國立臺灣大學共同教育中心統計碩士學位學程

碩士論文

Master's Program in Statistics

Center for General Education

National Taiwan University

Master's Thesis

以 AE 或 Beta-VAE 外加特徵方式提升 BERT 語言模型的內嵌訊息

Enhancing BERT Embeddings Using AE or Beta-VAE-Processed Additional Features

許維仁

Wei-Ren Hsu

指導教授: 陳彥賓 博士

Advisor: Yan-Bin Chen Ph.D.

中華民國 114 年 7 月

July, 2025

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

Acknowledgements

在撰寫這篇碩士論文的過程中，我經歷了無數次的挫折與迷惘。然而，也正是在一次次耐心解決問題、一步步摸索前行的過程中，我才能走到今日，將這份研究完整地呈現出來。

首先，我要誠摯感謝我的指導教授陳彥賓教授。在研究過程中，老師總是耐心地協助我釐清思路，幫助我判斷哪些問題該優先處理，哪些階段應該暫緩，讓我在這條漫長且容易迷失方向的道路上，能夠持續聚焦於核心目標。老師也經常分享學術研究中的寶貴經驗與文化細節，使我能以更穩定的心態去面對各種挑戰。老師的提點與引導，對我而言是一份極為重要的支撐與助力。

同時，我要感謝我的家人與女友。在這段日子裡，他們無條件的支持與鼓勵，是我能夠專心完成這篇論文、堅持走到最後的最大動力。每當我感到疲憊與灰心時，家人與伴侶的陪伴總能讓我重新振作，繼續向前。

最後，我也想感謝自己身處在這個時代。在這個人工智慧快速進步的年代，眾多大型語言模型的出現，成為我研究過程中無可取代的好夥伴。無論是想法的驗證、方向的討論，甚至是突發奇想的靈感碰撞，它們都陪伴著我思考與成長。可以說，這些智慧工具也在無聲之中，參與了這份研究的每一個重要時刻。

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

摘要

以 BERT 為代表的子詞語言模型，在自然語言理解任務中表現優異，但在面對需要字符級推理的任務時（如字符計數或字串比對），往往無法精確捕捉細微的字符級特徵。相比之下，字符級語言模型能夠細緻地建構這類訊息，但訓練字符級語言模型會帶來極高的運算負擔。本論文提出一套輕量化方法，將字符級視覺特徵融合至子詞單位的向量表示中。我們將每個子詞轉換為字形序列，並透過自編碼器 (AE) 與 beta-變分自編碼器 (beta-VAE) 進行視覺特徵的壓縮與表徵學習。這些特徵經由線性映射或多層感知器 (MLP) 映射至 BERT 的嵌入空間，並與原始子詞向量結合，僅需修改輸入層即可完成整合，無須調整原始模型架構。我們將此方法應用於一項字符數預測任務，並以遮罩語言模型 (Masked Language Modeling, MLM) 的形式進行訓練與評估。實驗涵蓋 460 萬筆樣本，結果顯示加入視覺特徵後，模型表現皆優於基準模型。透過 beta-變分自編碼器所取得之特徵在僅使用線性投影的情況下即可展現明顯效果，而透過自編碼器取得之特徵則需搭配多層感知器才能達到最佳表現。實驗結果驗證了將額外的視覺特徵注入子詞向量，能有效提升模型的字符推理能力，彌補子詞建模與字符細節之間的落差。本方法使模型能以極小的計算成本，學習到字符層級的特徵表現，適用於如計數、比對等需精細字元辨識的任務。

關鍵字：自編碼器、變分自編碼器、Beta-變分自編碼器、語言模型、模型微調、詞嵌入、字形結構

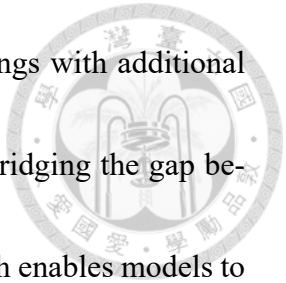
doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

Abstract

Subword-based language models like BERT achieve strong performance in natural language understanding but often miss fine-grained character-level cues essential for symbolic reasoning tasks like counting or string matching. While character-level representations can capture these subtle patterns more effectively, incorporating them directly into large models poses significant computational challenges. This thesis presents a lightweight framework to enrich subword token embeddings with character-level visual features. We render each token as a glyph sequence and train autoencoders or beta-VAEs to learn compact visual representations. These features are projected into BERT's embedding space via linear or MLP layers and added to the original embeddings, requiring no architectural changes beyond the input layer. We evaluate this integration on a character-counting task framed as masked language modeling. Experiments on 4.6 million examples show consistent improvements over the baseline. beta-VAE-based features are effective even with linear projections, while AE-based features benefit from non-linear

doi:10.6342/NTU202503286



mappings. Our findings indicate that augmenting subword embeddings with additional visual features significantly improves symbolic reasoning abilities, bridging the gap between subword efficiency and character-level precision. This approach enables models to capture fine-grained character patterns crucial for tasks like counting, without incurring the computational costs of full character-level architectures.

Keywords: Autoencoder, Beta-VAE, Variational Autoencoder, BERT, Embeddings, Glyph Structure, Fine-Tuning

doi:10.6342/NTU202503286

Contents

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

List of Figures

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

List of Tables

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

Chapter 1 Introduction

1.1 Background

Large language models (LLMs) such as GPT-3 [1] and LLaMA [13] have revolutionized natural language processing by demonstrating state-of-the-art performance on a wide range of tasks, from machine translation to open-domain question answering. These models leverage immense transformers trained on terabytes of text data to capture rich semantic and syntactic patterns. However, despite their remarkable knowledge and generative capabilities, LLMs exhibit surprising brittleness on tasks that humans consider trivial. In particular, they often produce inconsistent or incorrect results for questions that require simple symbolic manipulation or exact character-level reasoning, revealing a limitation in their internal representations.

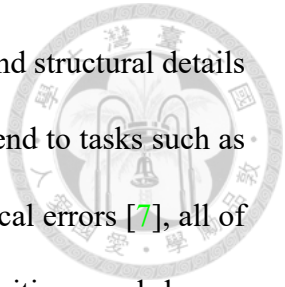
Consider, for example, prompts of the form:

"How many 'r' characters are in the word 'strawberry'?" [1]

While a human can easily provide the correct answer "3", pretrained LLMs frequently miscount, suggesting that the underlying tokenization and embedding mechanisms ob-

[1] This example comes from an online discussion of LLM failure cases: https://community.openai.com/t/incorrect-count-of-r-characters-in-the-word-strawberry/829618

doi:10.6342/NTU202503286



scure fine-grained character information, including the visual forms and structural details of glyphs. Such failures are not limited to counting problems but extend to tasks such as creating anagrams, generating palindromes, or identifying typographical errors [7], all of which require precise knowledge of individual character identities, positions, and shapes.

One fundamental contributor to this shortfall is the widespread adoption of subword tokenizers—e.g., Byte-Pair Encoding (BPE) [5] and SentencePiece [10]—which split words into arbitrary-length subword units. These tokenizers significantly reduce vocabulary size and accelerate training; however, they obscure character boundaries and disregard glyph structures, making it difficult to extract information at the level of individual characters. In contrast, character-level models retain full access to character-level signals by treating each character as a token, but they come with prohibitive computational costs: sequence lengths grow by an order of magnitude, and the model must learn longer-range dependencies from scratch.

These observations motivate the following research question:

> Is it possible to augment efficient subword-based LLMs that recover or approximate character-level information, enabling accurate character-level reasoning without the computational cost of full character-level modeling?

In this thesis, we investigate the feasibility of bridging the gap between coarse subword embeddings and fine-grained character reasoning by injecting additional character-level visual features into a pre-trained Transformer's embedding space. By augmenting token embeddings with glyph structure information, we enable the model to better handle tasks sensitive to such information.

doi:10.6342/NTU202503286

1.2 Contributions

Our main contributions are as follows:

1. A framework for enriching pre-trained token embeddings with additional character-level features using projection layers.

2. An experimental analysis demonstrating the effectiveness of the proposed approach, evaluating the trade-offs between injection complexity and task performance.

1.3 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 surveys related work on character-level and pixel-based modeling. Chapter 3 introduces our method for learning and integrating additional features, specifically character-level features. Chapter 4 outlines the experimental setup and presents empirical results on character-level reasoning tasks. Chapter 5 provides a comprehensive analysis of the experimental findings and their limitations and outlines avenues for future research.

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

Chapter 2 Literature Review

2.1 Limitations of Subword Tokenization

Large language models (LLMs) such as BERT [3] use subword tokenization schemes (e.g., WordPiece) to balance vocabulary size and coverage. However, subword tokenization can obscure character boundaries, limiting the model's ability to perform character-sensitive tasks such as spelling correction, anagram generation, and character counting. These tasks often require precise knowledge of character identity and position, which subword models cannot directly provide.

