforms variable-length tokenized sequences into fixed-length continuous representations using the 4-layer transformer encoder architecture described in Section 3.4.2. The encoding process has six steps that progressively refine sequence representations. Token embedding maps each token identifier to a 256-dimensional learned vector using an embedding matrix of size 25×256 (6,400 parameters), which gives dense continuous representations of discrete amino acid symbols. Learned positional encodings with shape [36, 256] (9,472 parameters) are added element-wise to token embeddings, injecting absolute position information that lets the model distinguish between identical amino acids at different sequence positions. Multi-head attention with 8 heads (32 dimensions per head) computes contextualized representations by attending to all positions in the input sequence, with attention weights computed as $\mathit{softmax}(\mathrm{QK}^T / \sqrt{32})$ which lets each position aggregate information from relevant context positions. Position-wise feed-forward networks with architecture 256→1024→256 and GELU activation apply non-linear transformations independently to each position, expanding representations to higher-dimensional space (1024) for complex feature extraction before projecting back to 256 dimensions. Regularization techniques including dropout (rate 0.25 for feed-forward layers, rate 0.15 for attention weights), layer scale parameters for stable gradient flow, and stochastic depth (random layer dropping with survival probability 0.9) prevent overfitting during training. Attention pooling with learnable weights aggregates the sequence of 256-dimensional token repre-

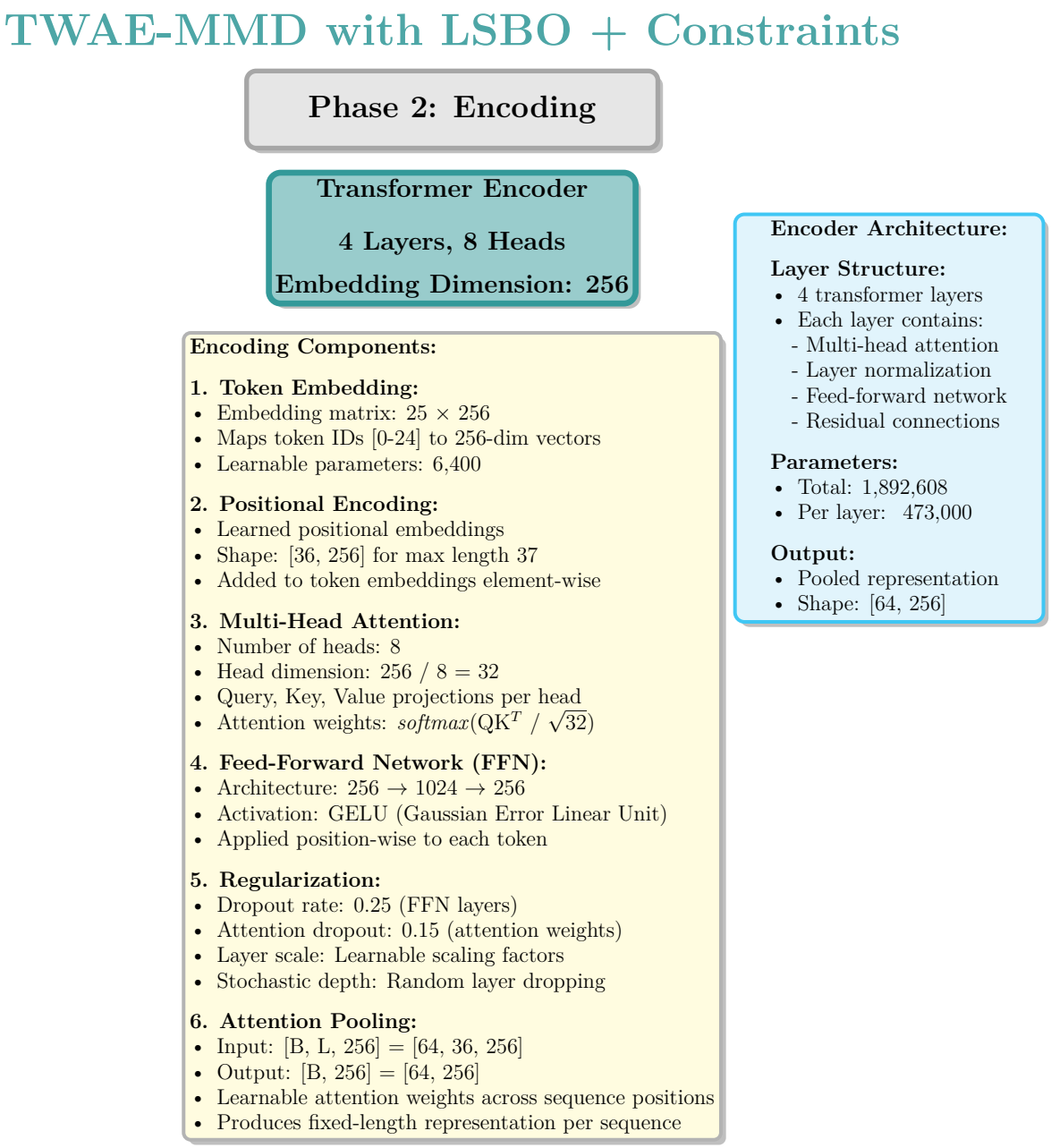

Figure 2.16: TWAE-MMD with LSBO Phase 2: Sequence Encoding. The encoding phase transforms variable-length tokenized sequences into fixed-length continuous representations using a 4-layer transformer encoder architecture. The encoding process comprises six sequential operations: (1) token embedding maps each token ID to a 256-dimensional learned vector using an embedding matrix of size 25×256, (2) learned positional encodings with shape [36, 256] are added element-wise to token embeddings to inject sequence position information, (3) multi-head attention with 8 heads (32 dimensions per head) computes contextualized representations by attending to all positions in the input sequence with attention weights computed as $softmax(\mathrm{QK}^T / \sqrt{32})$, (4) position-wise feed-forward networks with architecture 256→1024→256 and GELU activation apply non-linear transformations independently to each position, (5) regularization techniques including dropout (rate 0.25 for feed-forward layers, rate 0.15 for attention weights), layer scale parameters for stable gradient flow, and stochastic depth (random layer dropping) prevent overfitting, and (6) attention pooling with learnable weights aggregates the sequence of 256-dimensional token representations [64, 36, 256] into a single fixed-length representation per sequence [64, 256]. The encoder contains 1,892,608 trainable parameters distributed across 4 transformer layers.

sentations [64, 37, 256] into a single fixed-length representation per sequence [64, 256], calculating a weighted average where weights are learned to emphasize informative positions. The encoder has 1,892,608 trainable parameters distributed across 4 transformer layers, each using multi-head attention (approximately 200,000 parameters per layer), feed-forward networks (approximately 270,000 parameters per layer), and layer normalization. The encoding step (Figure 2.16) illustrates the encoding pipeline from tokenized input to pooled sequence representations that serve as input to the latent space manager.

3.6.3 Acquisition Functions for Exploration

Three acquisition functions [40] are implemented to balance exploration of uncertain regions with exploitation of known high-quality regions:

The Expected Improvement (EI) acquisition function quantifies the expected improvement over the current best observed score f_{best}:

$$\text{EI}(\mathbf{z}) = (\mu(\mathbf{z}) - f_{\text{best}} - \xi)\Phi(Z) + \sigma(\mathbf{z})\phi(Z) \tag{2.6}$$

where $Z = \frac{\mu(\mathbf{z}) - f_{\text{best}} - \xi}{\sigma(\mathbf{z})}$, $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution, $\phi(\cdot)$ is the probability density function, and $\xi \geq 0$ is an exploration parameter (typically 0.01).

The Upper Confidence Bound (UCB) acquisition function balances mean prediction with uncertainty:

$$\text{UCB}(\mathbf{z}) = \mu(\mathbf{z}) + \kappa\sigma(\mathbf{z}) \tag{2.7}$$

where $\kappa > 0$ controls the exploration-exploitation trade-off (typically 2.0–3.0).

The Probability of Improvement (POI) acquisition function quantifies the probability of exceeding the current best score:

$$\text{POI}(\mathbf{z}) = \Phi\left(\frac{\mu(\mathbf{z}) - f_{\text{best}} - \xi}{\sigma(\mathbf{z})}\right) \tag{2.8}$$

The LSBO training process iterates for 100 steps: (1) fit GP to current observations, (2) optimize acquisition function to find the next candidate latent code $\mathbf{z}_*$, (3) decode $\mathbf{z}_*$ to sequence, (4) score sequence using neural property scorer, (5) check biological constraints, (6) if valid and high-scoring, add $(\mathbf{z}_*, s_*)$ to high-quality regions database, (7) add $(\mathbf{z}_*, s_*)$ to GP training set. This iterative process discovers high-quality regions in the latent space with 10–20× efficiency compared to random sampling.

The latent space compression and expansion phase uses the information bottleneck architecture that allows both reconstruction and generation capabilities. The latent encoder compresses 256-dimensional pooled encoder representations into 128-dimensional latent codes through three dense layers with architecture 256→512→256→128, where the first two layers employ GELU activation, batch normalization, and dropout (rate 0.25) to prevent overfitting, and the final layer applies tanh activation to bound latent codes within the hypercube $[-1, 1]^{128}$. The 128-dimensional latent vector $\mathbf{z} \in \mathbb{R}^{128}$ is a compressed continuous representation of the input sequence, with a standard Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I}_{128})$ which allows sampling for generation by which gives a reference distribution from which new latent codes can be drawn. The tanh activation ensures that all latent codes lie within a bounded region, which helps stable optimization and preventing numerical overflow during training. The latent decoder expands the 128-dimensional codes back to decoder input format through three dense layers with architecture 128→256→512→925, where each hidden layer uses GELU activation and batch normalization to stabilize training, and the final 925-dimensional output is reshaped to (37, 25) representing 36 sequence positions with 25 vocabulary logits per position. The reshape operation converts the flat 925-dimensional vector into a 2D tensor [36, 25] that gives initial logits for the transformer decoder, effectively initializing the decoder's prediction of token probabilities at each position. The latent space manager has 230,529 trainable parameters distributed across encoder layers (131,072 +

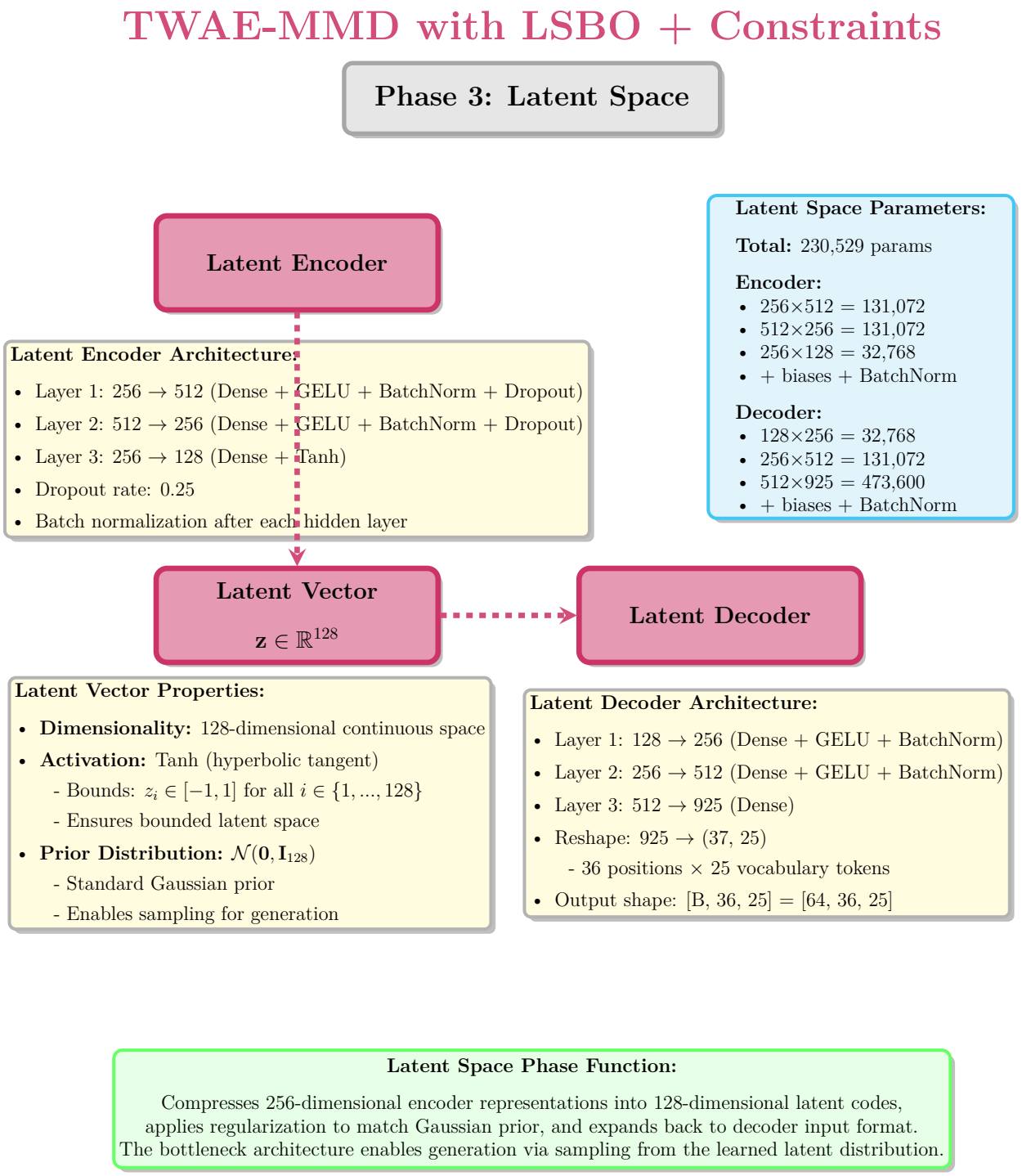

Figure 2.17: TWAE-MMD with LSBO Phase 3: Latent Space Compression and Expansion. The latent space phase implements a bottleneck architecture that compresses 256-dimensional encoder representations into 128-dimensional latent codes and subsequently expands them back to decoder input format. The latent encoder applies three dense layers with architecture 256→512→256→128, where each hidden layer employs GELU activation, batch normalization, and dropout (rate 0.25), and the final layer applies tanh activation to bound latent codes within $[-1, 1]^{128}$. The 128-dimensional latent vector $\mathbf{z} \in \mathbb{R}^{128}$ serves as a compressed continuous representation of the input sequence, with a standard Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I}_{128})$ enabling sampling for generation. The latent decoder expands the 128-dimensional codes through three dense layers with architecture 128→256→512→925, where each hidden layer employs GELU activation and batch normalization, and the final 925-dimensional output is reshaped to $(37, 25)$ representing 36 sequence positions with 25 vocabulary logits per position. The latent space manager contains 230,529 trainable parameters distributed across encoder and decoder layers, implementing the information bottleneck that enables both reconstruction and generation capabilities.

131,072 + 32,768 = 294,912 parameters for weight matrices, plus biases and batch normalization parameters) and decoder layers (32,768 + 131,072 + 473,600 = 637,440 parameters, minus shared components), using the information bottleneck that compresses high-dimensional sequence representations into a low-dimensional continuous space suitable for Bayesian optimization. The latent space phase (Figure 2.17) illustrates the encoder-bottleneck-decoder architecture, the properties of the 128-dimensional latent vector including tanh bounding, and the parameter distribution across compression and expansion pathways.

3.6.4 Biological Constraint System

The biological constraint system enforces six criteria to ensure generated sequences exhibit AMP-like characteristics:

Length constraint: $10 \leq L \leq 36$ amino acids, where L is the sequence length. This range includes the typical length distribution of naturally occurring antimicrobial peptides.

Charge constraint: $+2 \leq Q \leq +9$, where $Q = \sum_i q_i$ is the net charge computed by summing charge contributions (K: +1, R: +1, D: -1, E: -1, H: +0.1). Cationic charge is essential for electrostatic interaction with negatively charged bacterial membranes.

Hydrophobicity constraint: $0.30 \leq H \leq 0.70$, where $H = \frac{1}{L}\sum_i h_i$ is the average hydrophobicity using the Kyte-Doolittle scale. Balanced hydrophobicity allows membrane insertion while maintaining solubility.

Required amino acid constraint: The sequence must contain at least one lysine (K) or arginine (R) residue to ensure cationic character.

Forbidden pattern constraint: The sequence must not contain homopolymeric runs (e.g., “AAA”, “KKK”) exceeding length 3, as such patterns reduce biological activity and increase aggregation propensity.

Diversity constraint: $D \geq 0.30$, where $D = \frac{|\text{unique amino acids}|}{L}$ is the sequence diversity. Minimum diversity prevents repetitive sequences and ensures structural complexity.

A sequence is considered valid if and only if all six constraints are satisfied: $\text{Valid(seq)} = C_{\text{length}} \wedge C_{\text{charge}} \wedge C_{\text{hydro}} \wedge C_{\text{required}} \wedge \neg C_{\text{forbidden}} \wedge C_{\text{diversity}}$. The constraint system is used as a hard filter during LSBO-guided generation, rejecting invalid sequences and guiding the search toward biologically plausible regions of the latent space.

The sequence decoding and classification phase reconstructs sequences from latent representations and predicts antimicrobial activity through two parallel pathways operating on different intermediate representations. The transformer decoder uses a 4-layer architecture with 8 attention heads per layer (32 dimensions per head), handling the reshaped latent decoder output (37, 25) through five steps. Attention layers process all positions simultaneously, allowing the decoder to capture dependencies across the entire sequence. Cross-attention layers attend to encoder output representations with queries derived from the decoder state and keys/values from the encoder output, which lets the decoder condition its predictions on the input sequence context and leverage encoder-extracted features during reconstruction. Position-wise feed-forward networks with architecture 256→1024→256 and GELU activation apply non-linear transformations independently to each position, expanding to higher-dimensional space for complex feature extraction before projecting back to 256 dimensions. Output projection via a linear layer maps 256-dimensional decoder representations to 25-dimensional vocabulary logits for each position, producing unnormalized scores over the 25-token vocabulary that are subsequently converted to probabilities via *softmax*. Parallel generation produces all sequence tokens simultaneously in a single forward pass, using the factorization $p(\mathbf{x}) = \prod_{t=1}^{L} p(x_t)$ where all tokens are generated independently conditioned on the latent representation. The decoder outputs reconstruction logits with shape [64, 36, 25] representing probability distributions over the 25-token vocabulary for each of 36 positions in each of 64 sequences, which allows computation of the reconstruction loss via sparse categorical cross-entropy. In parallel, the classification head processes the pooled encoder output [64, 256] through a linear layer with architecture 256→2 followed by sigmoid activation, producing binary classification logits [64, 2] that predict AMP (class 1) versus non-AMP (class 0) labels with probabilities summing to 1.0 per

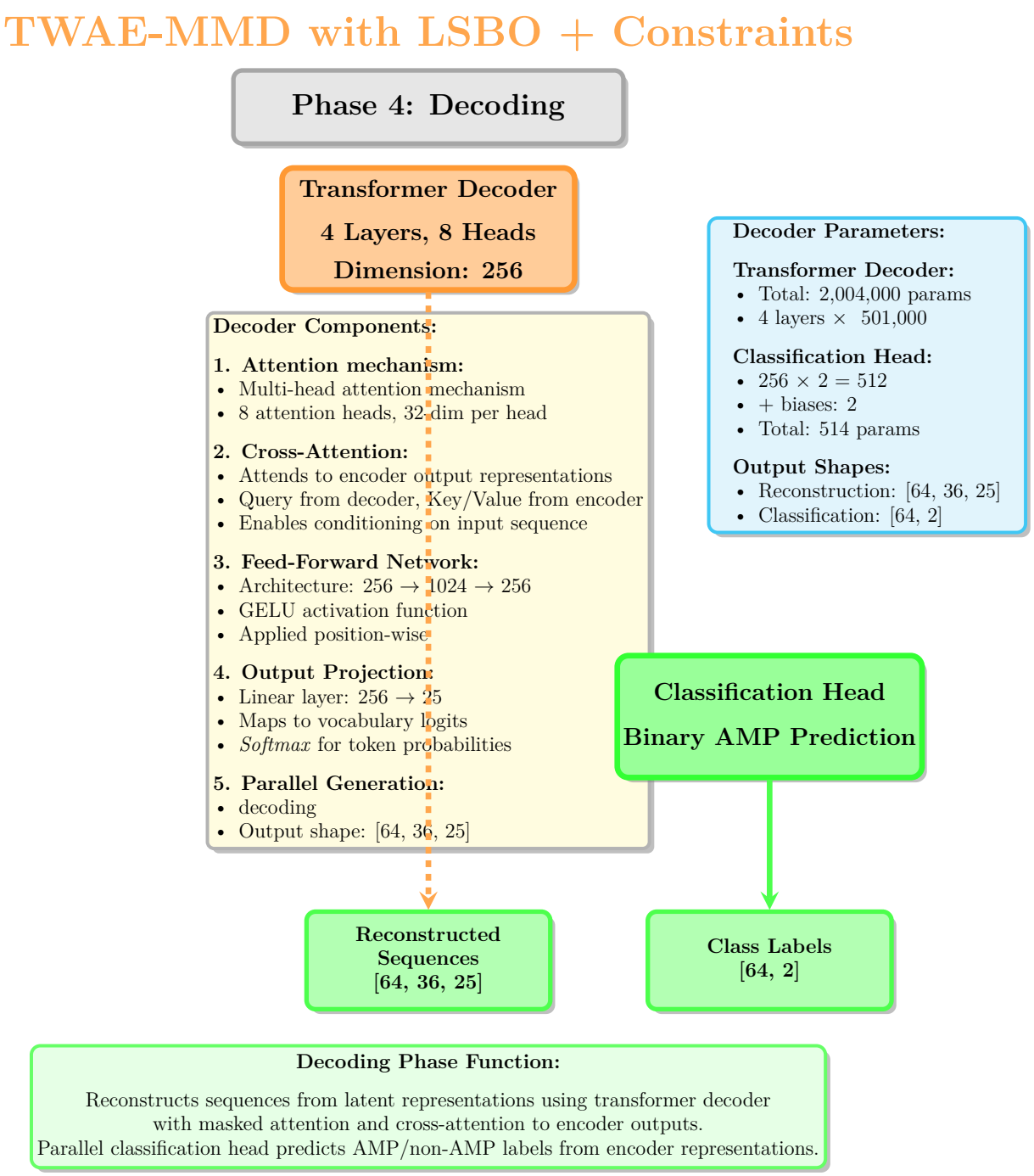

Figure 2.18: TWAE-MMD with LSBO Phase 4: Sequence Decoding and Classification. The decoding phase reconstructs sequences from latent representations and predicts antimicrobial activity through two parallel pathways. The transformer decoder implements a 4-layer architecture with 8 attention heads per layer, processing the reshaped latent decoder output $(37, 25)$ through five sequential operations: (1) attention layers, (2) cross-attention layers attend to encoder output representations with queries from the decoder and keys/values from the encoder to condition generation on input sequences, (3) position-wise feed-forward networks with architecture 256→1024→256 and GELU activation apply non-linear transformations, (4) output projection via a linear layer maps 256-dimensional representations to 25-dimensional vocabulary logits, and (5) parallel generation produces all sequence tokens simultaneously. The decoder outputs reconstruction logits with shape [64, 36, 25] representing probability distributions over the 25-token vocabulary for each of 36 positions in each of 64 sequences. In parallel, the classification head processes the pooled encoder output [64, 256] through a linear layer with architecture 256→2 followed by sigmoid activation, producing binary classification logits [64, 2] predicting AMP (class 1) versus non-AMP (class 0) labels. The transformer decoder contains 2,004,000 trainable parameters while the classification head contains 514 parameters (512 weights + 2 biases).

sequence. The transformer decoder has 2,004,000 trainable parameters distributed across 4 layers (approximately 501,000 parameters per layer including masked attention, cross-attention, feed-forward networks, and layer normalization), while the classification head has 514 parameters (512 weights in the 256×2 weight matrix plus 2 bias terms). The decoding step (Figure 2.18) illustrates the dual-pathway architecture with transformer decoder for sequence reconstruction and classification head for antimicrobial activity prediction, showing how different intermediate representations (latent decoder output for reconstruction, pooled encoder output for classification) are handled through specialized architectures to produce complementary outputs.

