Master Thesis

Towards a Foundation Model for the Analysis of Environmental Audio Data Using the Transformer and Mamba Architectures

Parmenion Koutsogeorgos

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Department of Advanced Computing Sciences
of the Maastricht University

Thesis Committee:

Aki Härmä

Philippe Dreesen

Maastricht University
Faculty of Science and Engineering
Department of Advanced Computing Sciences

June 27, 2025

Contents

List of Figures

List of Tables

Abstract

Environmental audio recordings captured via passive acoustic monitoring encompass diverse sounds such as bird vocalisations, weather phenomena and other outdoor sounds. While abundant, these recordings are inherently noisy, location-specific and often lack exhaustive annotations, posing challenges for traditional supervised methods. To mitigate these issues, this thesis investigates architectures and self-supervised pre-training techniques suitable for the development of foundation models for the purpose of learning generic, transferable feature representations from environmental audio data.

Our work compares two main self-supervised pre-training strategies on the GardenFiles23 dataset, which comprises approximately two years of stereo recordings from an urban garden. The strategies include masked spectrogram reconstruction, in which random patches of mel-spectrogram inputs are masked and the model learns to predict them, and a novel contrastive learning task, in which the model learns to align the two channels of stereo recordings that are masked in a complementary manner, meaning that the masked patches in one channel are unmasked in the other. We implement both strategies using two backbone architectures: a Self-Supervised Audio Spectrogram Transformer (SSAST), and a State-Space Model (SSM) variant, namely the Mamba model, which theoretically offers linear-time sequence modelling and improved efficiency over Transformers. The embeddings extracted through these methods are evaluated on three downstream tasks: bird detection, time-of-day prediction (using cyclical encoding), and weather metadata prediction (precipitation rate and average wind speed).

Experimental results reveal that masked reconstruction tends to produce more stable convergence and better performance on bird detection, whereas contrastive pre-training produces richer embeddings, which allow for stronger performance on temporal and weather metadata prediction. We also find that SSAST generally outperforms Mamba in all tasks when using short input sequences.

Chapter 1

Introduction

Environmental audio data includes a wide range of diverse audio signals and measurements captured from natural ecosystems or urban environments. These signals typically include recordings of animal activity, weather conditions, human interactions with the ecosystem and other environmental sounds. The analysis of environmental audio data offers valuable information for ecological monitoring, changes in biodiversity and the impact of human activities on natural habitats. While environmental monitoring is a particularly complex task involving multiple factors and modalities, audio data has the advantage of being abundant and relatively easy to collect and process, thus allowing for large-scale analysis of the environment, often assisted by deep learning techniques.

Environmental audio data is often collected on a large scale using passive acoustic monitoring (PAM) systems. Passive acoustic monitoring is a non-invasive method for recording and analysing sounds over extended periods using autonomous recording devices. These devices merely record the ambient soundscape without directly interacting with the environment, thus providing a realistic representation of the ecosystem.

While environmental audio data collected using PAM systems is abundant, there are multiple challenges associated with its analysis. In particular, the data is often noisy and unstructured, containing significant amounts of background noise due to weather conditions or multiple overlapping vocalisations from different species. Additionally, the information contained in the audio recordings is often location-specific, making generalisation across different environments difficult. Moreover, depending on the location of the recording devices, the data may be biased towards certain species and types of sounds, which can lead to imbalanced datasets. Finally, audio datasets collected using a PAM system suffer from the same challenge as any other large dataset, namely the difficulty of annotating the data.

Aiming to assist in addressing these challenges, this thesis explores the use of foundation models for the analysis of environmental audio data. Foundation models are large neural architectures designed to learn general-purpose representations from vast and diverse datasets. Their main advantage is that they can be pre-trained on large unlabelled datasets and then fine-tuned for specific tasks using smaller labelled datasets. Moreover, foundation models often show stronger generalisation capabilities than traditional models when applied to new tasks, even with limited labelled data.

Finally, foundation models often exhibit strong performance on few-shot learning, being able to adapt their generalised knowledge to novel tasks involving small and imbalanced datasets. These properties make foundation models particularly suitable for the analysis of environmental audio data.

The main architecture used in modern foundation models is the Transformer. Transformers have been shown to be highly effective in a wide range of tasks, including natural language processing, computer vision and audio processing. However, Transformers are often computationally expensive and require large amounts of data to train effectively. Recent work has established State-Space Models (SSMs) as a powerful alternative to Transformers. SSMs are a class of models that can capture long-range dependencies in sequential data, while being asymptotically more computationally efficient than Transformers. For this reason, most recent work focuses on their applications involving sequential data with very long context windows, such as natural language processing and time series analysis. In this thesis, we explore the use of SSMs for the analysis of environmental audio data and compare their performance to that of Transformers.

The inspiration for this thesis comes from the GardenFiles23 dataset collected and analysed by Härmä and Nazarenko [1] using a PAM setup in a garden located in an urban area of the Netherlands. The dataset consists of stereo recordings spanning a period of almost two years and includes a wide variety of environmental sounds, such as birdsong or other animal sounds, weather phenomena and human activity. Moreover, it includes a variety of associated metadata, such as timestamps and weather conditions at the time of recording. We use this dataset as a basis for our experiments.

Research questions. Through this work, we aim to answer the following research questions:

- **RQ1:** Which self-supervised pre-training tasks allow foundation models to extract useful features from environmental audio data?
- **RQ2:** How do SSMs compare to Transformers in the context of environmental audio data analysis?
- **RQ3:** What downstream tasks can be used to evaluate the quality of the learned representations and how do the models perform on these tasks?

Contributions. The main contributions of this thesis are the following:

- We establish a baseline for the development of foundation models in the context of environmental audio data analysis using a novel dataset.
- We compare masked reconstruction and a novel contrastive learning task as self-supervised pre-training tasks for environmental audio data, showing that both approaches can be effective in different contexts.
- We perform a comparison of SSM and Transformer-based foundation models for environmental audio data analysis, showing that the former offer comparable but generally inferior performance to the latter in this setup.
- We evaluate the learned representations on traditional tasks (bird detection) and novel downstream tasks (temporal and weather condition prediction), exploring the abilities and limita-

tions of foundation models in extracting complex and informative features in a self-supervised manner.

Thesis structure. The rest of this thesis is organised as follows. Chapter 2 presents an overview of related work in environmental audio processing. The following 4 chapters provide the theoretical background relevant to this work. Chapter 3 introduces the concept of foundation models, focusing on their design and feature extraction capabilities. Chapter 4 provides a detailed description of the Transformer architecture, including attention mechanisms and their computational characteristics. Chapter 5 introduces State-Space Models (SSMs), exploring their theoretical foundations, efficient implementations and recent developments, building up to the Mamba model, which has recently achieved state-of-the-art results in multiple sequence processing tasks. Chapter 6 reviews key deep learning techniques in audio processing, with emphasis on spectrogram and mel-spectrogram representations and existing foundation models for audio data.

The remaining chapters cover the relevant components of our own work. Chapter 7 describes the GardenFiles23 dataset and the pre-processing steps applied on both audio and metadata. Chapter 8 outlines the methodology, including model architectures, pre-training strategies and fine-tuning procedures. Chapter 9 presents the experimental setup and results for various pre-training tasks and downstream evaluations. Finally, Chapters 10 and 11 provide a discussion of findings and potential directions for future work.

Chapter 2

Related Work

Environmental sound classification has been a relatively unexplored topic in deep learning, with most datasets and models treating it as a subtask rather than a primary objective. Large audio datasets such as AudioSet [2] often include environmental samples, although these typically form a small part of the entire dataset. A notable benchmark is the ESC-50 dataset [3], which includes 2,000 recordings along with labels for 50 different classes, spanning categories such as animals, natural sounds, human-made sounds (excluding speech), domestic and urban sounds. Other outdoor-specific datasets such as UrbanSound8K [4] and SONYC-UST-V2 [5] focus more on human-made urban sounds and less on natural sounds. To the best of our knowledge, there is no large dataset covering the same scope as GardenFiles23, which includes a balance of natural and urban sounds, along with relevant weather and temporal metadata.

Significant progress has been made on the subtask of bird sound classification. Most notably, BirdNET [6] uses a CNN backbone trained in a supervised manner to identify 984 European and North American species by sound. Likewise, Perch [7] is a CNN model pre-trained on data from the Xeno-Canto website[1], which is a repository containing wildlife sounds. The model is trained on weakly labelled data, with an activity detector selecting bird sound segments from longer recordings for the purpose of predicting bird taxonomy. More recently, Vengrovski et al. introduce TweetyBERT [8], a Transformer-based foundation model designed for analysing birdsong. TweetyBERT employs self-supervised learning by predicting masked parts of birdsong sequences, allowing it to learn meaningful representations without requiring labelled data.

Deep learning has also been applied to weather-related acoustic signals, although dedicated datasets are scarcer. For example, Avanzato and Beritelli developed a CNN-based model that classifies rain sound timbres into four intensity categories. The associated dataset is collected via smartphone recordings and validated by a physical rain gauge. Additionally, Wang et al. introduce the Surveillance Audio Rainfall Intensity Dataset (SARID) [10], which contains six rainfall events segmented into 12,000 labelled audio clips, each annotated with ground-truth rain rate and environmental context. The authors also propose a Transformer-based architecture to predict rainfall intensity. Wind noise detection has also been explored, especially for filtering microphone data, though this

[1] `https://xeno-canto.org/`

is often treated as a noise-removal problem rather than target classification.

On the other hand, large foundation models developed for generic audio classification often include environmental labels as a part of their output. For example, the Audio Spectrogram Transformer (AST) [11, 12] is a Transformer-based model trained on the AudioSet dataset, which achieves state-of-the-art results on AudioSet and ESC-50 benchmarks. Closely related to our work is the Self-Supervised Audio Spectrogram Transformer (SSAST) [13], which has established a now widely adopted self-supervised pre-training pipeline for audio data. We explore foundation models for audio data in more detail in Chapter 6, where we review current approaches including both Transformer and SSM-based architectures. To the best of our knowledge, there is no existing work attempting to develop a foundation model aiming to assist in the analysis of environmental audio data using a dedicated dataset.

Chapter 3

Foundation Models

In this chapter we introduce the concept of foundation models and examine their role in generic feature extraction. We also mention some of the main self-supervised tasks which are used to train foundation models, two of which (namely, masked reconstruction and contrastive learning) we use as the basis of our own pre-training experiments.

3.1 The foundation model pipeline

Foundation models are large-scale neural networks that are trained on a large corpus for the purpose of building a robust backbone that can be fine-tuned for a wide variety of downstream tasks. Their main advantage is that once pre-trained, they frequently achieve state-of-the-art performance on new tasks with minimal fine-tuning or even none at all and are now widely used across natural language, vision, audio and multimodal applications.

To formalise the idea of foundation models, we note that all supervised learning tasks can be formulated as a function approximation problem, in which the goal is to learn a function $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ depending on parameters θ that best approximates an unknown target function $f^* : \mathbb{R}^n \to \mathbb{R}^m$ with respect to a certain loss criterion. We may view the function f_θ as a composition

$$f_\theta = h_\phi \circ g_\omega, \quad \theta = (\phi, \omega),$$

where g_ω is a feature extractor mapping the input x to a latent representation $z = g_\omega(x)$ and h_ϕ is a task-specific head mapping z to the desired output $y = h_\phi(z)$.

The goal of a foundation model is to learn a generic feature extractor g_ω that produces representations with enough statistical information about the underlying unlabelled data distribution, so that it can be reused as a backbone for multiple diverse downstream tasks. The pipeline for training and using a foundation model is summarised as follows:

1. **Pre-training**: The model is trained on a large dataset, often using self-supervised tasks with the ultimate goal of extracting useful features from the data.

2. **Fine-tuning**: The feature extractor g_ω of the pre-trained model is kept, while the head is replaced with a task-specific MLP h_ϕ. Typically for large models the feature extractor is frozen, while the head is fine-tuned on a smaller labelled dataset for downstream tasks such as classification or regression. Since h_ϕ is typically much smaller than g_ω, this can often be achieved using fewer labelled examples and in much less training time than training a new model from scratch.

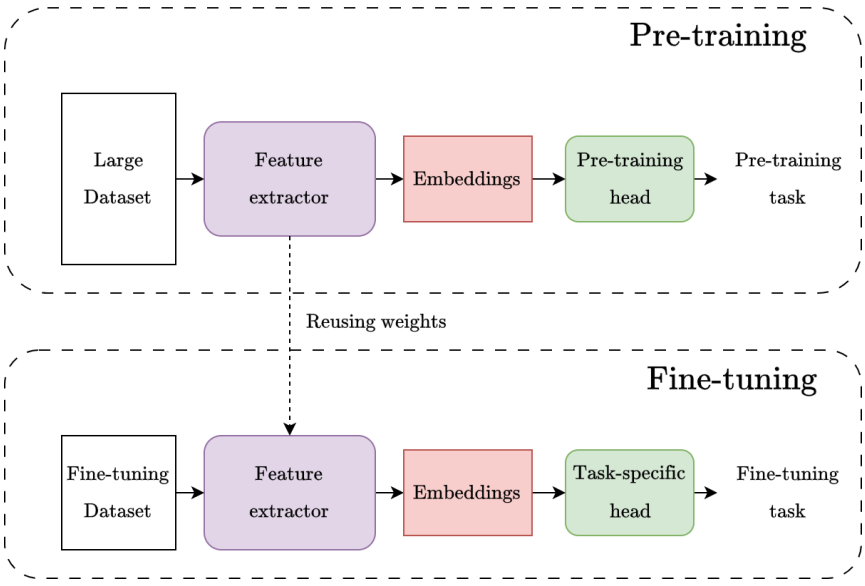


Figure 3.1: The foundation model pipeline.

Foundation models and transfer learning. The idea of foundation models is based on the common practice of transfer learning, in which a model is trained on a large generic dataset and then fine-tuned for a particular task. One key difference between generic feature extractors and traditional transfer learning is that in the latter case, the pre-training and the fine-tuning tasks are both supervised and closely related and often involve massive open-source labelled datasets that have been curated over time for training and evaluating such models. Early examples of transfer learning in the literature include CNNs such as VGG [14], ResNet [15] or the YOLO models [16], which are typically pre-trained on the labelled ImageNet [17] or COCO [18] datasets and used for tasks including image classification, object detection or segmentation. More recently, the Segment Anything model [19] has been proposed as a foundation model for image segmentation, trained on a semi-automatically generated dataset of mask-prompt pairs, where prompts may be internal points, bounding boxes or text descriptions of their associated masks.