2.2 Character and Byte-Level Modeling

One approach to address this limitation is to eliminate subword tokenization entirely. **ByT5** [14] operates purely at the byte level, demonstrating that pretrained encoder-decoder transformers can match or surpass traditional subword-based models without relying on a predefined vocabulary. While ByT5 offers improved robustness and generalization, its byte-level processing results in substantially longer input sequences compared to subword models. This leads to increased memory requirements and slower training and inference. Furthermore, since bytes lack inherent linguistic structure, the model must learn syntactic

doi:10.6342/NTU202503286

and morphological patterns from scratch, which raises training complexity and reduces efficiency.

CANINE [2] similarly avoids subword tokenization by operating on characters. It introduces a hierarchical model that encodes raw character sequences into token-like representations using downsampling. CANINE achieves performance comparable to BERT on many tasks while retaining character-level robustness, especially in noisy or multilingual settings. However, despite the downsampling mechanism, CANINE still suffers from significant computational overhead due to the inherently longer character sequences compared to subword tokens. The quadratic attention complexity in transformers becomes particularly problematic when processing character-level inputs, leading to substantially higher memory consumption and training time compared to subword-based models.

2.3 Character-Level Enhancements for Chinese NLP

For Chinese NLP, where characters encode rich visual and phonetic information, **ChineseBERT** [11] enhances a character-level Transformer model by injecting glyph and pinyin embeddings into the input layer. Each Chinese character is rendered as multiple font images, from which glyph embeddings are extracted via a CNN or FC layer. Similarly, pinyin sequences are encoded using a CNN to capture phonetic features. These additional features are concatenated with character embeddings to form a fused input. As a purely character-level model, ChineseBERT demonstrates strong performance across tasks, highlighting the value of incorporating intra-character structure. However, the additional glyph feature remains restricted to character-level modeling and fixed-resolution visual input.

doi:10.6342/NTU202503286

2.4 Vision-Based Text Modeling

Recent advances have explored a paradigm-shifting approach by processing text as visual renderings. **Pixel** [12] proposes to bypass tokenization altogether by converting entire text sequences into grayscale images and processing them with Vision Transformer (ViT) [4]. Each input sentence is rendered into a 2D pixel grid, allowing the model to capture visual and spatial patterns across character sequences. Despite having no textual embeddings or token boundaries, Pixel achieves competitive performance with BERT on several NLP benchmarks and demonstrates strong robustness to misspellings.

2.5 Technical Challenges and Research Motivation

Current approaches to character-aware language modeling present fundamental limitations that constrain their practical adoption:

1. Models such as **ByT5** and **CANINE** completely abandon subword tokenization in favor of byte or character-level inputs. While this allows fine-grained modeling, it results in significantly higher computational costs due to longer input sequences and lack of vocabulary compression.

2. **ChineseBERT** enriches Chinese character representations with glyph and pinyin information, but operates strictly at the character level and assumes fixed-length Chinese input. It is not compatible with subword tokenization used in multilingual or English-oriented models, limiting its general applicability.

3. **PIXEL** renders entire text sequences as images and processes them via vision trans-

doi:10.6342/NTU202503286

formers. Although conceptually innovative, the approach diverges from standard NLP architectures and involves considerable computational cost, posing challenges for large-scale or real-time use.

In contrast, our work introduces additional character-level visual features derived from rendered character sequences, effectively bridging the gap between subword embeddings and fine-grained character-level reasoning. Experimental results on a character-counting task demonstrate the effectiveness of the proposed approach.

doi:10.6342/NTU202503286

Chapter 3 Methodology

In this section, we describe the overall architecture of our proposed method, which aims to enrich subword-based large language models (LLMs) with additional character-level features extracted from their subword tokens. The core idea is to encode each subword token's character sequence using an autoencoder (AE) [15] or variational autoencoder (VAE) [9], producing a fixed-length vector that captures character-level patterns. These additional features are then integrated into the LLM's existing embeddings, augmenting them with finer-grained character-level information to improve performance on downstream tasks.

We begin by briefly reviewing autoencoders and variational autoencoders, emphasizing their differences and their role in learning compact feature representations. We then describe how these features are injected into the LLM's embedding layer, and introduce two integration strategies inspired by neural projection techniques. Finally, we detail which components are fine-tuned during training and explain how our method maintains full compatibility with existing pretrained embeddings and model checkpoints.

doi:10.6342/NTU202503286

3.1 Autoencoders

Autoencoders (AEs) are a class of neural networks designed to learn an efficient encoding of input data by compressing and reconstructing it in an unsupervised manner. Formally, let $\mathcal{X} \subseteq \mathbb{R}^d$ denote the input space, and let $\mathcal{Z} \subseteq \mathbb{R}^k$ denote the latent space, where typically $k \ll d$.

An autoencoder consists of two components:

an encoder $E_\phi : \mathcal{X} \to \mathcal{Z}$, parameterized by ϕ, which maps an input vector $\mathbf{x} \in \mathcal{X}$ to a latent code $\mathbf{z} = E_\phi(\mathbf{x}) \in \mathcal{Z}$, and

a decoder $D_\theta : \mathcal{Z} \to \mathcal{X}$, parameterized by θ, which reconstructs the input as $\hat{\mathbf{x}} = D_\theta(\mathbf{z}) = D_\theta(E_\phi(\mathbf{x}))$.

The parameters ϕ and θ are learned by minimizing a reconstruction loss over a dataset $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$:

$$\mathcal{L}_{\text{AE}}(\phi, \theta) = \frac{1}{N} \sum_{i=1}^{N} \ell\left(\mathbf{x}_i, D_\theta(E_\phi(\mathbf{x}_i))\right) \tag{3.1}$$

where $\ell : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a loss function such as the mean squared error (MSE) or cross-entropy, depending on the nature of the data.

The encoder E_ϕ maps each input $\mathbf{x} \in \mathcal{X}$ to a lower-dimensional encoding $\mathbf{z} = E_\phi(\mathbf{x}) \in \mathcal{Z}$ that preserves the information necessary for reconstruction. The decoder D_θ then attempts to recover the original input from the encoding, i.e., $\hat{\mathbf{x}} = D_\theta(\mathbf{z})$ (see Fig. 3.1 for an overview of the architecture). This encoding – decoding framework al-

doi:10.6342/NTU202503286

lows the model to learn compressive mappings of the input data and is widely used for dimensionality reduction.

A key limitation of standard autoencoder is that the encoding space $\mathcal{Z}$ is learned without any imposed structure. As a result, the encodings may be irregularly distributed and difficult to interpret. This lack of organization in $\mathcal{Z}$ can hinder tasks that require structure in the encoded features—such as smooth interpolation, feature disentanglement, or controlled generation—thus limiting the generalization ability of the model in downstream applications.

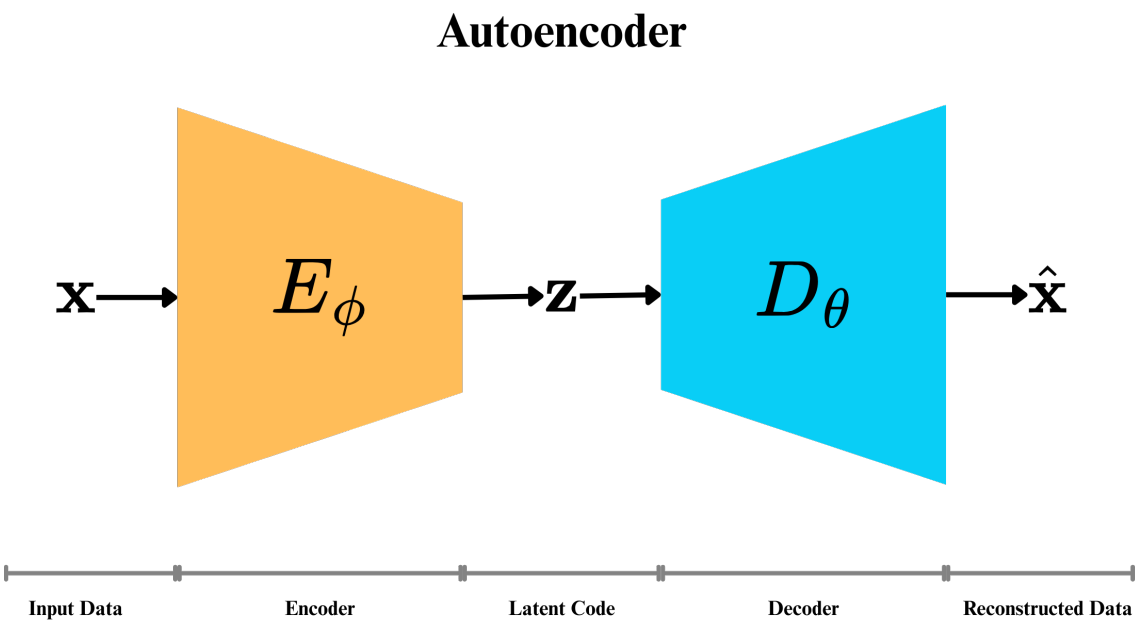


Figure 3.1: Illustration of a standard autoencoder architecture.