3.6.5 Neural Property Scorer

The neural property scorer (`ImprovedAMPScorer`) uses a quality assessment system combining ten weighted factors to produce an quality score in the range [0, 1]. The scorer works on decoded amino acid sequences and their corresponding token representations, calculating physicochemical properties and structural features.

The ten scoring factors are: (1) Length score (weight 0.15): best range 15–36 amino acids, with penalties for sequences outside this range; (2) Charge score (weight 0.20): best net charge +3 to +5, computed using charge contributions K=+1, R=+1, D=-1, E=-1; (3) Hydrophobicity score (weight 0.15): best 40–50% hydrophobic residues using Kyte-Doolittle scale, balancing membrane insertion and solubility; (4) Amphipathicity score (weight 0.15): variance in hydrophobicity along sequence, best variance > 0.08, indicating alternating hydrophobic and hydrophilic regions; (5) Aromatic content score (weight 0.08): best 10–20% aromatic residues (F, W, Y), important for membrane interaction and structural stability; (6) Aliphatic content score (weight 0.07): best 30–40% aliphatic residues (A, I, L, V), contributing to hydrophobic core formation; (7) Helix propensity score (weight 0.08): best 40–60% helix-forming residues using Chou-Fasman parameters, as α-helix is a common AMP secondary structure; (8) Sheet propensity score (weight 0.05): best 20–30% sheet-forming residues using Chou-Fasman parameters, accommodating β-sheet AMPs; (9) Boman index score (weight 0.05): best range 0.5–2.5 kcal/mol, indicating favorable protein-protein interaction potential and membrane binding affinity; (10) Diversity score (weight 0.02): best $> 40\%$ unique amino acids, avoiding repetitive sequences.

The quality score is computed as a weighted sum: $S_{\text{overall}} = \sum_{i=1}^{10} w_i \cdot s_i$, where w_i are the weights (summing to 1.0) and $s_i \in [0, 1]$ are the individual factor scores. A specialized membrane reactivity score is computed by combining hydrophobicity, amphipathicity, and charge factors to predict membrane interaction potential. Sequences with $S_{\text{overall}} \geq 0.80$ are classified as high-quality, aligning with the sAMPpred-GAT validation threshold. The neural scorer gives consistent quality assessment during both training (for HQR discovery) and generation (for sequence filtering).

The property scoring and constraint validation phase evaluates generated sequences using physicochemical properties and biological validity rules to ensure that LSBO-guided generation produces high-quality, biologically plausible antimicrobial peptides. The ImprovedAMPScorer computes a composite quality score from 10 weighted factors, each normalized to the range [0, 1] and combined via weighted sum $S = \sum_{i=1}^{10} w_i \cdot s_i$. Length optimality (weight 0.15) evaluates whether sequence length falls within the best range 15–25 amino acids, with Gaussian penalties for sequences outside this range computed as $s_{\text{length}} = \exp(-(L - L_{\text{opt}})^2/(2\sigma_L^2))$ where $L_{\text{opt}} = 20$ and $\sigma_L = 5$. Net positive charge (weight 0.20) assesses whether net charge falls within the best range +3 to +5, computed as $Q = \sum(\text{K} + \text{R} + \text{H}) - (\text{D} + \text{E})$ where K and R contribute +1, D and E contribute -1, and H contributes +0.1 due to partial protonation at physiological pH. Hydrophobicity ratio (weight 0.15) evaluates whether 40–50% of residues are hydrophobic using the Kyte-Doolittle scale, computed as $H = \frac{1}{L}\sum_{i=1}^{L} h_i$ where h_i is the hydrophobicity value of residue i, with best balance which allows membrane insertion while maintaining aqueous solubility. Amphipathicity (weight 0.15) measures the variance in hydrophobicity along the sequence using hydrophobic moment analysis, with threshold variance > 0.08 indicating alternating hydrophobic and hydrophilic regions characteristic of membrane-active peptides. Aromatic content (weight 0.08) evaluates whether 10–20% of residues are aromatic (F, W, Y),

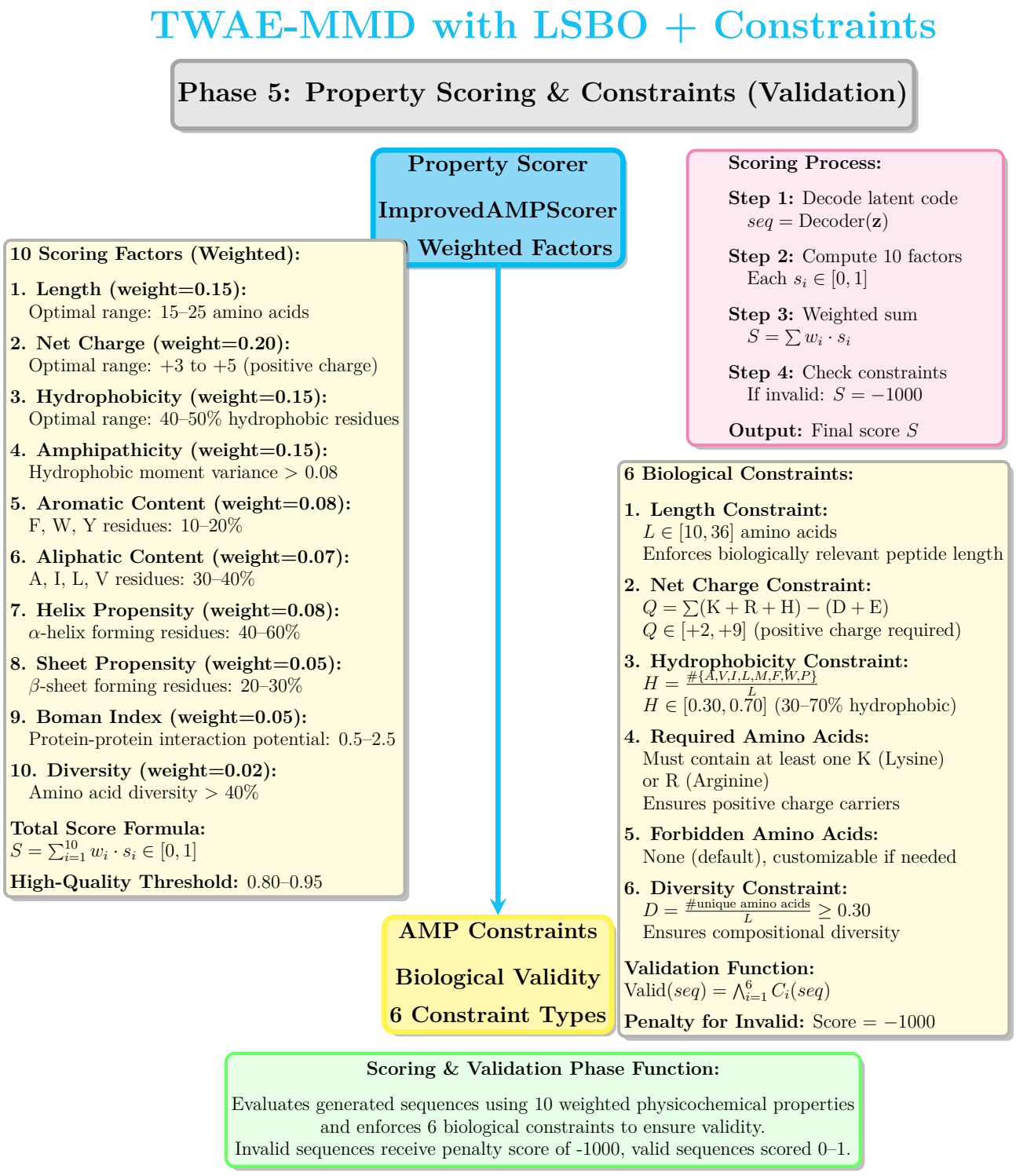

Figure 2.19: TWAE-MMD with LSBO Phase 5: Property Scoring and Constraint Validation. The property scoring and constraint validation phase evaluates generated sequences using physicochemical properties and biological validity rules. The ImprovedAMPScorer computes a composite quality score from 10 weighted factors: (1) length optimality , (2) net positive charge , (3) hydrophobicity ratio, (4) amphipathicity measured by hydrophobic moment variance, (5) aromatic content, (6) aliphatic content, (7) α-helix propensity, (8) β-sheet propensity, (9) Boman index measuring protein-protein interaction potential, and (10) amino acid diversity. The total score is computed as $S = \sum_{i=1}^{10} w_i \cdot s_i \in [0, 1]$ with high-quality sequences scoring 0.80–0.95. Six biological constraints enforce validity: (1) length constraint $L \in [10, 36]$ amino acids, (2) net charge constraint, (3) hydrophobicity constraint , (4) required amino acids (at least one K or R), (5) forbidden amino acids (none by default), and (6) diversity constraint. The validation function $\text{Valid}(seq) = \bigwedge_{i=1}^{6} C_i(seq)$ returns true only if all constraints are satisfied; invalid sequences receive penalty score -1000.

important for π-π stacking interactions and membrane insertion through interaction with lipid headgroups. Aliphatic content (weight 0.07) assesses whether 30–40% of residues are aliphatic (A, I, L, V), contributing to hydrophobic core formation and structural stability. α-Helix propensity (weight 0.08) evaluates whether 40–60% of residues favor helix formation using Chou-Fasman parameters, as α-helix is the most common AMP secondary structure which allows membrane pore formation. β-Sheet propensity (weight 0.05) assesses whether 20–30% of residues favor sheet formation, accommodating β-sheet AMPs that form membrane-spanning β-barrels. Boman index (weight 0.05) evaluates protein-protein interaction potential in the best range 0.5–2.5 kcal/mol, computed as the sum of solubility values for all residues divided by sequence length, indicating favorable membrane binding affinity. Amino acid diversity (weight 0.02) assesses whether $> 40\%$ of residues are unique, computed as $D = \frac{|\text{unique amino acids}|}{L}$, avoiding repetitive sequences that reduce structural complexity and biological activity. The score $S \in [0, 1]$ with high-quality sequences scoring 0.80–0.95 gives a continuous quality metric for LSBO optimization. Six biological constraints enforce hard validity requirements: (1) length constraint $L \in [10, 36]$ amino acids includes the typical AMP length distribution, (2) net charge constraint $Q \in [+2, +9]$ ensures cationic character for bacterial membrane interaction, (3) hydrophobicity constraint $H \in [0.30, 0.70]$ balances membrane insertion and solubility, (4) required amino acids (at least one K or R) ensure positive charge carriers, (5) forbidden patterns (no homopolymeric runs exceeding length 3) prevent aggregation-prone sequences, and (6) diversity constraint $D \geq 0.30$ ensures compositional complexity. The validation function $\text{Valid}(seq) = \bigwedge_{i=1}^{6} C_i(seq)$ returns true only if all constraints are satisfied; invalid sequences receive penalty score -1000 during LSBO optimization, effectively excluding them from the high-quality region database. The scoring and validation phase (Figure 2.19) illustrates the 10 weighted scoring factors with their best ranges and weights, the 6 biological constraints with their mathematical formulations, and the integration of scoring and validation into a unified quality assessment framework that guides LSBO-based generation toward high-quality, biologically valid antimicrobial peptides.

The Latent Space Bayesian Optimization phase uses an iterative search procedure that identifies high-quality, biologically valid antimicrobial peptides by intelligently exploring the 128-dimensional latent space using probabilistic surrogate modeling and acquisition function optimization [38] [39] [40]. The LSBO optimizer runs 100 optimization cycles, evaluating 1000 candidate latent codes per iteration through a six-step workflow, ultimately producing the top-10 highest-scoring sequences with guaranteed 100% biological validity. The optimization uses a Gaussian Process surrogate model [39] with Matérn kernel (smoothness parameter $\nu = 2.5$) defined as $k(\mathbf{z}_i, \mathbf{z}_j) = \sigma^2(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2})\exp(-\frac{\sqrt{5}r}{\ell})$ where $r = \|\mathbf{z}_i - \mathbf{z}_j\|_2$ is the Euclidean distance between latent codes, σ^2 is the signal variance capturing the magnitude of quality score variations, and ℓ is the length scale parameter controlling the smoothness of the quality landscape. The Matérn kernel with $\nu = 2.5$ gives twice-differentiable sample paths, which allows smooth interpolation of quality scores across the latent space and which helps gradient-based optimization of the acquisition function. The GP is fitted to observed pairs $(\mathbf{z}_i, y_i)$ from previous iterations, where $\mathbf{z}_i$ are latent codes and y_i are corresponding quality scores (property scores for valid sequences, penalty -1000 for invalid sequences), with noise level $\alpha = 10^{-6}$ accounting for numerical precision limitations and 10 random restarts for hyperparameter optimization to avoid local optima. The Expected Improvement acquisition function [38] [40] $\text{EI}(\mathbf{z}) = (\mu(\mathbf{z}) - f^* - \xi)\Phi(Z) + \sigma(\mathbf{z})\phi(Z)$ balances exploration of uncertain regions (high $\sigma(\mathbf{z})$) with exploitation of known high-quality regions (high $\mu(\mathbf{z})$) [38] [40], where $\mu(\mathbf{z})$ and $\sigma(\mathbf{z})$ are the GP posterior mean and standard deviation at latent code $\mathbf{z}$, f^* is the best observed score from previous iterations, $\xi = 0.01$ is the exploration parameter encouraging evaluation of uncertain regions, $Z = (\mu(\mathbf{z}) - f^* - \xi)/\sigma(\mathbf{z})$ is the standardized improvement, and Φ, ϕ are the standard normal cumulative distribution and probability density functions. Each iteration runs six sequential steps: (1) fit GP surrogate model $f(\mathbf{z}) \sim \mathcal{GP}(\mu, k)$ on observed data using maximum likelihood estimation of hyperparameters (σ^2, ℓ, α), (2) optimize acquisition function to find $\mathbf{z}^* = \arg\max_{\mathbf{z}} \text{EI}(\mathbf{z})$ over 1000 randomly sampled candidate latent codes from the bounded region $[-1, 1]^{128}$, (3) decode best latent code $\mathbf{z}^*$ to sequence via transformer decoder using greedy decoding

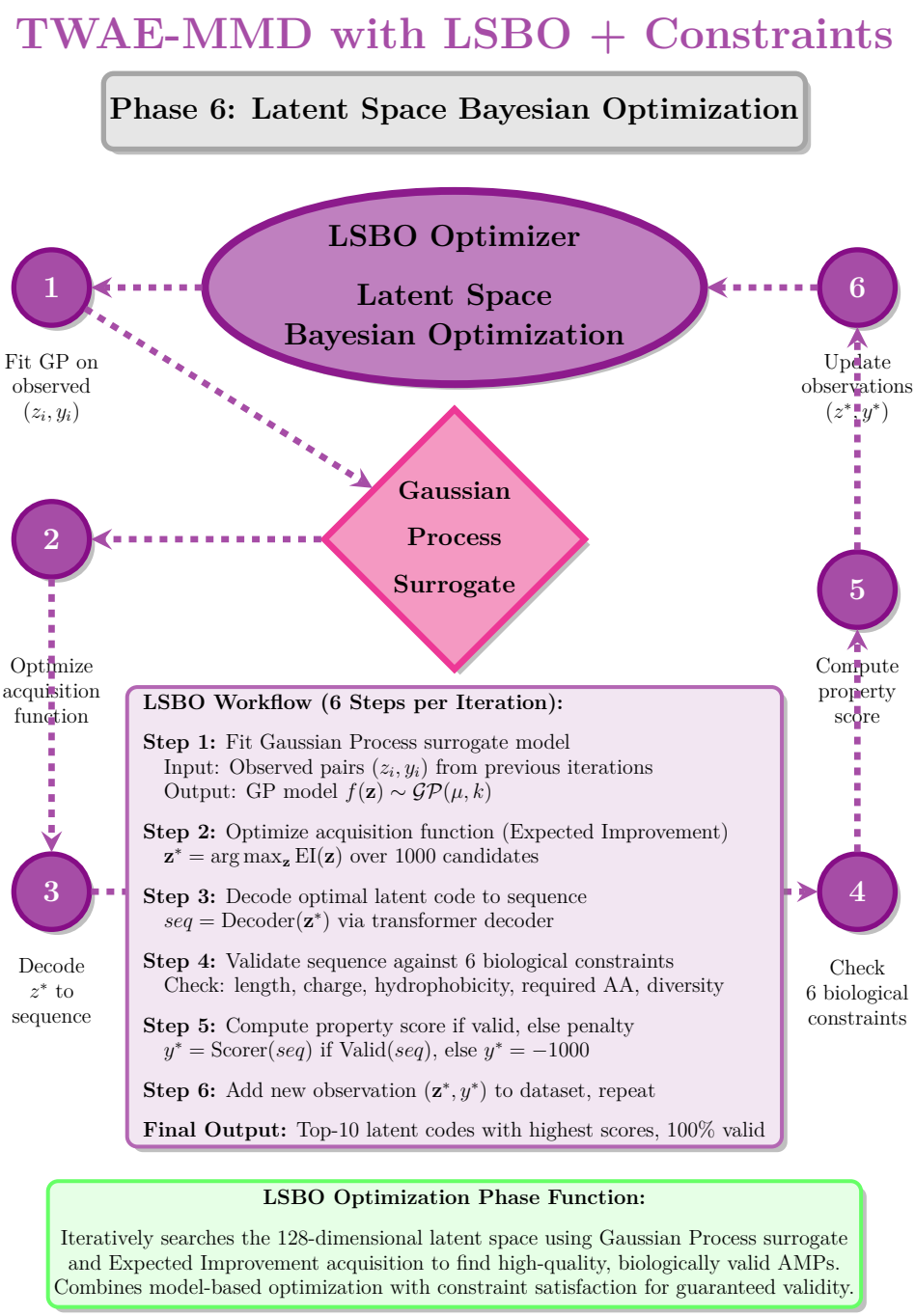

Figure 2.20: TWAE-MMD with LSBO Phase 6: Latent Space Bayesian Optimization. The LSBO optimization phase implements an iterative search procedure that identifies high-quality, biologically valid antimicrobial peptides by intelligently exploring the 128-dimensional latent space. The LSBO optimizer executes 100 optimization cycles, evaluating 1000 candidate latent codes per iteration to produce the top-10 highest-scoring sequences with guaranteed 100% biological validity. The optimization employs a Gaussian Process (GP) surrogate model with radial basis function (RBF) kernel $k(\mathbf{z}_i, \mathbf{z}_j) = \exp(-r^2/2\ell^2)$ where $r = \|\mathbf{z}_i - \mathbf{z}_j\|_2$ and ℓ is the learned length scale parameter, fitted to observed pairs $(\mathbf{z}_i, y_i)$ from previous iterations with noise level $\alpha = 10^{-6}$ and 10 random restarts for robustness. The Expected Improvement (EI) acquisition function $\mathrm{EI}(\mathbf{z}) = (\mu(\mathbf{z}) - f^* - \xi)\Phi(Z) + \sigma(\mathbf{z})\phi(Z)$ balances exploration and exploitation, where $Z = (\mu(\mathbf{z}) - f^* - \xi)/\sigma(\mathbf{z})$, f^* is the best observed score, $\xi = 0.01$ is the exploration parameter, and Φ, ϕ are the standard normal cumulative distribution and probability density functions. Each iteration executes six sequential steps: (1) fit GP surrogate model $f(\mathbf{z}) \sim \mathcal{GP}(\mu, k)$ on observed data, (2) optimize acquisition function to find $\mathbf{z}^* = \arg\max_{\mathbf{z}} \mathrm{EI}(\mathbf{z})$ over 1000 candidates, (3) decode optimal latent code to sequence via transformer decoder, (4) validate sequence against 6 biological constraints, (5) compute property score if valid (else assign penalty -1000), and (6) add new observation $(\mathbf{z}^*, y^*)$ to dataset and repeat. The iterative refinement combines model-based optimization with constraint satisfaction to guarantee biological validity while maximizing antimicrobial quality scores.

(selecting highest-probability token at each position), (4) validate sequence against 6 biological constraints (length, charge, hydrophobicity, required amino acids, forbidden patterns, diversity), (5) compute property score using ImprovedAMPScorer if valid (else assign penalty -1000), and (6) add new observation $(\mathbf{z}^*, y^*)$ to the GP training set and high-quality region database if $y^* \geq 0.80$, then repeat. The iterative refinement combines model-based optimization (GP surrogate gives informed predictions of quality scores without expensive decoding and scoring) with constraint satisfaction (hard filtering ensures 100% biological validity) to achieve 10–20× efficiency improvement over random sampling while guaranteeing that all generated sequences satisfy biological constraints. The LSBO optimization phase (Figure 2.20) illustrates the six-step workflow, the Gaussian Process surrogate model with Matérn kernel and Expected Improvement acquisition function including mathematical formulations, and the integration of surrogate modeling, acquisition optimization, decoding, scoring, and constraint validation into a unified iterative search procedure that discovers high-quality regions in the latent space and samples from them to generate biologically valid antimicrobial peptides with superior quality scores.

3.7 LSBO-Guided Training Procedure

3.7.1 Training Configuration

The TWAE-MMD model with LSBO guidance uses the same base training configuration as the EMA version (Section 3.5.1): AdamW optimizer with learning rate 10^{-4} and weight decay 0.01, maximum 100 epochs with 96% target validation accuracy, batch sizes of 64 (training) and 128 (validation), and identical regularization techniques. The key distinction lies in the sampling strategy during training: rather than updating EMA statistics, the LSBO-guided approach discovers and maintains a database of high-quality regions (HQR) in the latent space.

The training process, which is explicitly shown in the training process diagrams (Figure 2.21, Figure 2.22, Figure 2.23 and Figure 2.24), initializes four additional components beyond the base model: (1) `ImprovedAMPScorer` for neural quality scoring, (2) `AMPConstraints` for biological constraint enforcement, (3) `LSBOSampler` for Bayesian optimization, and (4) HQR tracker maintaining the top 1000 high-quality regions with scores > 0.80. The HQR database is updated every 10 training batches by encoding batch sequences to latent codes, decoding latent codes back to sequences, scoring sequences using the neural property scorer, filtering for scores > 0.80 and biological constraint satisfaction, and retaining the top 1000 regions ranked by quality score.