3.2 Foundation models for feature extraction

We now focus on foundation models trained for generic feature extraction. The pre-training stage often involves one or several self-supervised tasks, which has two main advantages:

- It allows for much larger datasets to be used for training, as it is often easier to collect unlabelled data than labelled data.
- It allows for the model to learn a more generic representation of the data, as it is not biased by the specific labels used in supervised training.

Typical self-supervised pre-training tasks include:

- **Masked token prediction/reconstruction**: The model is trained to reconstruct the original input from a masked version of it. This is often done by randomly masking a portion of the input and training the model to predict the masked tokens. In the case of a discrete vocabulary, such as in NLP, this is treated as a classification task, i.e., the model produces a probability distribution over the vocabulary for each masked token and the loss is computed as the cross-entropy between the predicted and true tokens. In the case of continuous inputs, such as images, the reconstruction is treated as a regression task. Typical losses used in the continuous case include the (masked) *Mean Squared Error* (MSE):

$$\text{MSE}(y, \hat{y}) = \frac{1}{nN} \sum_{i=1}^{N} \|y_i - \hat{y}_i\|_2^2 , \tag{3.1}$$

  and the (masked) *Mean Absolute Error* (MAE):

$$\text{MAE}(y, \hat{y}) = \frac{1}{nN} \sum_{i=1}^{N} \|y_i - \hat{y}_i\|_1 , \tag{3.2}$$

  where y_i is the ground-truth values of the i'th masked token, $\hat{y}_i$ is the corresponding predicted token, N is the total number of masked tokens and n is the dimensionality of $y_i, \hat{y}_i$.

- **Next token prediction/reconstruction**: The model is trained to predict the next token in a sequence given the previous tokens. This is particularly common in fields that involve sequences with a clear ordering, for example in NLP where the model is trained to predict the next word in a sentence given the previous words. Similar to masked token prediction, this is treated as a classification task in the case of discrete tokens and as a regression task in the case of continuous tokens.

- **Contrastive learning**: The model is trained to distinguish between similar and dissimilar samples. A common technique for contrastive learning is SimCLR [20], which constructs *positive pairs* by applying two different augmentations to the same input sample, while all other pairs in the batch where the two views come from different samples are treated as *negative pairs*. Let N be the total number of samples in the batch, so that $2N$ is the number of augmented samples and let $z = (z_1, \ldots, z_{2N})$ be the embeddings extracted from all the augmented samples. The model is trained using the *InfoNCE* loss [21], which for a single positive pair (z_i, z_j) is defined as:

$$\mathcal{L}_{i,j} = -\log \left(\frac{\exp\left(\text{sim}\left(z_i, z_j\right)/\tau\right)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} \exp\left(\text{sim}\left(z_i, z_k\right)/\tau\right)} \right), \tag{3.3}$$

  where

$$\text{sim}(z_i, z_j) = \frac{\langle z_i, z_j \rangle}{\|z_i\|_2 \|z_j\|_2}$$

is the cosine similarity between representations and τ is a temperature hyperparameter controlling the sharpness of the softmax distribution. The final loss is computed by averaging over all positive pairs in the batch:

$$\text{InfoNCE}(z) = \frac{1}{2N} \sum_{i=1}^{2N} \mathcal{L}_{i,\mathcal{P}(i)} \tag{3.4}$$

where $\mathcal{P}(i)$ denotes the index of the positive pair for sample i.

The advancement of the Transformer architecture and training paradigms has led to the development of many large-scale and multi-purpose foundation models. One of the earliest examples of the foundation model pipeline is BERT [22], which is an encoder-only Transformer that is trained on a large corpus of unlabelled sentences using a combination of a masked language modelling (i.e. predict masked tokens in a sentence) and next sentence prediction (i.e. predict if two sentences appear consecutively in the original corpus or not). BERT-based architectures are commonly used as embedding extractors in NLP tasks and have been successfully applied on a wide variety of tasks such as topic modelling, sentiment analysis, named entity recognition, question answering and more.

In the field of computer vision, many foundation models have been developed based on the Vision Transformer (ViT) architecture [23], which was originally developed for image classification. For example, DINOv2 [24] trains a ViT backbone using a teacher-student self-distillation framework, in which the model learns to align embeddings of different augmented views of the same image. Models from the field of computer vision are commonly used also in audio processing tasks, as they can process spectrograms as images. We discuss this practice in more detail in chapter 6.

Chapter 4

Transformers

In this chapter we explain the inner workings of the Transformer architecture, focusing in particular on the attention calculation. We also discuss the complexity of the attention mechanism and how it can be optimised for modern hardware, two concepts which we will refer to for comparison purposes when we discuss the SSM architecture in chapter 5.

4.1 The Transformer encoder

The *Transformer* [25] is a neural network architecture originally developed for machine translation that has been widely adopted in most deep learning domains, achieving state-of-the-art results in fields such as natural language processing, computer vision and audio processing. Its core component is the attention mechanism, which allows the model to weigh and aggregate information from different positions in the input sequence, enabling effective modelling of long-range dependencies. Unlike convolutional or recurrent networks, which process inputs locally or sequentially, the Transformer processes all input tokens in parallel and captures global context at every layer, making it more efficient and expressive even for inputs of high dimensionality.

The standard Transformer architecture consists of an encoder-decoder structure, where the encoder processes the input sequence and produces representations for it and the decoder generates the output sequence based on the encoder's representations. However, many Transformers consist only of an encoder or a decoder, depending on the task. In particular, encoder-only Transformers are often used as the backbone of foundation models, because of their ability to generate high-quality context-aware embeddings. The main components of an encoder-only Transformer architecture are:

- The input embedding layer.
- The positional encoding layer.
- A sequence of blocks, each of which consists of a multi-head (self-)attention layer, the outputs of which are combined via a multi-linear perceptron (MLP) layer.

This architecture is illustrated in Figure 4.1.

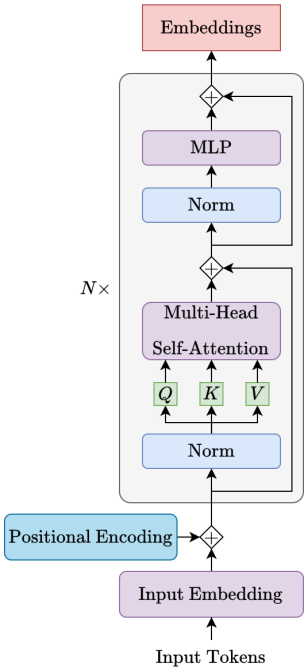


Figure 4.1: The Transformer encoder architecture.

Input representation. For NLP tasks, the input to the Transformer is a sequence of discrete tokens drawn from a fixed vocabulary. Each token can be represented as an indicator vector (one-hot encoding) and the input sequence is usually linearly ordered.[1] In other domains such as image or audio processing, the input is inherently continuous and lacks an obvious discrete vocabulary or natural token boundaries. In these cases, the raw input is typically divided into fixed-size (possibly overlapping) patches or windows. For example, an image may be partitioned into 2D patches, each of which is flattened into a vector and treated as a token in the sequence. Similarly, a spectrogram derived from an audio signal may be processed as a grid of time-frequency patches. These patches are typically derived directly from the raw input. Depending on the application however, quantisation techniques such as vector or product quantisation [26, 27], sometimes assisted by other neural networks [28], may be used to discretise the input tokens.

Input embedding layer. The purpose of the embedding layer is to map raw inputs into a shared embedding space that can be processed by the subsequent Transformer layers. Given an input sequence $x = (x_1, \ldots, x_l) \in \mathbb{R}^{l \times d}$, the embedding layer is a trainable function $E : \mathbb{R}^d \to \mathbb{R}^{d_E}$ that maps each input element x_i to a d_E-dimensional vector. When the input consists of discrete tokens, such as words or subwords in NLP, each x_i is typically represented as an one-hot vector over a fixed vocabulary. In this case, the embedding layer is essentially a trainable lookup table $E \in \mathbb{R}^{d \times d_E}$, where d becomes the total length of the vocabulary, the i'th row of which provides the

[1] However, even in the context of NLP, these assumptions are violated in many languages other than English, where word boundaries, morphology or writing systems differ significantly.

learned embedding of the i'th token in the vocabulary. For continuous inputs, such as flattened image patches or raw audio features, there is typically no fixed vocabulary or one-hot encoding. Instead, each input token x_i is already a real-valued vector and the embedding layer acts as a learned linear projection.

Positional encoding. The output values of the attention mechanism are permutation-invariant and do not inherently encode the order of the input tokens. To address this, a positional encoding is added to the input embeddings to incorporate information about token positions. Given an input sequence $x \in \mathbb{R}^{l \times d}$, the positional encoding matrix $P \in \mathbb{R}^{l \times d}$ is added element-wise to produce a location-aware representation $x_p = x + P$. This augmentation allows the Transformer to distinguish between tokens based not only on their content but also on their position in the sequence.

There are several common strategies for constructing positional embeddings. One approach introduced in the original Transformer paper [25] uses fixed *sinusoidal encodings*, where each position is mapped to a vector based on sine and cosine functions of different frequencies. These encodings can easily generalise to longer sequences than those seen during training and can also indirectly encode relative positional information, as for any fixed offset k there is a linear map R_k such that $P_{i+k} = R_k P_i$. Alternatively, *learned absolute positional encodings* [25] treat P as a set of trainable parameters. Since P remains invariant for all inputs, its tokens retain positional information which is additionally tailored to the data distribution. A third strategy is to use *relative positional encodings* [29], which represent the pairwise distance between tokens instead of their absolute position in the sequence and inject this information directly into the attention weights of each block, rather than modifying the input embeddings.

4.2 Attention mechanism

The attention mechanism is the core of the Transformer architecture, allowing the model to dynamically focus on different parts of the input sequence when producing contextualised representations. The attention mechanism operates by relating a set of *query* vectors to a set of *key* vectors and using their similarities to compute a weighted combination of corresponding *value* vectors. The naming convention of these components is inspired by the information retrieval domain, where queries are used to search for relevant documents (keys) and retrieve their content (values). Informally, the attention mechanism can be thought of as a type of retrieval process as well; for each output token (query), the model computes how relevant each input token (key) is and uses these relevance scores as weights to retrieve aggregated information from a generic encoding of the input (values). Figure 4.2 provides an illustration of the calculated attention between two sequences, using an example from machine translation, the domain in which Transformers were first used. The same concept applies to any sequence-to-sequence task.

Formally, let $Q \in \mathbb{R}^{l_Q \times d_K}$, $K \in \mathbb{R}^{l_K \times d_K}$ and $V \in \mathbb{R}^{l_K \times d_V}$ denote the matrices of queries, keys and values, where l_Q, l_K are the number of query and key/value tokens respectively and d_K, d_V are the dimensions of the query/key and value token embeddings respectively. The scaled dot-product attention is defined as

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_K}}\right) V \in \mathbb{R}^{l_Q \times d_V},$$

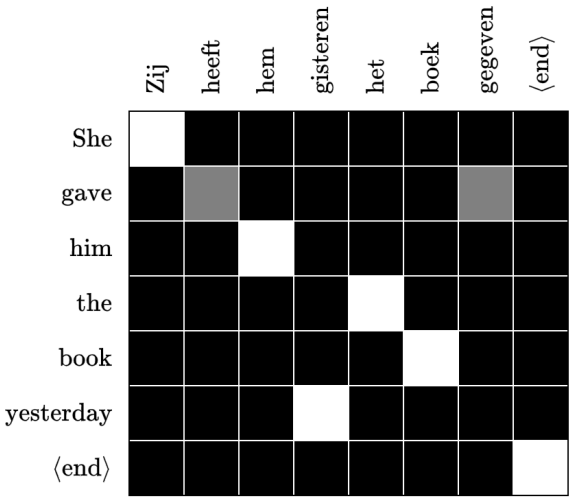


Figure 4.2: Attention heatmap during English-to-Dutch translation of "She gave him the book yesterday." Each row represents a query from an English token and each column represents a key from a Dutch token. White and gray squares indicate higher attention weights, illustrating how the attention mechanism learns to align English words with relevant Dutch words during translation.

where the inner product $QK^\top$ computes pairwise similarities between queries and keys, the scaling factor $\frac{1}{\sqrt{d_K}}$ prevents excessively large gradients for large d_K and the softmax is applied row-wise, ensuring that the weights applied to the values define a probability distribution.

Multi-head attention. To improve model capacity, the Transformer employs *multi-head attention*, in which multiple independent attention heads with different query/key/value matrices $Q_i/K_i/V_i$ of embedding dimensions $d'_K/d'_K/d'_V$ are applied and their outputs are concatenated and projected to form the final head output. The usage of multi-head attention allows the model to capture complex relationships between the input and output sequences by breaking them down into multiple small-scale similarity calculations $Q_i K_i^\top$ and to learn different token representations V_i, thus maintaining contextual information that could be lost due to averaging if only one head was used. Given h parallel attention heads, let

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) \in \mathbb{R}^{l_Q \times d'_V}, \quad i = 1, \ldots, h,$$

where

$$\begin{aligned} Q_i &= QW_i^Q \in \mathbb{R}^{l_Q \times d'_K}, \\ K_i &= KW_i^K \in \mathbb{R}^{l_K \times d'_K}, \\ V_i &= VW_i^V \in \mathbb{R}^{l_K \times d'_V}, \end{aligned}$$

for trainable projection matrices $W_i^Q \in \mathbb{R}^{d_K \times d'_K}$, $W_i^K \in \mathbb{R}^{d_K \times d'_K}$, $W_i^V \in \mathbb{R}^{d_V \times d'_V}$. Then the multi-head attention is calculated as:

$$\text{MultiHeadAttention}(X) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O \in \mathbb{R}^{l_Q \times d_M},$$

where d_M is the final embedding dimension of the head and $W^O \in \mathbb{R}^{(h \cdot d'_V) \times d_M}$ is once again a trainable projection.

Self-attention. *Self-attention* is a special case of attention in which the queries, keys and values are all derived from the same input sequence $x \in \mathbb{R}^{l \times d}$ via learned linear projections:

$$Q = xW_Q, \quad K = xW_K, \quad V = xW_V,$$

for $W_Q, W_K \in \mathbb{R}^{d \times d_K}$, $W_V \in \mathbb{R}^{d \times d_V}$. Self-attention forms the basis of a Transformer-based feature extractor, utilising the generic attention mechanism's ability to capture complex relationships within the same sequence in order to produce context-aware embeddings. Figure 4.3 shows a visualisation of the multi-head self-attention mechanism.