3.2 Variational Autoencoders

To address the unstructured encoding space of standard autoencoders, Variational Autoencoders(VAEs) [9] introduce a probabilistic formulation that enables both encoding and generation, while imposing a prior-driven structure on the learned encoding space $\mathcal{Z}$.

doi:10.6342/NTU202503286

Instead of mapping each input $\mathbf{x} \in \mathcal{X}$ deterministically to a point in the encoding space, VAEs encode $\mathbf{x}$ as a distribution over latent codes $\mathbf{z} \in \mathcal{Z} \subseteq \mathbb{R}^k$. Specifically, the encoder defines a variational posterior:

$$q_\phi(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}\big(\mathbf{z}; \boldsymbol{\mu}_\phi(\mathbf{x}), \mathrm{diag}(\boldsymbol{\sigma}^2_\phi(\mathbf{x}))\big) \tag{3.2}$$

where both the mean $\boldsymbol{\mu}_\phi(\mathbf{x})$ and standard deviation $\boldsymbol{\sigma}_\phi(\mathbf{x})$ are learned functions of the input. This stochastic encoding mechanism not only captures uncertainty, but also allows the model to regulate the geometry of the encoding space via the posterior' s shape.

To impose structure on $\mathcal{Z}$, VAEs assume a fixed prior distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. By minimizing the Kullback–Leibler (KL) divergence between the learned encoding distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$ and this prior, the model encourages the encodings to lie in a continuous, smooth, and compact region of $\mathcal{Z}$. Given a sampled latent code $\mathbf{z}$ from the encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$, the decoder then reconstructs the input through the likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$, enabling end-to-end learning of both representation and reconstruction.

Compared to standard autoencoders, where encodings can be arbitrarily scattered or sparse, VAEs explicitly shape the encoding space to support generalization. For instance, nearby points in $\mathcal{Z}$ tend to decode to similar data, and interpolations between encodings yield smooth transitions in data space. Such geometric regularity makes the encoding space more interpretable and better suited for downstream tasks like clustering, disentanglement, or controlled generation.

The training goal of VAE is to maximize the log marginal likelihood $\log p_\theta(\mathbf{x})$ under

doi:10.6342/NTU202503286

the generative model:

$$\log p_\theta(\mathbf{x}) = \log \int p_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})\, d\mathbf{z} \tag{3.3}$$

As this quantity is intractable, variational autoencoders (VAEs) maximize a tractable lower bound, known as the evidence lower bound (ELBO):

$$\text{ELBO}(\phi, \theta; \mathbf{x}) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right]}_{\text{reconstruction}} - \underbrace{\text{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})\right)}_{\text{regularization}} \tag{3.4}$$

which is maximized during training. The first term encourages the decoder to faithfully reconstruct the input, while the second term ensures that the learned encodings $q_\phi(\mathbf{z} \mid \mathbf{x})$ stay close to the prior $p(\mathbf{z})$. This regularization constrains the encoding space to be continuous, smooth, and densely populated, which are essential properties for effective generalization and structured representation learning. In practice, we minimize the negative ELBO, which defines the VAE loss:

$$\mathcal{L}_{\text{VAE}}(\phi, \theta; \mathbf{x}) = -\text{ELBO}(\phi, \theta; \mathbf{x}) = \text{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})\right) - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] \tag{3.5}$$

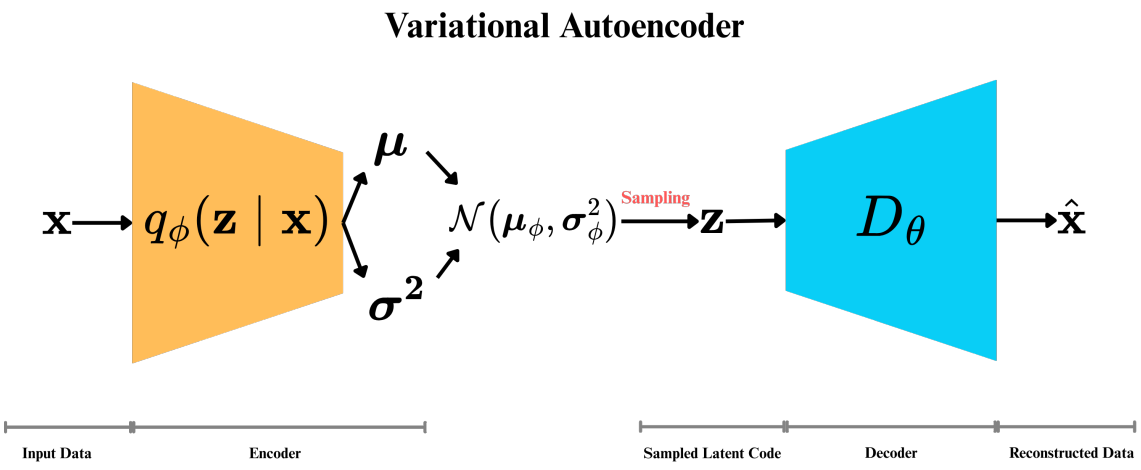


Figure 3.2: Illustration of a standard variational autoencoder architecture.

To further encourage structured representations in the encoding space, the β-VAE [8]

doi:10.6342/NTU202503286

introduces a tunable hyperparameter β that scales the KL term:

$$\mathcal{L}_{\beta\text{-VAE}} = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] + \beta \cdot D_{\mathrm{KL}}(q_\phi(z \mid x) \parallel p(z)) \tag{3.6}$$

A larger β enforces stronger alignment between the posterior and the prior, which can improve the organization of the encoding space but may reduce reconstruction quality. Despite this trade-off, β-VAE and related variants have been shown to yield more interpretable and robust representations.

In our work, we leverage these models to encode each subword token into a low-dimensional vector that captures character-level information. These vectors serve as additional features that are injected into the embedding space of a large language model, with the goal of improving its performance on downstream tasks, especially those requiring sensitivity to character-level patterns such as spelling or character frequency.

3.3 Injecting Additional Character-Level Features into Embeddings

Once character-level features are learned using an autoencoder or β-VAE, the next step is to integrate them into a language model to enhance the embeddings of subword tokens.

In an ideal scenario, one could train a large language model from scratch with an augmented embeddings that incorporate character-level features. Such a setup would provide a valuable opportunity to study the role of additional feature throughout the entire training process. However, training LLMs from scratch requires massive computational

doi:10.6342/NTU202503286

resources, making this approach impractical for most research settings.

Instead, we adopt a more feasible strategy by injecting character-level features into the embedding space of pretrained models. Specifically, we augment the original token embeddings with additional features using projection layers.

To ensure compatibility with the pretrained model architecture, the injected features must be projected to match the original embedding dimension n. We therefore project the learned character-level feature vector (of dimension d) into the same embedding space before combining it with the original token embedding via vector addition. This results in an augmented embeddings that retains the positional and semantic context of the original token embedding, while incorporating additional character-level features.

We consider two projection strategies for integrating character-level features into the token embedding space, as illustrated in Figure 3.3:

- **Linear Projection (Shallow Injection):** This method applies a single linear layer to project the d-dimensional character-level vector into the n-dimensional token embedding space. It is computationally efficient and simple to implement, but may have limited capacity to model complex interactions between sub-token structure and token semantics.

- **MLP-based Projection (Deep Injection):** This variant replaces the projection block with a multi-layer perceptron (MLP), enabling nonlinear mappings from character-level features to token embeddings. The added depth and activation functions allow the model to capture richer patterns, potentially benefiting tasks requiring more nuanced sub-token reasoning.