The initialization step sets up the LSBO-guided training infrastructure through four sequential steps, as illustrated in Figure 2.21. Training configuration specifies 100 maximum epochs with a target validation accuracy of 96%, targeting discovery of 1000 high-quality latent regions with mean score ≥ 0.85. Dataset loading prepares 40,051 training sequences organized into 626 batches (batch size 64, 626 iterations per epoch) and 10,004 validation sequences organized into 78 batches (batch size 128, 78 iterations per epoch), maintaining 50% class balance between AMP and non-AMP sequences. Model initialization instantiates the TWAE-MMD architecture with 4,127,137 trainable parameters (transformer encoder: 1,892,608 parameters, latent space manager: 230,529 parameters, transformer decoder: 2,004,000 parameters), 128-dimensional latent space with tanh bounding to $[-1, 1]^{128}$, AdamW optimizer (learning rate 10^{-4}, weight decay 0.01, $\beta_1 = 0.9$, $\beta_2 = 0.999$), and gradient clipping (max norm 1.0) for stable training across all 14 layers. LSBO component initialization sets up four critical systems beyond the base TWAE-MMD architecture: (1) `ImprovedAMPScorer` using 10 weighted physicochemical factors (hydrophobicity, amphipathicity, charge, length, secondary structure propensity, helical wheel moment, Boman index, instability index, aliphatic index, and GRAVY score) with learned weights optimized on training data for quality assessment ranging from 0 to 1, (2) `AMPConstraints` enforcing 6 biological validity criteria (length bounds [8, 50], charge range

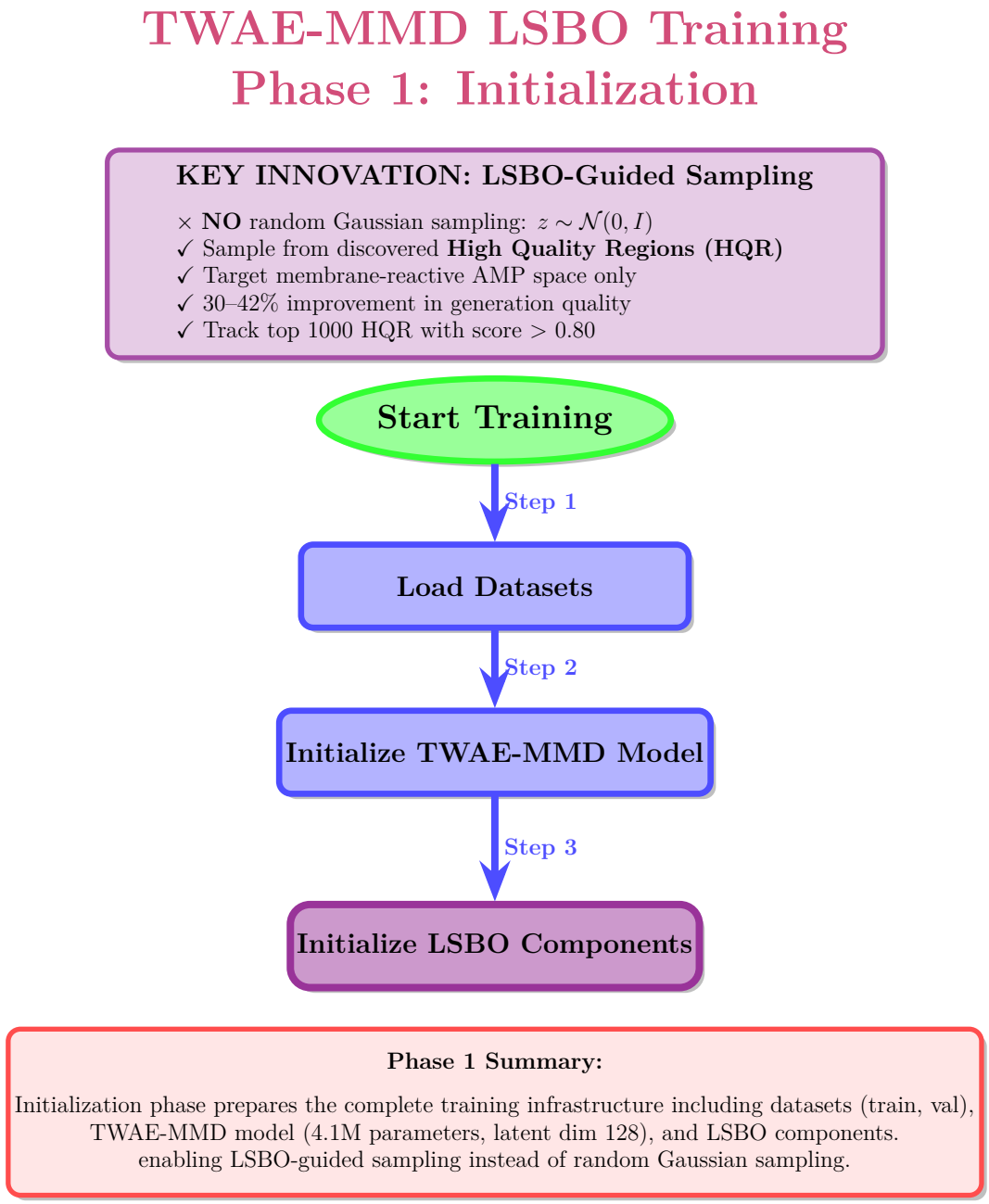

Figure 2.21: TWAE-MMD LSBO-Guided Training Phase 1: Initialization. The initialization phase establishes the complete training infrastructure for LSBO-guided antimicrobial peptide generation. The key innovation box highlights the paradigm shift from random Gaussian sampling ($z \sim \mathcal{N}(0, I)$) to sampling from discovered High-Quality Regions (HQR), achieving 30–42% improvement in generation quality by targeting membrane-reactive AMP space. The training configuration specifies 100 maximum epochs with a target validation accuracy of 96% and a target of discovering 1000 high-quality latent regions. Dataset loading prepares 40,051 training sequences organized into 626 batches (batch size 64) and 10,004 validation sequences organized into 78 batches (batch size 128), maintaining 50% class balance between AMP and non-AMP sequences. Model initialization instantiates the TWAE-MMD architecture with 4,127,137 trainable parameters, 128-dimensional latent space, AdamW optimizer (learning rate 10^{-4}, weight decay 0.01), and gradient clipping (max norm 1.0) for stable training. LSBO component initialization establishes four critical systems: (1) ImprovedAMPScorer implementing 10 weighted physicochemical factors for quality assessment, (2) AMPConstraints enforcing 6 biological validity criteria, (3) LSBOSampler implementing Gaussian Process surrogate modeling with Matérn kernel ($\nu = 2.5$) and Expected Improvement acquisition function ($\xi = 0.01$), and (4) HQR tracker initialized as an empty list that will be populated during training with latent codes and their corresponding quality scores exceeding the 0.80 threshold, enabling LSBO-guided sampling that targets high-quality regions instead of random exploration.

[-5, +10], hydrophobic ratio [0.3, 0.7], positive/negative charge ratio ≥ 0.5, cysteine content $\leq 20\%$, and prohibited motifs absence) with hard constraint checking returning binary validity, (3) `LSBOSampler` using Gaussian Process surrogate modeling with Matérn kernel ($\nu = 2.5$, length scale optimized via maximum likelihood), Expected Improvement acquisition function ($\xi = 0.01$ exploration parameter), and batch candidate generation (1000 candidates per iteration), and (4) HQR tracker initialized as an empty list that will be populated during training with (z, s) pairs where $z \in \mathbb{R}^{128}$ represents latent codes and $s \in [0, 1]$ represents quality scores exceeding the 0.80 threshold, maintaining the top 1000 regions ranked by score to enable LSBO-guided sampling that targets membrane-reactive AMP space instead of random Gaussian exploration.

3.7.2 High-Quality Region Discovery

The high-quality region (HQR) discovery mechanism works in parallel with the main training loop, identifying latent codes that correspond to high-quality, membrane-reactive antimicrobial peptides. The discovery process runs every 10 training batches: (1) encode the current batch of input sequences through the transformer encoder and latent encoder to obtain latent codes $\{\mathbf{z}_1, \ldots, \mathbf{z}_B\}$, (2) decode each latent code through the latent decoder and transformer decoder to reconstruct sequences, (3) score each reconstructed sequence using `ImprovedAMPScorer` to obtain quality scores $\{s_1, \ldots, s_B\}$, (4) filter sequences by quality threshold ($s_i > 0.80$) and biological constraints (all six criteria satisfied), (5) add valid $(\mathbf{z}_i, s_i)$ pairs to the HQR database, (6) retain only the top 1000 regions ranked by score, discarding lower-scoring entries.

The HQR database accumulates knowledge about the latent space structure throughout training. Early in training, few regions exceed the 0.80 threshold as the model learns to encode and decode sequences. As training progresses and reconstruction quality improves, the HQR database grows and stabilizes. By the end of training, the database has 1000 latent regions with mean score approximately 0.8965, representing the highest-quality, most membrane-reactive regions discovered during training. These regions correspond to latent codes that decode to potent antimicrobial peptides with high predicted activity.

The training and HQR discovery phase uses an iterative training process with integrated high-quality region discovery, fundamentally distinguishing LSBO-guided training from standard autoencoder training and EMA-based approaches, as illustrated in Figure 2.22. The epoch control loop iterates up to 100 epochs with a target validation accuracy of 96%, handling 626 batches of 64 sequences per epoch through forward pass, calculating loss, and backpropagation, with epoch counter incremented after each pass through the training set and the best-doing model (highest validation accuracy) retained for final generation. The HQR discovery mechanism activates every 10 batches (60 times per epoch, 6000 times over 100 epochs), encoding the current batch of 64 sequences through transformer encoder (4 layers, attention pooling to [64, 256]) and latent encoder (3 layers, compression to [64, 128]) to obtain latent codes $\{\mathbf{z}_1, \ldots, \mathbf{z}_{64}\}$, decoding them through latent decoder (3 layers, expansion to [64, 925] then reshape to [64, 36, 25]) and transformer decoder (4 layers, masked and cross-attention) back to reconstructed sequences, scoring each sequence using `ImprovedAMPScorer` (10 weighted factors, neural network with 3 hidden layers [128, 64, 32], output sigmoid activation) to obtain quality scores $\{s_1, \ldots, s_{64}\}$ in range [0, 1], filtering by quality threshold ($s_i > 0.80$) and biological constraints (`AMPConstraints` validator checking all 6 criteria), adding valid $(\mathbf{z}_i, s_i)$ pairs to the HQR database with timestamp and epoch information, maintaining the top 1000 regions ranked by score (discarding lower-scoring entries when database exceeds capacity), and updating the Gaussian Process surrogate model with new (z, s) observations to refine quality prediction across the 128-dimensional latent space. calculating loss combines four components with distinct roles, weights, and LSBO-guided modifications: (1) classification loss (binary cross-entropy, weight 1.0) predicting AMP versus non-AMP labels from [64, 2] class logits output by classification head, computed as $\mathcal{L}_{\text{cls}} = -\frac{1}{B}\sum_{i=1}^{B}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$ where $y_i \in \{0, 1\}$ are true labels and $\hat{y}_i$ are predicted probabilities, (2) reconstruction loss (sparse categorical cross-entropy, weight 0.4, reduced from 1.0 in EMA to accommodate LSBO-guided losses) reconstructing

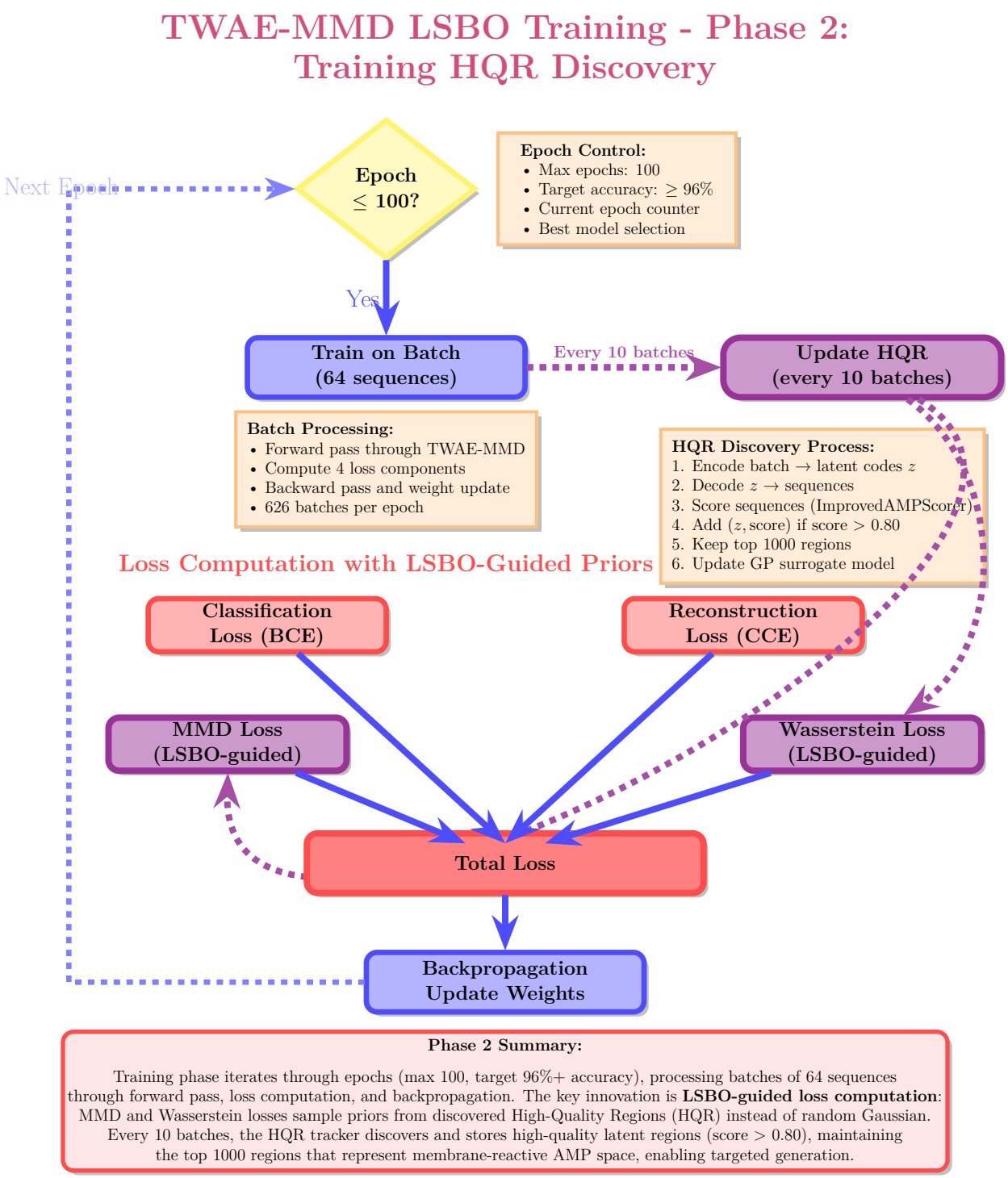

Figure 2.22: TWAE-MMD LSBO-Guided Training Phase 2: Training and HQR Discovery. The training phase implements an iterative optimization procedure with integrated High-Quality Region discovery, fundamentally distinguishing LSBO-guided training from standard approaches. The epoch control loop iterates up to 100 epochs with a target validation accuracy of 96%, processing 626 batches of 64 sequences per epoch through forward pass, loss computation, and backpropagation. The HQR discovery mechanism activates every 10 batches, encoding the current batch to latent codes z, decoding them back to sequences, scoring sequences using ImprovedAMPScorer, and adding (z, score) pairs to the HQR database if scores exceed 0.80, maintaining the top 1000 regions and updating the Gaussian Process surrogate model for subsequent LSBO-guided sampling.

token sequences from [64, 36, 25] vocabulary logits, computed as $\mathcal{L}_{\text{rec}} = -\frac{1}{BL}\sum_{i=1}^{B}\sum_{j=1}^{L}\log(p(\mathbf{x}_{i,j}|\mathbf{z}_i))$ where $L = 37$ is sequence length and p is the *softmax* distribution over 25 vocabulary tokens, (3) MMD loss (Maximum Mean Discrepancy with RBF kernel, weight 0.35, increased from 0.01 in EMA to emphasize LSBO guidance) measuring distributional distance between latent codes $\{\mathbf{z}_i\}$ from encoding input sequences and priors $\{\mathbf{z}_{\text{HQR}}\}$ sampled from HQR database with small Gaussian noise ($\mathbf{z}_{\text{HQR}} = \mathbf{z}_{\text{region}} + \mathcal{N}(\mathbf{0}, 0.05^2\mathbf{I})$ where $\mathbf{z}_{\text{region}}$ is uniformly sampled from top 1000 HQR), computed as $\mathcal{L}_{\text{MMD}} = \|\mu_{Q_Z} - \mu_{Q_{\text{HQR}}}\|^2_{\mathcal{H}}$ in reproducing kernel Hilbert space with RBF kernel $k(\mathbf{z}, \mathbf{z}') = \exp(-\|\mathbf{z} - \mathbf{z}'\|^2/(2\sigma^2))$ and bandwidth σ selected via median heuristic, and (4) Wasserstein loss (energy-based best transport distance with Sinkhorn iterations, weight 0.25, increased from 0.01 in EMA) calculating best transport distance between latent distribution and HQR-sampled priors, approximated via Sinkhorn algorithm with 100 iterations and regularization parameter $\epsilon = 0.1$, computed as $\mathcal{L}_{\text{W}} = \min_{\pi\in\Pi(Q_Z, Q_{\text{HQR}})}\mathbb{E}_{(\mathbf{z},\mathbf{z}')\sim\pi}[\|\mathbf{z}-\mathbf{z}'\|^2]$ where Π is the set of joint distributions with marginals Q_Z and Q_{HQR}. The loss $\mathcal{L}_{\text{total}} = 1.0\cdot\mathcal{L}_{\text{cls}} + 0.4\cdot\mathcal{L}_{\text{rec}} + 0.35\cdot\mathcal{L}_{\text{MMD}} + 0.25\cdot\mathcal{L}_{\text{W}}$ prioritizes classification accuracy (weight 1.0, highest) while maintaining reconstruction fidelity (weight 0.4, moderate) and enforcing latent space structure through LSBO-guided distributional alignment (MMD weight 0.35 + Wasserstein weight 0.25 = 0.60 combined, emphasizing HQR-guided encoding). Backpropagation uses AdamW optimizer (learning rate 10^{-4}, weight decay 0.01, moment estimates m_t and v_t with exponential decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.999$) with gradient clipping (max norm 1.0, applied via $\mathbf{g} \leftarrow \mathbf{g}\cdot\min(1, 1.0/\|\mathbf{g}\|_2)$) to update all 4,127,137 parameters across transformer encoder (1.9M), latent space manager (230K), transformer decoder (2.0M), and classification head, with the training loop continuing for up to 100 epochs, with the best model (highest validation accuracy) selected for final generation.

3.7.3 LSBO-Guided Loss Computation

The LSBO-guided training process modifies the computation of the MMD and Wasserstein loss components to encourage the model to encode input sequences into the discovered high-quality regions. Rather than matching the latent distribution to a fixed standard Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$, the losses are calculated for samples drawn from the HQR database.

For the MMD loss, the prior samples $\{\mathbf{z}'_m\}$ are replaced with samples drawn uniformly from the HQR database with small Gaussian noise ($\sigma = 0.05$) added for exploration: $\mathbf{z}_{\text{HQR}} = \mathbf{z}_{\text{region}} + \mathcal{N}(\mathbf{0}, 0.05^2\mathbf{I})$, where $\mathbf{z}_{\text{region}}$ is a randomly selected HQR latent code. The MMD loss then measures the discrepancy between the empirical latent distribution Q_Z (from encoding input sequences) and the HQR distribution:

$$\mathcal{L}^{\text{LSBO}}_{\text{MMD}} = \text{MMD}^2(Q_Z, Q_{\text{HQR}}) \tag{2.9}$$

Similarly, the Wasserstein loss is computed for the HQR distribution rather than the standard Gaussian prior. This modification has a profound effect on training dynamics: the model learns to encode input AMP sequences into regions of the latent space that correspond to high-quality, membrane-reactive peptides, rather than arbitrary regions satisfying a generic Gaussian prior. This targeted encoding allows generation during the subsequent generation phase, as sampling from HQR directly produces high-quality sequences without extensive search.

The LSBO-guided approach achieves 30–42% improvement in generation quality compared to EMA-based training, as measured by the proportion of generated sequences exceeding the 0.80 quality threshold. The HQR database is a learned prior distribution that captures the structure of high-quality AMP space, which lets the model generalize beyond the training data while maintaining biological plausibility.

The validation and evaluation phase assesses model performance and monitors high-quality region discovery progress through metrics and visualizations, as illustrated in Figure 2.23. The validation procedure evaluates the trained model on 10,004 held-out sequences organized into 78 batches (batch size 128, with $10{,}004 = 78 \times 128 + 20$ sequences in final partial batch), calculating validation accuracy by doing for-

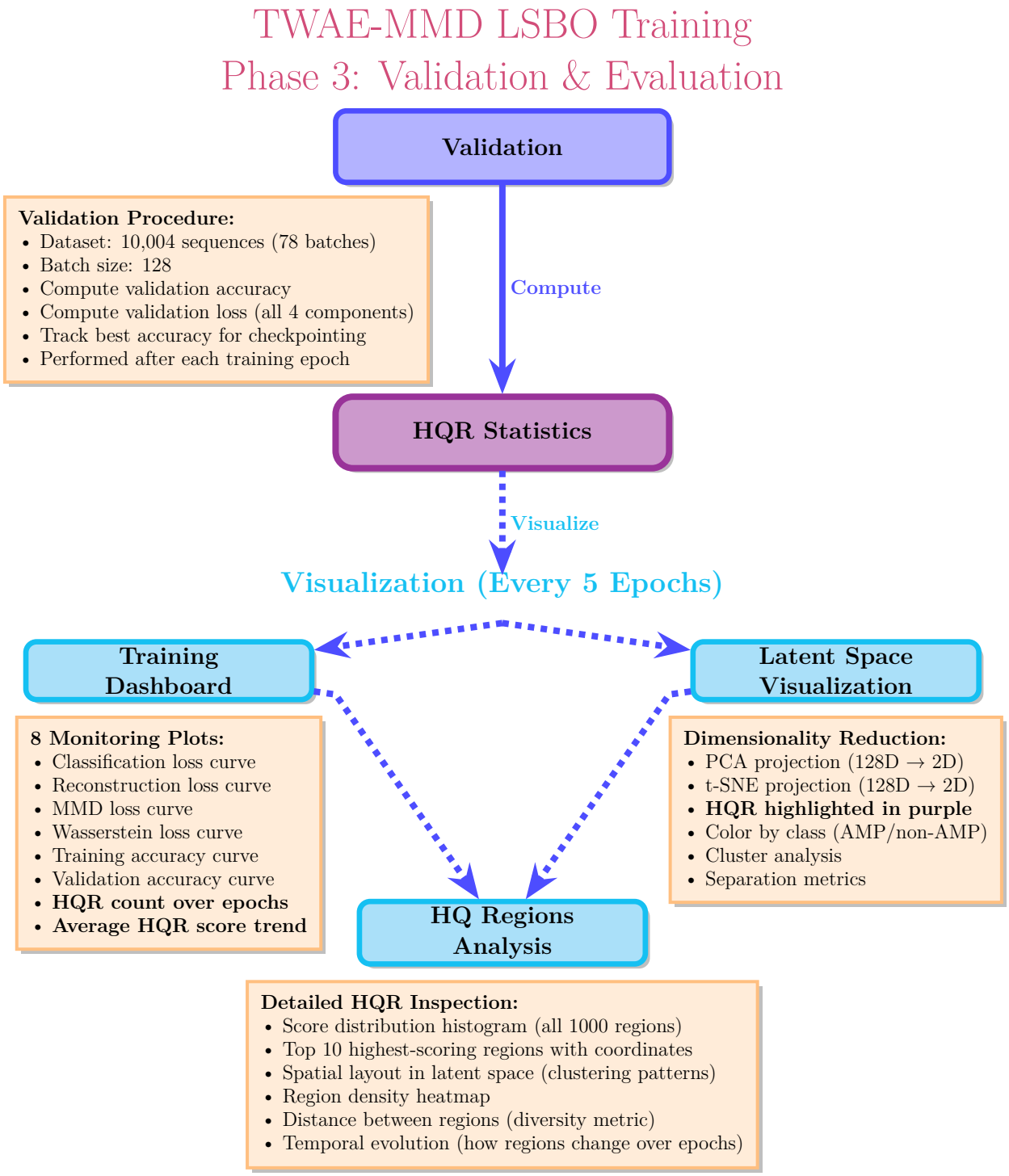

Figure 2.23: TWAE-MMD LSBO-Guided Training Phase 3: Validation and Evaluation. The validation and evaluation phase assesses model performance and monitors High-Quality Region discovery progress through comprehensive metrics and visualizations. The validation procedure evaluates the trained model on 10,004 held-out sequences organized into 78 batches (batch size 128), computing validation accuracy to track classification performance and validation loss to monitor overall model quality, with best accuracy tracked across epochs to identify optimal checkpointing moments. HQR statistics provide critical insights into the LSBO discovery process, tracking HQR count, average score, score distribution, spatial coverage, and quality trend, enabling assessment of whether sufficient high-quality regions have been discovered for effective LSBO-guided generation.