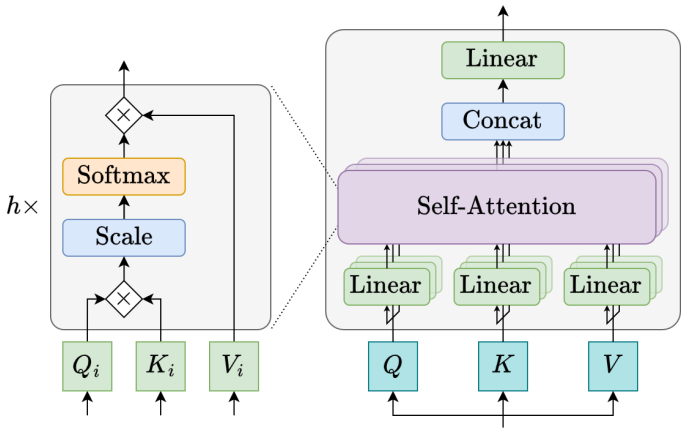


Figure 4.3: Visualisation of Multi-Head Self-Attention.

4.3 Attention complexity analysis

The main advantage of the self-attention mechanism is its ability to process the entire input sequence at once, computing attention scores between all pairs of tokens in parallel. However, this expressiveness comes at the cost of quadratic complexity in the sequence length l, since it involves pairwise comparisons between all tokens in the input sequence. To illustrate this, let $x \in \mathbb{R}^{l \times d}$ be the input sequence and assume without loss of generality that all embedding dimensions are $\mathcal{O}(d)$. Ignoring optimised matrix multiplication implementations which are typically not used in this context, we may assume that multiplying a matrix of size $p \times q$ with one of size $q \times r$ requires $\mathcal{O}(pqr)$ operations. The self-attention mechanism then proceeds through the following steps:

1. It computes the query/key/value matrices Q, K, V, which involves multiplication of the input x with the projection matrices W_Q, W_K, W_V and requires a total of $\mathcal{O}\left(ld^2\right)$ operations.
2. It computes the pairwise similarities $S = QK^\top$, which requires $\mathcal{O}\left(l^2 d\right)$ operations.
3. It computes the attention weights $A = \text{Softmax}\left(\frac{S}{\sqrt{d_K}}\right)$ by applying the softmax row-wise on l rows of size l, which requires $\mathcal{O}\left(l^2\right)$ operations.

4. It computes the weighted sum of values $Z = AV$, which requires $\mathcal{O}\left(l^2 d\right)$ operations.

In total, the self-attention mechanism has a complexity of $\mathcal{O}\left(l^2 d + l d^2\right)$ both during training and inference. Although the above steps are all fully parallelizable, which allows for their efficient implementation in modern hardware such as GPUs or TPUs, this quadratic complexity may be a significant limitation for long sequences, especially under constrained resources, as it can lead to high memory and computational costs in practice. As a result, a wide range of recent work focuses on reducing this cost through approximations or hardware-aware algorithms.

4.4 Hardware-aware implementations

Modern implementations of deep neural networks aim to make efficient use of the GPU memory hierarchy by fusing certain operations in order to minimise the number of slow memory accesses and they have become an essential part of training for many more architectures apart from the Transformer. For example, the Mamba model [30], which is the variant of SSMs that we use in this work, also uses a hardware-aware implementation, which is in fact one of the key innovations of the paper. Such implementations are particularly relevant for foundation models, due to the large scales involved in their training. Describing these algorithms in detail is beyond the scope of this thesis. However, we provide an illustration of how they aim to optimise GPU usage using the FlashAttention [31] implementation of the attention mechanism as an example.

FlashAttention. To understand how FlashAttention and similar optimisation algorithms work, we need to examine the memory hierarchy of a GPU, which involves two relevant components.

- The *high-bandwidth memory* (HBM) is an external memory used in GPUs to store large volumes of data such as model weights and intermediate activations. HBM provides large storage capacity (e.g., up to 80 GB on an NVIDIA A100), but it has higher *latency* (i.e., time between issuing a memory access request and receiving the data) and lower *throughput* (i.e., amount of data that can be transferred per unit of time) compared to on-chip memory, making repeated access to large tensors a significant performance bottleneck.
- The *static random-access memory* (SRAM) is a fast on-chip memory physically integrated within GPU cores. It is used to store frequently accessed data during computation, such as shared memory or registers. While SRAM has much lower latency and higher access speed than HBM, it is much more limited in size (e.g., up to 164 KB per streaming multiprocessor on an NVIDIA A100).

Standard attention implementations perform the computations in separate steps, each of which loads and stores all intermediate tensors (including the full attention matrix, which has size l^2) from HBM. FlashAttention on the other hand exploits the GPU memory hierarchy using two key ideas:

- **Tiled matrix multiplication.** It performs matrix multiplications in small blocks, sharing intermediate results in SRAM, which significantly reduces the number of HBM access requests.
- **Online softmax.** It fuses the attention operations into a single kernel by calculating the softmax incrementally using a numerically stable on-line algorithm, which eliminates the need to materialise the full attention matrix in HBM at any point.

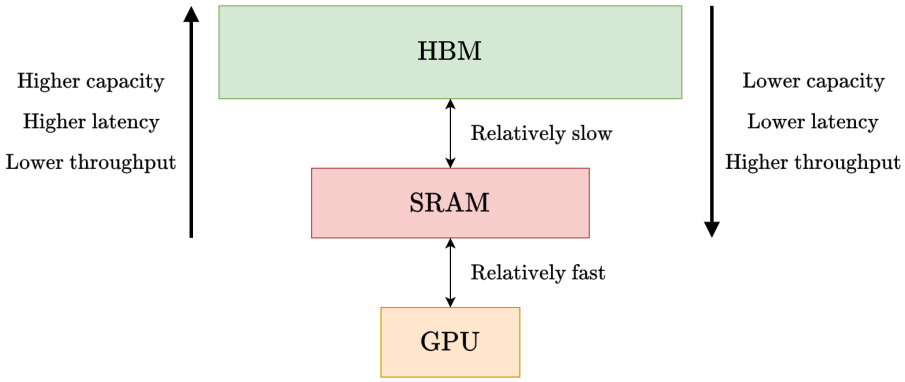


Figure 4.4: GPU memory hierarchy. The SRAM is fast but limited in capacity, while the HBM is large but slow.

Although the algorithm maintains the original theoretical time complexity of $\mathcal{O}\left(l^2d + ld^2\right)$, in practice FlashAttention offers state-of-the-art improvements in both speed and memory usage.

Chapter 5

State-Space Models

We will now discuss the SSM architecture in the context of deep learning, starting with its theoretical foundations and then moving on to its early adoption in deep learning, as well as recent developments that have led to the success of SSMs in sequence processing tasks.

5.1 Theoretical foundation

In the context of deep learning, *State-Space Models* (SSMs) refer to a class of sequence-to-sequence models inspired by traditional techniques from control theory and signal processing. From a theoretical point of view, SSMs model the evolution of a system over time using a set of latent variables (or states) that capture the system's dynamics. In recent years, SSMs have gained renewed interest in deep learning as an alternative to attention-based models, aiming to achieve similar performance without the quadratic complexity of the attention mechanism. In order to understand the modern usage of SSMs in deep learning, we need to build upon multiple theoretical ideas and practical improvements that have been made to their basic design over time. We start with an overview of the theoretical background of SSMs.

State-Space models are inspired by the state-space representation of a linear system of continuous differential equations in one variable t (time), which relates an input vector $x(t) \in \mathbb{R}^d$ with an output vector $y(t) \in \mathbb{R}^n$ through a state vector $h(t) \in \mathbb{R}^{d_M}$ as follows:

$$\begin{aligned} \dot{h}(t) &= \mathbf{A}h(t) + \mathbf{B}x(t), \\ y(t) &= \mathbf{C}h(t) + \mathbf{D}x(t), \end{aligned} \tag{5.1}$$

for parameters

$$\begin{aligned} &\mathbf{A} \in \mathbb{R}^{d_M \times d_M} && \text{(state matrix)}, \\ &\mathbf{B} \in \mathbb{R}^{d_M \times d} && \text{(input matrix)}, \\ &\mathbf{C} \in \mathbb{R}^{n \times d_M} && \text{(output matrix)}, \\ &\mathbf{D} \in \mathbb{R}^{n \times d} && \text{(feed-forward matrix)}. \end{aligned}$$

In practice the continuous system is discretised, transforming into a sequence-to-sequence mapping

$$x = (x_1, \ldots, x_l) \in \mathbb{R}^{l \times d} \mapsto y = (y_1, \ldots, y_l) \in \mathbb{R}^{l \times n}$$

which is reformulated for $t = 1, \ldots, l$ as follows:

$$\begin{aligned} h_t &= \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t, \\ y_t &= \mathbf{C} h_t + \mathbf{D} x_t. \end{aligned} \tag{5.2}$$

By convention in (5.2) we set $h_0 = 0$.

Various well-known ODE discretisation methods can be used to obtain (5.2) from (5.1), through the use of a timescale parameter Δ which controls the granularity of the discretisation. For example the *generalised bilinear transform* is defined as:

$$\begin{aligned} \overline{\mathbf{A}} &= \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1} \left(\mathbf{I} + \frac{\Delta}{2}\mathbf{A}\right), \\ \overline{\mathbf{B}} &= \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1} \Delta \mathbf{B}, \end{aligned} \tag{5.3}$$

while the *zero-order hold* discretisation is given by:

$$\begin{aligned} \overline{\mathbf{A}} &= \exp(\Delta \mathbf{A}), \\ \overline{\mathbf{B}} &= (\Delta \mathbf{A})^{-1} \left(\exp(\Delta \mathbf{A}) - I\right) \Delta \mathbf{B}. \end{aligned} \tag{5.4}$$

Equation (5.2) can be replicated by a neural network, referred to as a State-Space Model (SSM), with input x, output y, hidden state representation h and trainable parameters

$$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \Delta \;\leftrightarrow\; \overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C}, \mathbf{D}.$$

In this context, parameter $\mathbf{D}$ is often omitted from the formulation, as it can be viewed as a skip connection between the network's layers. Thus, from now on we will be adopting the following setup for SSMs in the context of deep learning:

$$\begin{aligned} h_t &= \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t, \\ y_t &= \mathbf{C} h_t. \end{aligned} \tag{5.5}$$

A significant advantage of the SSM is that in principle it can be computed both as a recurrence as well as a convolution, combining the fast inference time of RNNs with the parallelisation capabilities of CNNs and Transformers.

Recurrent computation of the SSM. For a given Δ, (5.5) defines a recurrent formula that can be used to iteratively calculate the output of the system, using constant memory and having constant computational requirements per time step (in l), similar to the RNN architecture. This is important upon inference, especially when generating long sequences (e.g., when generating text token by token given a long prompt and history). Figure 5.1 illustrates the SSM model as a recurrent network.

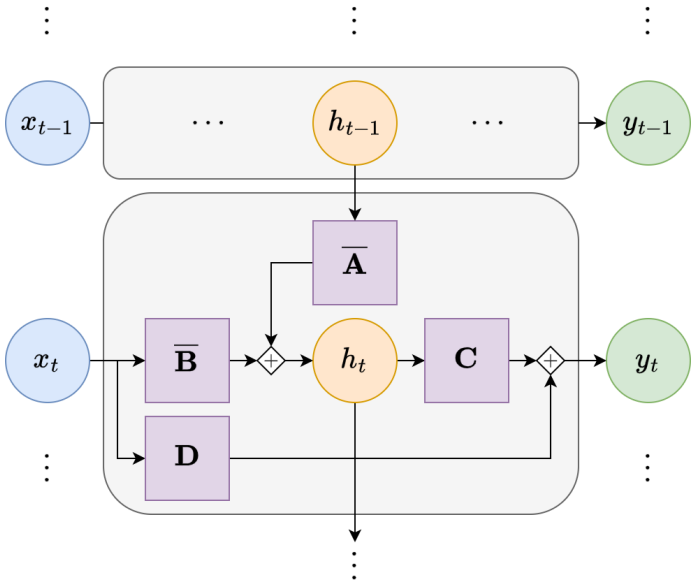


Figure 5.1: The SSM model as a recurrence.

Assuming without loss of generality that n, d_M are $\mathcal{O}(d)$ and taking into account all the necessary matrix multiplications, the computational complexity of the SSM is $\mathcal{O}(d^2)$ per time step, for a total of $\mathcal{O}(ld^2)$ complexity for the entire sequence. This is significantly more efficient than the $\mathcal{O}(l^2d + ld^2)$ complexity of the attention mechanism, especially for long sequences.

Convolutional computation of the SSM. It is not immediately obvious how the iterative inference can be parallelised. This becomes useful during training, when the sequences x and y are both known and we are interested in making full use of the computational capabilities of a GPU. Fortunately, as we will now show, the SSM is equivalent to a convolution applied on the input sequence at once. Indeed, expanding the first few iterations of Equation (5.5) we obtain:

$$\begin{aligned}
h_1 &= \overline{\mathbf{A}}h_0 + \overline{\mathbf{B}}x_1 = \overline{\mathbf{B}}x_1, \\
y_1 &= \mathbf{C}h_1 \qquad\quad = \mathbf{C}\overline{\mathbf{B}}x_1, \\
\\
h_2 &= \overline{\mathbf{A}}h_1 + \overline{\mathbf{B}}x_2 = \overline{\mathbf{A}\mathbf{B}}x_1 + \overline{\mathbf{B}}x_2, \\
y_2 &= \mathbf{C}h_2 \qquad\quad = \mathbf{C}\overline{\mathbf{A}\mathbf{B}}x_1 + \mathbf{C}\overline{\mathbf{B}}x_2, \\
\\
h_3 &= \overline{\mathbf{A}}h_2 + \overline{\mathbf{B}}x_3 = \overline{\mathbf{A}}^2\overline{\mathbf{B}}x_1 + \overline{\mathbf{A}\mathbf{B}}x_2 + \overline{\mathbf{B}}x_3, \\
y_3 &= \mathbf{C}h_3 \qquad\quad = \mathbf{C}\overline{\mathbf{A}}^2\overline{\mathbf{B}}x_1 + \mathbf{C}\overline{\mathbf{A}\mathbf{B}}x_2 + \mathbf{C}\overline{\mathbf{B}}x_3,
\end{aligned} \tag{5.6}$$

and so on. Let

$$\overline{\mathbf{K}} = \left(\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{l-1}\overline{\mathbf{B}}\right) \in \mathbb{R}^{l \times n \times d}. \tag{5.7}$$

Inductively, for $1 \leq t \leq l$ we get:

$$\begin{aligned} y_t &= \sum_{i=1}^{t} \mathbf{C}\overline{\mathbf{A}}^{t-i}\overline{\mathbf{B}}\, x_i \\ &= \sum_{i=1}^{t} \overline{\mathbf{K}}_{t-i}\, x_i, \\ &= \left(\overline{\mathbf{K}} * x\right)_t. \end{aligned} \tag{5.8}$$

In other words, the full output y of the system can be calculated as a convolution

$$y = \overline{\mathbf{K}} * x \tag{5.9}$$

of the kernel $\overline{\mathbf{K}}$ with the input x. Importantly, $\overline{\mathbf{K}}$ can be calculated once before each parameter update, while each subsequent convolution can be computed in $\mathcal{O}\left(d^2 \cdot l \log l\right)$ operations using the Fast Fourier Transform (FFT), assuming that n is $\mathcal{O}(d)$. Moreover, convolutions can be trivially parallelised. Figure 5.2 illustrates the SSM model as a convolution.