doi:10.6342/NTU202503286

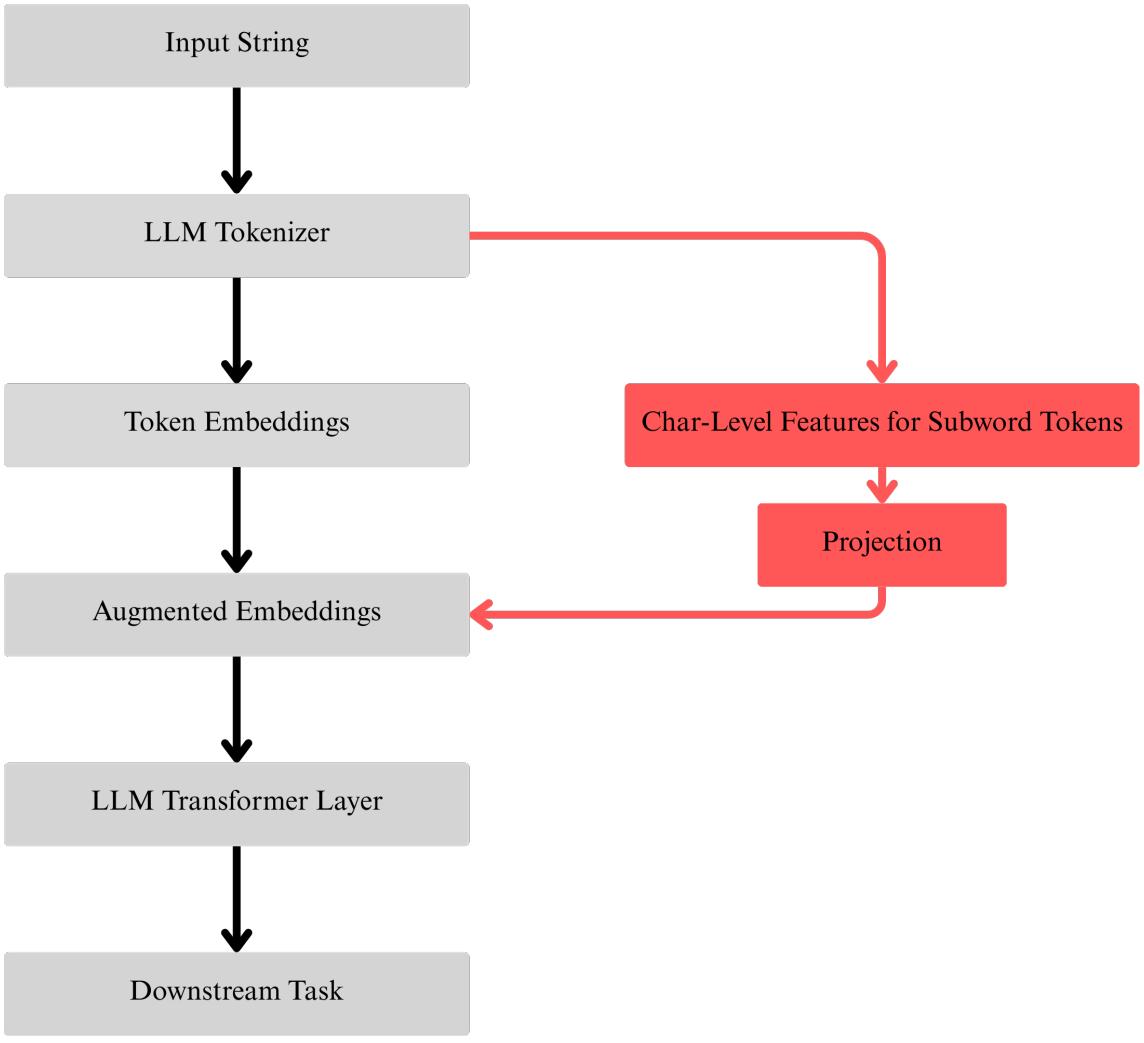


Figure 3.3: Overview of character-level feature injection into a pretrained LLM. For each subword token, a character-level additional feature is derived using an external encoder (e.g., autoencoder or β-VAE), then projected and added element-wise to the original token embedding. The augmented embeddings are passed through the frozen LLM for downstream fine-tuning.

doi:10.6342/NTU202503286

In both cases, the projected vector is added element-wise to the original token embedding to form the final augmented embedding. During fine-tuning, we update the augmented embedding layer and the task-specific classification head, while keeping the rest of the LLM frozen to reduce computational cost.

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

Chapter 4 Experiments

4.1 Overview and Motivation

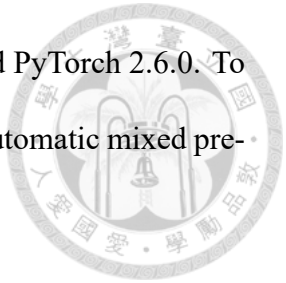
Building on the character-level additional features introduced in Chapter 3, this chapter explores how to integrate these features into the BERT architecture to enrich its token embeddings. While BERT's token embeddings are highly effective at capturing semantic and syntactic patterns from large-scale text corpora, they do not explicitly encode character-level visual information, even though such information can be crucial in languages with complex scripts or visually distinctive characters.

As outlined in Section 3.3, we consider two projection-based strategies for injecting character-level visual features into BERT' s token embeddings. These methods differ in expressiveness, ranging from a simple linear projection to a more expressive multilayer perceptron (MLP), but they share the same core idea: fusing additional character-level visual features with BERT's pre-trained token embeddings via vector addition. This allows the model to incorporate both pre-trained semantic knowledge and character-level visual structure with minimal architectural modifications.

Experimental Environment All experiments were conducted on a computing server equipped with one-eighth allocation of an NVIDIA A100-SXM4-40GB GPU. The soft-

doi:10.6342/NTU202503286

ware environment was configured with Python 3.10, CUDA 12.6, and PyTorch 2.6.0. To improve training efficiency and reduce GPU memory consumption, automatic mixed precision (AMP) was employed.



The following section outlines the construction of character-level visual features from glyph images, beginning with the dataset and preprocessing setup.

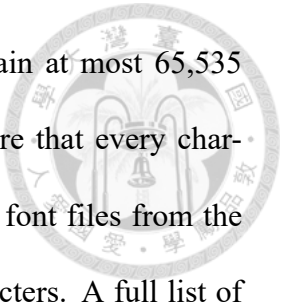
4.2 Building Character-Level Visual Features for Subword Tokens

To enrich subword embeddings with visual information, we aim to learn compact visual features that capture the glyph-level structure of their constituent characters. This is achieved by training convolutional autoencoders and β-VAEs on synthetic sequences of character images, allowing the model to compress a sequence of glyphs into a fixed-size visual feature vector.

4.2.1 Dataset and Preprocessing

The training data consists of synthetic sequences of Unicode characters rendered as grayscale images. To ensure broad coverage across scripts and writing systems, we sample characters from the Unicode range `U+0020` to `U+2FFFF`, which includes Latin, Chinese, Japanese, Korean, and many others. This range covers a total of 196,576 unique characters. Each character is rendered as a 64×64 grayscale image using fonts from the Noto Sans family [6], which was chosen for its wide Unicode coverage and consistent visual style across scripts.

doi:10.6342/NTU202503286



However, due to the limitation that a single font file can contain at most 65,535 glyphs, a single font is insufficient to cover the entire set. To ensure that every character in our training set has a corresponding glyph, we combined 19 font files from the Noto Sans family, achieving complete coverage of the 196,576 characters. A full list of the fonts used is provided in Appendix A.0.2.

With full glyph coverage ensured, we proceed to construct the training samples. Rather than using real subword tokens tied to a specific tokenizer, we construct synthetic samples by randomly selecting sequences of 18 characters from the Unicode set. This design decouples the visual representation from any particular vocabulary and makes it easier to generalize across different tokenization schemes. Each sample forms a tensor of shape $(18, 64, 64)$, simulating a subword-like unit composed of multiple glyphs. Figure 4.1 shows an example of such a sequence.

Figure 4.1: Example of a synthetic character sequence used for training the AE/β-VAE models. The sequence consists of 18 randomly sampled Unicode characters, including symbols from multiple scripts.

With this synthetic dataset, we now train autoencoder-based models to learn compact visual features of subword-like character sequences.

4.2.2 Visual Autoencoder and β-VAE Architecture

To extract compact visual features from character image sequences, we train convolutional autoencoders and β-VAEs on synthetic grayscale character sequences. We treat the entire input as a sequence of visual units and aim to learn a compact character-level feature suitable for downstream tasks.

doi:10.6342/NTU202503286

Figure 4.2 shows the architecture of the convolutional autoencoder. The encoder consists of four convolutional blocks, each reducing the spatial resolution by a factor of two. Each character is encoded into a 128-dimensional feature vector. These 18 feature vectors are concatenated and then projected down to a single 128-dimensional feature, which serves as the learned visual representation of the input character sequence. The decoder mirrors the encoder using transposed convolutions to reconstruct the original character sequence. This means that an input originally represented as a $18 \times 64 \times 64 = 73{,}728$-dimensional image tensor is ultimately compressed into a 128-dimensional feature, reflecting a substantial reduction in dimensionality while aiming to retain core visual characteristics.

To enable structured latent representation learning, we further extend the autoencoder into a variational version (β-VAE), as shown in Figure 4.3. The encoder follows the same architecture as the AE but learns both the mean and log-variance of a latent Gaussian distribution. A sampled 128-dimensional latent vector is obtained via the reparameterization trick. The decoder is identical to the AE's, reconstructing the full character image sequence from this feature embedding. The KL divergence term is scaled by a β hyperparameter to control the weight.

Once training is complete, we construct a visual feature lookup table by encoding the character sequence of each subword token in the BERT vocabulary using the encoder of the trained autoencoder or β-VAE. As illustrated in Figure 4.4, we enumerate all subword tokens from the BERT tokenizer and render each as a sequence of glyph images using the Noto Sans Regular font. Each sequence is padded to a fixed length of 18 characters to match the encoder' s input shape; this length is sufficient to cover the longest token in the BERT vocabulary. The encoder then maps each glyph sequence to a 128-dimensional