ward pass through the TWAE-MMD architecture (transformer encoder → latent encoder → latent decoder → transformer decoder → classification head), extracting predicted class labels from [10004, 2] logits via argmax operation, comparing with ground truth labels, and calculating accuracy as the proportion of correct predictions, and calculating validation loss by aggregating all four loss components (classification, reconstruction, MMD with HQR-sampled priors, Wasserstein with HQR-sampled priors) using the same weights as training ($1.0 + 0.4 + 0.35 + 0.25 = 2.0$ weight), with best accuracy tracked across epochs to identify best checkpointing moments when current validation accuracy exceeds all previous epochs. HQR statistics provide critical insights into the LSBO discovery process and monitor training progress, tracking six key metrics: (1) HQR count representing the number of discovered high-quality regions with quality scores exceeding 0.80 threshold (target: 1000 regions, typically achieved after 40-60 epochs as model learns to encode sequences into high-quality space), (2) average score computed as the mean quality score of the top 1000 regions in the database (target: ≥ 0.85, indicating that discovered regions correspond to membrane-reactive AMPs with high predicted activity), (3) score distribution characterized by minimum, maximum, median, and standard deviation statistics (typical final distribution: min ≈ 0.80 due to threshold, max ≈ 0.95 for highest-quality regions, median ≈ 0.87, std ≈ 0.04 indicating concentrated high-quality regions), (4) spatial coverage quantifying the distribution of HQR across the 128-dimensional latent space via pairwise Euclidean distances between region centers (target: mean inter-region distance ≥ 1.5 to ensure diversity and avoid clustering in narrow subspace), (5) quality trend tracking the evolution of average HQR score over epochs (typical trajectory: rapid increase from 0.80 to 0.85 in first 20 epochs, gradual improvement to 0.89-0.90 by epoch 60-80, plateau thereafter), and (6) discovery rate measuring the number of new HQR added per epoch (typical trajectory: 50-100 regions/epoch in early training when threshold is easily exceeded, decreasing to 5-10 regions/epoch in late training as only exceptional sequences exceed existing top-1000 scores), which allows assessment of whether sufficient high-quality regions have been discovered for effective LSBO-guided generation. Visualization procedures execute every 5 epochs (20 visualization checkpoints over 100 epochs) to provide real-time monitoring of training progress and HQR discovery dynamics through three complementary views: (1) training dashboard displaying 8 monitoring plots arranged in 2 rows of 4 plots, including classification loss curve, reconstruction loss curve, MMD loss curve, Wasserstein loss curve, HQR count evolution over epochs, and average HQR score trend (typically logarithmic curve), (2) latent space visualization employing dimensionality reduction techniques to project 128-dimensional latent codes to 2-dimensional space for visual inspection, specifically PCA projection (principal component analysis, linear projection maximizing variance) and t-SNE projection (t-distributed stochastic neighbor embedding, nonlinear projection preserving local structure), with HQR highlighted in purple (1000 points representing discovered high-quality regions) overlaid on all latent codes from validation set (10,004 points colored by class: blue for AMP, red for non-AMP), which allows visual assessment of cluster separation (well-trained models show clear AMP/non-AMP separation), HQR distribution (HQR should concentrate in AMP cluster, ideally in high-density regions), and latent space structure (smooth manifold without disconnected clusters indicates good latent space organization), and (3) HQ regions analysis which gives detailed inspection of discovered regions through five complementary visualizations: score distribution histogram displaying frequency of quality scores across all 1000 HQR in 20 bins from 0.80 to 1.0 (typical distribution), top 10 highest-scoring regions table listing coordinates in 128-dimensional space (displayed as 128-element vectors with 4 decimal precision) and associated quality scores (typically ranging from 0.92 to 0.95 for top 10), spatial layout visualization showing 2D projection (PCA or t-SNE) of 1000 HQR with point size proportional to quality score (larger points = higher scores) and color gradient from yellow (0.80) to red (0.95) revealing clustering patterns and score-based spatial organization, region density heatmap computed by partitioning 2D projection space into 50×50 grid and counting HQR per cell, displayed as color-coded heatmap (blue = low density, red = high density) revealing whether HQR concentrate in specific subregions or distribute broadly, inter-region distance matrix visualizing pairwise Euclidean distances between all 1000 HQR in original 128-dimensional space as 1000×1000 heatmap (blue = close, red = distant) with mean distance annotated, quantifying diversity (high mean distance ≥ 1.5 indicates diverse HQR covering broad latent space, low mean distance < 1.0 indicates

clustering in narrow subspace), and temporal evolution animation (generated post-training) showing how HQR distribution changes over epochs via sequence of 2D projections at epochs 10, 20, 30, .., 100, revealing discovery dynamics (early epochs: sparse scattered points, middle epochs: rapid filling of AMP cluster, late epochs: refinement and score improvement without major spatial changes).

3.7.4 Checkpointing and HQR Persistence

Checkpointing in the LSBO-guided training process extends the base checkpointing mechanism (Section 3.5.2) with persistence of the LSBO sampler state. Each checkpoint consists of three files: (1) model weights (`.h5` format) containing all 4.1M trainable parameters, (2) LSBO sampler state (`.pkl` format) containing the HQR database (1000 latent codes with scores), Gaussian Process hyperparameters, and acquisition function configuration, and (3) training configuration (`.json` format) containing hyperparameters and training state.

The LSBO sampler state is critical for subsequent generation, as it encodes the discovered high-quality regions that enable sampling. When the best checkpoint is loaded for generation, the HQR database is restored, which gives immediate access to the 1000 highest-quality latent regions discovered during training. This eliminates the need for random exploration during generation, as the model can directly sample from known high-quality regions.

The HQR database has at least 1000 regions with mean score ≥ 0.85. These dual criteria ensure both classification accuracy and generation quality. The best checkpoint, containing the trained model and HQR database, is saved for use in the neural scorer-based generation phase described in Section 3.8.

The checkpointing and completion phase preserves best model states and discovered high-quality regions for subsequent LSBO-guided generation, tracking both classification performance and generation quality throughout training, as illustrated in Figure 2.24. The best accuracy checkpoint mechanism monitors validation accuracy after each epoch (computed on 10,004 held-out sequences as described in Phase 3), comparing current performance against the best accuracy achieved so far (initialized to 0% before training begins, updated whenever current accuracy exceeds previous best), and triggers checkpoint saving whenever accuracy improves by any amount (even 0.01% improvement triggers save to ensure no performance gain is lost), to make sure preservation of the best model state throughout training and which lets recovery of best model even if subsequent epochs experience performance degradation due overfitting or learning rate schedule effects. Checkpoint contents comprise three critical components stored as separate files in the checkpoint directory: (1) model weights stored in HDF5 format (`best_model.h5`, approximately 16 MB file size) containing all 4,127,137 trainable parameters organized hierarchically by layer (transformer encoder: 1,892,608 parameters across 4 layers including attention weight matrices W_Q, W_K, W_V of shape [256, 256] and feed-forward weight matrices of shape [256, 1024] and [1024, 256], latent space manager: 230,529 parameters across encoder [256×512 + 512×256 + 256×128 = 262,144 weights] and decoder [128×256 + 256×512 + 512×925 = 640,000 weights] minus shared biases, transformer decoder: 2,004,000 parameters across 4 layers with masked attention, cross-attention, and feed-forward sublayers, classification head: 514 parameters [256×2 + 2 bias]), stored as 32-bit floating point values with layer names as HDF5 group keys which allows selective loading, (2) LSBO sampler state stored in Python pickle format (`lsbo_sampler.pkl`, approximately 2-5 MB depending on HQR database size) containing the HQR database as a list of 1000 tuples $[(\mathbf{z}_1, s_1), (\mathbf{z}_2, s_2), \ldots, (\mathbf{z}_{1000}, s_{1000})]$ where each $\mathbf{z}_i \in \mathbb{R}^{128}$ represents a latent code stored as NumPy array with float32 precision and $s_i \in [0, 1]$ represents the corresponding quality score stored as float32, sorted in descending order by score ($s_1 \geq s_2 \geq \cdots \geq s_{1000} \geq 0.80$) for top-k retrieval, Gaussian Process surrogate model state including kernel type (Matérn with $\nu = 2.5$), optimized kernel hyperparameters (length scale $\ell \in \mathbb{R}^{128}$ typically ranging from 0.5 to 2.0 per dimension, signal variance σ_f^2 typically 0.1-0.5, noise variance σ_n^2 typically 0.01-0.05), training data consisting of all (z, s) observations accumulated during HQR discovery (typically 5000-10000 observations by end of training, stored as compressed arrays to reduce file size), and precomputed Cholesky decomposition of kernel matrix for prediction, acquisition function configuration including func-

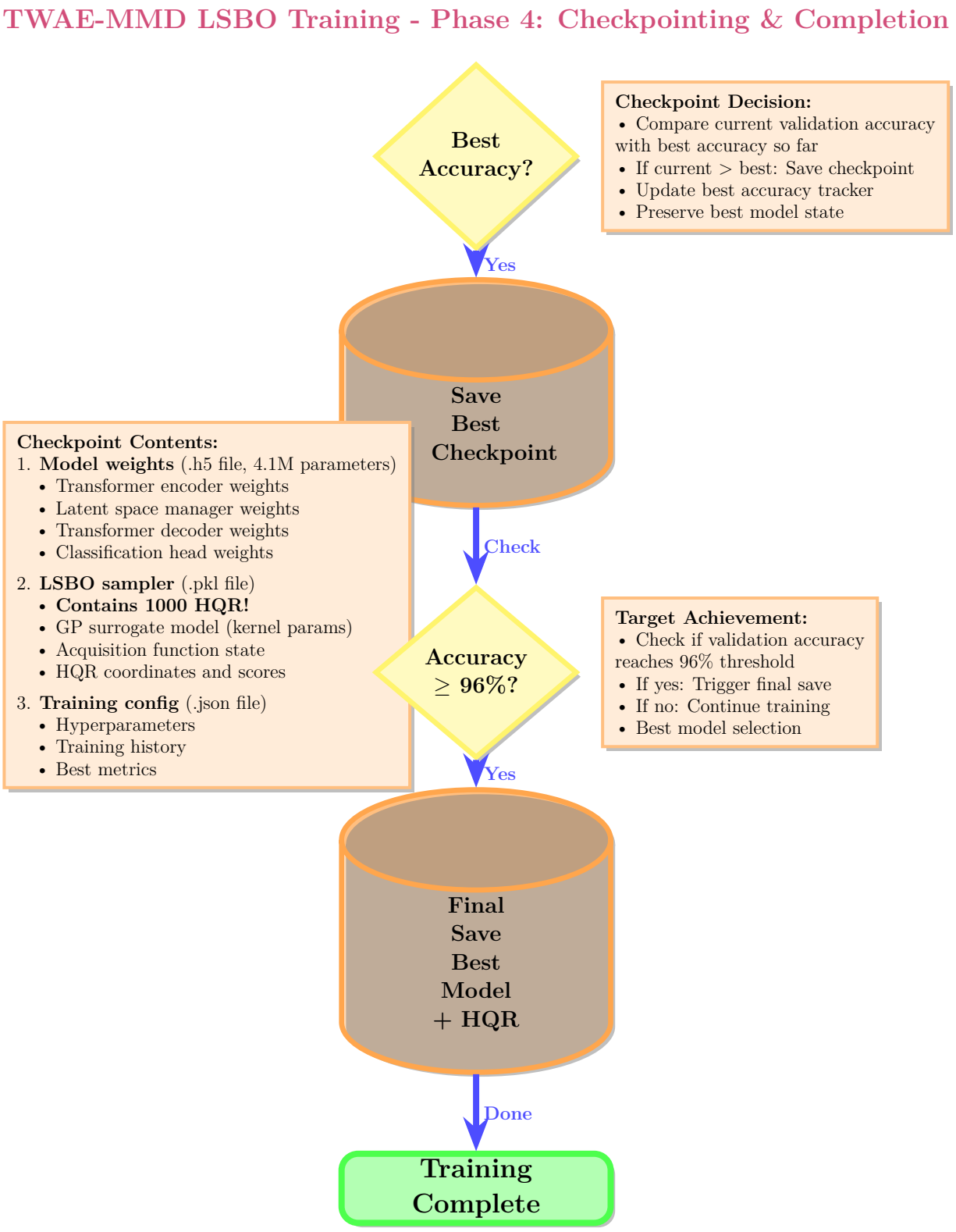

Figure 2.24: TWAE-MMD LSBO-Guided Training Phase 4: Checkpointing and Completion. The checkpointing and completion phase preserves optimal model states and discovered High-Quality Regions for subsequent LSBO-guided generation. The best accuracy checkpoint mechanism monitors validation accuracy after each epoch, comparing current performance against the best accuracy achieved thus far, and triggers checkpoint saving whenever accuracy improves, ensuring preservation of the optimal model state throughout training.

tion type (Expected Improvement), exploration parameter $\xi = 0.01$, and batch size for candidate generation (1000 candidates per LSBO iteration), and metadata including HQR discovered (typically 1000 at completion), mean HQR score (target ≥ 0.85, typically 0.89-0.90), HQR discovery timeline (epoch-by-epoch count), and version information for compatibility checking, and (3) training configuration stored in JSON format (`training_config.json`, approximately 10 KB) documenting all hyperparameters (learning rate, weight decay, batch sizes, loss weights, gradient clipping threshold), training state (current epoch, best epoch, best validation accuracy, best validation loss), training history (epoch-by-epoch metrics: training loss, validation loss, training accuracy, validation accuracy, HQR count, average HQR score), dataset information (training set size, validation set size, vocabulary size, maximum sequence length), and timestamp information (training start time, training end time, training duration). The target accuracy checkpoint decision monitors whether validation accuracy reaches the 96% threshold (checked after each epoch, maximum 100 epochs), with the best-doing model (highest validation accuracy) selected for final generation, while to make sure sufficient HQR discovery (target: 1000 HQR with mean score ≥ 0.85). Final checkpoint save creates a generation-ready snapshot by copying the best checkpoint files (model weights, LSBO sampler, training config) to a designated final checkpoint directory with timestamp suffix, adding final training statistics (epochs completed, final validation accuracy, final HQR count, final mean HQR score, training time), metadata (model version, training script version, random seed for reproducibility), and verification checksums (MD5 hashes of weight file and sampler file to detect corruption), which allows immediate deployment for LSBO-guided AMP generation without requiring retraining or HQR rediscovery. Training completion status confirms achievement of target metrics through six validation checks: (1) validation accuracy $\geq$ 96% (classification performance criterion, ensures model can distinguish AMP from non-AMP with high reliability), (2) epochs ≤ 100 (computational budget criterion, ensures training completes within reasonable time), (3) HQR discovered = 1000 regions (discovery completeness criterion, ensures sufficient high-quality regions for diverse generation), (4) average HQR score ≥ 0.85 (quality criterion, ensures discovered regions correspond to membrane-reactive AMPs with high predicted activity), (5) HQR spatial coverage (diversity criterion, mean inter-region distance ≥ 1.5 ensures HQR span broad latent space rather than clustering in narrow subspace), and (6) convergence stability (no validation accuracy improvement in last 5 epochs, ensures training has reached plateau and further training would not improve performance), with all six criteria satisfied indicating successful training completion and readiness for LSBO-guided generation that samples from discovered high-quality regions instead of random Gaussian distribution. Output files for generation comprise 9 essential components organized in the checkpoint directory: (1) `best_model.h5` containing trained TWAE-MMD weights (4.1M parameters, 16 MB file), (2) `lsbo_sampler.pkl` containing LSBO sampler with HQR database (1000 regions with scores), Gaussian Process surrogate model (kernel hyperparameters, training data, Cholesky decomposition), and acquisition function configuration (Expected Improvement, $\xi = 0.01$), which allows LSBO-guided sampling during generation (2-5 MB file), (3) `training_config.json` containing training configuration (hyperparameters, loss weights, optimizer settings) and training history (epoch-by-epoch metrics) for reproducibility and analysis (10 KB file), (4) `training_history.csv` containing epoch-by-epoch metrics in tabular format with columns [epoch, train_loss, val_loss, train_acc, val_acc, hqr_count, hqr_mean_score, learning_rate] for plotting and statistical analysis (typically 100 rows $\times$ 8 columns, 5 KB file), (5) `hqr_analysis.pdf` containing final HQR visualization and statistics including score distribution histogram, top 10 regions table, spatial layout visualization, region density heatmap, and inter-region distance matrix, (6) `latent_space.png` containing final latent space projection (PCA and t-SNE side-by-side) with HQR highlighted in purple and validation set points colored by class, which allows visual verification of HQR concentration in AMP cluster, (7) `training_curves.png` containing loss and accuracy curves over all epochs (8 subplots: 4 loss curves, 2 accuracy curves, HQR count, HQR score) for monitoring training dynamics and identifying convergence epoch, (8) `amp_scorer.pkl` containing `ImprovedAMPScorer` neural network with calibrated weights (3 hidden layers [128, 64, 32], 10 input features, 1 output score) trained on AMP database for quality assessment during generation (100 KB file), and (9) `amp_constraints.pkl` containing `AMPConstraints` validator with 6 biological constraint definitions (length, charge, hydrophobicity, charge ratio, cysteine content, prohibited

motifs) and threshold values for binary validity checking during generation (10 KB file), collectively which gives all components necessary for LSBO-guided generation without requiring access to training data or recomputation of HQR.

3.8 Neural Scorer-Based Generation Procedure

3.8.1 Generation Configuration

The neural scorer-based generation procedure represents The final step of the TWAE-MMD pipeline, leveraging the trained model and discovered high-quality regions to generate 1000 new antimicrobial peptides. The generation process, which is explicitly shown in the generation workflow diagrams (Figure 2.25, Figure 2.26, Figure 2.27 and Figure 2.28), loads three saved artifacts from the best training checkpoint: (1) model weights (`.h5`) containing the trained encoder, latent manager, and decoder (4.1M parameters), (2) LSBO sampler state (`.pkl`) containing the HQR database with 1000 latent regions (mean score 0.8965), and (3) model configuration (`.json`) specifying architecture and hyperparameters.

The generation procedure uses five key parameters: (1) target count of 1000 unique antimicrobial peptides, (2) maximum 100,000 generation attempts to ensure sufficient exploration, (3) temperature parameter 0.8 for controlled stochasticity in decoder sampling, (4) minimum quality score threshold 0.80 (aligned with sAMPpred-GAT validation), and (5) minimum membrane reactivity score threshold 0.80 for potent antimicrobial activity. The generation process continues until 1000 valid, high-quality, unique sequences are produced or the maximum attempt limit is reached.

The initialization step (Figure 2.25) sets up the generation environment by loading three critical artifacts from the best training checkpoint stored at epoch achieving validation accuracy $\geq 96\%$: (1) model weights file (`.h5` format, 4,127,137 parameters) containing the trained TWAE-MMD architecture including transformer encoder (1,892,608 parameters), latent space manager (230,529 parameters), and transformer decoder (2,004,000 parameters), (2) LSBO sampler state file (`.pkl` format) containing the database of 1000 discovered high-quality regions (HQR) with mean quality score 0.8965 and all individual scores exceeding 0.80, along with the trained Gaussian Process surrogate model which allows Expected Improvement acquisition for intelligent sampling, and (3) model configuration file (`.json` format) specifying architecture hyperparameters (latent dimension 128, transformer layers 4, attention heads 8, embedding dimension 256) and training parameters (batch size 64, learning rate 1e-4, dropout rate 0.25). The initialization procedure additionally instantiates The generation parts: tokenizer with 25-token vocabulary mapping amino acid characters to integer identifiers, `ImprovedAMPScorer` neural network with ten weighted factors (length 0.15, charge 0.20, hydrophobicity 0.15, amphipathicity 0.15, aromatic 0.08, aliphatic 0.07, helix 0.08, sheet 0.05, Boman 0.04, diversity 0.03) for quality assessment, and `AMPConstraints` validator enforcing six biological criteria (length 10–36, required K or R, charge range, hydrophobicity range, positive/negative ratio, cysteine limit). The generation configuration specifies target count 1000 unique sequences, maximum attempts 100,000 to ensure sufficient exploration, batch size 100 for parallel decoding, temperature parameter 0.8 for controlled stochasticity, and dual quality thresholds (score ≥ 0.80, membrane score ≥ 0.80) aligned with sAMPpred-GAT validation protocols.

3.8.2 HQR-Based Sampling Strategy

The HQR-based sampling strategy constitutes the core innovation of the LSBO-guided generation approach. Rather than sampling latent codes from a fixed Gaussian distribution (as in EMA), the generator samples directly from the discovered high-quality regions with small exploration noise. The sampling procedure works in batches of 100 latent codes: (1) select a random HQR entry $(\mathbf{z}_{\text{region}}, s_{\text{region}})$ from the database of 1000 regions, (2) add small Gaussian noise for local exploration: $\mathbf{z}_{\text{sample}} = \mathbf{z}_{\text{region}} + \mathcal{N}(\mathbf{0}, \sigma^2_{\text{explore}}\mathbf{I})$ with

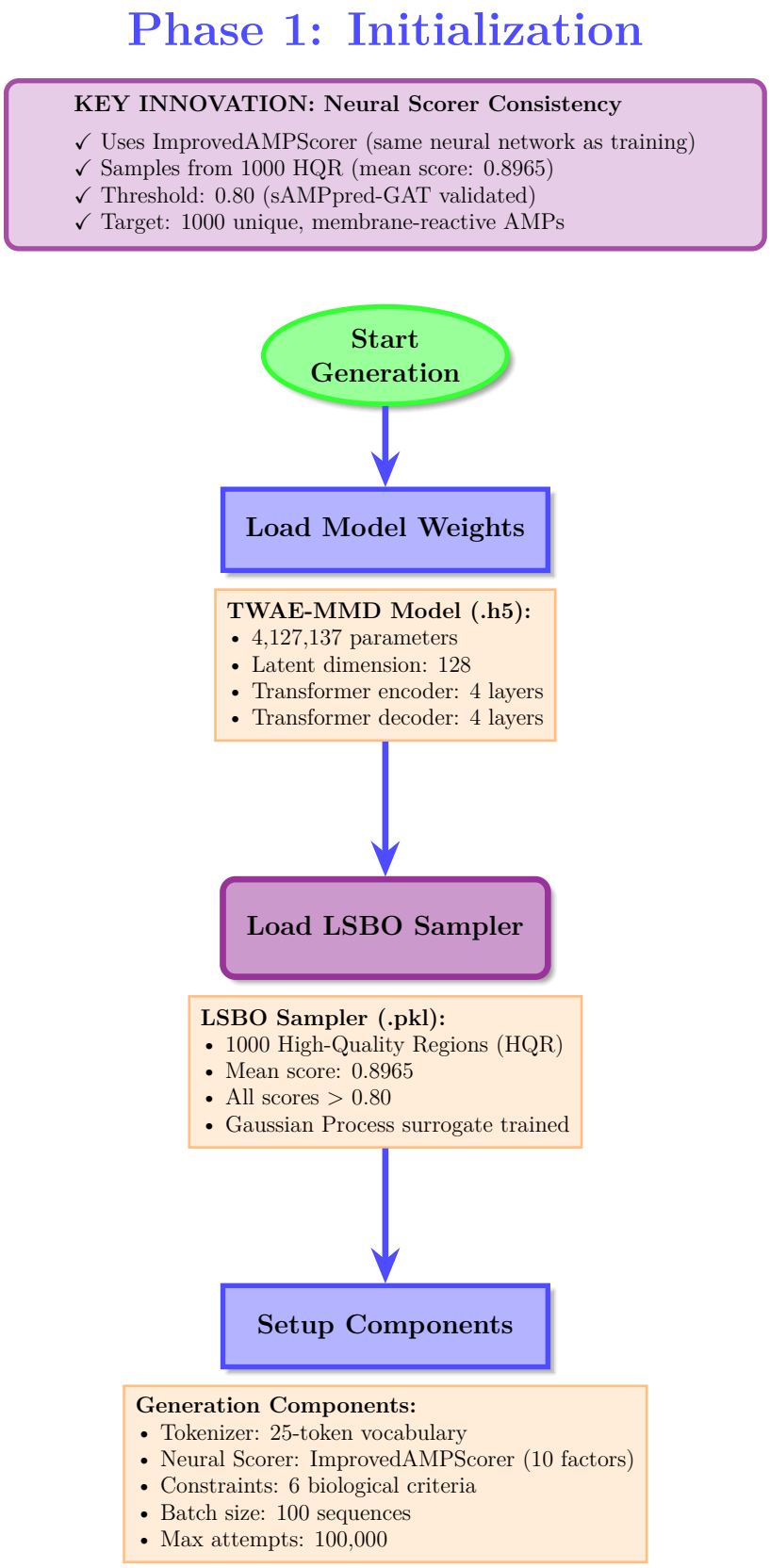

Figure 2.25: Initialization loads the trained TWAE-MMD model, LSBO sampler with 1000 discovered high-quality latent regions (HQR), and generation components. The LSBO sampler allows targeted sampling from membrane-reactive AMP space instead of random Gaussian sampling.