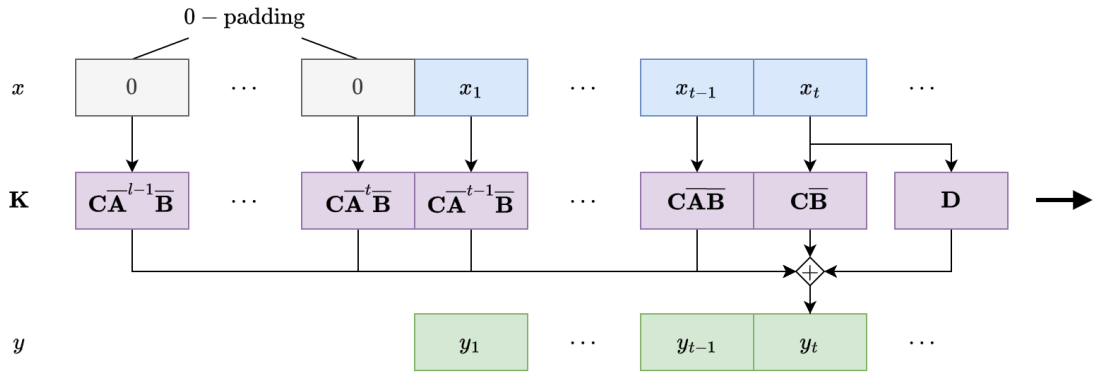


Figure 5.2: The SSM model as a convolution.

5.2 Early adoption of SSMs in deep learning (LSSL)

Despite the rich theoretical background behind SSMs, deep neural networks consisting of simple SSM blocks were shown to struggle with long-range dependency tasks [32]. Gu et al. [32] introduce the Linear State-Space Layer (LSSL), which significantly improves on the plain SSM's performance by imposing restrictions on the structure of the parameters $\mathbf{A}, \mathbf{B}$. In particular, instead of using randomly initialised matrices $\mathbf{A}, \mathbf{B}$ which are optimised during training, Gu et al. apply the High-Order Polynomial Projection Operator (HiPPO) theory of continuous-time memorisation [33], which specifies a class of parameters that allow the SSM to effectively represent the history of the input sequence $(x_1, \ldots, x_t)$ using the hidden state h_t at time t.

The LSSL is formulated for an one-dimensional input sequence $x \in \mathbb{R}^{l\times 1}$. To construct an LSSL with input $x \in \mathbb{R}^{l\times d}$ for $d > 1$, Gu et al. [32] employ d distinct LSSLs processing each input dimension independently. A linear layer is then used to combine the outputs of the d LSSLs into a single output $y \in \mathbb{R}^{l\times n}$.

5.2.1 HiPPO theory

The *HiPPO* theory is a theoretical framework that provides a systematic way to design SSMs that can effectively capture complex relationships in sequential data. It is based on the idea of approximating the history of a continuous function by polynomials and using their coefficients as a finite-dimensional representation. The HiPPO framework is crucial for capturing complex, long-range dependencies in State-Space Models: without it, a recurrent SSM risks information loss over time similar to traditional RNNs, as it must compress the entire input history into a single hidden state at each time step. In contrast, Transformers avoid this by attending to all tokens simultaneously.

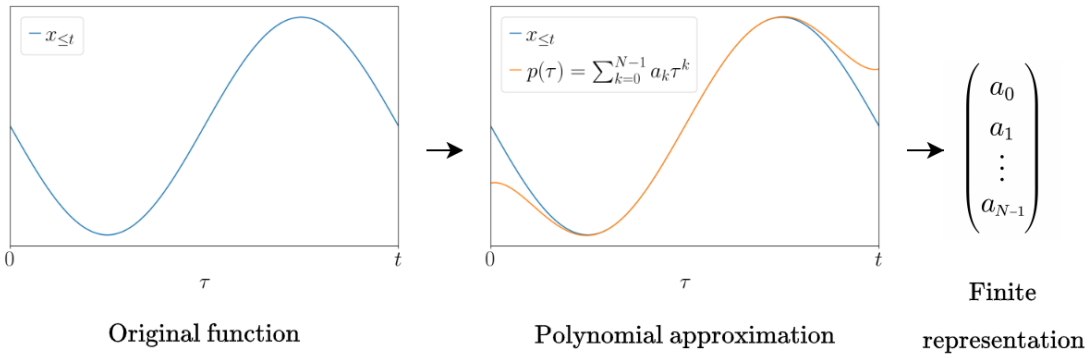


Figure 5.3: The idea behind the HiPPO theory is to approximate the history of a continuous function using polynomials. The coefficients of the polynomial are used as a finite-dimensional representation of the history. The HiPPO theory provides a framework for modelling the evolution of these coefficients over time in the form of a continuous SSM.

Formally, the HiPPO theory is concerned with the problem of effectively representing the cumulative history

$$x_{\leq t} = x(\tau)\big|_{\tau\leq t}$$

of a continuous function $x : [0, \infty) \to \mathbb{R}$ using finite-dimensional representations. In order to explain how this is achieved, first we need to set up some notation and terminology.

For each $t \in (0, \infty)$ we equip the truncated time domain $[0, t]$ with a measure μ_t and define an inner-product $\langle\cdot,\cdot\rangle_{\mu_t}$ with corresponding norm $\|\cdot\|_{\mu_t}$ as follows:

$$\begin{aligned} \langle x, y\rangle_{\mu_t} &= \int_0^t x(\tau)y(\tau)\, d\mu_t(\tau), \\ \|x\|_{\mu_t} &= \sqrt{\langle x, x\rangle_{\mu_t}} = \left(\int_0^t |x(\tau)|^2\, d\mu_t(\tau)\right)^{\frac{1}{2}}. \end{aligned} \tag{5.10}$$

We can now evaluate the effectiveness of a continuous approximation z_t for our history $x_{\leq t}$ at time t as follows:

$$\|x_{\leq t} - z_t\|_{\mu_t} = \left(\int_0^t |x(\tau) - z_t(\tau)|^2 \, d\mu_t(\tau) \right)^{\frac{1}{2}}. \tag{5.11}$$

The following key questions now arise:

- What is a good choice of μ_t?
- What is a good choice of approximation z_t for a given measure μ_t that can easily be represented in a finite-dimensional space?

Gu et al. [33] propose a generic methodology to solve the history approximation problem, which can be described as follows. For $t \in (0, \infty)$, let $\mathcal{C}[0,t]$ be the space of continuous functions over $[0,t]$ and for $N \geq 1$, let $\mathcal{P}_N[0,t]$ be the N-dimensional space of polynomials of degree less than N, defined over $[0,t]$. HiPPO defines an operator

$$\text{hippo}_t : \mathcal{C}[0,t] \to \mathbb{R}^N, \quad x_{\leq t} \mapsto \text{coef}_t(\text{proj}_t(x_{\leq t})),$$

composed of a projection

$$\text{proj}_t : \mathcal{C}[0,t] \to \mathcal{P}_N[0,t]$$

followed by a coefficient extractor

$$\text{coef}_t : \mathcal{P}_N[0,t] \to \mathbb{R}^N$$

such that:

- $p_t = \text{proj}_t(x_{\leq t})$ is the polynomial $p_t \in \mathcal{P}_N[0,t]$ that minimises the error $\|x_{\leq t} - p_t\|_{\mu_t}$.
- $c_t = \text{coef}_t(p_t)$ is the vector of coefficients corresponding to p_t written in terms of an orthogonal basis of $\mathcal{P}_N[0,t]$ with respect to the inner product $\langle \cdot, \cdot \rangle_{\mu_t}$.

Expanding upon the work of Gu et al. [33], Theorem 1 from Gu et al. [32] shows that the HiPPO operator evolves in a way that naturally defines a continuous SSM using parameters $\mathbf{A}, \mathbf{B}$ of a certain structure. This formulation can then be discretised and used to define an SSM in one dimension. In other words, for any choice of measure μ_t, any desired hidden dimension $N \geq 1$ and for any input signal $x(\tau) \in \mathbb{R}$, if we define the state as $h(t) = \text{hippo}_t(x_{\leq t})$, then there exist $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\mathbf{B} \in \mathbb{R}^{N \times 1}$ such that:

$$\dot{h}(t) = \mathbf{A}h(t) + \mathbf{B}x(t). \tag{5.12}$$

For instance, using a uniform measure

$$d\mu_t(\tau) = \frac{1}{t} \mathbb{I}_{[0,t]}(\tau) \, d\tau,$$

and choosing the scaled Legendre polynomials as an orthogonal basis for $\mathcal{P}_N[0,t]$, defined as

$$P_n(\tau) = \sqrt{2n+1} \, Q_n \left(\frac{2\tau}{t} - 1 \right), \quad \tau \in [0,t],$$

for $n = 0, \ldots, N-1$, where

$$Q_n(\tau) = \frac{1}{2^n n!} \frac{d^n}{d\tau^n} \left(\tau^2 - 1 \right)^n, \quad \tau \in [-1,1],$$

gives rise to the following matrix structures for $\mathbf{A}, \mathbf{B}$ (up to a scaling factor of t^{-1}):

$$\mathbf{A}^{(H)}_{m,n} = -\begin{cases} \sqrt{(2m+1)(2n+1)} & \text{if } m > n, \\ m+1 & \text{if } m = n, \\ 0 & \text{if } m < n, \end{cases} \qquad \mathbf{B}^{(H)}_n = \sqrt{2n+1}, \tag{5.13}$$

for $m, n = 0, \ldots, N-1$. For example, for $N = 3$ we get:

$$\mathbf{A}^{(H)} = \begin{pmatrix} -1 & 0 & 0 \\ -\sqrt{3} & -2 & 0 \\ -\sqrt{5} & -\sqrt{15} & -3 \end{pmatrix}, \qquad \mathbf{B}^{(H)} = \begin{pmatrix} 1 \\ \sqrt{3} \\ \sqrt{5} \end{pmatrix}.$$

Gu et al. [32] find that initialising $\mathbf{A}$ with $\mathbf{A}^{(H)}$ improves the performance of the SSM on the sequential MNIST benchmark from 60% to 98% compared to a randomly initialised $\mathbf{A}$.

5.3 Structured and diagonal SSMs (S4, DSS)

The LSSL introduces a new class of theoretically performant SSMs by initialising their parameters based on the HiPPO theory. However, the convolutional view of the SSM delegates a lot of computational complexity into the calculation of $\overline{\mathbf{K}}$, which is not trivial and can be quite slow for general $\overline{\mathbf{A}}$. Indeed, when calculated naively, $\overline{\mathbf{K}}$ requires $\mathcal{O}\left(ld^3\right)$ operations due to the repeated matrix multiplications by $\overline{\mathbf{A}}$.

A typical way to calculate powers of a matrix efficiently is to diagonalise it whenever possible. Indeed, suppose that for some (possibly complex) diagonal $\mathbf{\Lambda}$ and some unitary matrix V we have:

$$\mathbf{\Lambda} = V^* \overline{\mathbf{A}} V.$$

Then we can compute the k-th power of $\overline{\mathbf{A}}$ as follows:

$$\overline{\mathbf{A}}^k = \left(V \mathbf{\Lambda} V^*\right)^k = V \mathbf{\Lambda}^k V^* = V \operatorname{diag}\left(\lambda_1^k, \ldots, \lambda_{d_M}^k\right) V^*.$$

Focusing on the case of an one-dimensional SSM, where $\mathbf{C} \in \mathbb{R}^{1 \times d_M}$ and $\overline{\mathbf{B}} \in \mathbb{R}^{d_M \times 1}$, the calculation of $\overline{\mathbf{K}}$ can be done efficiently in $\mathcal{O}\left((l+d)\log^2(l+d)\right)$ operations (when d_M is $\mathcal{O}(d)$) using a Vandermonde product [34]:

$$\overline{\mathbf{K}} = \begin{pmatrix} \mathbf{C}_0 \overline{\mathbf{B}}_0 & \cdots & \mathbf{C}_{d_M-1} \overline{\mathbf{B}}_{d_M-1} \end{pmatrix} \begin{pmatrix} 1 & \lambda_0 & \lambda_0^2 & \cdots & \lambda_0^{l-1} \\ 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^{l-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_{d_M-1} & \lambda_{d_M-1}^2 & \cdots & \lambda_{d_M-1}^{l-1} \end{pmatrix}.$$

Applying d independent SSM blocks and mixing them with a linear layer would then amount to a total complexity of

$$\mathcal{O}\left(d(l+d)\log^2(l+d) + ld^2\right) = \mathcal{O}\left(d(l+d)\log^2(l+d)\right).$$

Theoretically this method seems ideal for the HiPPO matrix $\mathbf{A}^{(H)}$ as defined in (5.13), which is diagonalisable, since it is lower triangular with distinct diagonal entries (hence distinct eigenvalues). However, Gu, Goel, and Ré [35] show that the diagonalisation of $\mathbf{A}^{(H)}$ is not numerically stable and cannot be used in practice, requiring V with exponentially large entries with respect to the hidden dimension d_M. Instead, they propose a new parametrisation of $\mathbf{A}^{(H)}$ using its linear-algebraic properties, along with an efficient and numerically stable algorithm for the calculation of $\overline{\mathbf{K}}$, thus introducing the class of *Structured State-Space Sequence (S4)* models.