doi:10.6342/NTU202503286

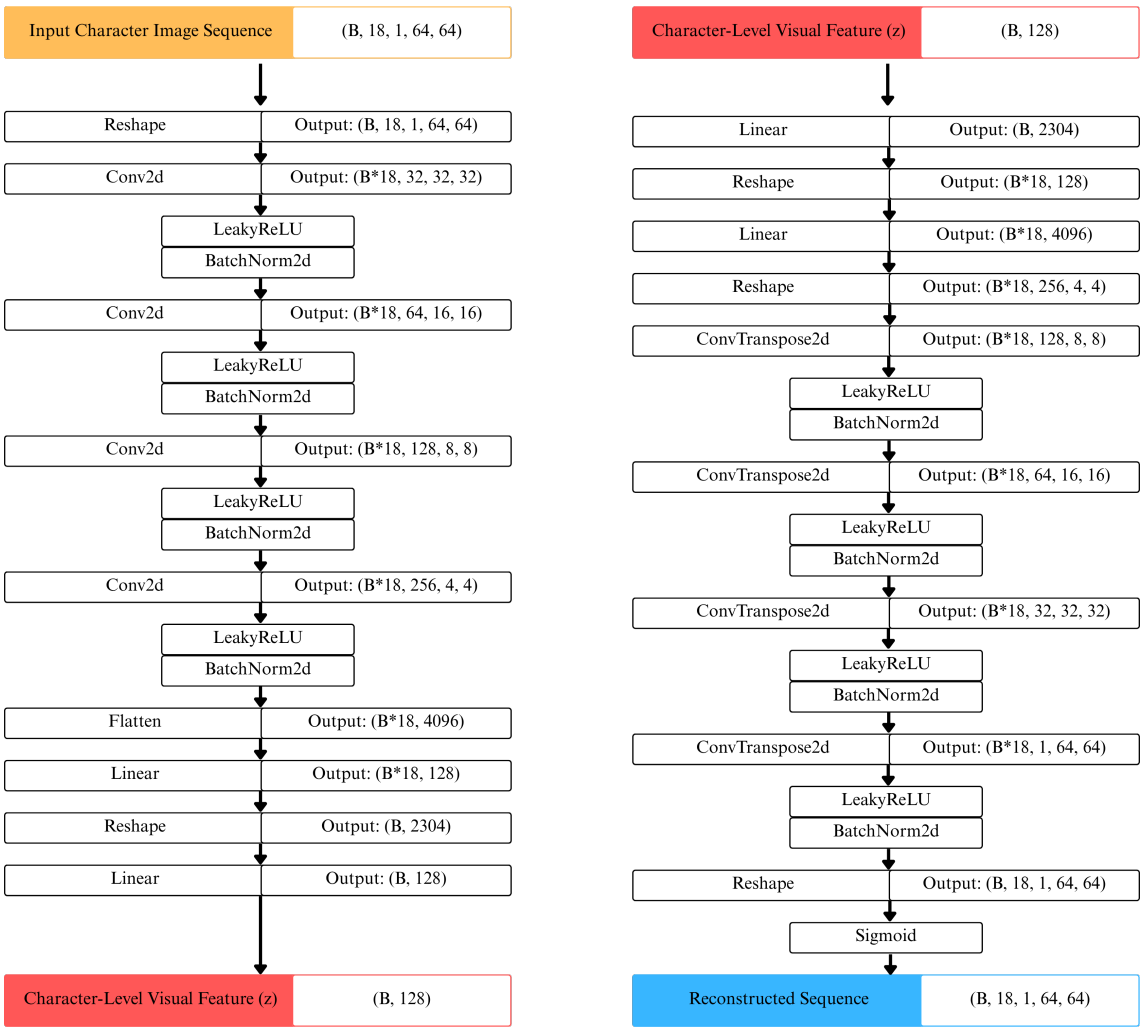


Figure 4.2: **Architecture of the autoencoder.** Orange denotes the input character image sequence; red indicates the learned character-level visual feature z; blue denotes the reconstructed output. Here, B denotes the batch size.

doi:10.6342/NTU202503286

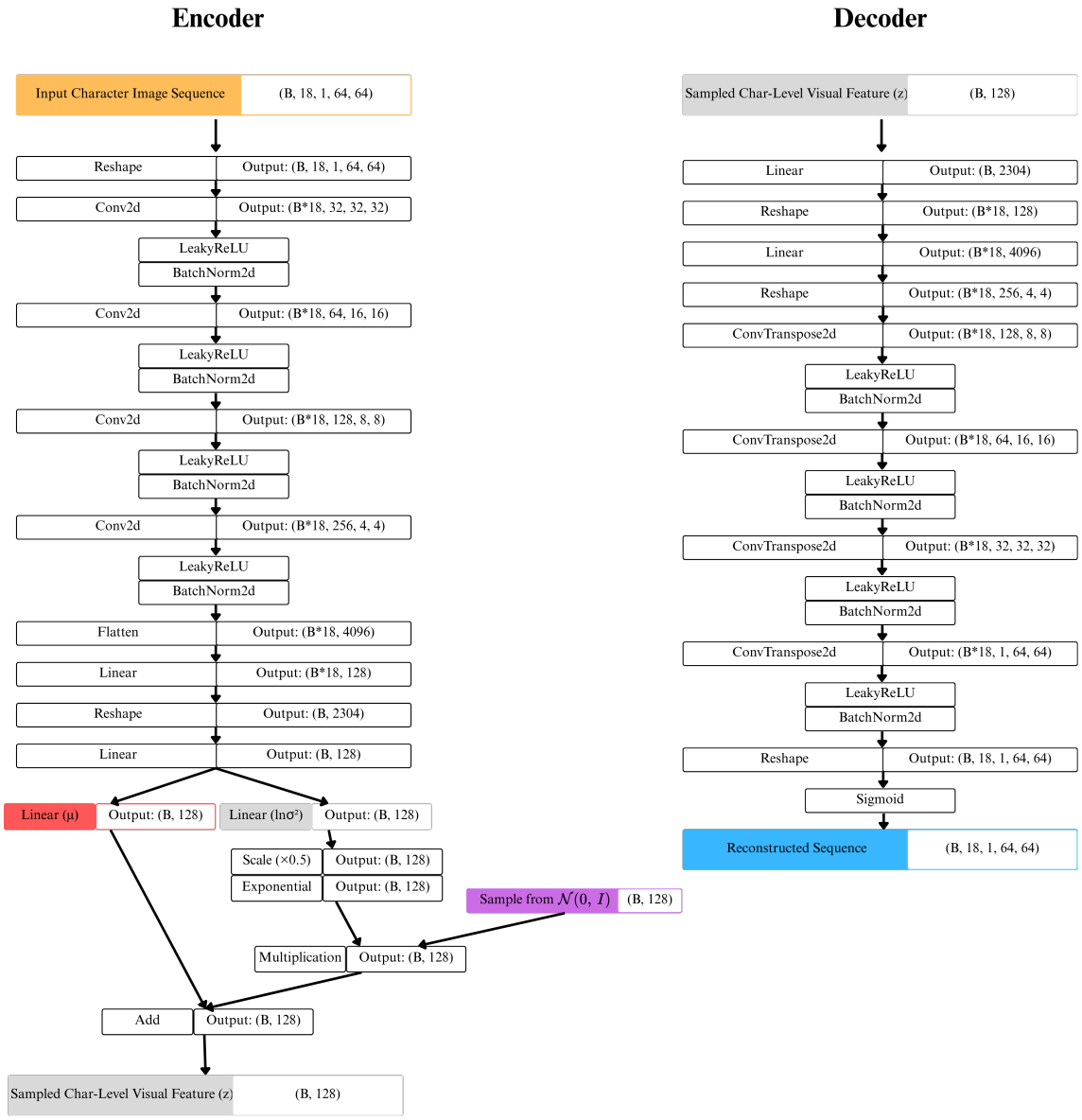


Figure 4.3: **Architecture of the β-VAE.** Orange denotes the input character image sequence; red indicates the mean μ from the encoder; grey indicates the sampled latent embedding z ; blue denotes the reconstructed output. The sampled feature is obtained via the reparameterization trick. Here, B denotes the batch size.

doi:10.6342/NTU202503286

visual feature, which is cached into a lookup table for fast retrieval during downstream task.

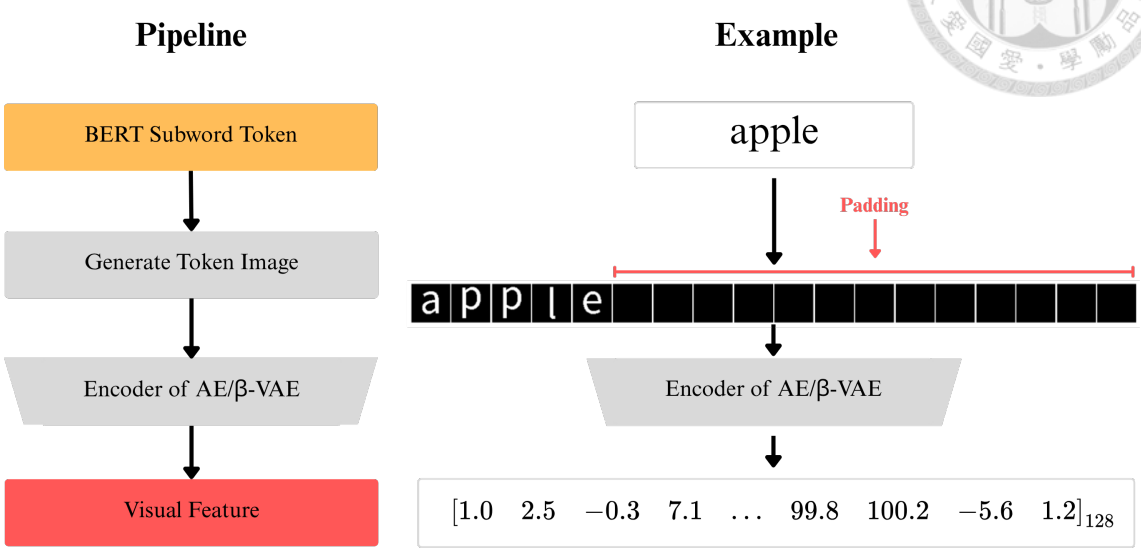


Figure 4.4: Pipeline for constructing the visual feature lookup table. The left side shows the offline preprocessing steps, while the right side illustrates an example token being rendered into a sequence of character images (padded to length 18) and mapped to a 128-dimensional feature. Example numeric values are illustrative only and do not represent actual features.