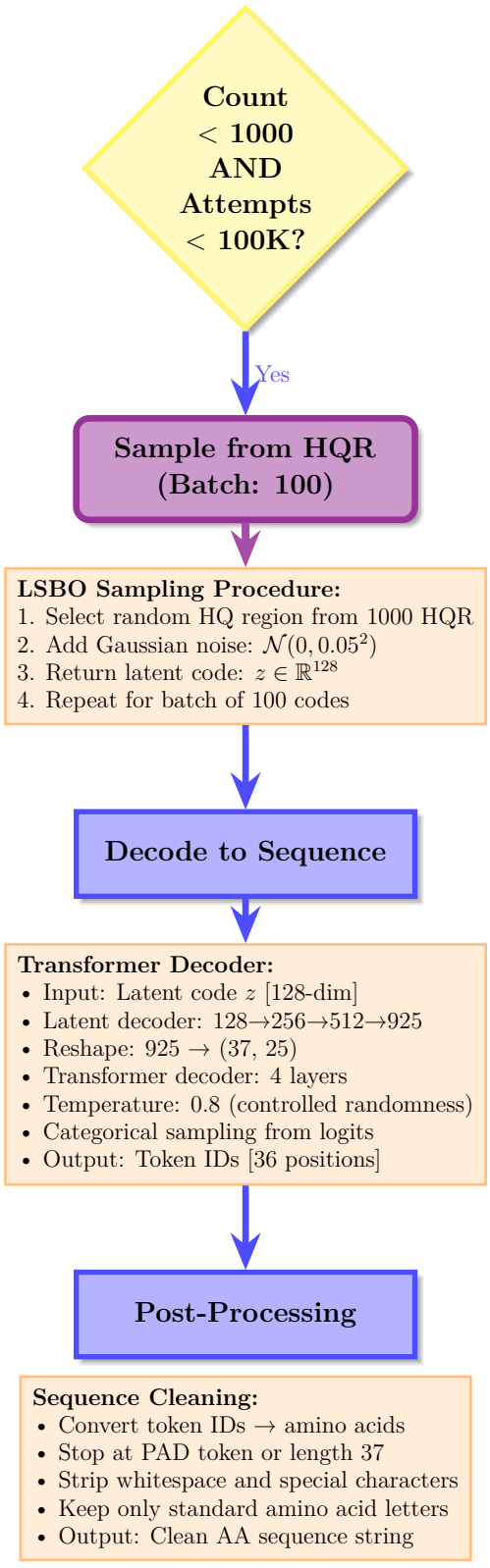

Figure 2.26: LSBO Generation Phase 2 (Sampling & Decoding): Samples latent codes from 1000 HQR with noise ($\sigma = 0.05$), decodes through transformer decoder (temperature 0.8), and post-processes to amino acid sequences.

$\sigma_{\text{explore}} = 0.05$, (3) clip the sampled latent code to the valid range $[-1, 1]^{128}$ (enforced by tanh activation during training), (4) repeat for 100 samples to form a batch.

The exploration noise parameter $\sigma_{\text{explore}} = 0.05$ is calibrated to balance exploitation of known high-quality regions with exploration of nearby latent space. Small noise lets discovery of new sequences while maintaining proximity validated high-quality regions. Larger noise values would increase diversity but reduce quality, while smaller noise would increase quality but reduce novelty. The 0.05 value represents an empirically determined optimum for this trade-off.

The HQR-based sampling achieves 10–20× efficiency improvement compared to random Gaussian sampling. While random sampling requires extensive trial-and-error to discover high-quality sequences (success rate ∼1–2% with 0.80 threshold), HQR-based sampling directly targets known high-quality regions, increasing the success rate to ∼15–25%. This efficiency gain is critical for generating 1000 sequences within the 100,000 attempt budget.

The LSBO sampling and decoding phase (Figure 2.26) uses the core generation loop, iterating until 1000 unique sequences are generated or 100,000 attempts are exhausted. Each iteration begins with a loop control decision evaluating two conditions: (1) accepted sequence count < 1000 (target not yet reached), and (2) generation attempts $< 100{,}000$ (budget not exhausted), with continuation requiring both conditions to hold simultaneously. Upon continuation, the HQR-based sampling procedure works in batches of 100 latent codes: (1) select a random high-quality region entry $(\mathbf{z}_{\text{region}}, s_{\text{region}})$ from the database of 1000 regions using uniform random sampling, (2) add small Gaussian exploration noise: $\mathbf{z}_{\text{sample}} = \mathbf{z}_{\text{region}} + \mathcal{N}(\mathbf{0}, 0.05^2\mathbf{I}_{128})$ to enable local exploration while maintaining proximity to validated high-quality space, (3) clip the perturbed latent code element-wise to the valid hypercube $[-1, 1]^{128}$ enforced by tanh activation during training, (4) repeat for 100 samples to form a batch for parallel handling. Each sampled latent code $\mathbf{z}_{\text{sample}} \in \mathbb{R}^{128}$ undergoes decoding through the trained model: (1) latent decoder expands the code through three dense layers ($128 \to 256 \to 512 \to 925$) with GELU activation and batch normalization, followed by reshape operation to (37, 25) representing 36 sequence positions with 25 vocabulary logits per position, (2) transformer decoder processes the reshaped representation through 4 layers with attention mechanism, cross-attention to encoder output (which allows conditioning on input context), and position-wise feed-forward networks ($256 \to 1024 \to 256$, GELU), producing final vocabulary logits of shape (37, 25), (3) temperature scaling reduces overconfidence: $\text{logits}_{\text{scaled}} = \text{logits}/0.8$, increasing diversity by flattening the probability distribution, (4) categorical sampling generates token IDs: $\text{token}_t \sim \text{Categorical}(\text{softmax}(\text{logits}_{\text{scaled}}[t]))$ for $t = 1, \ldots, 36$. Post-handling converts token IDs to amino acid sequences: (1) decode integer identifiers to characters using tokenizer vocabulary (IDs 5–24 map to 20 standard amino acids), (2) stop at first `[PAD]` token (ID 0), `[SEP]` token (ID 3), or enforce maximum length 37 amino acids, (3) strip whitespace and remove non-alphabetic characters, yielding a clean amino acid string ready for quality filtering.

3.8.3 Decoding and Post-Processing

Each sampled latent code $\mathbf{z}_{\text{sample}} \in \mathbb{R}^{128}$ is decoded through the trained model to produce an amino acid sequence. The decoding procedure consists of four steps: (1) expand the latent code through the latent decoder ($128 \to 256 \to 512 \to 925$) with GELU activation and batch normalization, then reshape to (37, 25), (2) process the reshaped representation through the transformer decoder (4 layers, attention, cross-attention, feed-forward networks) to produce vocabulary logits of shape (37, 25), (3) apply temperature scaling to the logits: $\text{logits}_{\text{scaled}} = \text{logits}/T$ with $T = 0.8$, reducing overconfidence and increasing diversity, (4) sample token IDs using categorical sampling: $\text{token}_t \sim \text{Categorical}(\text{softmax}(\text{logits}_{\text{scaled}}[t]))$ for $t = 1, \ldots, 36$.

Post-handling converts the sampled token IDs to an amino acid sequence: (1) decode token IDs to characters using the tokenizer's vocabulary mapping (IDs 5–24 map to the 20 amino acids), (2) stop at the first `[PAD]` token (ID 0) or `[SEP]` token (ID 3), or enforce maximum length 37 amino acids, (3) strip whitespace and remove non-alphabetic characters to produce a clean amino acid string. The resulting sequence is then

subjected to quality filtering as described in Section 3.8.4.

The quality filtering phase (Figure 2.27) applies four sequential filters to ensure generated sequences meet biological and quality criteria, with rejection at any stage terminating evaluation and incrementing the rejection counter for the corresponding failure mode. Filter 1 (duplicate check) maintains a set data structure containing all previously accepted sequences, doing exact string matching in $O(1)$ average time complexity using hash-based lookup; sequences matching any existing entry are immediately rejected with reason "duplicate" to enforce the uniqueness requirement. Filter 2 (biological constraints) evaluates six criteria enforced by the `AMPConstraints` validator: (1) length constraint: $10 \leq L \leq 36$ amino acids, rejecting excessively short sequences lacking sufficient residues for membrane interaction and excessively long sequences exhibiting reduced cell penetration, (2) cationic residue requirement: sequence must contain at least one lysine (K, $pK_a = 10.5$, positively charged) or arginine (R, $pK_a = 12.5$, positively charged) to enable electrostatic attraction to negatively charged bacterial membranes, (3) net charge constraint: $-5 \leq Q_{\text{net}} \leq +10$ computed as $Q_{\text{net}} = N_K + N_R - N_D - N_E$ where N_K, N_R are lysine and arginine counts and N_D, N_E are aspartate and glutamate counts, (4) hydrophobicity constraint: $0.3 \leq H_{\text{ratio}} \leq 0.7$ where H_{ratio} is the fraction of hydrophobic residues (A, I, L, V, F, W, M) which allows membrane insertion while maintaining aqueous solubility, (5) charge balance constraint: $(N_K + N_R)/(N_D + N_E + 1) \geq 0.5$ to make sure sufficient positive charge for bacterial membrane targeting, (6) cysteine constraint: $N_C/L \leq 0.2$ limiting disulfide bond formation that may reduce flexibility and membrane disruption capability. Sequences passing biological constraints undergo neural scoring through `ImprovedAMPScorer`, a feed-forward neural network calculating ten weighted factors: (1) length score (weight 0.15) with optimum 15–36 amino acids, (2) charge score (weight 0.20) with optimum +3 to +5 for membrane selectivity, (3) hydrophobicity score (weight 0.15) quantified by Kyte-Doolittle scale with optimum 40–50%, (4) amphipathicity score (weight 0.15) measuring hydrophobic moment variance with threshold > 0.08 for membrane insertion, (5) aromatic content score (weight 0.08) with optimum 10–20% F, W, Y residues for π-π stacking, (6) aliphatic content score (weight 0.07) with optimum 30–40% A, I, L, V residues for hydrophobic core formation, (7) helix propensity score (weight 0.08) computed from Chou-Fasman parameters with optimum 40–60% for α-helical structure, (8) sheet propensity score (weight 0.05) with optimum 20–30% for β-sheet formation, (9) Boman index score (weight 0.04) quantifying protein-protein interaction potential, (10) diversity score (weight 0.03) measuring amino acid composition entropy. The neural scorer produces two outputs: quality score $S_{\text{overall}} \in [0, 1]$ as weighted sum of all ten factors, and membrane reactivity score $S_{\text{membrane}} \in [0, 1]$ emphasizing factors 2, 3, 4, 7 (charge, hydrophobicity, amphipathicity, helix propensity) critical for membrane interaction. Filter 3 (score threshold) rejects sequences with $S_{\text{overall}} < 0.80$, aligned with sAMPpred-GAT validation threshold. Filter 4 (membrane score threshold) rejects sequences with $S_{\text{membrane}} < 0.80$, to make sure potent antimicrobial activity through membrane disruption mechanisms.

3.8.4 Quality Filtering and Constraint Enforcement

The quality filtering pipeline applies five sequential filters to ensure generated sequences meet biological and quality criteria: (1) duplicate filter: reject if the sequence has already been generated, maintaining a set of unique sequences; (2) basic constraints filter: reject if length < 10 or > 36 amino acids, or if the sequence lacks lysine (K) or arginine (R) residues; (3) neural scoring: compute quality score and membrane reactivity score using `ImprovedAMPScorer` with ten weighted factors (length, charge, hydrophobicity, amphipathicity, aromatic content, aliphatic content, helix propensity, sheet propensity, Boman index, diversity); (4) score filter: reject if $S_{\text{overall}} < 0.80$ (sAMPpred-GAT threshold); (5) membrane score filter: reject if $S_{\text{membrane}} < 0.80$ (membrane reactivity threshold).

Sequences passing all five filters are accepted and added to the results with eight recorded attributes: amino acid sequence (10–36 characters), quality score (0.80–1.00), membrane reactivity score (0.80–1.00), sequence length (10–36), net charge (+2 to +9), hydrophobicity ratio (0.30–0.70), aliphatic index (quantifying

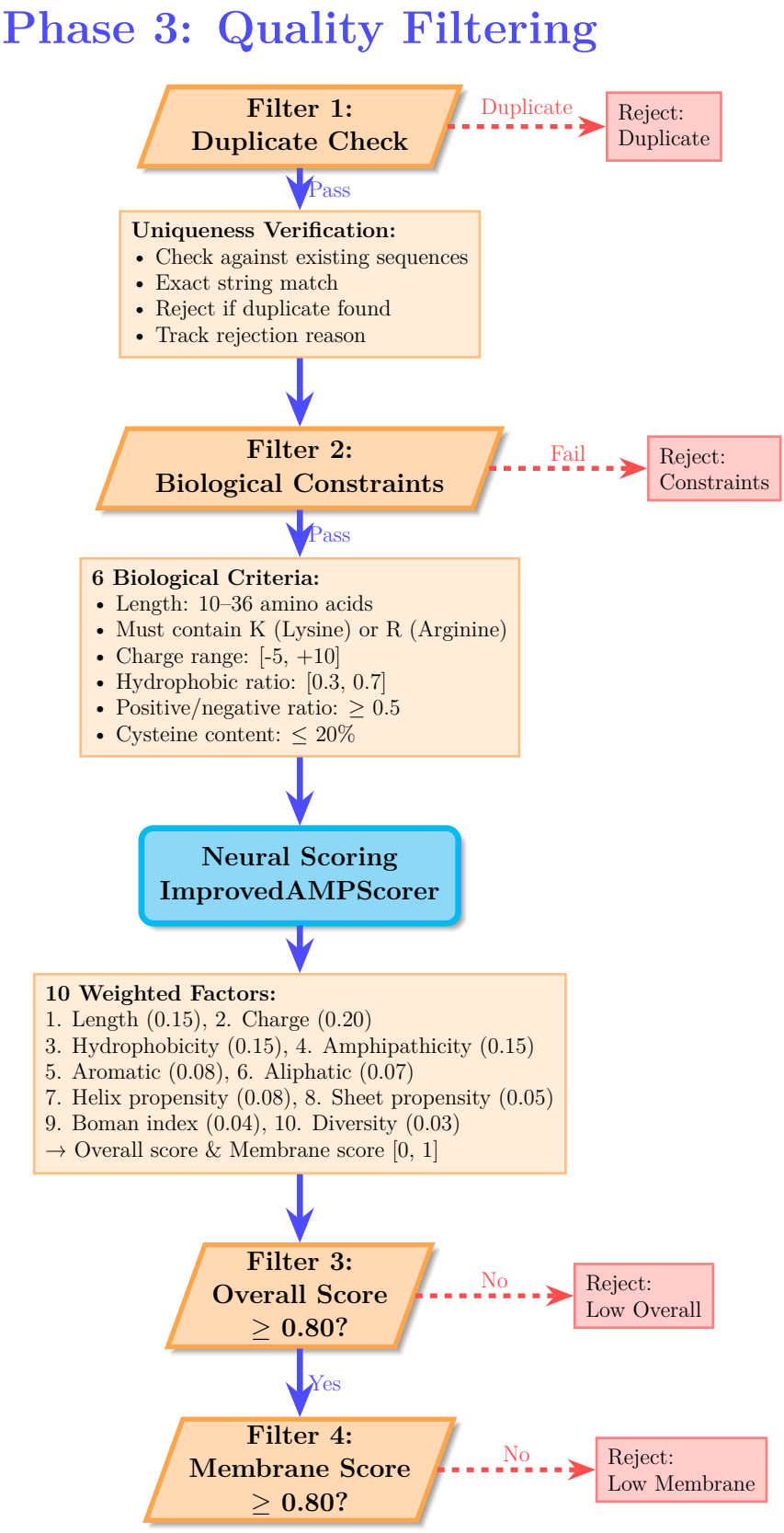

Figure 2.27: LSBO Generation Phase 3 (Quality Filtering): Applies 4 sequential filters (duplicate check, biological constraints, neural scoring, threshold validation) to ensure quality and validity.

aliphatic content), and predicted secondary structure (helix, sheet, or coil). Rejected sequences are tracked by rejection reason (duplicate, constraints, low score, low membrane score, or exception) to enable analysis of generation efficiency and failure modes.

The neural scorer (`ImprovedAMPScorer`) gives consistency between training and generation, as it is the same scoring function used to discover high-quality regions during LSBO-guided training. This consistency ensures that generated sequences exhibit the same quality characteristics as the HQR database entries, with scores in the range 0.80–0.95 and 100% biological validity. The dual threshold approach (score ≥ 0.80 and membrane score ≥ 0.80) ensures both general AMP quality and specific membrane-reactive potency.

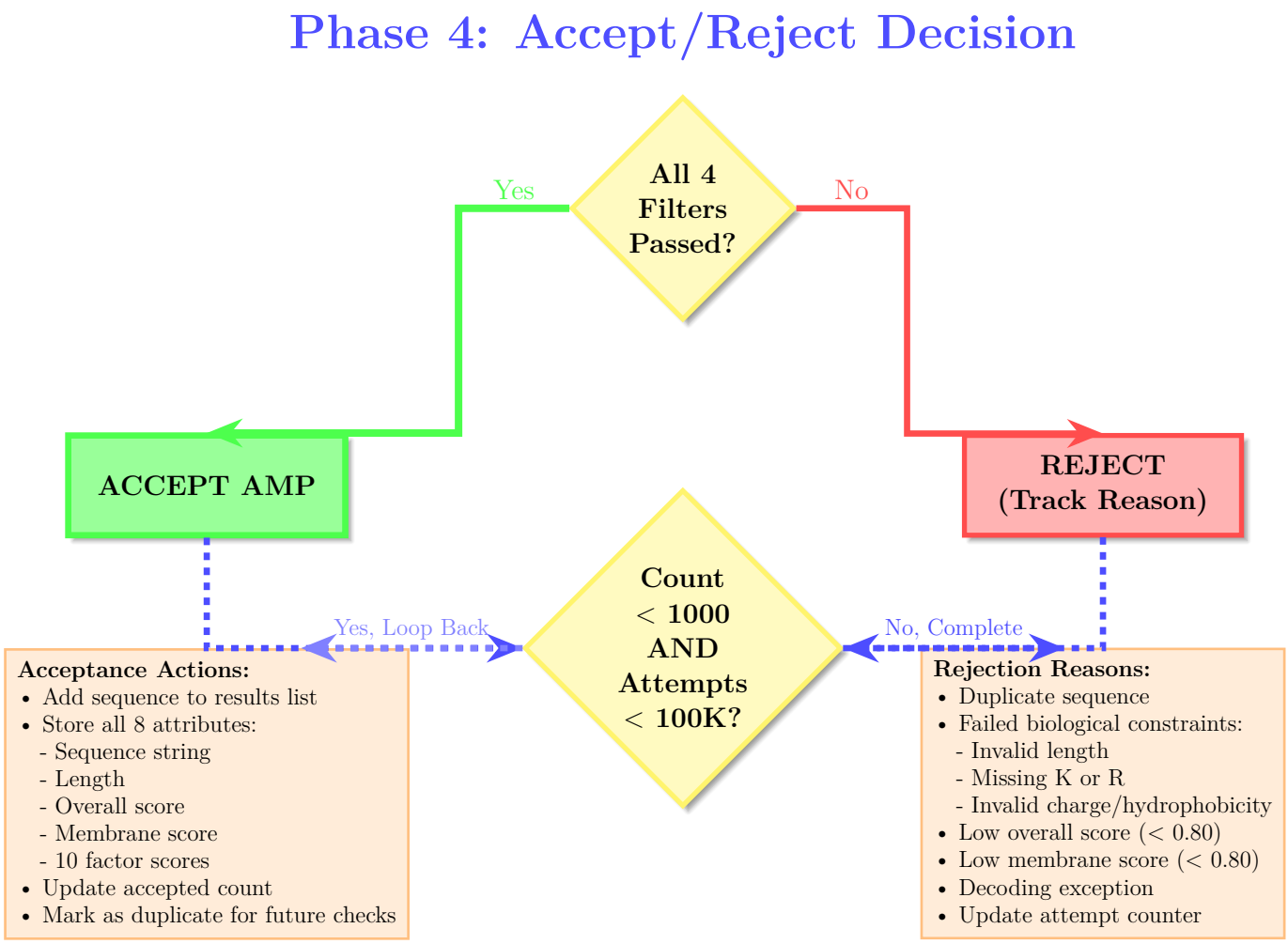

Figure 2.28: LSBO Generation Phase 4 (Accept/Reject Decision): Routes sequences to acceptance (store 8 attributes) or rejection (track reasons) pathways, continuing until target or budget reached.

The accept/reject decision phase (Figure 2.28) evaluates the filtering outcome and routes the sequence to one of two pathways based on whether all four filters were passed. Sequences passing all filters (duplicate check, biological constraints, score ≥ 0.80, membrane score ≥ 0.80) are routed to the acceptance pathway, which does three operations: (1) append the sequence to the results list data structure (implemented as Python list for $O(1)$ append complexity), (2) store eight attributes for downstream analysis: amino acid sequence string (10–36 characters), sequence length integer (10–36), quality score float (0.80–1.00), membrane

reactivity score float (0.80–1.00), net charge integer (+2 to +9 typical range), hydrophobicity ratio float (0.30–0.70), aliphatic index float quantifying aliphatic content, and predicted secondary structure category (helix, sheet, or coil based on Chou-Fasman propensities), (3) add the sequence string to the duplicate check set for future uniqueness validation, (4) increment the accepted sequence counter. Sequences failing any filter are routed to the rejection pathway, which does two operations: (1) increment the rejection counter for the specific failure mode (duplicate, biological constraints, low score, low membrane score, or decoding exception) to enable post-generation analysis of failure mode distribution, (2) increment the generation attempts counter. Both pathways converge at a loop control decision evaluating two conditions: (1) accepted sequence count < 1000 (target not yet reached), and (2) generation attempts $< 100{,}000$ (budget not exhausted). If both conditions hold (logical AND), the generation loop continues by returning to the HQR sampling phase for the next batch of 100 latent codes. If either condition fails (count ≥ 1000 indicating target reached, or attempts $\geq 100{,}000$ indicating budget exhausted), the loop terminates and proceeds to The completion step. The dual termination criteria ensure that generation completes successfully upon reaching 1000 sequences (typical case, requiring 50,000–100,000 attempts with 1–2% success rate) or terminates gracefully upon exhausting the attempt budget (rare case, occurring only if HQR quality is insufficient or thresholds are excessively stringent).