S4. The innovation of Gu, Goel, and Ré [35] lies in the following key ideas:

- It is always possible to split a HiPPO matrix $\mathbf{A}^{(H)}$ into a sum of a normal matrix $\mathbf{A}^{(N)}$ and a low-rank matrix factorisation $\mathbf{A}^{(LR)}$:

  $$\mathbf{A}^{(H)} = \mathbf{A}^{(N)} + \mathbf{A}^{(LR)},$$

  where

  $$\mathbf{A}^{(N)} = V\mathbf{\Lambda}^{(H)}V^*$$

  for unitary $V \in \mathbb{C}^{d_M \times d_M}$ and diagonal $\mathbf{\Lambda}^{(H)} \in \mathbb{C}^{d_M \times d_M}$ and

  $$\mathbf{A}^{(LR)} = -PQ^\top$$

  for $P, Q \in \mathbb{R}^{d_M \times r}$ with $r \ll d_M$.
- Rather than computing powers of $\mathbf{A}^{(H)}$, it is possible define the kernel via a generating function whose evaluation reduces to a matrix inverse.
- The low-rank correction can be handled using the Woodbury identity, which calculates inverses of the form $\left(\mathbf{A} + PQ^\top\right)^{-1}$ in terms of $\mathbf{A}^{-1}$, further reducing the computation to the diagonal case.
- The resulting expression can be evaluated efficiently and stably in the frequency domain, while $\overline{\mathbf{K}}$ can be recovered using an inverse FFT.

Explaining the details of the S4 algorithm is beyond the scope of this thesis and we refer the reader to Gu, Goel, and Ré [35] for a detailed description. Taking into account the kernel computation and application per input dimension and the final mixing layer over all d input dimensions, the resulting frequency-domain expression for the kernel requires $\mathcal{O}\left(d^2 \cdot l \log l + d(l+d)\log^2(l+d)\right)$ operations to evaluate efficiently and stably using FFTs and structured matrix multiplications. The convolution of the kernel with the input then requires an additional $\mathcal{O}\left(d^2 \cdot l \log l\right)$ operations. Thus, the total complexity of the S4 layer becomes $\mathcal{O}\left(d^2 \cdot l \log l + d(l+d)\log^2(l+d)\right)$.

DSS. A natural question that arises from the S4 framework is whether the low-rank correction is necessary at all. Gupta, Gu, and Berant [36] show that in fact it is not and that similar performance can be achieved by using diagonal matrices of the form $\exp\left(\mathbf{\Lambda}^{(H)}\right)$, where the real part of the elements of $\mathbf{\Lambda}^{(H)}$ is restricted to be negative for numerical stability. This greatly simplifies the S4 architecture and decreases its computational requirements, leading to the class of *Diagonal State-Space* (DSS) models [36]. Gu et al. [37] further simplify and improve upon the DSS implementation using Vandermonde products for the kernel calculation and provide a mathematical analysis of possible DSS parametrisations and initialisations.

Since DSS sets $\mathbf{A}$ to be diagonal from the start, it eliminates the need for low-rank corrections or structured matrix inversions. The kernel $\overline{\mathbf{K}}$ can then be computed over all input dimensions d in $\mathcal{O}\left(d(l+d)\log^2(l+d)\right)$ operations using the Vandermonde matrix multiplication [34]. This improves upon the S4 training complexity by removing the $\mathcal{O}\left(d^2 \cdot l \log l\right)$ term associated with the inverse FFT of a non-diagonal $\mathbf{A}$. The final convolution with the input $x \in \mathbb{R}^{l \times d}$ still requires $\mathcal{O}\left(d^2 \cdot l \log l\right)$ operations due to the FFT convolution across all input-output channel pairs, giving once again a total complexity of $\mathcal{O}\left(d^2 \cdot l \log l + d(l+d)\log^2(l+d)\right)$. Note that this matches the S4 in asymptotic terms, but with reduced constants and smaller implementation overhead.

5.4 Multi-dimensional SSMs (S5) and the parallel scan algorithm

Similar to the LSSL, S4 and DSS handle multi-dimensional input $x \in \mathbb{R}^{l \times d}$ by using d independent S4 blocks, each processing a single input dimension. The dimensions are then mixed after each layer using a position-wise MLP. This setup is known as single-input single-output (SISO). Smith, Warrington, and Linderman [38] instead propose a multi-input, multi-output (MIMO) SSM architecture based on the S4/DSS framework, introducing the *S5* layer.

The S5 layer implements the full SSM formulation (5.5) with a single set of parameters $\mathbf{A}, \mathbf{B}, \mathbf{C}$ for all input dimensions. The change from SISO to MIMO reduces the total number of parameters required from the full d-dimensional SSM calculation per layer, which allows for dropping the complicated convolutional calculation of S4 and replacing it with a *parallel scan* algorithm applied on the recurrent calculation. Assuming that n, d_M are $\mathcal{O}(d)$ and that $\mathbf{A}$ is diagonal, the scan-based implementation allows for computing the recurrence in $\mathcal{O}(d \cdot l \log l)$ operations for the entire sequence [39].

5.4.1 Parallel scan

Parallel scan refers to a class of algorithms that can be used to parallelise the computation of an associative binary operator applied iteratively over a sequence. Formally, given a sequence $x = (x_1, \ldots, x_l)$ and an associative binary operator $\oplus$, define the *(inclusive) scan* of x with respect to $\oplus$ as the sequence

$$\text{scan}(x, \oplus) = (x_1,\ x_1 \oplus x_2,\ x_1 \oplus x_2 \oplus x_3,\ \ldots,\ x_1 \oplus x_2 \oplus \cdots \oplus x_l).$$

For example, if $\oplus = +$ is the addition operator and

$$x = (1, 2, 3, 4, 5, 6, 7, 8),$$

then the scan of x with respect to addition is given by:

$$\text{scan}(x, +) = (1, 3, 6, 10, 15, 21, 28, 36).$$

Associativity of the operator is a key requirement for the above calculation to be executable in parallel, as it allows for the reordering of the operations without changing the final result. Recall that a binary operator $\oplus$ is associative if for all a, b, c we have:

$$(a \oplus b) \oplus c = a \oplus (b \oplus c).$$

It turns out that the SSM recurrence (5.5) can be formulated as a scan problem. To see this, define the binary operator $\oplus$ as

$$\left(\overline{\mathbf{A}}^i,\, x\right) \oplus \left(\overline{\mathbf{A}}^j,\, y\right) = \left(\overline{\mathbf{A}}^{i+j},\, \overline{\mathbf{A}}^j x + y\right) \tag{5.14}$$

for $x, y \in \mathbb{R}^{d_M}$. This operator is associative:

$$\begin{aligned}
\left[\left(\overline{\mathbf{A}}^i,\, x\right) \oplus \left(\overline{\mathbf{A}}^j,\, y\right)\right] \oplus \left(\overline{\mathbf{A}}^k,\, z\right) &= \left(\overline{\mathbf{A}}^{i+j},\, \overline{\mathbf{A}}^j x + y\right) \oplus \left(\overline{\mathbf{A}}^k,\, z\right) \\
&= \left(\overline{\mathbf{A}}^{i+j+k},\, \overline{\mathbf{A}}^k \left(\overline{\mathbf{A}}^j x + y\right) + z\right) \\
&= \left(\overline{\mathbf{A}}^{i+j+k},\, \overline{\mathbf{A}}^{j+k} x + \overline{\mathbf{A}}^k y + z\right) \\
&= \left(\overline{\mathbf{A}}^i,\, x\right) \oplus \left(\overline{\mathbf{A}}^{j+k},\, \overline{\mathbf{A}}^k y + z\right) \\
&= \left(\overline{\mathbf{A}}^i,\, x\right) \oplus \left[\left(\overline{\mathbf{A}}^j,\, y\right) \oplus \left(\overline{\mathbf{A}}^k,\, z\right)\right].
\end{aligned} \tag{5.15}$$

The key in formulating the SSM as a scan problem lies in the following observation:

$$\begin{aligned}
(I,\, 0) \oplus \left(\overline{\mathbf{A}},\, \overline{\mathbf{B}} x_1\right) &= \left(\overline{\mathbf{A}},\, \overline{\mathbf{B}} x_1\right) &&= \left(\overline{\mathbf{A}},\, h_1\right), \\
\left(\overline{\mathbf{A}},\, h_1\right) \oplus \left(\overline{\mathbf{A}},\, \overline{\mathbf{B}} x_2\right) &= \left(\overline{\mathbf{A}}^2,\, \overline{\mathbf{A}} h_1 + \overline{\mathbf{B}} x_2\right) &&= \left(\overline{\mathbf{A}}^2,\, h_2\right), \\
\left(\overline{\mathbf{A}}^2,\, h_2\right) \oplus \left(\overline{\mathbf{A}},\, \overline{\mathbf{B}} x_3\right) &= \left(\overline{\mathbf{A}}^3,\, \overline{\mathbf{A}} h_2 + \overline{\mathbf{B}} x_3\right) &&= \left(\overline{\mathbf{A}}^3,\, h_3\right),
\end{aligned}$$

and so on. In general, for $t = 1, \ldots, l$ we have:

$$\left(\overline{\mathbf{A}}^{t-1},\, h_{t-1}\right) \oplus \left(\overline{\mathbf{A}},\, \overline{\mathbf{B}} x_t\right) = \left(\overline{\mathbf{A}}^t,\, \overline{\mathbf{A}} h_{t-1} + \overline{\mathbf{B}} x_t\right) = \left(\overline{\mathbf{A}}^t,\, h_t\right). \tag{5.16}$$

Therefore, the hidden states $h_1, \ldots, h_l$ can be retrieved from the output of the scan over the sequence

$$X = \left[\left(\overline{\mathbf{A}},\, \overline{\mathbf{B}} x_1\right),\, \left(\overline{\mathbf{A}},\, \overline{\mathbf{B}} x_2\right),\, \ldots\, \left(\overline{\mathbf{A}},\, \overline{\mathbf{B}} x_l\right)\right] \tag{5.17}$$

with respect to our operator $\oplus$:

$$\operatorname{scan}(X, \oplus) = \left[\left(\overline{\mathbf{A}},\, h_1\right),\, \left(\overline{\mathbf{A}}^2,\, h_2\right),\, \ldots\, \left(\overline{\mathbf{A}}^l,\, h_l\right)\right]. \tag{5.18}$$

Note that precomputing the sequence X requires $\mathcal{O}\left(ld^2\right)$ operations.

We now explain how this operation can be parallelised. Assume for simplicity that the input sequence x has length equal to a power of 2, i.e., $l = 2^k$ for some k. The parallel scan constructs a binary tree out of the input sequence x, where each leaf stores the elements of the sequence and each internal node stores the result of applying the operator $\oplus$ to its children. The algorithm then iteratively updates x in place by applying two sweeps of this tree:

1. **Up-sweep**: The algorithm moves from the leaves towards the root, calculating in parallel for each inner node at each level the sum of its children and updating x accordingly. In particular, at level $r = 1, \ldots, k$ we update

   $$x\left(2^r t\right) \leftarrow x\left(2^r t\right) \oplus x\left(2^r (t-1)\right),$$

   for $t = 1, \ldots, 2^{k-r}$. Note that this calculation already fills in $x\left(2^r\right)$ with the correct scan output at level r.

2. **Down-sweep**: The algorithm moves from the root towards the leaves, calculating in parallel any leftover sums at each level $r = k-1, \ldots, 1$ as follows:

$$x\left(2^{r-1}(2t+1)\right) \leftarrow x\left(2^{r-1}(2t+1)\right) \oplus x\left(2^{r}t\right)$$

for $t = 1, \ldots, 2^{k-r} - 1$.

Figure 5.4 shows an example of the parallel scan algorithm when $x = (1, 2, 3, 4, 5, 6, 7, 8)$ and $\oplus$ is the addition operator.

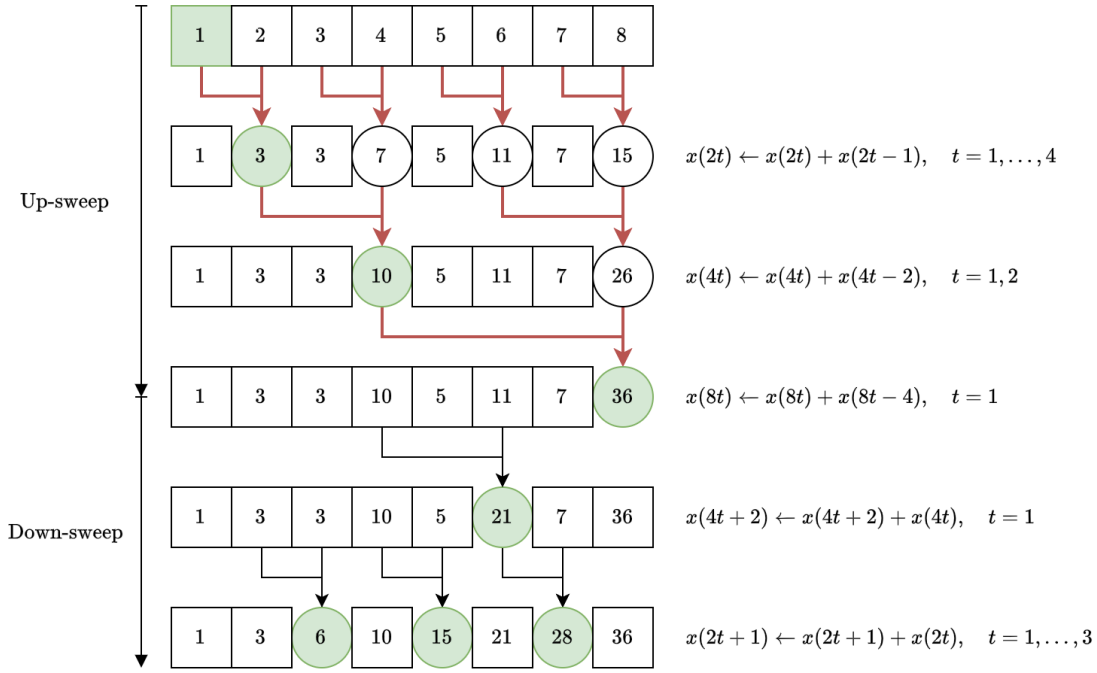


Figure 5.4: Example of the parallel scan algorithm when $x = (1, 2, 3, 4, 5, 6, 7, 8)$ and $\oplus$ is the addition operator. The time at which a position is filled with its correct scan output is highlighted in green. The underlying binary tree structure is indicated by the red lines.