To visualize the reconstruction ability of different latent-variable models, we compare reconstruction results across several trained models, including a standard autoencoder and multiple VAEs with varying β values. As shown in Figure 4.5, the top row shows ground truth input sequences, while subsequent rows depict reconstructions from different models. This qualitative comparison illustrates the trade-off between the reconstruction error and the KL divergence term in the loss function as β increases.

4.3 Integrating Visual Embeddings into BERT

We now integrate the learned 128-dimensional visual features into BERT for downstream tasks. Each token in BERT's vocabulary has an original 768-dimensional embedding from the pre-trained embedding layer. To incorporate character-level visual informa-

doi:10.6342/NTU202503286

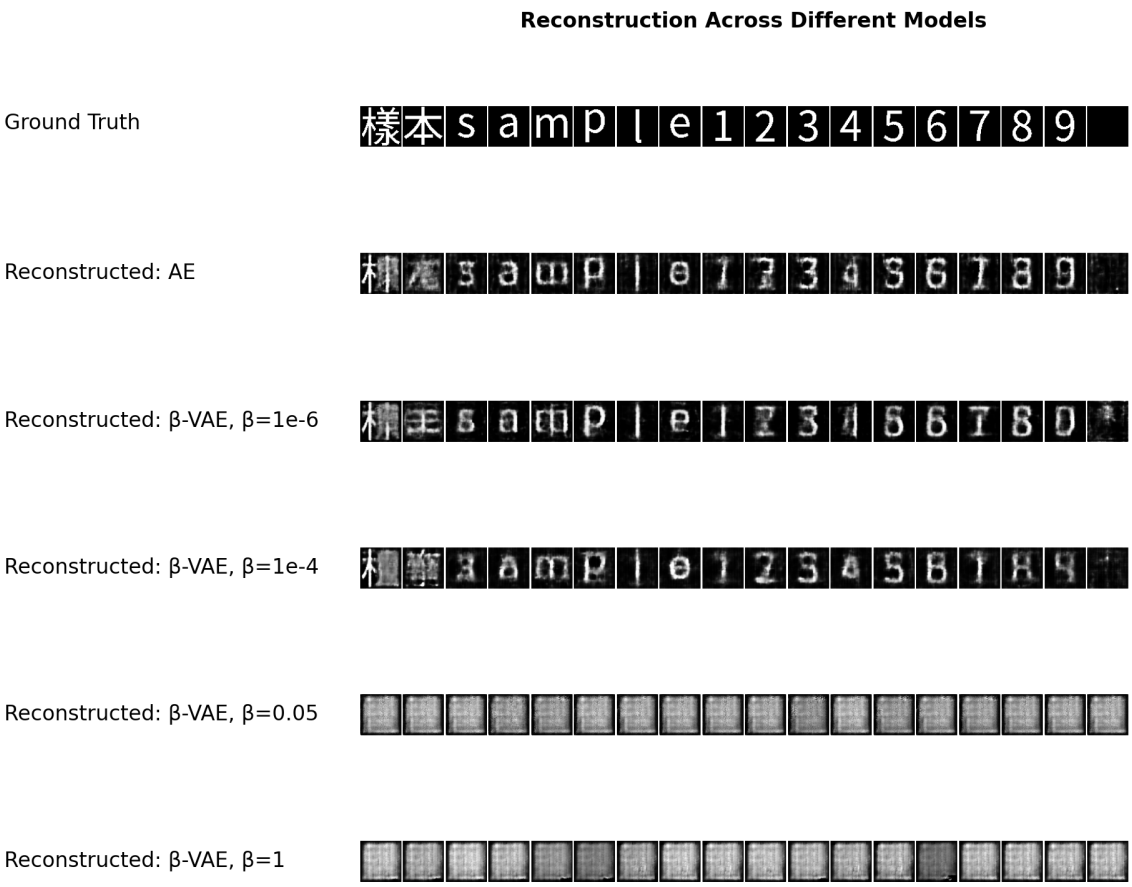


Figure 4.5: The top row shows the original input. The rows below display reconstructions from an Autoencoder (AE) and different β-VAE models (with varying β parameters). The feature's dimension for all models is set to 128.

doi:10.6342/NTU202503286

tion, we project the 128-dimensional visual feature to 768 dimensions using the projection architectures described in Section 3.3 (see also Figure 3.3). The projected features are then added to the original token embeddings via element-wise addition, enriching BERT's token embeddings with character-level visual features while preserving its pre-trained semantic representations. The overall integration process is illustrated in Figure 4.6.

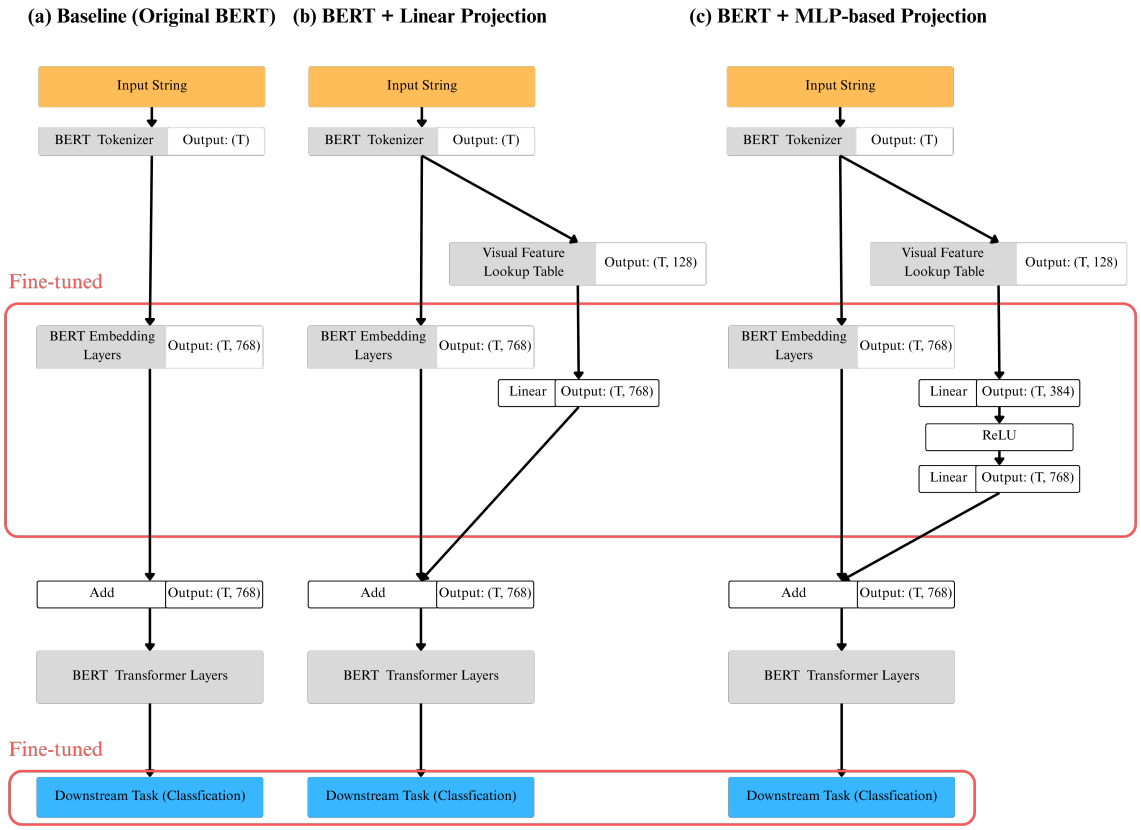


Figure 4.6: Comparison of three architectures: (a) baseline BERT, (b) linear projection, and (c) MLP-based projection for integrating character-level visual features. Visual features are projected to match BERT's embedding dimension and fused via addition at the input layer. Red boxes indicate components updated during fine-tuning, while the rest of the model remains frozen.

doi:10.6342/NTU202503286

4.4 Evaluating Visual Integration on Fine-grained Token Tasks

To evaluate the usefulness of the integrated visual features, we formulate a character-counting classification task as a downstream fine-tuning objective for BERT. Specifically, the model is trained to predict how many times a specific character appears in a given word, cast as a masked language modeling problem.

4.4.1 Real English Word Counting

Data Source. We construct the dataset from the SCOWL (Spell Checker Oriented Word Lists) and Friends database,[1] yielding a vocabulary of `614,557` English words. For each word, we generate prompts of the form:

```
There are [MASK] <char> in <word>.
```

The model is trained to predict the number of times the target character occurs in the word, using the `[MASK]` position as a classification label. For example, given the word `apple` and the target character `p`, the prompt becomes:

```
There are [MASK] p in apple.
```

In this case, the correct label is `2`, since the character `p` appears twice.