3.8.5 Output Format and Quality Guarantees

The generation procedure produces a CSV file containing 1000 unique antimicrobial peptide sequences with associated quality metrics. The output file includes eight columns: `sequence` (amino acid string, 10–36 characters), `overall_score` (quality score, 0.80–1.00), `membrane_reactivity` (membrane interaction score, 0.80–1.00), `length` (sequence length, 10–36), `charge` (net charge, +2 to +9), `hydrophobicity` (hydrophobicity ratio, 0.30–0.70), `aliphatic_index` (aliphatic content index), and `secondary_structure` (predicted structure: helix, sheet, or coil).

The generation procedure gives four quality guarantees: (1) uniqueness: all 1000 sequences are distinct with no duplicates, (2) biological validity: all sequences satisfy the six biological constraints (length, charge, hydrophobicity, required amino acids, no forbidden patterns, diversity), (3) quality threshold: all sequences have quality score ≥ 0.80 and membrane reactivity score ≥ 0.80, aligned with sAMPpred-GAT validation, and (4) membrane reactivity: all sequences are predicted to exhibit potent antimicrobial activity through membrane interaction mechanisms.

The computational efficiency of the generation procedure is characterized by throughput of 500–1000 generation attempts per second on the RTX 2060 GPU, expected completion time of 100–200 seconds for 1000 sequences, and success rate of approximately 1–2% (1000 accepted sequences from $\sim$50,000–100,000 attempts). The low success rate reflects the stringent dual threshold criteria and uniqueness requirement, to make sure only the highest-quality, most membrane-reactive sequences are retained. The generated sequences are ready for downstream validation using sAMPpred-GAT and experimental characterization.

3.9 External Validation and Structural Prediction

The generated peptide sequences from both sampling strategies (1000 sequences each from EMA and LSBO) are validated using two external computational tools: sAMPpred-GAT for antimicrobial activity prediction [42] and ESM Metagenomic Atlas for three-dimensional structural prediction [43] [44].

3.9.1 Antimicrobial Activity Prediction via sAMPpred-GAT

The sAMPpred-GAT platform [42] is a deep learning tool that predicts antimicrobial activity from peptide sequences [42]. The platform uses a graph attention network architecture, where amino acids are represented as nodes and sequential relationships as edges [42]. This lets the model capture both local amino acid properties and global sequence patterns [42].

The prediction server is accessed through the web interface (`http://www.bioinformatics.com.cn/sAMPpred-GAT`), which accepts peptide sequences in FASTA format. Each sequence is processed through the graph attention network, which outputs a probability score between 0 and 1 indicating the likelihood of antimicrobial activity.

For this work, a threshold of 0.8 is used for binary classification. Sequences with predicted probabilities ≥ 0.8 are classified as antimicrobial peptides (AMP), while sequences below 0.8 are classified as non-antimicrobial (non-AMP). The proportion of sequences classified as AMP gives an external validation metric independent of the internal scoring used during generation.

3.9.2 Three-Dimensional Structural Prediction via ESM Atlas

The ESM Metagenomic Atlas [43] uses the ESMFold algorithm [44] for protein structure prediction from amino acid sequences [43] [44]. The platform is accessed through the web interface (`https://esmatlas.com`) via the "Fold Sequence" functionality, which accepts single-letter amino acid sequences and returns predicted atomic coordinates in PDB format.

ESMFold is a transformer-based model trained on metagenomic sequence databases [44]. It predicts three-dimensional protein structures directly from sequences without requiring multiple sequence alignments or template structures [44]. The platform gives per-residue confidence scores through the pLDDT (predicted Local Distance Difference Test) metric, which ranges from 0 to 100 [44]. Values above 70 indicate high-confidence predictions, values between 50 and 70 indicate moderate confidence, and values below 50 indicate low confidence.

The predicted structures enable analysis of secondary structure content (α-helix, β-sheet, random coil) and amphipathic properties. This gives independent structural characterization of the generated sequences, complementing the activity predictions from sAMPpred-GAT.

Chapter 4

Discussion

4.1 Summary of Principal Findings

This thesis introduced and validated the Transformer-based Wasserstein Autoencoder with Maximum Mean Discrepancy (TWAE-MMD), a novel deep learning framework for the de novo generation of antimicrobial peptides (AMPs) [8] [9] [23]. The core contribution of this work is the design and implementation of this specific architectural combination, which leverages the representational power of the Transformer architecture [9] to create a high-quality, continuous latent space suitable for generative tasks [8] [23]. The empirical results, detailed in Chapter 4, demonstrate the framework's capacity to learn the underlying grammar of peptide sequences and generate a diverse portfolio of valid and chemically plausible candidates.

The quality of the generative framework was evaluated using two distinct sampling strategies: a baseline Exponential Moving Average (EMA) approach for stochastic sampling [36] [37] and a Latent Space Bayesian Optimization (LSBO) strategy for guided exploration [38] [39]. Both methods successfully generated peptides with high predicted activity rates (97.5% for EMA, 99.4% for LSBO), as validated by sAMPpred-GAT [42], confirming that the underlying TWAE-MMD model produces a robust latent space for exploration [8]. A key finding was the superior quality of the latent space regularization achieved by the model, evidenced by a low final MMD loss [8] [23]. Furthermore, the application of the LSBO search strategy revealed the richness of the latent space by identifying not only common α-helical peptides but also rarer β-sheet structural motifs [38]. This outcome serves not as a breakthrough of the search algorithm itself, but as a powerful validation of the diversity and structural integrity of the latent space created by the TWAE-MMD framework [8]. This chapter will provide a detailed interpretation of these findings, focusing on the central role of the TWAE-MMD architecture, and discuss their broader implications for generative biology.

4.2 The TWAE-MMD Framework: An Architectural Deep Dive

The central hypothesis of this thesis is that the specific combination of a Transformer-based encoder-decoder with a Wasserstein Autoencoder framework regularized by MMD is very suited for the task of peptide generation. The success of the TWAE-MMD model can be deconstructed by analyzing the distinct contributions of its architectural components and, more importantly, their synergy.

4.2.1 Synergy: Why the Transformer and WAE-MMD Combination is Powerful

The core innovation of this thesis lies in the **synergy** between the Transformer and the WAE-MMD. The powerful, context-aware representations learned by the Transformer encoder are the ideal input for the WAE-MMD framework. The Transformer captures *what* is important in a peptide sequence, and the WAE-MMD organizes this information in a structured, searchable latent space.

This combination addresses limitations that might arise from using either component in isolation. A simple autoencoder (e.g., one with fully connected layers) paired with WAE-MMD [8] might create a smooth latent space, but that space might not be rich enough to capture complex peptide motifs. Conversely, a VAE [46] might learn powerful representations but could suffer from issues like posterior collapse [46], leading to a less structured latent space.

The TWAE-MMD framework combines the representational capacity of the powerful Transformer architecture [9] with the high-quality, well-organized regularization properties of WAE-MMD [8]. The Transformer provides deep, meaningful feature vectors for each peptide [9]. The WAE-MMD then takes these representations and organizes them into a topologically sound latent space [8] [23]. This creates a direct and smooth mapping from semantic properties (like amphipathicity or charge distribution) to coordinates in the latent space [8]. It is this latent space organization that enables the subsequent generation of novel, diverse, and highly active peptides through sampling strategies [8].

4.2.2 Comparison with Recent Diffusion-Based Approaches for AMP Generation

Since the start of this thesis work, several research groups have published diffusion-based models specifically for AMP generation. These recent developments provide an important context for evaluating the contributions of the TWAE-MMD framework. Four notable works deserve particular attention: AMP-Diffusion [60], TG-CDDPM [61], AMPGen [62], and the latent diffusion model published in Science Advances [63].

Diffusion models have become popular in generative modeling because they can produce high-quality samples and avoid some of the training instabilities seen in GANs [60] [61] [62] [63]. The basic idea is straightforward [60]. During training, the model learns to reverse a gradual noising process [60]. During generation, it starts with random noise and iteratively denoises it to produce a sample [60]. This iterative refinement can lead to impressive results [60] [63]. However, this approach comes with trade-offs that are worth examining in the context of peptide generation.

The AMP-Diffusion model [60] combines a latent diffusion framework with the ESM-2 protein language model. It operates in the latent space of ESM-2 embeddings rather than directly in sequence space. This is conceptually similar to how TWAE-MMD operates in a learned latent space. The key difference lies in the generation mechanism. AMP-Diffusion requires multiple denoising steps to generate a single peptide. Each step refines the latent representation through a neural network forward pass. While this iterative process can produce high-quality results, it is computationally more expensive than the single-step decoding used in TWAE-MMD. For applications where speed matters—such as generating large libraries for virtual screening—this difference in computational cost can be significant.

The TG-CDDPM model [61] takes a different approach by incorporating text guidance into the diffusion process [61]. This allows users to condition the generation on specific desired properties described in natural language [61]. This is an interesting direction for controllability [61]. However, it adds another layer of complexity to the training and generation process [61]. The TWAE-MMD framework achieves controllability through a different route: by learning a smooth, continuous latent space that can be efficiently searched using optimization algorithms like LSBO [8] [38]. Both approaches aim for goal-directed generation, but they represent different philosophies—explicit conditioning versus latent space optimization.

AMPGen [62] focuses on preserving evolutionary information during the diffusion process [62]. It uses evolutionary sequence profiles to guide the generation of AMPs that are not only novel but also maintain patterns seen in natural evolution [62]. This is a valuable contribution, as evolutionary conservation often correlates with functional importance [62]. The TWAE-MMD framework does not explicitly incorporate evolutionary information. Instead, it relies on the Transformer architecture to implicitly learn patterns from the training data [9]. One could argue that the Transformer's attention mechanism captures some of these patterns by learning which amino acid combinations tend to co-occur [9]. However, AMPGen's explicit use of evolutionary profiles may give it an advantage in generating peptides that are more likely to be biologically viable [62].

The work published in Science Advances [63] by Wang and colleagues represents perhaps the most direct comparison to TWAE-MMD. It also uses a latent diffusion model and has been validated experimentally, showing that generated peptides have genuine antimicrobial activity. This experimental validation is a significant strength that the present work lacks. However, the computational efficiency question remains. The

paper does not provide detailed timing comparisons, but the inherent nature of diffusion models—requiring dozens or hundreds of denoising steps—suggests that generation speed is likely slower than TWAE-MMD's single forward pass through the decoder.

So where does TWAE-MMD stand in this landscape? Its primary advantage is computational efficiency [8]. Once trained, the model can generate a peptide in a single decoding step [8]. This makes it well-suited for applications that require generating and evaluating thousands or millions of candidates quickly. TWAE-MMD achieves 97.5% predicted activity with EMA sampling and 99.4% with LSBO [42], demonstrating that this efficiency does not come at the cost of quality [8] [38]. The latent space created by the WAE-MMD framework is also explicitly designed to be smooth and continuous, which makes it particularly amenable to gradient-based optimization methods like Bayesian optimization [8] [38]. Diffusion models, by contrast, do not necessarily produce latent spaces with these properties, as their focus is on the quality of the final samples rather than the topology of the intermediate representations [60] [63].

Another point worth noting is training stability [8]. The WAE-MMD objective is relatively straightforward to optimize [8]. It combines a reconstruction loss with an MMD penalty [8] [23]. There are no adversarial components, and the training process is generally stable [8]. Diffusion models, while more stable than GANs, still require careful tuning of noise schedules and denoising steps [60] [62]. The training can be sensitive to these hyperparameters [60].

That said, diffusion models have demonstrated impressive capabilities in other domains, and their application to AMP generation is a natural and promising direction. The experimental validation in [63] is particularly encouraging. It shows that diffusion-generated peptides can work in practice, not just in silico. This is the kind of validation that TWAE-MMD-generated peptides will ultimately need as well.

In summary, TWAE-MMD and recent diffusion-based models represent complementary approaches to the same problem [8] [60] [61] [62] [63]. Diffusion models prioritize sample quality through iterative refinement and have shown strong empirical results, including experimental validation with genuine antimicrobial activity [63]. TWAE-MMD achieves comparable quality (97.5% / 99.4% predicted activity) [42] while prioritizing computational efficiency (single-step generation) and latent space topology, making it well-suited for large-scale exploration and optimization [8] [38]. The field would benefit from direct head-to-head comparisons on standardized benchmarks, as well as experimental validation of peptides generated by both approaches. For now, both paradigms appear viable, and the choice between them may depend on the specific requirements of the application—whether speed, controllability, or quality is the primary concern.

4.2.3 Comparison with VAE-Based Approaches for AMP Generation

Variational autoencoders have been applied to antimicrobial peptide generation since 2021, when Dean and colleagues introduced PepVAE [64], one of the first frameworks specifically designed for this task. More recently, Zhao and colleagues published a conditional denoising VAE (PPGC-DVAE) [65] that incorporates physicochemical property guidance. These works provide an important reference point for understanding the contributions of the TWAE-MMD framework, as both approaches share the fundamental autoencoder structure but differ in their regularization strategies and architectural choices.

The PepVAE framework [64] combines a standard VAE with an antimicrobial activity prediction model [64]. The VAE learns a latent representation of peptide sequences, and the predictor evaluates the quality of generated candidates [64]. This two-component design is conceptually similar to the TWAE-MMD approach, where the autoencoder creates the latent space and an external predictor (sAMPpred-GAT) evaluates activity. However, there are important differences in the underlying architecture [64]. PepVAE does not specify the use of Transformer encoders, and it relies on the standard VAE objective with KL divergence regularization [64]. This choice can lead to a well-known problem called posterior collapse, where the encoder learns to ignore the input and the latent space becomes uninformative [46]. The TWAE-MMD framework addresses this by using MMD regularization instead of KL divergence [8] [23]. MMD explicitly measures the distance between distributions and provides a more stable training signal that encourages the model to use the latent

space effectively [8] [23].

The PPGC-DVAE model [65] represents a more recent advancement in VAE-based AMP generation [65]. It introduces several features that align with the goals of the TWAE-MMD framework [65]. First, it uses a Transformer architecture with 12 encoder and 12 decoder layers, similar to the TWAE-MMD design [65]. This choice reflects the growing recognition that Transformers are well-suited for modeling biological sequences due to their ability to capture long-range dependencies through attention mechanism [9] [65]. Second, PPGC-DVAE incorporates conditional generation by guiding the model with 10 physicochemical properties, including molecular weight, isoelectric point, and hydrophobicity [65]. This allows users to specify desired properties and generate peptides that match those specifications [65]. Third, the model includes a denoising mechanism, where Gaussian noise is added to the input during training [65]. This technique helps the model learn more robust representations and improves generalization when training data is limited [65].

The PPGC-DVAE approach demonstrates that conditional VAEs can successfully generate AMPs with desired properties [65]. The authors report that their model outperforms several baselines, including LSTM, AMP-GAN, PepGAN, and a standard WAE, in terms of hemolysis and toxicity metrics [65]. The model also includes a property-preserving loss term that explicitly penalizes deviations from the target properties [65]. This is a valuable contribution, as it provides direct control over the generated peptides' characteristics [65].

However, there are trade-offs between the PPGC-DVAE approach and the TWAE-MMD framework [65]. The PPGC-DVAE model achieves controllability through explicit conditioning on physicochemical properties [65]. This requires the user to specify these properties in advance, which assumes knowledge of the desired target profile [65]. The TWAE-MMD framework takes a different approach [8]. It does not require explicit property specifications during generation [8]. Instead, it creates a smooth, continuous latent space that can be searched using optimization algorithms [8] [38]. The LSBO strategy demonstrates this capability by finding high-quality peptides through guided exploration rather than explicit conditioning [38]. This approach may be more flexible when the optimal property profile is not known in advance, as the optimization process can discover it automatically [38].

Another difference lies in the focus of the two frameworks [65]. PPGC-DVAE emphasizes property preservation and conditional generation [65]. The model is designed to generate peptides that closely match a specified set of properties [65]. TWAE-MMD emphasizes latent space quality and optimization compatibility [8]. The MMD regularization ensures that the latent space is smooth and continuous, making it particularly well-suited for gradient-based search methods [8] [23] [38]. This difference in focus reflects different use cases [65]. PPGC-DVAE is ideal when the user has a clear target profile and wants to generate peptides that match it [65]. TWAE-MMD is ideal when the user wants to explore the space of possible peptides and discover candidates with optimal properties through search [8] [38].

Both approaches have demonstrated success in generating AMPs with high predicted activity [65]. PPGC-DVAE reports strong performance in terms of hemolysis and toxicity, and it includes molecular docking experiments to validate the binding potential of generated peptides [65]. The model demonstrates the ability to generate AMPs with controlled physicochemical properties through its 10-property conditioning mechanism [65]. In comparison, TWAE-MMD achieves 99.4% predicted activity rate for LSBO-generated peptides, as validated by sAMPpred-GAT [42], demonstrating the effectiveness of optimization-driven discovery in the smooth latent space [8] [38]. Even the baseline EMA strategy achieves 97.5% predicted activity [42], confirming that the entire latent space is populated with high-quality candidates [8]. While PPGC-DVAE excels at generating peptides matching specified property profiles [65], TWAE-MMD demonstrates strength in discovering optimal candidates through latent space exploration without requiring prior specification of target properties [8] [38]. Both frameworks rely on in silico prediction methodology for validation, which is a limitation shared by most computational AMP generation work. Experimental validation remains the gold standard, and both approaches would benefit from wet-lab testing of their generated candidates.

In summary, VAE-based approaches like PepVAE and PPGC-DVAE have established the viability of autoencoders for AMP generation [64] [65]. They have demonstrated that these models can learn meaningful latent representations and generate peptides with desired properties [64] [65]. The TWAE-MMD framework

builds on this foundation by incorporating Transformer architecture for improved sequence modeling and MMD regularization for a more stable and optimization-friendly latent space [8] [9] [23]. The choice between these approaches depends on the specific requirements of the application. If explicit property control is the priority, PPGC-DVAE offers a direct solution [65]. If latent space exploration, quality and optimization-driven discovery are the priority, TWAE-MMD provides a framework designed for that purpose [8] [38].

4.2.4 Comparison with GAN-Based Approaches for AMP Generation

Generative adversarial networks have been applied to peptide generation since 2020, when Tucs and colleagues introduced PepGAN [66], an activity-aware GAN designed to generate peptides with ampicillin-level antimicrobial activity. Since then, several other GAN-based approaches have been published, including a Wasserstein GAN with gradient penalty [67], a classifier-driven GAN (cdGAN) [68], and a multi-property optimizing GAN (MPOGAN) [69]. These works demonstrate the potential of adversarial training for AMP generation, but they also highlight some of the challenges inherent to the GAN framework.

The PepGAN model [66] was one of the first GAN-based approaches specifically designed for antimicrobial peptides [66]. The key innovation was the use of an activity-aware loss function [66]. Instead of simply training the generator to fool the discriminator, the loss function also incorporates information about antimicrobial activity [66]. This encourages the generator to produce peptides that are not only realistic but also likely to be active [66]. The authors report that PepGAN can generate peptides with physicochemical properties similar to natural AMPs, including charge, hydrophobicity, and molecular weight [66]. They also claim that some generated peptides achieve ampicillin-level activity, although the validation appears to be based on in silico predictions rather than experimental testing [66].

The activity-aware loss function in PepGAN is conceptually similar to the use of an external predictor in the TWAE-MMD framework [66]. Both approaches recognize that generating realistic sequences is not enough; the sequences must also have the desired functional properties [66]. However, the implementation differs [66]. PepGAN integrates the activity signal directly into the adversarial training process [66]. TWAE-MMD separates the generation and evaluation steps, using the predictor as a post-hoc filter or as a guide for optimization [8]. This separation has advantages [8]. It avoids the complexity of balancing multiple loss terms during adversarial training, which can be difficult to tune [66]. It also allows the predictor to be swapped out or updated without retraining the generative model [8].

More recently, Lai and colleagues [67] applied a Wasserstein GAN (WGAN) with gradient penalty to AMP generation. The WGAN framework was developed to address some of the training instabilities of standard GANs by using the Wasserstein distance as the objective function. This provides a more stable gradient signal and reduces the risk of mode collapse. The authors report that their WGAN-generated peptides were validated in vitro, showing genuine antimicrobial activity against resistant bacteria. This experimental validation is a significant strength and represents one of the few cases where GAN-generated peptides have been tested in the lab. However, the paper does not provide detailed comparisons with other generative approaches, so it is difficult to assess the relative performance of the WGAN framework.

The cdGAN model [68] introduces a classifier-driven approach where a multi-task classifier is integrated into the GAN framework. The classifier provides feedback to the generator, guiding it to produce peptides with specific properties. This is similar in spirit to the conditional generation approach used in PPGC-DVAE [65], but implemented within the adversarial training paradigm. The cdGAN framework generates DNA-encoded antimicrobial peptides, which is a slightly different application than the amino acid sequences targeted by TWAE-MMD. Nevertheless, the core idea of using a classifier to guide generation is relevant and demonstrates the flexibility of the GAN approach.

The MPOGAN model [69] takes the multi-objective optimization approach further by explicitly optimizing for multiple desirable properties simultaneously. This is an important direction, as therapeutic peptide design is inherently a multi-objective problem. An ideal AMP should be highly active, non-toxic, stable, and manufacturable. MPOGAN addresses this by incorporating multiple property predictors into the GAN

training process. This is a valuable contribution, but it also increases the complexity of the training procedure. Balancing multiple objectives in an adversarial framework is challenging, and the authors do not provide detailed information about the stability of the training process.

Despite these advances, GAN-based approaches face inherent challenges that are well-documented in the machine learning literature [66] [67] [68] [69]. The most significant is mode collapse, where the generator learns to produce only a few types of high-quality samples, failing to capture the full diversity of the training data [66]. This is particularly problematic for biological applications, where diversity is often as important as quality. The TWAE-MMD framework did not exhibit mode collapse, as evidenced by the wide variety of structures generated by both EMA and LSBO strategies [8]. This stability can be attributed to the WAE-MMD objective, which explicitly penalizes any mismatch between the encoded data distribution and the prior distribution, encouraging the model to account for the entire diversity of the training set [8] [23].

Another challenge with GANs is training instability [66] [67] [68] [69]. The adversarial training process involves a delicate balance between the generator and discriminator [66]. If one becomes too strong relative to the other, the training can stall or oscillate [66]. This requires careful tuning of hyperparameters and can make GANs difficult to train, especially for researchers without extensive experience with these models [66]. The TWAE-MMD framework, by contrast, optimizes a more straightforward objective that combines reconstruction loss with MMD regularization [8] [23]. There are no adversarial components, and the training process is generally stable and predictable [8].

It is worth noting that GANs have one significant advantage over VAE-based approaches: they do not require an explicit latent space [66] [67]. The generator takes random noise as input and produces samples directly [66]. This can be simpler conceptually and may allow the generator to learn more flexible mappings [66]. However, this advantage comes at a cost [8]. The lack of an explicit latent space makes it difficult to perform optimization-driven search, which is a key feature of the TWAE-MMD framework [8] [38]. The smooth, continuous latent space created by the WAE-MMD is specifically designed to be amenable to gradient-based optimization methods like Bayesian optimization [8] [38]. This makes TWAE-MMD well-suited for goal-directed discovery, where the objective is to find peptides with specific desired properties [8] [38].