5.5 Selective State-Space Models (Mamba)

In the setup described so far, all the models mentioned have been *linear-time invariant (LTI)*, meaning that the parameters $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \Delta$ are fixed for each $t = 1, \ldots, l$. Gu and Dao [30] identify a key limitation of this architecture, in that it is unable to perform context-aware reasoning. To illustrate this, the authors present three tasks of increasing complexity.

- **Copying task:** The copying task involves a sequence of tokens that the model must reproduce later in the output after a fixed delay. The spacing between input and output is constant, making the task solvable by simple time-invariant models like linear RNNs, convolutions, or LTI SSMs.

- **Selective copying task:** In contrast, the selective copying task presents a long sequence containing scattered "relevant" tokens marked by cues. The model must copy only those marked tokens to the output. Since the spacing between relevant inputs is random, the model must be time-aware and selectively remember important content.
- **Induction heads task:** Finally, the induction heads task requires associative recall: the model is shown input-output pairs and must later infer the output of a new input by matching it with earlier context. This tests the model's ability to recognise patterns and generalise from them.

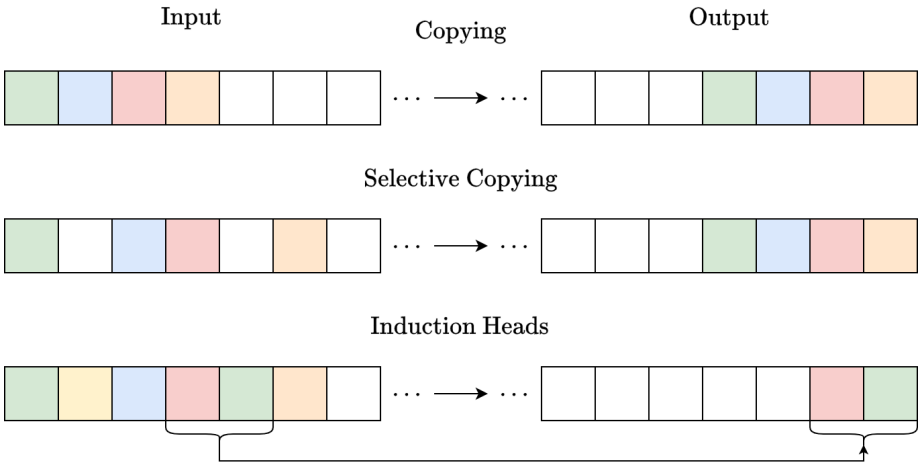


Figure 5.5: Illustration of the copying, selective copying and the induction heads tasks, used to evaluate context-aware reasoning in sequence models.

In particular, Gu and Dao identify the selective copying task and the induction heads task as examples in which the LTI SSM is insufficient and a model which is able to selectively focus on inputs is better suited. In the Transformer architecture, this is achieved through the use of the attention mechanism.

Gu and Dao [30] relax the LTI requirement and propose the Mamba SSM, which once again initialises $\mathbf{A}$ based on the diagonal HiPPO operator and uses a selection mechanism which makes the parameters $\mathbf{B}, \mathbf{C}, \Delta$ input dependent as follows:

$$\begin{aligned} \mathbf{B}_t &= W_B x_t, \\ \mathbf{C}_t &= W_C x_t, \\ \Delta_t &= \text{Softplus}\left(Q_\Delta + \text{Broadcast}_d\left(W_\Delta x_t\right)\right), \end{aligned} \tag{5.19}$$

for $t = 1, \ldots, l$ and trainable parameters $W_B, W_C \in \mathbb{R}^{d_M \times d}$, $Q_\Delta, W_\Delta \in \mathbb{R}^{1 \times d}$, where

$$\text{Broadcast}_d(a) = (a, a, \ldots, a) \in \mathbb{R}^{1 \times d}.$$

Similar to S4, Mamba applies d one-dimensional SSMs per input dimension and mixes the outputs using a linear layer. The full Mamba block is a residual module which operates as follows:

1. The input is split into two branches: the first branch is the one that passes through the SSM, while the second branch bypasses it to form a multiplicative gating path later on.
2. Two linear projections that expand the hidden dimension from d_M to $a \cdot d_M$ by a factor $a > 1$ are applied on each branch.
3. A depth-wise 1D convolution is applied to the first branch, introducing local temporal context into the representation. A non-linear activation is applied to the output of the convolution.
4. The selective SSM module is applied to the output of the convolution, using the input-dependent parameters $\mathbf{B}_t, \mathbf{C}_t, \Delta_t$ defined in (5.19).
5. The SSM output is multiplied element-wise with the second branch, after the latter has passed through its own non-linear activation. This operation acts as a gating mechanism, enhancing or suppressing features in the SSM output based on the input.
6. The gated output is passed through a final linear projection, which reduces the feature dimension back to its original size.

The Mamba block architecture is shown in Figure 5.6.

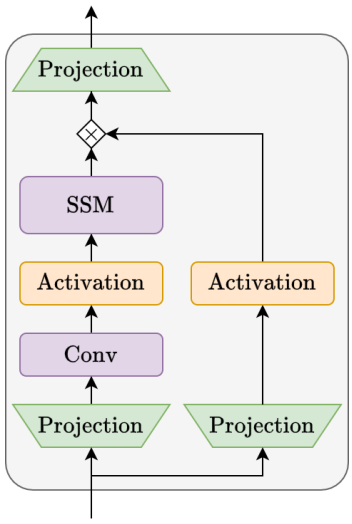


Figure 5.6: The Mamba block architecture.

A consequence of the selection mechanism is that the convolutional calculation of the SSM is no longer possible, as the kernel differs for each input and cannot be calculated in advance. As a result, only the recurrent calculation of the SSM can be performed. In order to recover the parallelisation capabilities and speed improvements of the base SSM, Gu and Dao use the parallel scan algorithm[1], similar to Smith, Warrington, and Linderman [38]. Moreover, the authors propose a hardware-aware algorithm that efficiently utilises the HBM-SRAM memory hierarchy, by fusing the discretisation

[1]Associativity of the defined operator is guaranteed even though it is not the same $\mathbf{A}_t$ in each step t of the input sequence, since $\mathbf{A}_t$ is assumed to be diagonal.

step, the scan and the multiplication with $\mathbf{C}$ into one kernel, analogous to FlashAttention [31] (cf. section 4.3). For more details on the implementation of these algorithms, we refer the reader to Gu and Dao [30].

5.6 SSMs summary

So far we examined the SSM from a theoretical point of view and we introduced the main components used by SSM architectures in deep learning by examining its evolution from the original LSSL model to the Mamba model. We now summarise the key features and innovations of the various SSM architectures.

- **LSSL** [32]: Introduces HiPPO-initialised SSMs for capturing long-range dependencies. Supports both recurrent and convolutional computation, but the kernel calculation is computationally expensive.
- **S4** [35]: Builds on LSSL by proposing a structured decomposition of the HiPPO matrix into a normal plus low-rank (NPLR) form. Enables efficient and stable kernel computation in the frequency domain using FFT.
- **DSS** [36]: Simplifies S4 by dropping the low-rank correction and using diagonal matrices directly. Improves computational efficiency and preserves expressivity through careful parametrisation and initialisation.
- **S5** [38]: Replaces SISO with MIMO architecture using shared parameters across dimensions. Avoids convolutions entirely by expressing the SSM recurrence as a parallel scan, enabling efficient training and inference.
- **Mamba** [30]: Retains the SISO setup and introduces selective (input-dependent) SSMs with dynamic parameters $\mathbf{B}, \mathbf{C}, \Delta$. Drops the convolutional view and extends the parallel scan approach using a hardware-efficient fused kernel implementation.

Table 5.1 summarises the approaches used by the various SSM architectures discussed in this section, including their computational complexity for inference and parallelised training. The table shows that all models are recurrent, but only S5 and Mamba use a parallel scan algorithm for efficient training and inference. The S4 and DSS models are convolutional, but they have higher computational complexity due to the kernel calculation. Mamba is the most efficient in terms of both training and inference, as it uses a hardware-efficient fused kernel implementation. Importantly, for all models the computational complexity is linear in the input sequence length l, which is a key advantage over the Transformer architecture for long input sequences, since the latter has a quadratic complexity in l (cf. section 4.3).

Comparison with RNNs. SSMs are closely related to recurrent neural networks (RNNs), in that they both produce a sequence of outputs y_t from a sequence of inputs x_t using a hidden state h_t in an iterative manner. In fact, Gu et al. [32] show that the sigmoid gating mechanism of RNNs can actually be viewed as a special case of discretisation of a continuous SSM. A key difference between SSMs and RNNs is that the latter typically use a non-linear activation function σ in the recurrence

[2]This complexity refers to the naive kernel calculation, as the subsequent more efficient implementations are not directly applicable to the LSSL due to numeric instability [35].

Model	Recurrent	Convolution	Parallel Scan	Inference	Parallel Training
LSSL	Yes	Yes	No	$\mathcal{O}\left(ld^2\right)$	$\mathcal{O}\left(ld^3\right)^2$
S4	Yes	Yes	No	$\mathcal{O}\left(ld^2\right)$	$\mathcal{O}\left(d^2 \cdot l \log l + d(l+d) \log^2(l+d)\right)$
DSS	Yes	Yes	No	$\mathcal{O}\left(ld^2\right)$	$\mathcal{O}\left(d^2 \cdot l \log l + d(l+d) \log^2(l+d)\right)$
S5	Yes	No	Yes	$\mathcal{O}\left(ld^2\right)$	$\mathcal{O}\left(d \cdot l \log l + ld^2\right)$
Mamba	Yes	No	Yes	$\mathcal{O}\left(ld^2\right)$	$\mathcal{O}\left(d \cdot l \log l + ld^2\right)$

Table 5.1: Comparison of various SSM-based architectures in terms of structure and computational complexity for input sequence $x \in \mathbb{R}^{l \times d}$ of length l and dimension d. Hidden and output dimensions are assumed to be $\mathcal{O}(d)$.

relation, or use various gating mechanisms, while SSMs use a fully linear recurrence. The latter allows for parallelised implementation of the SSM recurrence during training, through convolutional or parallel scan calculations, which is typically not possible for RNNs. In addition, SSMs make use of advanced theoretical foundations that allow for careful parameter initialisation, mitigating issues such as information loss or the vanishing gradient problem that are often encountered in RNNs. Parallelisation and robustness are key advantages of the SSM architecture over RNNs, which is particularly important for long input sequences.

Chapter 6

Deep Learning and Audio Processing

In this chapter we introduce the final piece of background relevant to this thesis, namely deep learning techniques for audio processing. We focus on the use of spectrograms and mel-spectrograms as input representations and review existing foundation models for audio data, including both Transformer-based and SSM-based architectures.

6.1 Introduction to audio processing

Audio processing has been traditionally done using classical signal processing techniques, often relying on domain knowledge and heuristics for feature extraction. More recently, the advancements in deep learning have enabled large-scale end-to-end learning of audio representations, allowing for the automatic extraction of features from raw audio signals. This has led to significant improvements in various audio processing tasks, such as speech recognition, music tagging, sound classification and segmentation.

Many architectures have been developed for learning directly from raw audio signals. Early examples of such work include Dieleman and Schrauwen [40], who use waveforms as input to a CNN for music tagging, Sainath et al. [41], who combine LSTMs with CNNs for the task of speech recognition, Dai et al. [42] who employ deep residual CNNs for environmental sound classification and Oord et al. [43], who use dilated causal convolutions for waveform generation. More recently, self-supervised learning networks have been applied on raw audio signals for the purpose of generic feature extraction. Such approaches include Wav2Vec [44, 45] a series of models that employ contrastive or masked prediction tasks to learn high-level audio representations from unlabelled data and HuBERT [46], a BERT-inspired architecture that learns audio embeddings by predicting pseudo-labels which are generated by an unsupervised clustering algorithm for masked portions of the input.

Although powerful, models that operate on raw waveforms have recently fallen out of favour compared to approaches that process spectrograms generated from the audio signals. This is largely due to the fact that spectrograms provide a meaningful and compact representation of audio by

converting it into a time-frequency representation that is more robust in presence of small variations in the raw input. By discarding phase information and summarising frequency content over time, spectrograms reduce the burden on neural networks to learn invariances such as local phase shifts or harmonic structure from scratch. Moreover, spectrograms are naturally compatible with 2D CNNs, allowing models originally developed for computer vision to be effectively repurposed for audio tasks. In fact, in deep learning spectrograms are often treated exactly as images and the same architectures and pre-processing techniques that are used for image processing are often applied to audio processing tasks.

6.2 Spectrograms and mel-spectrograms

Spectrograms are a common representation of audio signals, obtained by applying the short-time Fourier transform (STFT) to the audio waveform. The STFT divides the audio signal into overlapping segments and computes the Fourier transform for each segment, resulting in a time-frequency representation of the signal. The spectrogram is typically visualised as a 2D heatmap, where the x-axis represents time, the y-axis represents frequency and the intensity of each pixel corresponds to the intensity of the frequency at that time.

Formally, let $x(t)$ for $t = 0, \dots, T-1$ be a discrete audio signal of length T sampled at frequency F_s and let $w(n)$ for $n = 0, \dots, N-1$ be a window function of length $N \ll T$ applied to consecutive overlapping segments of the signal. Let H be the hop size, i.e., the number of samples between the starts of two consecutive windows. The short-time Fourier transform (STFT) of the signal is defined for window index $t \in \left\{0, \dots, \left\lceil \frac{T}{H} \right\rceil\right\}$ and frequency bin $f \in \{0, \dots, N-1\}$ as:

$$\mathrm{STFT}(x)(t, f) = \sum_{n=0}^{N-1} x(tH + n) \cdot w(n) \cdot e^{-2\pi i f n / N}. \tag{6.1}$$

Each frequency bin f corresponds to a physical frequency in the range:

$$\left[f \cdot \frac{F_s}{N},\, (f+1) \cdot \frac{F_s}{N} \right] \text{ Hz}.$$

The spectrogram of the signal is then obtained by taking the squared magnitude of the STFT:

$$\mathrm{Spectrogram}(x)(t, f) = \left|\mathrm{STFT}(x)(t, f)\right|^2. \tag{6.2}$$

For real-valued input signals, the STFT satisfies the conjugate symmetry property:

$$\mathrm{STFT}(x)(t, f) = \mathrm{STFT}(x)(t, N-f)^*,$$

which implies

$$\mathrm{Spectrogram}(x)(t, f) = \mathrm{Spectrogram}(x)(t, N-f).$$

Therefore, only the first $\left\lfloor \frac{N}{2} \right\rfloor + 1$ frequency bins (i.e., $f \in \{0, \dots, \left\lfloor \frac{N}{2} \right\rfloor\}$), corresponding to physical frequencies up to the Nyquist frequency $\frac{F_s}{2}$ Hz, contain non-redundant information and the spectrogram representation is restricted to these bins.