In total, this procedure produces `4,645,756` questions. We visualize the distribution of letter frequencies in Figure 4.7, which shows that most questions have answers in the

[1] http://wordlist.aspell.net/

doi:10.6342/NTU202503286

range of 1, 2, or 3. For simplicity, we exclude questions where the target character appears more than 10 times in the word.

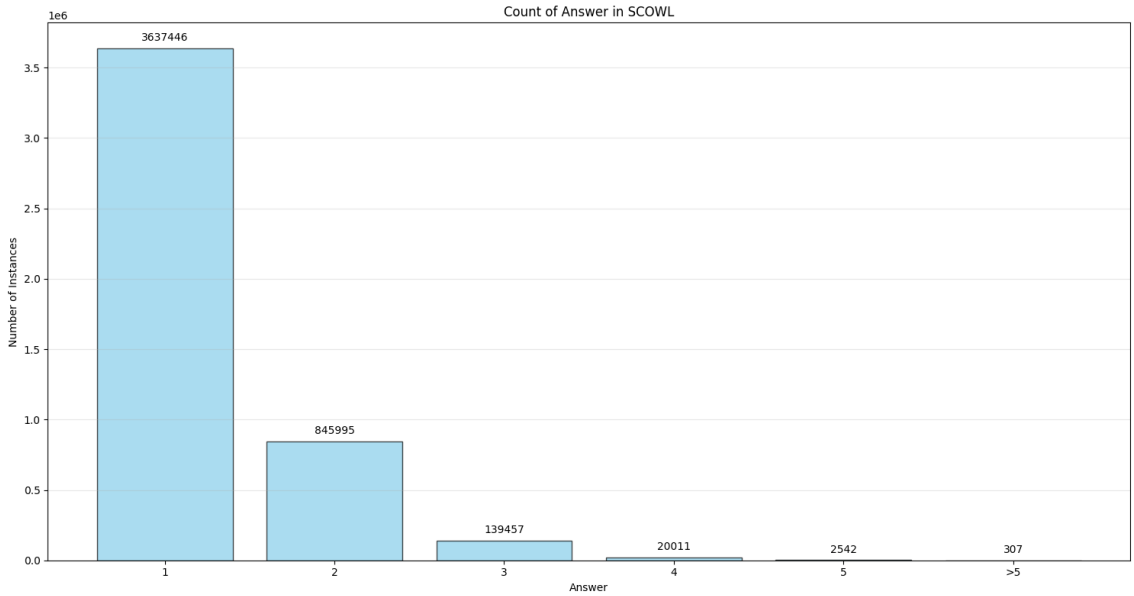


Figure 4.7: Distribution of character counts per word in the dataset.

Fine-tuning Procedure. As shown earlier, we update only a small subset of parameters during fine-tuning: the BERT token embeddings, visual feature projection layers, and a new classification head. The rest of the BERT encoder remains frozen. The classifier projects the hidden representation at the `[MASK]` position to logits over counts 1–10, optimized with cross-entropy loss.

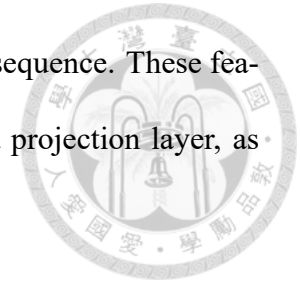
Comparative Evaluation We compare the baseline BERT model against variants integrated with visual embeddings from two generative models:

- Autoencoder (AE)
- Beta-VAE ($\beta = 10^{-6}$)

The choice of a very small β is intended to prioritize reconstruction quality, so that the

doi:10.6342/NTU202503286

visual features effectively capture the overall shape of the character sequence. These features are then injected into BERT via either a linear or MLP-based projection layer, as discussed below.



Results. Figure 4.8 and Figure 4.9 summarize the test loss and accuracy across training epochs. Results averaged over 10 random seeds show:

- **Using AE-Based Visual Features:** MLP projection substantially improves over the baseline, while linear projection struggles-suggesting AE embeddings need non-linear mapping.
- **Using β-VAE-Based Visual Features:** Both projection types outperform baseline, benefiting from VAE' s structured latent space. MLP projection further improves performance.

These results confirm that integrating structured visual features can enhance character-level reasoning in language models, especially when paired with an expressive projection mechanism.

doi:10.6342/NTU202503286

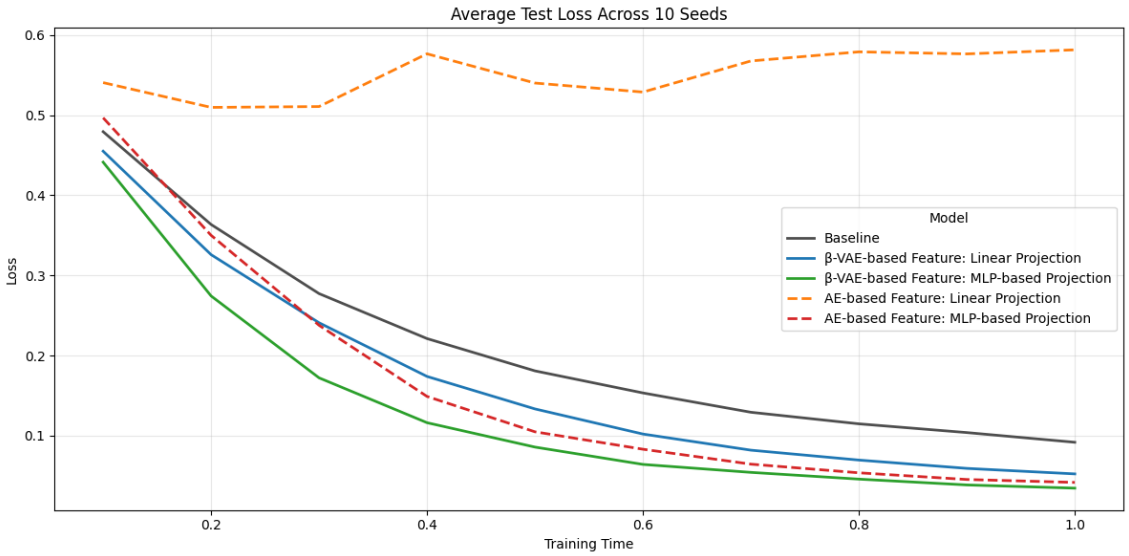


Figure 4.8: Test loss curves during fine-tuning for baseline and visual feature variants (AE, β-VAE) using linear and MLP projections. Results are averaged over 10 seeds.

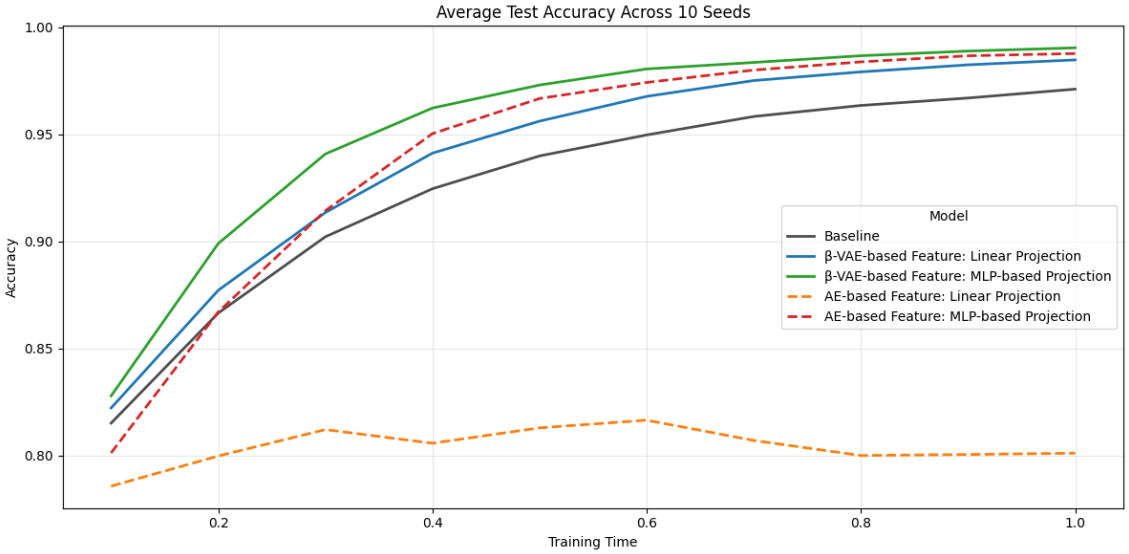


Figure 4.9: Test accuracy curves during fine-tuning for baseline and visual feature variants (AE, β-VAE) using linear and MLP projections. Results are averaged over 10 seeds.

doi:10.6342/NTU202503286

Statistical Comparison. We conducted paired t tests on 10 random seeds to assess the significance of the accuracy differences between each projection method and the BERT-only baseline. Results from both the β-VAE and standard autoencoder (AE) variants are summarized in Table 4.1.

For the β-VAE setting, both the Linear and MLP projections significantly outperform the baseline ($p < 0.0001$). In the AE setting, the MLP projection also achieves significant improvement ($p < 0.0001$), while the Linear projection unexpectedly leads to worse accuracy ($p < 0.0001$). Detailed per-seed accuracies and test statistics are provided in Appendix 5.4.