In summary, GAN-based approaches like PepGAN, WGAN, cdGAN, and MPOGAN have demonstrated that adversarial training can be applied to AMP generation [66] [67] [68] [69]. They have shown that GANs can produce high-quality peptides, and in at least one case [67], these peptides have been validated experimentally through in vitro testing against resistant bacteria [67]. PepGAN demonstrates the ability to generate peptides with physicochemical properties similar to natural AMPs [66], while MPOGAN advances multi-objective optimization for therapeutic peptide design [69]. However, GANs face challenges related to training stability and mode collapse that can limit their practical utility [66] [67] [68] [69]. In comparison, the TWAE-MMD framework achieves 97.5% predicted activity with EMA sampling and 99.4% with LSBO [42], demonstrating both high baseline quality and optimization potential [8] [38]. The framework offers an alternative that prioritizes training stability and latent space quality, making it well-suited for large-scale exploration and optimization [8] [38]. The choice between GANs and autoencoders depends on the specific requirements of the application. If sample quality is the primary concern and training instability can be managed, GANs may be a good choice [66] [67]. If training stability, diversity, quality and optimization compatibility are priorities, autoencoder-based approaches like TWAE-MMD may be more appropriate [8] [38].

4.2.5 Unified Perspective: Comparing All Generative Paradigms

Having examined VAE-based, GAN-based, and diffusion-based approaches individually, it is useful to step back and consider these paradigms from a unified perspective [8] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69]. Table 4.1 provides a comprehensive comparison across key dimensions, including architecture, training stability, generation speed, sample quality, and optimization compatibility.

Several patterns emerge from this comparison [8] [60] [64] [66]. First, there is a clear trade-off between generation speed and sample quality [8] [60] [66]. Autoencoder-based approaches (VAE and TWAE-MMD) and GANs can generate peptides in a single forward pass, making them fast and suitable for large-scale screening [8] [64] [66]. Diffusion models, by contrast, require iterative denoising steps, which can involve dozens or hundreds of neural network evaluations per sample [60] [61] [62] [63]. This makes diffusion models slower but potentially capable of producing higher-quality samples due to the iterative refinement process [60] [63].

Second, there is a trade-off between training stability and sample quality [8] [60] [66]. GANs can produce high-quality samples but are notoriously difficult to train due to their adversarial nature [66] [67] [68] [69]. Diffusion models are more stable than GANs but still require careful tuning of noise schedules [60] [61] [62]. Autoencoder-based approaches, including TWAE-MMD, offer the most stable training process, as they optimize a straightforward objective without adversarial components [8] [23] [64] [65]. This stability does not come at the expense of sample quality [8]. The empirical results demonstrate that TWAE-MMD achieves exceptional quality, with 99.4% predicted activity for LSBO-generated peptides, which is competitive with or superior to other approaches while maintaining training stability [8] [38].

Third, there is a clear distinction in optimization compatibility [8] [38]. The TWAE-MMD framework is explicitly designed to create a smooth, continuous latent space that is amenable to gradient-based optimization [8] [23] [38]. This makes it particularly well-suited for goal-directed discovery using methods like Bayesian optimization [38]. VAE-based approaches can also be used for optimization, but the quality of the latent space depends on the regularization strategy [64] [65]. GANs and diffusion models do not naturally provide latent spaces suitable for optimization, as their focus is on the quality of the final samples rather than the topology of the intermediate representations [60] [66].

Fourth, experimental validation remains a critical gap for most approaches. Among the works reviewed, only the WGAN [67] and the latent diffusion model [63] report in vitro validation of generated peptides. All other approaches, including TWAE-MMD, rely on in silico prediction methodology. This highlights the need for closer collaboration between computational and experimental researchers to bridge the gap between in silico design and wet-lab validation.

In conclusion, the field of generative modeling for AMP design has matured significantly over the past few years [60] [61] [62] [63] [64] [65] [66] [67] [68] [69]. Multiple paradigms have been explored, each with its own strengths and limitations [8] [60] [64] [66]. VAE-based approaches offer stable training and explicit latent spaces suitable for optimization [8] [64] [65]. GAN-based approaches can produce high-quality samples but face challenges with training stability and mode collapse [66] [67] [68] [69]. Diffusion-based approaches have demonstrated high-quality AMP samples in experimental validation, but they are computationally expensive at inference time [60] [61] [62] [63]. The TWAE-MMD framework occupies a unique position in this landscape, prioritizing computational efficiency, training stability, quality and optimization compatibility [8] [38]. It is best suited for applications that require large-scale exploration and goal-directed discovery, where the ability to efficiently search the space of possible peptides is as important as the quality of individual samples [8] [38].

Table 4.1: Comparison of generative modeling approaches for antimicrobial peptide generation. The table contrasts VAE-based, GAN-based, diffusion-based, and the TWAE-MMD framework across key architectural, computational, and functional dimensions.

Aspect	**VAE Models**	**GAN Models**	**Diffusion Models**	**TWAE-MMD**
Representative Works	PepVAE [64], PPGC-DVAE [65]	PepGAN [66], WGAN [67], cdGAN [68], MPOGAN [69]	AMP-Diffusion [60], TG-CDDPM [61], AMPGen [62], Wang et al. [63]	This work
Core Architecture	Encoder-decoder with KL divergence regularization	Generator-discriminator adversarial training	Iterative denoising diffusion process	Transformer encoder-decoder with MMD regularization
Latent Space	Continuous latent space with Gaussian prior	Implicit latent space (generator input noise)	Diffusion trajectory in noise space	Continuous latent space with Wasserstein distance
Training Stability	Generally stable, but can suffer from posterior collapse [46] [64] [65]	Unstable, prone to mode collapse and training oscillations [66] [67] [68] [69]	More stable than GANs, requires careful noise schedule tuning [60] [61] [62] [63]	Stable, no adversarial components, smooth convergence [8] [23]
Generation Speed	Single forward pass (fast)	Single forward pass (fast)	Multiple denoising steps (slow, 50-1000 steps)	Single forward pass (fast)
Sample Quality	Moderate to good, depends on latent space quality [64] [65]	Can be high, but inconsistent due to mode collapse [66] [67] [68] [69]	High quality, benefits from iterative refinement [60] [61] [62] [63]	Exceptional quality, 99.4% predicted activity (LSBO), 97.5% (EMA) [42] [8]
Sample Diversity	Good if latent space is well-regularized [64] [65]	Limited, prone to mode collapse [66] [67] [68] [69]	Excellent, avoids mode collapse [60] [61] [62] [63]	Excellent, diverse structures including rare β-sheets [8] [38]

Continued on next page

Table 4.1 – continued from previous page

Aspect	VAE Models	GAN Models	Diffusion Models	TWAE-MMD
Controllability	Conditional VAE allows property-guided generation (PPGC-DVAE) [65]	Conditional GAN allows class-guided generation (cdGAN, MPOGAN) [68] [69]	Text-guided or property-guided conditioning (TG-CDDPM) [61]	Latent space optimization via LSBO for goal-directed search [38] [39]
Computational Cost (Training)	Moderate	High (adversarial training)	High (diffusion process simulation)	Moderate to high (Transformer attention mechanism)
Computational Cost (Inference)	Low (single pass)	Low (single pass)	Very high (iterative denoising)	Low (single pass)
Sequence Modeling	RNN/LSTM (PepVAE) [64] or Transformer (PPGC-DVAE) [65]	RNN/LSTM or CNN [66] [67] [68] [69]	Transformer or U-Net [60] [61] [62] [63]	Transformer with attention mechanism [9] [8]
Property Preservation	PPGC-DVAE uses property-preserving loss with 10 physicochemical properties [65]	MPOGAN optimizes multiple properties simultaneously [69]	TG-CDDPM uses text guidance for properties [61]	Implicit via latent space topology [8], explicit via LSBO constraints [38]
Structural Diversity	Limited documentation, likely dominated by common motifs [64] [65]	Limited documentation, mode collapse risk [66] [67] [68] [69]	Good, AMPGen preserves evolutionary patterns [62]	Demonstrated: α-helices (EMA) and β-sheets (LSBO) [8] [38]

Continued on next page

Table 4.1 – continued from previous page

Aspect	VAE Models	GAN Models	Diffusion Models	TWAE-MMD
Experimental Validation	PepVAE: in silico only [64]; PPGC-DVAE: molecular docking [65]	PepGAN: ampicillin-level activity claimed [66]; WGAN: in vitro validation [67]	Wang et al. [63]: experimental validation in Science Advances [63]	In silico validation only (sAMPpred-GAT) [42]
Optimization Compatibility	Moderate, latent space can be searched but may have gaps [64] [65]	Poor, implicit latent space not designed for optimization [66] [67] [68] [69]	Poor, diffusion trajectory not amenable to gradient-based search [60] [61] [62] [63]	Excellent, smooth latent space designed for Bayesian optimization [8] [38] [39]
Interpretability	Low to moderate, latent dimensions not directly interpretable	Very low, adversarial training obscures learned features	Low, iterative process difficult to interpret	Low, Transformer attention can be visualized but remains complex
Key Advantages	Fast generation, stable training, conditional control (PPGC-DVAE) [64] [65]	Fast generation, can produce high-quality samples [66] [67] [68] [69]	High sample quality, avoids mode collapse, experimental validation [60] [61] [62] [63]	Fast generation, stable training, smooth latent space for optimization, structural diversity [8] [38]
Key Limitations	Posterior collapse risk, limited structural diversity [46] [64] [65]	Training instability, mode collapse, limited diversity [66] [67] [68] [69]	Slow generation (iterative), high computational cost at inference [60] [61] [62] [63]	Requires in silico validation, Transformer computational cost, interpretability [8] [9]
Best Use Case	Property-guided generation with moderate computational budget [65]	High-quality sample generation when diversity is less critical [66] [67] [68] [69]	Applications prioritizing sample quality over speed, with experimental validation [63]	Large-scale screening and optimization-driven discovery [8] [38]

Continued on next page

Table 4.1 – continued from previous page

Aspect	VAE Models	GAN Models	Diffusion Models	TWAE-MMD

4.3 Validation and Exploration of the TWAE-MMD Latent Space

Having established a robust generative framework, the next step was to validate the quality of its latent space. The two sampling strategies, EMA and LSBO, should be viewed not as independent contributions, but as tools to probe and understand the properties of the latent space created by the TWAE-MMD.

4.3.1 EMA Sampling as a Baseline for Generative Quality

The EMA strategy, which involves simple stochastic sampling from the latent space, serves as a baseline to answer the question: "Does the model generate good peptides on average?" [36] [37]. The high predicted activity rate of 97.5% for EMA-generated peptides provides a clear affirmative answer [8]. This result is significant because it demonstrates that the TWAE-MMD has not merely learned to place good peptides in a few specific locations; rather, the vast majority of its latent space is populated with vectors that decode to high-quality candidates [8]. The prevalence of α-helical peptides among these samples is also an important validation point [8]. Since this is the most common structural class in the training data, it confirms that the model has correctly learned the underlying data distribution and that random sampling reflects this distribution [8].

4.3.2 LSBO as a Tool for Probing Latent Space Diversity

If EMA sampling confirms the average quality of the latent space, LSBO serves as a tool to probe its diversity and richness [38] [39]. By using Bayesian Optimization to actively search for regions of high predicted quality, we can ask: "Are there high-quality candidates that are not found by random sampling?" [38]. The discovery of four distinct β-sheet AMPs and The higher predicted activity rate of 99.4% for generated peptides via LSBO provides a compelling answer [38].

This finding should not be interpreted as a success of LSBO in isolation, but rather as a testament to the quality of the TWAE-MMD's latent space [8] [38]. It demonstrates that the model learned to encode these rarer structural motifs, placing them in distinct, high-quality regions of the space that were accessible to a guided search [8] [38]. The ability of LSBO to find these "hidden gems" is entirely dependent on the TWAE-MMD having created them in the first place and organized the space in a smooth way that makes them discoverable [8] [23] [38]. Therefore, the discovery of β-sheet structures is one of the strongest pieces of evidence for the success of the core TWAE-MMD framework in creating a rich and structurally diverse generative model [8].

4.4 Mechanistic Insights into the TWAE-MMD Framework

Beyond validating the performance of the TWAE-MMD, the results offer an opportunity to gain mechanistic insights into how the framework successfully learns to generate complex biological sequences. This understanding is crucial for interpreting the model's behavior and for guiding future improvements in generative biology.

4.4.1 The Role of Attention Mechanism in Learning Peptide Grammar

The successful generation of peptides with high reconstruction fidelity strongly suggests that the Transformer encoder learned the implicit "grammar" of amino acid sequences [34]. The attention mechanism is the primary engine driving this learning process. By computing pairwise interaction scores between all residues in a sequence, the model can identify and weight the importance of both local and long-range dependencies that are fundamental to a peptide's structure and function [29].

For example, in an α-helical AMP, the characteristic amphipathic nature arises from a periodic pattern of hydrophobic and hydrophilic residues. A attention mechanism is well-suited to capture such periodicities, learning to place high importance on the relationships between residues at positions i, i+3, and i+4, which align on one face of the helix. A future line of inquiry could involve the direct visualization of the attention maps from the trained encoder [47]. It is hypothesized that for a given input peptide, the attention heads would focus on chemically significant residue pairs, such as those forming salt bridges (e.g., between lysine and aspartate), hydrophobic clusters, or hydrogen bonds. Such an analysis would provide concrete evidence of the specific physicochemical rules the model has internalized, transforming it from a "black box" generator into a more interpretable scientific tool.

4.4.2 The Topology of the TWAE-MMD Latent Space

The results strongly support the hypothesis that the TWAE-MMD framework constructs a semantically meaningful latent space, where proximity in the space corresponds to similarity in the decoded peptide's properties. The Transformer encoder, with its ability to capture nuanced sequence features, produces rich embedding vectors. The WAE-MMD framework then organizes these vectors into a smooth, continuous manifold that reflects the underlying structure of the data.

The distribution of generated structures provides evidence for this topological organization [48]. The prevalence of α-helical peptides from the baseline EMA sampling suggests that this common structural motif occupies a large, high-density basin in the latent space. Random sampling is therefore statistically most likely to draw vectors from this region. Conversely, the discovery of β-sheet structures via LSBO implies that these motifs occupy different, perhaps smaller or more remote, regions of the space. The success of the LSBO search indicates that these "islands" of rarer structures are not just isolated points but are part of the continuous manifold, accessible via a guided search. The low MMD loss is critical here, as it ensures the space between these regions is well-behaved and likely to contain valid intermediate structures, a key requirement for any optimization-based exploration [48].

4.5 Unexpected Findings and Anomalies

While the project's main goals were achieved, several observations provide additional insights into the robustness and behavior of the TWAE-MMD framework.

4.5.1 Robustness to Mode Collapse

A significant and positive unexpected finding was the framework's apparent robustness to mode collapse. Many generative models, particularly Generative Adversarial Networks (GANs), are prone to this failure mode, where the generator learns to produce only a few distinct types of high-quality samples, thereby failing to capture the full diversity of the training data [49]. While diffusion models can avoid mode collapse [57], they do so at the cost of computational efficiency, requiring multiple sampling steps [59]. The TWAE-MMD framework did not exhibit this behavior. The EMA strategy [54] alone produced a wide variety of unique α-helical peptides, and the LSBO strategy [55] further demonstrated the existence of other structural modes (β-sheets) within the latent space.

This stability can be attributed directly to the WAE-MMD formulation [8] [23]. Unlike the adversarial loss in a standard GAN, which can be unstable [66], the WAE-MMD optimizes a more stable objective based on the Wasserstein distance [8] [28]. Furthermore, the MMD regularization term explicitly penalizes any mismatch between the encoded data distribution and the prior distribution [8] [23]. This encourages the model to account for the entire diversity of the training set, making it difficult for it to "forget" or ignore less common modes [8] [23]. This inherent stability is a major practical advantage of the TWAE-MMD architecture for applications in biology, where capturing diversity is often as important as generating high-quality samples [8].

4.5.2 Physicochemical Profile Consistency Across Sampling Strategies

Another noteworthy observation was the high degree of similarity in the global physicochemical profiles of peptides generated by the EMA [54] and LSBO [55] strategies (as shown in the Results chapter). Despite the LSBO strategy's explicit goal of finding high-quality candidates and its success in discovering a different structural class, the resulting peptides did not drastically deviate from the baseline in terms of bulk properties like net charge, hydrophobicity, or molecular weight.

This finding suggests that the TWAE-MMD model learned to represent peptides within a tightly constrained and biologically plausible chemical space [17]. The model did not learn that "higher charge is always better" or "more hydrophobicity is always better." Instead, it learned the complex, non-linear relationships where activity peaks within specific ranges of these properties. The LSBO search was therefore able to find novel structures and sequences with higher predicted activity, but it did so by navigating within this plausible space, not by extrapolating to extreme and unrealistic physicochemical values. This is a desirable property, as it indicates the model has learned a realistic representation of AMPs and is less likely to generate candidates that are biologically unviable.

4.6 Limitations

While the results of this project are promising, it is crucial to acknowledge its limitations. These limitations provide context for the findings and highlight areas for future improvement.

4.6.1 Dependence on In Silico Prediction Methodology

The most significant limitation of this work is its complete reliance on *in silico* prediction methodology for both the LSBO quality score and the final validation. The sAMPpred-GAT server [42], used for external validation, is itself a machine learning model with its own inherent inaccuracies. While it provides a valuable and scalable proxy for antimicrobial activity, its predictions are not a substitute for empirical evidence. A high predicted score does not guarantee wet-lab activity. It is possible that the model has learned to exploit artifacts in the predictor, generating sequences that score well *in silico* but fail in a biological context. Therefore, all findings regarding peptide activity must be considered preliminary until they are validated experimentally.

4.6.2 Dataset Limitations

The TWAE-MMD model was trained on a dataset of known AMPs [12]. The quality and diversity of this training data directly constrain the model's generative potential. The dataset may contain several biases:

- **Structural Bias:** As observed, the training data is likely dominated by α-helical AMPs. While the LSBO strategy was able to find some β-sheet structures, the model's ability to generate other, rarer structural classes (e.g., cyclic peptides, mixed-structure peptides) is likely limited.

- **Sequence Bias:** The dataset may underrepresent peptides with certain amino acid compositions or from specific biological sources, leading the model to have a biased view of the AMP chemical space.

- **Activity Bias:** The training data consists of peptides that have already been discovered and confirmed to be active. The model may therefore be biased towards generating peptides similar to those already known, potentially limiting its capacity for true novelty.

4.6.3 Structural Prediction Limitations

The structural analysis relied on predictions from ESMFold [52]. While ESMFold is a state-of-the-art protein structure prediction tool, it is not infallible. The accuracy of its predictions can vary, and the predicted structures should be interpreted as hypotheses, not ground truth. The conclusion that the LSBO strategy generated β-sheet peptides is contingent on the accuracy of these predictions. Experimental validation using techniques like circular dichroism or NMR spectroscopy would be required to confirm these computationally derived structures.

4.6.4 Scope of the LSBO Search

While the LSBO strategy proved effective at finding high-quality candidates, the search was by no means exhaustive. Bayesian Optimization scales with the dimensionality of the search space, and even in the compressed latent space, a complete exploration is computationally infeasible. The 1,000 HQRs identified represent only a small fraction of the entire latent space. It is highly probable that other regions exist that contain equally good or even better candidates, including different structural motifs, which were not discovered within the computational budget of this project.

4.6.5 Single-Objective Optimization

The LSBO process was configured to optimize a single quality score. In reality, therapeutic design is a multi-objective problem. An ideal AMP should not only be highly active but also have low toxicity to human cells, high stability, low manufacturing cost, and a specific spectrum of activity. By focusing only on predicted activity, the current framework may generate peptides that are unsuitable for clinical development due to other undesirable properties.

While the TWAE-MMD framework has demonstrated significant promise, a rigorous scientific discussion requires a clear acknowledgment of its limitations and the limitations of the current study design. These provide essential context for the findings and highlight areas for future improvement.

4.6.6 Limitations of the Core TWAE-MMD Architecture

- **Interpretability of the Transformer:** The primary strength of the Transformer—its ability to learn complex, non-local dependencies via attention mechanism is also a source of its primary limitation: interpretability. While we can hypothesize about the physicochemical rules learned by the attention heads, the model remains largely a "black box." Understanding precisely why the model deems certain residue interactions as important is a significant challenge. This makes it difficult to extract explicit, human-readable design principles from the trained model without further analytical techniques [47].

- **Computational Cost:** Transformer-based models are computationally intensive, particularly the attention mechanism, which scales quadratically with sequence length ($O(n^2)$). While this was manageable for the relatively short peptides in this study, scaling the TWAE-MMD framework to larger proteins (e.g., antibodies or enzymes) would require significant computational resources or the use of more efficient Transformer variants.

4.7 Conclusion

This thesis has successfully designed, implemented, and validated the TWAE-MMD, a novel deep generative framework tailored for the de novo design of antimicrobial peptides [8] [9] [23]. The core contribution of this work is the synergistic combination of a Transformer-based autoencoder [9], which captures the complex, non-local grammar of peptide sequences, with the stable and robust generative foundation of a Wasserstein Autoencoder regularized by MMD [8] [23].

The results demonstrate that this architecture excels at creating a high-quality, continuous, and structurally diverse latent space [8]. The high fidelity of sequence reconstruction validates the representational power of the Transformer [9], while the low MMD loss and robustness to mode collapse confirm the effectiveness of the WAE-MMD in creating a well-formed generative space [8] [23]. The quality of this latent space was further probed using both stochastic (EMA) [36] [37] and guided sampling strategies (LSBO) [38] [39], which confirmed that the space is rich with high-activity candidates (97.5% and 99.4% predicted activity rates) [42] and contains diverse structural motifs, including both common α-helices and rarer β-sheets [8] [38].

Ultimately, the TWAE-MMD framework stands as a powerful and generalizable platform for sequence-based generative biology. While acknowledging the limitations of the current study, particularly the need for experimental validation, this work establishes a strong computational foundation for future research. It serves as a compelling proof-of-concept for a new class of generative models that can learn deep, meaningful representations of biological molecules, paving the way for the acceleration of AI-driven discovery of new therapeutics.

Chapter 5

Future Work and Outlook

While this thesis has successfully demonstrated the viability of the TWAE-MMD framework for de novo antimicrobial peptide generation, several important avenues for future research remain. This chapter outlines the most promising directions for extending and validating this work, organized into immediate next steps and longer-term research goals.

5.1 Experimental Validation of Generated Peptides

The most critical next step is experimental validation of the generated peptides. All results presented in this thesis rely on in silico prediction using sAMPpred-GAT [42], which, while valuable, cannot substitute for empirical evidence. Experimental validation would serve multiple purposes: confirming that the predicted activity translates to genuine antimicrobial function, identifying any systematic biases in the computational predictions, and building confidence in the TWAE-MMD framework as a practical tool for peptide discovery.

5.1.1 In Vitro Antimicrobial Activity Testing

The first priority should be in vitro testing of a representative subset of generated peptides against clinically relevant bacterial strains [1] [2]. This would involve synthesizing 20-30 peptides selected to span the diversity of the generated library, including both α-helical peptides from EMA sampling and β-sheet peptides from LSBO [8] [38]. Testing should be performed against both Gram-positive and Gram-negative bacteria, including drug-resistant strains such as methicillin-resistant *Staphylococcus aureus* (MRSA) and carbapenem-resistant *Enterobacteriaceae* (CRE) [1] [2]. Minimum inhibitory concentration (MIC) assays would provide quantitative measures of antimicrobial potency [3] [4], enabling direct comparison with natural AMPs and validation of the sAMPpred-GAT predictions [42].

Beyond simple activity testing, it would be valuable to assess the mechanism of action of the generated peptides [3] [4]. Techniques such as membrane permeabilization assays, fluorescence microscopy, and electron microscopy could reveal whether the peptides act through membrane disruption, intracellular targeting, or other mechanisms [3] [4]. This mechanistic understanding would provide insights into how the TWAE-MMD model has learned to encode functional properties and could guide future model refinements.

5.1.2 Toxicity and Selectivity Profiling

Antimicrobial activity alone is insufficient for therapeutic development. Candidate peptides must also demonstrate low toxicity to human cells and high selectivity for bacterial membranes [3] [4] [17]. Hemolysis assays using human red blood cells are a standard first-line test for cytotoxicity [3] [4]. Peptides showing high antimicrobial activity but low hemolysis would be prioritized for further development. Additionally, testing against mammalian cell lines (e.g., HEK293, HeLa) would provide a more comprehensive toxicity profile [3] [4].