Often the range of magnitudes within a spectrogram can be spread over several orders of magnitude, with sparse high-power components dominating the representation, while all other patterns (such as harmonics) are compressed into a narrow range of magnitudes close to zero. To make finer details more discernible, a logarithmic scaling is applied to the magnitudes of the spectrogram, converting them to decibels (dB):

$$\text{Spectrogram}_{dB}(x)(t,f) = 10\log_{10}\big(\text{Spectrogram}(x)(t,f)\big). \tag{6.3}$$

For example, Figure 6.1a shows the waveform of a raw audio signal which is sampled at $F_s = 48,000\ Hz$ over 2.75 seconds, for a total of $T = 48,000 \cdot 2.75 = 132,000$ samples. Figure 6.1b shows the spectrogram of the same signal obtained with a window of size $N = 4,096$ and a hop of size $H = \frac{N}{2} = 2,048$, resulting in a total of $\lceil \frac{T}{H} \rceil = 65$ time steps and $\frac{N}{2} + 1 = 2,049$ frequency bins. The low visibility of the spectrogram is due to the wide dynamic range of the magnitudes. Figure 6.1c shows the same spectrogram after applying dB scaling, which makes time-frequency energy patterns more visible.

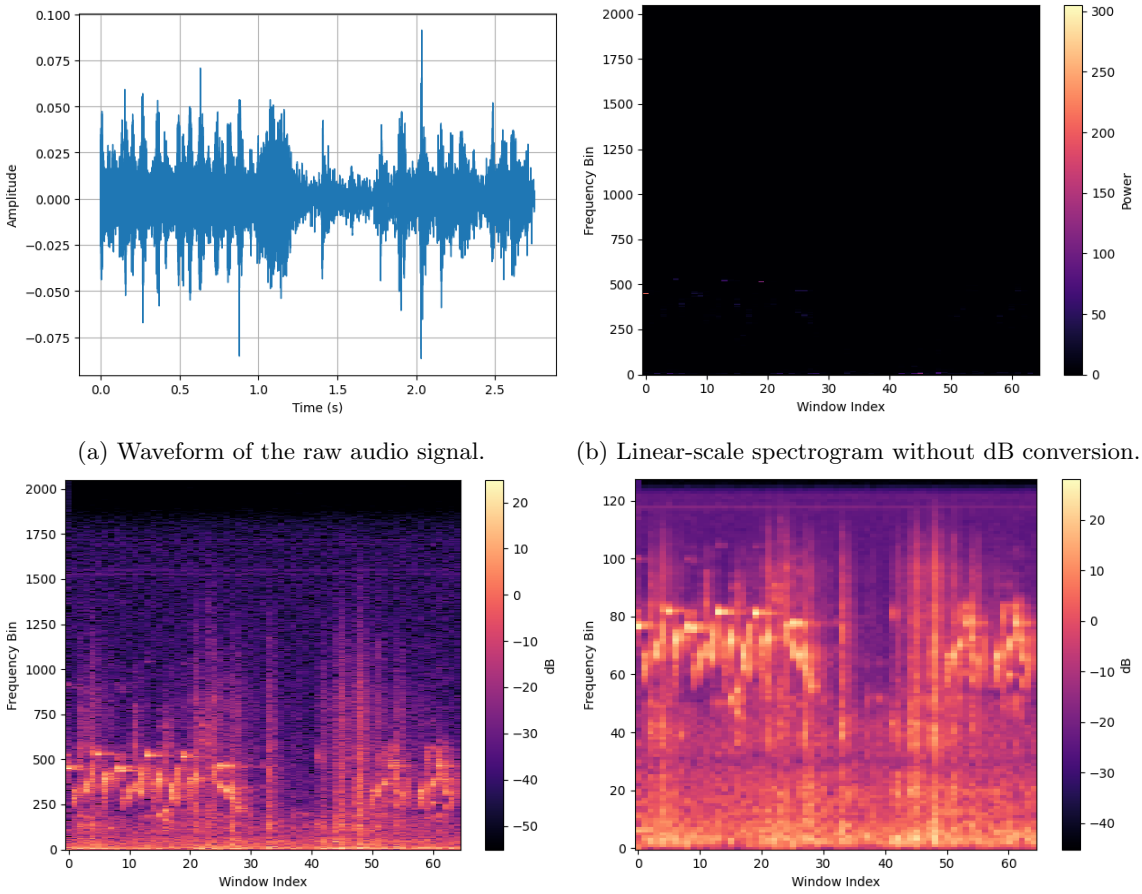
Mel-spectrograms. The mel scale is a perceptual scale of pitches, designed to better align with how humans perceive sound. In particular, the mel scale is based on the observation that humans perceive frequency differences logarithmically rather than linearly, being more sensitive to lower frequencies and less sensitive to higher frequencies. For example, the difference between 500 Hz and $1,000\ Hz$ is perceptually greater than the difference between $5,000Hz$ and $5,500Hz$, even though both are 500 Hz apart. The mel scale reflects this perceptual non-linearity by mapping frequencies to mel values in a way that compresses higher frequencies and expands lower frequencies. A frequency f in Hertz (Hz) can be converted to the mel scale as follows:

$$\text{Mel}(f) = 2595\log_{10}\left(1 + \frac{f}{700}\right).$$

Define a set of M (typically overlapping) triangular filters $v_0(f), \ldots, v_{M-1}(f)$ on the frequency domain, each centred at a frequency f_m such that the points $\{\text{Mel}(f_m) : m = 0, \ldots, M-1\}$ are uniformly spaced in the interval $\big[0, \text{Mel}\big(\frac{F_s}{2}\big)\big]$. The mel-spectrogram is then defined for $m = 0, \ldots, M-1$ as follows:

$$\text{MelSpectrogram}(x)(t,m) = \sum_{f=0}^{N-1} v_m(f) \cdot \text{Spectrogram}(x)(t,f), \tag{6.4}$$

Many modern deep learning architectures for audio processing use mel-spectrograms instead of linear spectrograms as input. This practice has been observed to improve performance on various tasks, as the mel-spectrogram provides high resolution in lower frequencies, where most of the high-energy and perceptually relevant information is located, while compressing higher frequencies, which tend to contain low energy patterns. This is particularly relevant in tasks that align with human perception, such as emotion recognition from speech [47], but it has also been shown to improve performance in other domains, including environmental sound classification [48]. Moreover, converting the frequencies into the mel scale acts as an extra feature extraction/pre-processing step, reducing the dimensionality of the input and making it more robust to small variations. Figure 6.1d shows an 128-bin mel-spectrogram of the waveform compared to its corresponding linear scale spectrogram.



(a) Waveform of the raw audio signal.

(b) Linear-scale spectrogram without dB conversion.

(c) Spectrogram after dB scaling.

(d) Mel-scaled spectrogram.

Figure 6.1: Comparison of different time-frequency representations of the same audio signal. Converting to dB scale improves discernibility of patterns in lower amplitudes and mel scaling enhances the patterns in lower frequencies, where high energy content is mostly located.

6.3 Foundation models for audio processing

In recent years, large-scale pre-trained foundation models have been developed for spectrogram-based audio processing. These models are typically based on architectures that have been successful in the field of computer vision, so the two fields are closely related and should be reviewed together.

Transformer-based models. Motivated by the ViT architecture [23], which demonstrated the effectiveness of Transformers for image classification, several Transformer-based models have been proposed for audio processing. These architectures typically split the input spectrogram into a grid of two-dimensional patches, which are then treated as tokens for the Transformer architecture, in the same manner as images are processed. Gong, Chung, and Glass [11] propose the audio spectrogram Transformer (AST) which uses a ViT-based backbone (DeiT [49]) pre-trained on ImageNet [17] and retrains it on spectrograms for the task of audio classification, achieving state-of-the-art results on various classification benchmarks. Building on the AST architecture, Gong et al. [13] propose the self-supervised audio spectrogram Transformer (SSAST), which is pre-trained using a combination of a masked prediction task with a contrastive learning task for the purpose of robust feature extraction.

SSM-based models. Very recently, researchers have been experimenting with replacing the ViT backbone by Mamba-based architectures for audio processing. These models are typically pre-trained using methods similar to those used for the AST and SSAST models for comparison purposes, as the use of SSMs and their potential as an alternative to the Transformer are still very new and active areas of research. For example, Yadav and Tan [50] propose a Mamba-based foundation model (SSAM) inspired by SSAST. SSAM follows a similar pre-training pipeline as SSAST, with the only differences being that SSAM replaces the ViT backbone with a sequence of Mamba blocks and is pre-trained using only a masked reconstruction task.

A major drawback of pure SSMs compared to attention in the fields of image and spectrogram processing is that the former always treat the input as a sequence with clear temporal ordering, while the latter treats the entire input concurrently. Due to the nature of self-attention, the Transformer learns relationships between all 2D patches in the input image/spectrogram, which creates context-aware embeddings for each token/patch. On the other hand, SSMs are forced to compress the entire history of the input sequence into the hidden state at each time step. While the HiPPO theory provides a principled and mathematically proven way to do this, the issue still remains that the hidden embedding produced by the SSM at each time step is only aware of what the model has seen so far and not of the entire input sequence. Intuitively, this approach is better suited for time-series data, where the future samples are not available at the time of prediction and an accurate representation of the past is crucial for making predictions. In contrast, images and spectrograms exhibit complex relationships between multiple 2D patches, which might not be effectively captured by a strictly one-directional sequential processing. For this reason, researchers have been experimenting with architectures that process the input sequences in more than one direction.

For example, Liu et al. [51] propose the 2D Selective scan module (SS2D), which processes images in four different paths, i.e., starting from top-left and moving to the right, starting from top-left and moving downwards, starting from bottom-right and moving to the left and starting from bottom-right and moving upwards, applying a separate Mamba block on each sequence and merging their

outputs using a linear layer. Lin and Hu [52] repurpose the SS2D module for audio processing and propose the AudioMamba architecture, trained in a supervised manner for audio classification. Other works use simpler bi-directional architectures, which first construct the input sequence in the typical way of starting from top-left and moving towards the right and then apply a forward and a backward SSM pass to it. Examples of such architectures include AuM [53], which uses a bi-directional SSM for audio classification and SPMamba [54], which is trained for speech separation. Particularly relevant to our work is the SSAMBA architecture [55], which trains a bi-directional SSM in a self-supervised manner for embedding extraction, using the same pre-training pipeline as SSAST. We examine the bi-directional Mamba architecture in more detail in chapter 8, in which we describe our full methodology.

Chapter 7

Dataset

In this chapter we describe the GardenFiles23 dataset used in our experiments. We provide a brief overview of the dataset, including its collection method and the metadata included, some of which we use as targets in order to evaluate the performance of the embeddings produced by our models. We also describe the pre-processing steps applied to the audio recordings, including the generation of spectrograms and the extraction of training targets from the metadata features.

7.1 The GardenFiles23 dataset

The *GardenFiles23* dataset [1, 56] is a collection of 3-second stereo audio recordings designed to facilitate research in ecoacoustics. It consists of stereo audio recordings collected from a passive acoustic monitoring (PAM) system installed in a residential back garden in the Netherlands. The system uses a two-microphone array and operates continuously from August 2023 onwards, capturing 3-second audio clips in response to detected acoustic activity. Each recording is stored as a 32-bit WAV file sampled at 48 kHz. In addition to the audio data, the dataset includes metadata such as precise timestamps, the type of trigger that initiated each recording and environmental context.

Data collection. The waveforms in the GardenFiles23 dataset are segmented using a real-time trigger-based recording mechanism, designed to efficiently capture relevant acoustic events while reducing storage of irrelevant background noise. This system continuously computes power spectra in 2048-sample frames using a squared cosine window with a hop size of 1024, generating a dynamic spectrogram representation of the incoming audio stream. A background noise model is maintained and updated with an exponential smoothing filter and subtracted from the incoming signal to create noise-compensated spectra. Based on these, three complementary acoustic triggers are evaluated:

- **Normalised spectral peak ratio:** detects sharp peaks in the spectrogram, often associated with bird calls or distinct anthropogenic sounds.
- **Spectral flatness:** identifies segments with high variability, such as rustling or composite events, by comparing geometric and arithmetic means of spectral energy.

- **Hellinger divergence:** captures statistical changes in local spectrogram blocks, serving as a generalised change detector.

When any of these triggers exceeds a defined threshold, a 3-second stereo recording is stored.

In this work we use an expanded and further processed version of the GardenFiles23 dataset. In particular, our data consists of around 1.3 million recordings, along with their environmental metadata, collected from August 2023 to March 2025 using the same pipeline. Each recording has been trimmed to 2.75 seconds. For the extended dataset we only have access to pre-processed spectrograms, which are generated as described in section 7.2. The original dataset, which includes around 750,000 raw audio files recorded between August 2023 and September 2024, is publicly available on DataverseNL [56].

Environmental metadata. Environmental measurements are provided by a Froggit WH3000 SE weather station installed adjacent to the microphone array in the garden. The station is connected to the WunderGround service and continuously uploads atmospheric data at intervals of a few seconds. Logged parameters included temperature, wind speed, humidity, barometric pressure and other standard weather metrics. For each audio recording, the temporally closest weather measurement is linked to the metadata. The integration of synchronised weather data allows for detailed analysis of how meteorological conditions influence acoustic activity. Table 7.1 summarises the weather variables recorded with each 3-second stereo clip, along with their possible effects on the audio data.