Table 4.1: Paired t-test results vs. baseline across 10 seeds. "↑" means improvement, "↓" means decrease in accuracy.

Visual Feature	Comparison	t statistic	p-value	Effect
β-VAE-based	Linear vs. Baseline	-6.087	1.8×10^{-4}	↑ significant
β-VAE-based	MLP vs. Baseline	-15.402	$< 10^{-6}$	↑ significant
AE-based	Linear vs. Baseline	9.807	$< 10^{-6}$	↓ significant
AE-based	MLP vs. Baseline	-10.978	$< 10^{-6}$	↑ significant

doi:10.6342/NTU202503286

Chapter 5 Conclusion and Discussion

5.1 Discussion and Limitations

Our comparative analysis of AE and VAE integrations reveals several key insights:

5.1.1 Representation Quality and Structural Feature Space

- **Autoencoder (AE)**: Prioritizing reconstruction fidelity, AE-based visual features capture detailed visual patterns of character sequences but may bundle diverse visual factors (e.g., stroke orientation, curvature) into shared dimensions. Consequently, a non-linear MLP projection is needed to disentangle task-relevant aspects from the richer AE feature space, and linear mappings alone can sometimes degrade downstream accuracy.

- **β-Variational Autoencoder (β-VAE)**: The structured feature space encouraged by KL-divergence regularization in the VAE seems to yield features that lend themselves more naturally to downstream use, possibly by organizing visual features in a more consistent manner. This structural regularity reduces the reliance on complex projection networks—linear mappings suffice to extract meaningful signals—resulting in robust task improvements even with shallow injection architectures.

doi:10.6342/NTU202503286

5.1.2 Computational and Practical Constraints

Our offline preprocessing requires rendering and encoding every token in the vocabulary, which scales linearly with vocabulary size but remains feasible for typical subword vocabularies (e.g., BERT's 30K tokens).

In terms of model size, we compare three variants used in the experiment. The additional components introduce only a minor increase in the number of trainable and total parameters.

For the linear projection variant, the number of trainable parameters increased by approximately 0.4%, and the total parameter count increased by about 0.1%.

For the MLP variant, the increase was slightly larger, with trainable parameters growing by 1.4% and total parameters by 0.3%.

In both cases, the increases in trainable parameters remain well under 2% and the increases in total parameters remain under 0.3%, suggesting that the additional capacity is negligible in practice. Given the potential gains in character-aware representation learning, we consider our approach a lightweight extension to standard transformer-based architectures.

5.2 Future Work

Building on these findings, several avenues remain to be explored:

1. **Data Diversity**: Extend the visual feature pipeline to multiple fonts and styles (italic, bold) to assess robustness across typographic variations.

doi:10.6342/NTU202503286

2. **Model Scale and Architecture**: Evaluate the injection framework on larger transformer models (e.g., GPT-3, LLaMA) and on subword vocabularies derived from BPE or SentencePiece. This could reveal how character-level features interact with scale and tokenization schemes.

3. **Semantic Preservation**: Measure the impact of visual feature injection on models' original semantic capabilities (e.g., GLUE benchmarks) to quantify any trade-offs between fine-grained reasoning and general language understanding.

4. **Task Generalization**: Explore other character-sensitive tasks beyond counting, such as anagram generation, palindrome recognition, or typographical error correction, to test the generality of the additional feature injection.

5.3 Conclusion

In this thesis, we proposed a novel framework for enriching pre-trained subword-based language models with compact character-level visual features. By training lightweight autoencoders and β on rendered character sequences and injecting their visual features into a frozen BERT model via linear and MLP projection layers, we demonstrated that:

- **Character-Level Injection Improves Counting Accuracy**: Both AE and β-VAE-based features significantly improved BERT's performance on a fine-grained character-counting classification task compared to the baseline.

- **Projection Complexity Matters**: MLP-based projection consistently outperformed simple linear mapping for AE-based features, indicating that non-linear transformations better capture complex relationships between visual and semantic spaces.

doi:10.6342/NTU202503286

- **Structured Feature Spaces Help**: Even a shallow linear projection of β-VAE-based features delivered gains, suggesting that the structured feature space resulting from the β-VAE's KL-divergence regularization yields features that better reflect consistent visual patterns useful for downstream tasks.

Overall, the results support the hypothesis that subword token embeddings can be effectively augmented with additional visual features to enhance character-sensitive reasoning without incurring the full computational cost of pure character- or pixel-level models.

5.4 Overall Reflection

This work demonstrates the promise of lightweight, hybrid subword–character modeling, showing that a modest injection of visual features can materially enhance fine-grained text reasoning. While several practical and theoretical questions remain open, the proposed framework offers a flexible foundation for future research at the intersection of vision and language modeling.

doi:10.6342/NTU202503286

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

[2] J. H. Clark, D. Garrette, I. Turc, and J. Wieting. <scp>canine</scp>: Pre-training an efficient tokenization-free encoder for language representation. Transactions of the Association for Computational Linguistics, 10:73–91, 2022.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[5] P. Gage. A new algorithm for data compression. C Users Journal, 1994. http://www.pennelynn.com/Documents/CUJ/HTML/94HTML/19940045.HTM.

doi:10.6342/NTU202503286

[6] Google. Noto Sans. https://fonts.google.com/noto/specimen/Noto+Sans, 2024. Version used: Regular. Accessed: July 2025.

[7] A. Gourabathina, W. Gerych, E. Pan, and M. Ghassemi. The medium is the message: How non-clinical information shapes clinical decisions in llms. In Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency, FAccT '25, page 1805–1828, New York, NY, USA, 2025. Association for Computing Machinery.

[8] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In International Conference on Learning Representations, 2017.

[9] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022.

[10] T. Kudo and J. Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 2018.

[11] Z. Sun, X. Li, X. Sun, Y. Meng, X. Ao, Q. He, F. Wu, and J. Li. ChineseBERT: Chinese pretraining enhanced by glyph and Pinyin information. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 2065–2075, Online, Aug. 2021. Association for Computational Linguistics.

[12] Y. Tai, X. Liao, A. Suglia, and A. Vergari. Pixar: Auto-regressive language modeling in pixel space, 2024.

[13] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.

doi:10.6342/NTU202503286

[14] L. Xue, A. Barua, N. Constant, R. Al-Rfou, S. Narang, M. Kale, A. Roberts, and C. Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models, 2022.

[15] Y. Zhang. A better autoencoder for image: Convolutional autoencoder. In ICONIP17-DCEC. Available online: http://users. cecs. anu. edu. au/Tom. Gedeon/conf/ABCs2018/paper/ABCs2018_paper_58. pdf (accessed on 23 March 2017), 2018.

doi:10.6342/NTU202503286

doi:10.6342/NTU202503286

Appendix A — Detailed Results and Statistical Tests

A.0.1 Per-seed Evaluation Accuracy (β-VAE)

Table A.1: Evaluation accuracy across 10 random seeds for β-VAE variants.

Seed	Baseline	Linear	MLP
0	0.9749	0.9833	0.9901
1	0.9766	0.9778	0.9902
2	0.9739	0.9832	0.9911
3	0.9669	0.9878	0.9917
4	0.9711	0.9878	0.9908
5	0.9642	0.9858	0.9897
6	0.9714	0.9887	0.9911
7	0.9746	0.9800	0.9914
8	0.9722	0.9868	0.9899
9	0.9722	0.9856	0.9896

A.0.2 Per-seed Evaluation Accuracy (AE)

doi:10.6342/NTU202503286

Table A.2: Evaluation accuracy across 10 random seeds for AE variants.

Seed	Baseline	Linear	MLP
0	0.9749	0.9607	0.9862
1	0.9766	0.7837	0.9836
2	0.9739	0.7834	0.9880
3	0.9711	0.7817	0.9905
4	0.9711	0.7831	0.9884
5	0.9711	0.7832	0.9904
6	0.9714	0.7835	0.9896
7	0.9746	0.7842	0.9883
8	0.9722	0.7842	0.9825
9	0.9722	0.7828	0.9902

doi:10.6342/NTU202503286

Appendix B — List of Font Files Utilized in Glyph Rendering

- `Noto_Sans_CJK_Regular.otf`
- `NotoSans_Condensed-Regular.ttf`
- `NotoSans_ExtraCondensed-Regular.ttf`
- `NotoSans_SemiCondensed-Regular.ttf`
- `NotoSans-Regular.ttf`
- `NotoSansArabic-Regular.ttf`
- `NotoSansArmenian-Regular.ttf`
- `NotoSansBengali-Regular.ttf`
- `NotoSansGeorgian-Regular.ttf`
- `NotoSansHebrew-Regular.ttf`
- `NotoSansJP-Regular.ttf`
- `NotoSansKannada-Regular.ttf`
- `NotoSansKR-Regular.ttf`

doi:10.6342/NTU202503286

- NotoSansMath-Regular.ttf
- NotoSansMyanmar-Regular.ttf
- NotoSansSinhala-Regular.ttf
- NotoSansTamil-Regular.ttf
- NotoSansThai-Regular.ttf
- NotoSerifTibetan-Regular.ttf

doi:10.6342/NTU202503286