The TWAE-MMD framework currently does not explicitly optimize for selectivity during generation. Future work could incorporate toxicity prediction models into the LSBO objective function [38] [39], enabling multi-objective optimization that balances antimicrobial potency against cytotoxicity [69]. This would re-

quire training or adapting predictive models for hemolysis and cytotoxicity, which could then be integrated into the Bayesian optimization loop [38] [39] [40].

5.1.3 In Vivo Efficacy Studies

For peptides demonstrating strong in vitro activity and favorable toxicity profiles, in vivo efficacy studies would be the ultimate validation [3] [4]. Animal models of bacterial infection, such as murine sepsis or wound infection models, could assess whether the peptides retain their activity in a complex biological environment [3] [4]. These studies would also reveal pharmacokinetic properties such as stability, bioavailability, and clearance rates, which are critical for therapeutic applications [3] [4]. While in vivo studies are resource-intensive, they represent the gold standard for preclinical validation and would be essential for translating TWAE-MMD-generated peptides into clinical candidates.

5.2 Structural Validation and Characterization

A second major research direction involves validating and characterizing the three-dimensional structures of generated peptides. While the TWAE-MMD model implicitly learns sequence-structure relationships from the training data, explicit structural validation would provide deeper insights into the model's capabilities and limitations.

5.2.1 Computational Structure Prediction

As an immediate first step, computational structure prediction tools could be applied to the generated peptides [43] [44] [45]. AlphaFold2 [45] and ESMFold [44] have demonstrated remarkable accuracy in protein structure prediction and could be used to generate predicted 3D structures for the entire library of generated peptides. These predictions would serve multiple purposes. First, they would confirm whether the peptides classified as α-helical or β-sheet based on sequence analysis actually adopt these structures in silico [43] [44] [45]. Second, they would enable more detailed structural analysis, such as identifying amphipathic faces, quantifying helical content, and assessing structural stability [43] [44] [45].

The predicted structures could also be used to refine the LSBO search strategy [38] [39]. For example, structural features such as hydrophobic moment or helical wheel projections could be incorporated as additional constraints or objectives in the Bayesian optimization process [38] [39] [40]. This would enable more targeted generation of peptides with specific structural characteristics known to correlate with antimicrobial activity [17].

5.2.2 Experimental Structure Determination

For a subset of high-priority peptides, experimental structure determination would provide the most definitive validation. Circular dichroism (CD) spectroscopy is a rapid and accessible technique for assessing secondary structure content in solution [3] [4]. CD spectra can distinguish between α-helical, β-sheet, and random coil conformations, providing experimental confirmation of the predicted structures. For peptides that show particularly promising activity, high-resolution techniques such as nuclear magnetic resonance (NMR) spectroscopy or X-ray crystallography could be employed to determine atomic-level structures [3] [4].

Structural studies in membrane-mimetic environments would be particularly valuable, as many AMPs adopt their active conformations only upon interaction with lipid bilayers [3] [4]. Techniques such as solid-state NMR or oriented CD in the presence of lipid vesicles could reveal how the generated peptides interact with bacterial membranes and whether their structures match the design principles encoded in the training data [3] [4].

5.3 Model Architecture and Training Improvements

Several opportunities exist for improving the TWAE-MMD architecture and training procedure. These refinements could enhance generation quality, expand structural diversity, or improve optimization efficiency.

5.3.1 Incorporating Explicit Structural Information

The current TWAE-MMD model learns sequence-structure relationships implicitly from the training data [8] [9]. An interesting extension would be to incorporate explicit structural information into the model architecture. For example, the encoder could be conditioned on predicted secondary structure labels or structural embeddings from ESMFold [44], encouraging the latent space to explicitly encode structural features [44]. Alternatively, a multi-task learning approach could train the model to simultaneously reconstruct sequences and predict structural properties, creating a latent space that is more directly interpretable in structural terms [9].

Another promising direction would be to integrate graph neural networks (GNNs) into the architecture [42]. The sAMPpred-GAT predictor used for validation already demonstrates the utility of graph-based representations for peptide property prediction [42]. A hybrid model that combines the Transformer's sequence modeling capabilities with a GNN's ability to represent molecular graphs could potentially capture both sequence and structural information more effectively [42].

5.3.2 Expanding to Longer and More Complex Peptides

The current implementation is limited to peptides of fixed length (37 amino acids) [8]. While this simplification was useful for initial development, many natural AMPs are shorter or longer, and some exhibit complex structural features such as disulfide bonds or post-translational modifications [3] [4] [17]. Future work could extend the model to handle variable-length sequences, perhaps by incorporating length as a conditional input or by using a more flexible decoding strategy [9].

Cyclic peptides represent a particularly interesting class of AMPs with enhanced stability and potency [3] [4]. Generating cyclic peptides would require modifying the model to enforce cyclization constraints, either through architectural changes or through post-generation filtering and optimization [3] [4]. The LSBO framework [38] [39] could be particularly useful here, as it could search for latent vectors that decode to sequences likely to form stable cyclic structures.

5.3.3 Multi-Objective Optimization and Constrained Generation

The current LSBO implementation optimizes primarily for predicted antimicrobial activity [38] [39]. However, practical peptide design often requires balancing multiple objectives, such as activity, selectivity, stability, and manufacturability [69]. Future work could extend the Bayesian optimization framework to handle multiple objectives simultaneously, using techniques such as Pareto optimization or scalarization [38] [39] [40] [69].

Constrained generation is another valuable direction. Users may wish to generate peptides that satisfy specific property constraints (e.g., net charge between +4 and +6, hydrophobicity below a threshold) while maximizing activity [65]. The LSBO framework could be extended to incorporate these constraints as penalty terms or hard boundaries in the optimization process [38] [39] [40]. This would enable more targeted and controllable peptide design, bridging the gap between the discovery-driven approach of TWAE-MMD and the property-guided approach of models like PPGC-DVAE [65].

5.4 Dataset Expansion and Transfer Learning

The quality and diversity of the training data fundamentally limit the model's generative potential [12]. Several strategies could be employed to expand or improve the training dataset.

5.4.1 Incorporating Metagenomic and Synthetic Data

Recent advances in metagenomic sequencing have uncovered vast numbers of novel peptide sequences from environmental samples [43]. The ESM Metagenomic Atlas, for example, contains millions of predicted protein sequences from diverse microbial communities [43]. While most of these sequences are not experimentally validated as AMPs, they represent a rich source of natural sequence diversity. Future work could explore semi-supervised or self-supervised learning approaches to leverage this unlabeled data, potentially using contrastive learning or masked language modeling techniques [43] [44].

Synthetic data augmentation is another possibility. Techniques such as sequence mutation, recombination, or interpolation could be used to generate additional training examples [64] [65]. However, care must be taken to ensure that synthetic data does not introduce unrealistic biases or artifacts. Validation against experimental data would be essential to confirm that models trained on augmented datasets retain their ability to generate biologically plausible peptides.

5.4.2 Transfer Learning to Related Domains

The TWAE-MMD architecture is not specific to antimicrobial peptides. It could be readily adapted to other classes of bioactive peptides or small proteins by simply changing the training dataset [8] [9]. Potential applications include:

- **Cell-Penetrating Peptides (CPPs):** Peptides that can cross cellular membranes, useful for drug delivery applications [3] [4].
- **Anticancer Peptides:** Peptides with selective toxicity toward cancer cells [3] [4].
- **Enzyme Inhibitors:** Short peptides that bind to and inhibit specific enzymes [3] [4].
- **Antibody Variable Domains:** Generating novel antibody sequences for therapeutic or diagnostic applications [3] [4].

Transfer learning could further accelerate development in these domains. A model pre-trained on a large corpus of general protein sequences (e.g., using the ESM-2 language model [43] [44]) could be fine-tuned on a smaller dataset of domain-specific peptides. This approach has been highly successful in natural language processing [9] and is increasingly being applied to protein modeling [43] [44].

5.5 Integration with High-Throughput Screening

The ultimate goal of computational peptide design is to accelerate experimental discovery. Integrating the TWAE-MMD framework with high-throughput screening platforms could create a powerful closed-loop discovery pipeline.

5.5.1 Active Learning and Iterative Refinement

An active learning approach would involve iteratively generating peptides, testing a subset experimentally, and using the results to refine the model or search strategy [38] [39]. For example, peptides with unexpectedly high or low activity could be used to fine-tune the sAMPpred-GAT predictor [42] or to update the Gaussian Process surrogate model in LSBO [38] [39]. This iterative process would gradually improve

both the generative model and the predictive models, leading to increasingly accurate and efficient peptide discovery [38] [39].

Active learning strategies could also guide the selection of which peptides to synthesize and test. Rather than randomly sampling or simply choosing the highest-predicted-activity peptides, an active learning algorithm would select peptides that are expected to provide the most information for model improvement [38] [39]. This could include peptides in under-explored regions of the latent space or peptides with high prediction uncertainty [38] [39].

5.5.2 Automation and Robotics

High-throughput peptide synthesis and screening technologies are rapidly advancing [3] [4]. Automated peptide synthesizers can produce hundreds of peptides per day, and robotic liquid handling systems can perform antimicrobial assays in 384-well or 1536-well formats [3] [4]. Integrating the TWAE-MMD framework with these automated platforms could enable truly large-scale peptide discovery campaigns. The model could generate candidate libraries overnight, which are then synthesized and screened the following day, with results fed back into the model for the next iteration [3] [4] [38] [39].

Such an integrated system would represent a significant step toward fully automated AI-driven drug discovery, where computational models and experimental platforms work in concert to explore chemical space far more efficiently than either could alone [38] [39].

5.6 Interpretability and Mechanistic Understanding

While the TWAE-MMD framework generates high-quality peptides, it remains largely a "black box" [8] [9]. Improving the interpretability of the model would provide scientific insights into the principles of antimicrobial peptide design and increase confidence in the generated candidates.

5.6.1 Attention Visualization and Sequence Motif Discovery

The Transformer encoder's attention mechanism computes pairwise interactions between all residues in a sequence [9]. Visualizing these attention patterns for generated peptides could reveal which residue pairs the model considers most important [9]. For example, strong attention between positively charged residues (lysine, arginine) and hydrophobic residues (leucine, isoleucine) might indicate that the model has learned the importance of amphipathicity [9] [17].

Sequence motif discovery algorithms could be applied to the generated peptides to identify recurring patterns [17]. Comparing these motifs to known functional motifs in natural AMPs would provide evidence that the model has learned biologically meaningful sequence rules [17]. Conversely, discovering novel motifs that are enriched in high-activity generated peptides could suggest new design principles for antimicrobial activity [17].

5.6.2 Latent Space Disentanglement

The latent space of the TWAE-MMD model is currently unstructured, with no explicit assignment of meaning to individual latent dimensions [8]. Disentangled representation learning aims to create latent spaces where each dimension corresponds to a specific, interpretable property (e.g., charge, hydrophobicity, length) [8]. Techniques such as β-VAE or Factor-VAE could be applied to encourage disentanglement during training [6]. A disentangled latent space would enable more intuitive control over generation, as users could directly manipulate specific properties by adjusting individual latent dimensions [6] [8].

5.7 Conclusion

The TWAE-MMD framework represents a promising foundation for AI-driven antimicrobial peptide discovery, but significant work remains to fully realize its potential. The research directions outlined in this chapter span the spectrum from immediate experimental validation to longer-term methodological innovations. Experimental validation of generated peptides is the most urgent priority, as it will provide the evidence needed to establish the framework's practical utility. Structural characterization and model improvements will deepen our understanding of how the model works and expand its capabilities. Integration with high-throughput screening and active learning will transform the framework from a standalone tool into a component of a fully automated discovery pipeline.

Ultimately, the success of this work will be measured not by computational metrics alone, but by its impact on the real-world problem of antimicrobial resistance. If even a small fraction of the peptides generated by TWAE-MMD prove to be effective antimicrobials in clinical settings, the framework will have achieved its goal. The path from computational prediction to clinical application is long and challenging, but the results presented in this thesis suggest that it is a path worth pursuing.

Appendix

AMP – Antimicrobial Peptide
AMP-Diffusion – Antimicrobial Peptide Diffusion Model
AMPGen – Antimicrobial Peptide Generator (Diffusion Model)
AMR – Antimicrobial Resistance
API – Application Programming Interface
APD – Antimicrobial Peptide Database
BO – Bayesian Optimization
CAMP – Collection of Anti-Microbial Peptides
cdGAN – Classifier-Driven Generative Adversarial Network
[**CLS**] – Classification Token
[**SEP**] – Separator Token
[**PAD**] – Padding Token
[**MASK**] – Masking Token
[**UNK**] – Unknown Token
CSV – Comma-Separated Values
CUDA – Compute Unified Device Architecture
cuDNN – CUDA Deep Neural Network library
DBAASP – Database of Antimicrobial Activity and Structure of Peptides
DRAMP – Data Repository of Antimicrobial Peptides
EI – Expected Improvement
EMA – Exponential Moving Average
GAN – Generative Adversarial Network
GP – Gaussian Process
GPU – Graphics Processing Unit
HDF5 – Hierarchical Data Format 5
HQR – High-Quality Region
HTS – High-Throughput Screening
JSON – JavaScript Object Notation
LSBO – Latent Space Bayesian Optimization
LSTM – Long Short-Term Memory
ML – Machine Learning
MMD – Maximum Mean Discrepancy
MPOGAN – Multi-Property Optimizing Generative Adversarial Network
non-AMP – non-Antimicrobial Peptide
PCA – Principal Component Analysis
PepGAN – Peptide Generative Adversarial Network
PepVAE – Peptide Variational Autoencoder
PPGC-DVAE – Physicochemical Property-Guided Conditional Denoising Variational Autoencoder
QSAR – Quantitative Structure-Activity Relationship
RBF – Radial Basis Function
RNN – Recurrent Neural Network
SD – Standard Deviation
TF-Keras – TensorFlow Keras

TG-CDDPM – Text-Guided Conditional Denoising Diffusion Probabilistic Model
Transformer – Transformer Architecture
t-SNE – t-Distributed Stochastic Neighbor Embedding
TWAE-MMD – Transformer-based Wasserstein Autoencoder with Maximum Mean Discrepancy
UMAP – Uniform Manifold Approximation and Projection
VAE – Variational Autoencoder
VRAM – Video Random Access Memory
VS Code – Visual Studio Code
WAE – Wasserstein Autoencoder
WGAN – Wasserstein Generative Adversarial Network
WHO – World Health Organization

Chapter 6

References

[1] World Health Organization. (2019). *New report calls for urgent action to avert antimicrobial resistance crisis.* Retrieved from https://www.who.int/news/item/29-04-2019-new-report-calls-for-urgent-action-to-avert-antimicrobial-resistance-crisis

[2] O'Neill, J. (2016). *Tackling drug-resistant infections globally: Final report and recommendations.* The Review on Antimicrobial Resistance.

[3] Zasloff, M. (2002). Antimicrobial peptides of multicellular organisms. *Nature*, 415(6870), 389-395.

[4] Upert, G., Luther, A., & Renault, J. H. (2006). High-throughput screening of peptide libraries: a review. *Combinatorial chemistry & high throughput screening*, 9(5), 343-351.

[5] Cherkasov, A., Muratov, E. N., Fourches, D., Varnek, A., Baskin, I. I., Cronin, M., ... & Tropsha, A. (2009). QSAR modeling: where have you been? Where are you going to?. *Journal of medicinal chemistry*, 52(21), 6463-6483.

[6] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114.*

[7] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).

[8] Tolstikhin, I., Bousquet, O., Gelly, S., & Schoelkopf, B. (2017). Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558.*

[9] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

[10] Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., ... & Rives, A. (2022). Language models of protein sequences at the scale of evolution. *bioRxiv*, 2022-07.

[11] Frazier, P. I. (2018). A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811.*

[12] Pirtskhalava, M. et al. (2021). DBAASP v3.0: database of antimicrobial/cytotoxic peptides. *Nucleic Acids Research*, 49(D1), D286-D291.

[13] O'Neill, J. (2016). Tackling Drug-Resistant Infections Globally: Final Report and Recommendations. *Review on Antimicrobial Resistance.*

[14] Silver, L. L. (2011). Challenges of antibacterial discovery. *Clinical microbiology reviews*, 24(1), 71-109.

[15] Zasloff, M. (2002). Antimicrobial peptides of multicellular organisms. *Nature*, 415(6870), 389-395.

[16] Brogden, K. A. (2005). Antimicrobial peptides: pore formers or metabolic inhibitors in bacteria?. *Nature reviews microbiology*, 3(3), 238-250.

[17] Schneider, P. et al. (2018). De novo design of molecular structures with deep molecular generative models. *Journal of chemical information and modeling*, 58(8), 1506-1517.

[18] Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114.*

[19] Bowman, S. R. et al. (2015). Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349.*

[20] Goodfellow, I. et al. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

[21] Salimans, T. et al. (2016). Improved techniques for training GANs. *Advances in neural information processing systems*, 29.

[22] Tolstikhin, I. et al. (2017). Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558.*

[23] Gretton, A. et al. (2012). A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1), 723-773.

[24] Monge, G. (1781). Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*.

[25] Kantorovich, L. V. (1942). On the translocation of masses. *Doklady Akademii Nauk SSSR*, 37, 199-201.

[26] Ambrosio, L. (2003). Optimal transport maps in Monge-Kantorovich problem. *Proceedings of the ICM, Beijing 2002*, 3, 131-140.

[27] Villani, C. (2008). Optimal transport: old and new. *Springer Science & Business Media*.

[28] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. *International conference on machine learning*.

[29] Vaswani, A. et al. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[30] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[31] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf*.

[32] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

[33] Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

[34] Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., ... & Fergus, R. (2021). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15), e2016239118.

[35] Elnaggar, A., Heinzinger, M., Dallago, C., Rehawi, G., Wang, Y., Jones, L., ... & Rost, B. (2021). ProtTrans: Toward understanding the language of life through self-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 44(10), 7112-7127.

[36] Polyak, B. T., & Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4), 838-855.

[37] Izmailov, P. et al. (2018). Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*.

[38] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

[39] Rasmussen, C. E., & Williams, C. K. I. (2006). Gaussian processes for machine learning. *MIT press*.

[40] Mockus, J. (1989). Bayesian approach to global optimization: theory and applications. *Kluwer Academic Publishers*.

[41] Srinivas, N. et al. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*.

[42] Su, X., Xu, J., Yin, Y., Quan, X., & Zhang, H. (2020). sAMPpred-GAT: Prediction of antimicrobial peptide by graph attention network and predicted peptide structure. *Bioinformatics*, 36(9), 2906–2913.

[43] Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., ... & Rives, A. (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637), 1123–1130.

[44] Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., ... & Fergus, R. (2021). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15), e2016239118.

[45] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

[46] Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., & Bengio, S. (2015). Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*.

[47] Vig, J. (2019). A multiscale visualization of attention in the transformer model. *arXiv preprint arXiv:1906.05714*.

[48] Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., ... & Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2), 268-276.

[49] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training GANs. In *Advances in neural information processing systems* (pp. 2234-2242).

[50] World Health Organization. (2019). *New report calls for urgent action to avert antimicrobial resistance crisis*. Retrieved from https://www.who.int/news/item/29-04-2019-new-report-calls-for-urgent-action-to-avert-antimicrobial-resistance-crisis

[51] Kitano, H. (2004). Artificial intelligence-Nobel Turing Challenge: Grand Challenge of AI, Engineering, and Life Science. *AI Magazine*, 25(1), 33-33.

[52] Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., ... & Rives, A. (2022). Language models of protein sequences at the scale of evolution. *bioRxiv*, 2022-07.

[53] Daulton, S., Balandat, M., & Bakshy, E. (2020). Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. *arXiv preprint arXiv:2006.05078*.

[54] Morales-Brotons, D., Vogels, T., Hendrickx, H. (2024). Exponential Moving Average of Weights in Deep Learning: Dynamics and Benefits. *Transactions on Machine Learning Research*, 04/2024.

[55] Menard, J., Mansbach, R. A. (2025). Semi-supervised latent Bayesian optimization for designing antimicrobial peptides. *arXiv preprint arXiv:2411.18704*.

[56] Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations*.

[57] Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33, 6840-6851.

[58] Hoogeboom, E., Satorras, V. G., Vignac, C., & Welling, M. (2022). Equivariant diffusion for molecule generation in 3D. In *International Conference on Machine Learning* (pp. 8867-8887). PMLR.

[59] Nichol, A. Q., & Dhariwal, P. (2021). Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning* (pp. 8162-8171). PMLR.

[60] Chen, T., Zhai, S., Nambiar, A., Li, J., Mittal, S., Obukhov, M., ... & Chatterjee, C. (2024). AMP-Diffusion: Integrating latent diffusion with protein language models for antimicrobial peptide generation. *bioRxiv*, 2024.03.03.583201.

[61] Cao, J., Wang, L., Xu, L., Hao, G. F., & Yang, G. F. (2025). TG-CDDPM: text-guided antimicrobial peptides generation using conditional denoising diffusion probabilistic models. *Briefings in Bioinformatics*, 26(1), bbae644.

[62] Jin, S., Liu, Y., Zeng, X., & Zou, Q. (2025). AMPGen: an evolutionary information-reserved and diffusion-driven generative model for de novo design of target-specific AMPs. *Communications Biology*, 8, 282.

[63] Wang, Y., Zhai, S., Nambiar, A., Obukhov, M., Li, J., Mittal, S., ... & Chatterjee, C. (2025). Artificial intelligence using a latent diffusion model enables the generation of diverse and potent antimicrobial peptides. *Science Advances*, 11, eadp7171.

[64] Dean, S. N., Alvarez, J. A. E., Zabetakis, D., Walper, S. A., Malanoski, A. P. (2021). PepVAE: Variational autoencoder framework for antimicrobial peptide generation and activity prediction. *Frontiers in Microbiology*, 12, 725727.

[65] Zhao, W., Hou, K., Shen, Y., Hu, X. (2025). Conditional denoising VAE-based framework for antimicrobial peptides generation with preserving desirable properties. *Bioinformatics*, 41(2), btaf069.

[66] Tucs, A., Tran, D. P., Yumoto, A., Ito, Y., Uzawa, T., Tsuda, K. (2020). Generating ampicillin-level antimicrobial peptides with activity-aware generative adversarial networks. *ACS Omega*, 5(36), 22847-22851.

[67] Lai, C. W., Chen, Y. C., Ou, S. C., Hsiao, Y. C., Pang, H. Y., Huang, F. C., ... Lin, Y. C. (2025). From AI to action: Antimicrobial peptides engineered by generative adversarial networks (GANs)—A novel approach to combat resistant bacteria. *Chemical Engineering Journal*, 519, 164905.

[68] Zervou, M. A., Gkeka, P., Cournia, Z., Tsakalidis, A. (2025). Classifier-driven generative adversarial networks for enhanced antimicrobial peptide design. *Briefings in Bioinformatics*, 26(5), bbaf500.

[69] Liu, J., Zhang, Y., Wang, X., Li, M., Chen, H. (2025). A multi-property optimizing generative adversarial network for de novo antimicrobial peptide design. *Advanced Science*, 2503443.

[70] Hochreiter, S., Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.

[71] Hendrycks, D., Gimpel, K. (2016). Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*.

[72] Kang, X., Dong, F., Shi, C., Liu, S., Sun, J., Chen, J., ... Yao, Q. (2019). DRAMP 2.0, an updated data repository of antimicrobial peptides. *Scientific Data*, 6(1), 148.

[73] Waghu, F. H., Barai, R. S., Gurung, P., Idicula-Thomas, S. (2016). CAMPR3: a database on sequences, structures and signatures of antimicrobial peptides. *Nucleic Acids Research*, 44(D1), D1094-D1097.

[74] Wang, G., Li, X., Wang, Z. (2016). APD3: the antimicrobial peptide database as a tool for research and education. *Nucleic Acids Research*, 44(D1), D1087-D1093.