Generated labels. Each recording is automatically annotated using two pre-trained deep learning models:

- **AST** [11, 12], an audio spectrogram Transformer model trained for general-purpose sound classification. It includes labels for a wide range of geophonic, biophonic and anthropophonic sources.
- **BirdNET** [6], a convolutional neural network specialised in bird sound detection, trained on expert-annotated bird vocalisations from a global species dataset.

All samples for which the pre-trained AST model detects human speech have been filtered out from the dataset, so that the main focus is on environmental and other typical outdoor sounds. Excluding human vocalisations, the model finds 312 unique labels on the extended dataset. Figure 7.1 shows the distribution of the 20 most common labels predicted by the AST model. Multiple bird-related samples are found in the dataset, with *bird* being the most common detection and many other labels such as *owl*, *chirp* and *birdsong* being among the most common. Other common labels include environmental sounds such as *rustling leaves* and *raindrop*, as well as anthropogenic sounds such as *vehicle* and *door*. Interestingly, among the most common labels we also find three outliers, namely *heartbeat/heart murmur* and *clip-clop*, which are not expected to be present in the dataset. Manual inspection of a small random sample of the associated recordings indicates that heart-related sounds may refer to ambient background noise, often involving wind, while clip-clop may be associated with heavy storms.

BirdNET detects 190 unique bird species in the dataset, many of which are most likely misclassified due to similar vocal patterns as birds that are found in the Netherlands. Figure 7.3 shows the distribution of the confidence scores over all detections. We observe that the model produces a

Label	**Unit**	**Possible effect on the audio**
Precipitation rate	mmh^{-1}	Rain introduces easily detectable broadband patter/splash noise and masks high-frequency animal and insect sounds.
Peak wind-gust speed	kmh^{-1}	Gusts create low-frequency microphone buffeting and brief bursts of leaf rustling that should be detectable in spectrograms.
Average wind speed	kmh^{-1}	Sustained wind adds continuous low-frequency turbulence and can mask animal sounds at higher speeds.
Average wind direction	$^{\circ}$	In stereo recordings, wind direction could potentially be inferred from inter-channel amplitude and time differences in wind noise.
Average air temperature	$^{\circ}C$	Moderate temperature values correlate with increased animal and insect activity, while extreme values tend to suppress it.
Maximum barometric pressure	hPa	High pressure is associated with calm weather, whereas falling pressure indicates fronts with stronger wind and lower bird activity.
Average dew-point temperature	$^{\circ}C$	Higher absolute humidity reduces high-frequency air absorption and is often accompanied by increased insect activity.
Average relative humidity	N/A	
Maximum UV index	N/A	The rise of solar radiation in the morning is correlated with increased bird activity.
Maximum solar radiation	Wm^{-2}	

Table 7.1: Weather variables recorded with each 3s stereo clip and their expected detectable influence on the audio data.

large number of predictions with low confidence. Figure 7.2 shows the distribution of the 20 most common BirdNET detections in the dataset, all of which are found in the Netherlands.

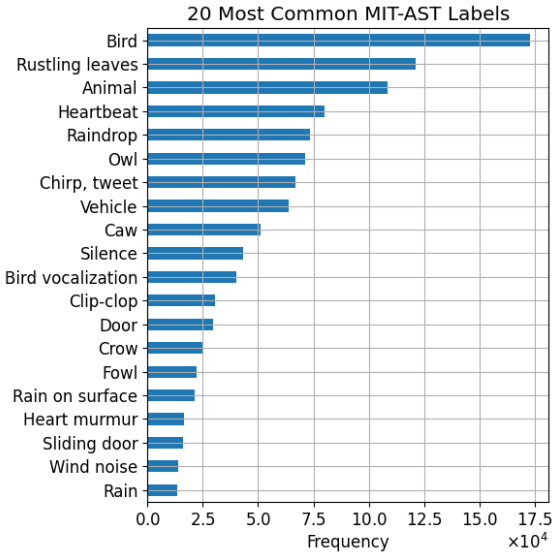


Figure 7.1: Distribution of the 20 most common MIT-AST labels in the dataset.

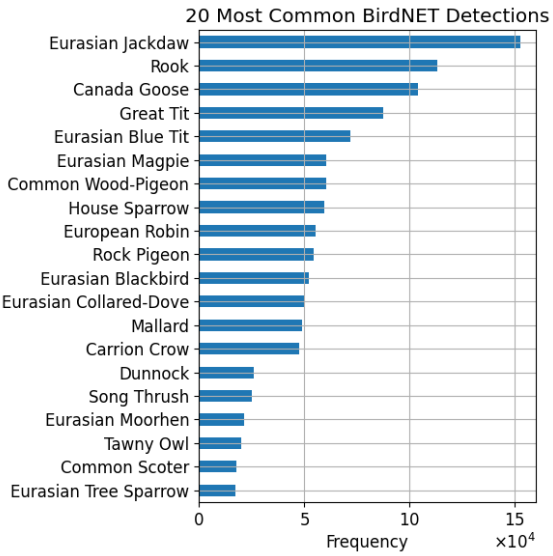


Figure 7.2: Distribution of the 20 most common BirdNET detections in the dataset.

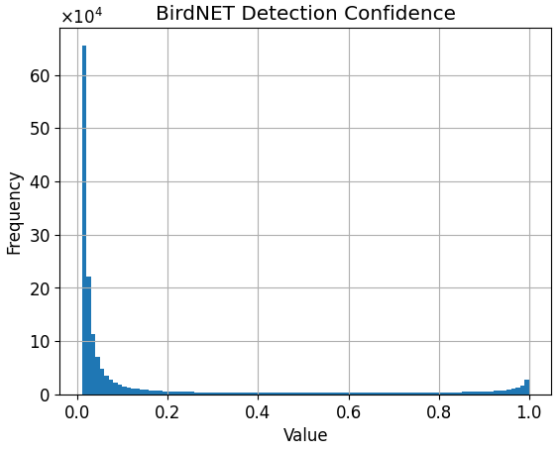


Figure 7.3: Distribution of the confidence scores over all detections.

7.2 Spectrogram pre-processing

Two pipelines were used to convert the audio recordings into mel-spectrograms for our experiments.

Expanded dataset pipeline. The first pipeline was applied to the expanded dataset. One of the two stereo channels is selected from each 2.75-second audio clip and then it is resampled to 24 kHz. A mel-spectrogram with window size 2,048, hop size 1,024 and 128 mel bands is computed. The amplitudes of the spectrogram are also converted to decibels, producing a log-mel-spectrogram of size 65×128. The time resolution of each spectrogram is

$$\Delta t = \frac{2048}{24000} \approx 0.0853 \ s,$$

meaning that each column of the spectrogram corresponds to approximately 85 ms of audio. For this data we only have access to the generated mel-spectrograms.

Original dataset pipeline. For the subset of the expanded dataset that is publicly available, we apply a second spectrogram generation pipeline on the waveform files. In particular, the mel-spectrograms are generated from the original 48 kHz stereo audio files, using a window size of 4,096, a hop size of 2,048 and 128 mel bands. Once again, the amplitudes of the spectrogram are converted to decibels, producing a log-mel-spectrogram of size 65×128 with time resolution $\Delta t \approx 0.0853 \ s$. This pipeline is applied both on each stereo channel individually, as well as the mono channel obtained by averaging the two channels.

Normalisation. In order to avoid extreme values in the spectrograms destabilising the training, we normalise the spectrograms to the range $[0, 1]$. During the normalisation step it is important to truncate extreme values so that they do not skew the normalised spectrograms towards the edges of the range. However, if truncation is too aggressive it may lead to loss of information in the spectrograms. For this reason, we sample a set of 100,000 spectrograms from the expanded dataset and compute the 1st and 99th percentiles of the log amplitude values as approximately -49.25 and 60.9 respectively, which we round to $x_{min} = -50$ and $x_{max} = 60$. We then truncate the spectrograms with respect to x_{min}, x_{max}:

$$x \mapsto \begin{cases} x_{min}, & \text{if } x < x_{min}, \\ x, & \text{if } x_{min} \leq x \leq x_{max}, \\ x_{max}, & \text{if } x > x_{max}. \end{cases}$$

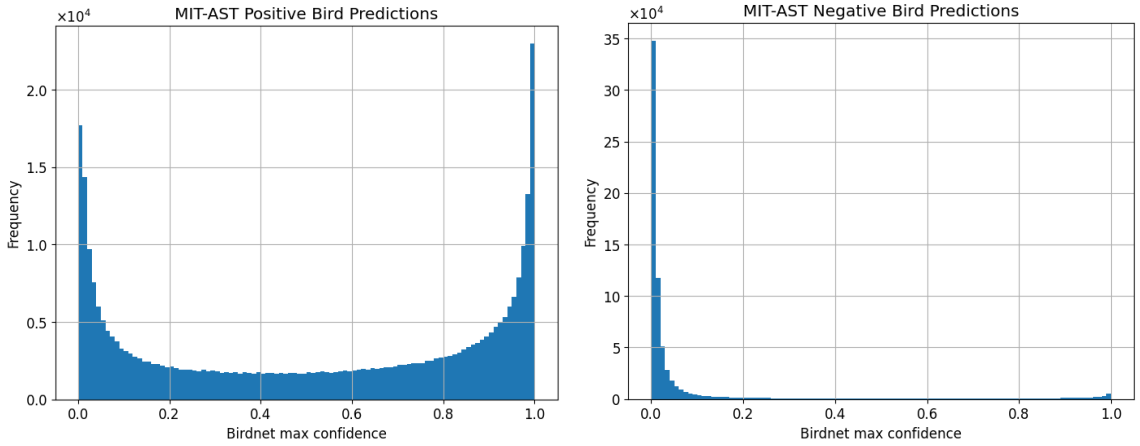
Finally, we normalise the spectrograms to the range $[0, 1]$ using min-max normalisation:

$$x \mapsto \frac{x - x_{min}}{x_{max} - x_{min}}.$$

7.3 Metadata pre-processing

Bird label. We leverage the predictions of both AST and BirdNET models to generate a binary label indicating the presence or absence of birds in each recording. First, we derive an auxiliary label, *MIT_AST_Bird*, by aggregating all bird-related categories predicted by MIT-AST, which

includes 18 labels such as *bird*, *crow*, *owl*, *chirp*, etc. Then, we compare this indicator with the maximum confidence score produced by BirdNET for any species in the clip. Figure 7.4 shows the distribution of BirdNET's maximum confidence score when MIT-AST detects a bird (left) and when it does not (right).



(a) BirdNET maximum confidence score when MIT-AST detects a bird.

(b) BirdNET maximum confidence score when MIT-AST does not detect a bird.

Figure 7.4: Distribution of BirdNET maximum confidence score compared to the *MIT_AST_Bird* label.

We find that both models typically agree when the MIT-AST does not detect a bird, with BirdNET's confidence score distribution being heavily skewed towards low values. However, when MIT-AST detects a bird, BirdNET's confidence score distribution is bimodal, with a large number of clips having high confidence scores and a smaller but significant number of clips with low confidence scores. We assign the final binary label based on the agreement between the two models. The assignment is done as follows:

- Assign label 0 (absence) if

$$MIT\_AST\_Bird = 0 \text{ and } BirdNET\_confidence < 0.5,$$

  or

$$MIT\_AST\_Bird = 1 \text{ and } BirdNET\_confidence < 0.2.$$

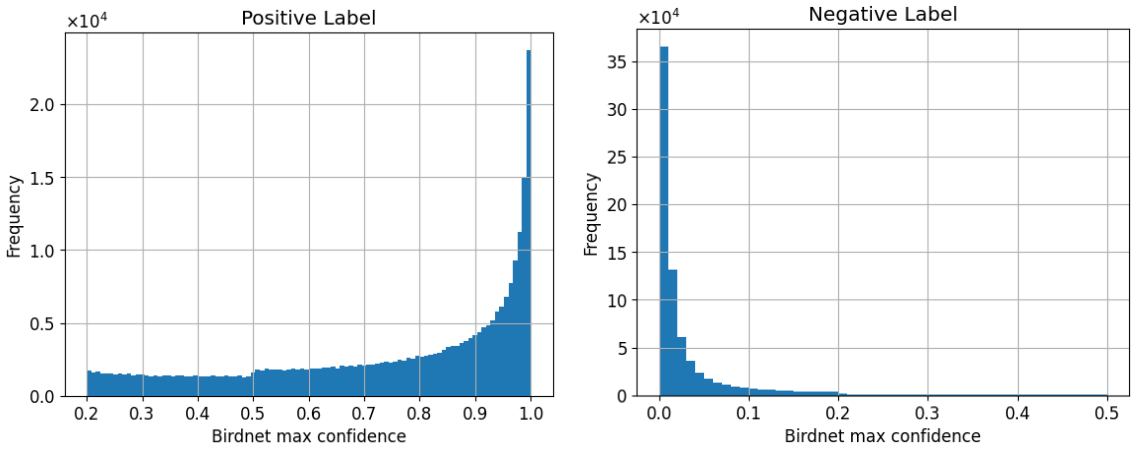
- Assign label 1 (presence) if

$$MIT\_AST\_Bird = 0 \text{ and } BirdNET\_confidence \geq 0.5,$$

  or

$$MIT\_AST\_Bird = 1 \text{ and } BirdNET\_confidence \geq 0.2.$$

The BirdNET confidence scores for the final binary label are shown in Figure 7.5.



(a) BirdNET maximum confidence score for recordings with label 1 (presence).

(b) BirdNET maximum confidence score for recordings with label 0 (absence).

Figure 7.5: Distribution of BirdNET maximum confidence score compared to final bird presence label.

The label distribution is shown in Table 7.2.

Label	**Count**	**Percentage**
0 (absence)	747554	72.9%
1 (presence)	278009	27.1%

Table 7.2: Distribution of the final bird presence label.

Temporal metadata. The metadata includes the timestamp of each recording. We speculate that different periods of the day may present with different characteristics on the audio segments. For example, animal activity (including bird vocalisations or insect sounds), as well as anthropogenic activity both vary throughout the day. We are interested if this correlation can be expressed in the embeddings produced by our models. Since the time of day is a cyclical feature, we choose to encode it as an angle:

$$time\_of\_day\_angle = 2\pi \cdot \frac{hour \cdot 3600 + minute \cdot 60 + second}{86400}. \tag{7.1}$$

The distribution of the timestamp angle is shown in Figure 7.6. We see that the majority of the recordings is concentrated between the morning and the afternoon, with a peak around the early afternoon hours. This is expected given the recording selection process, which emphasises the detection of unique events that differ from the baseline noise and is likely to be more active during the day.

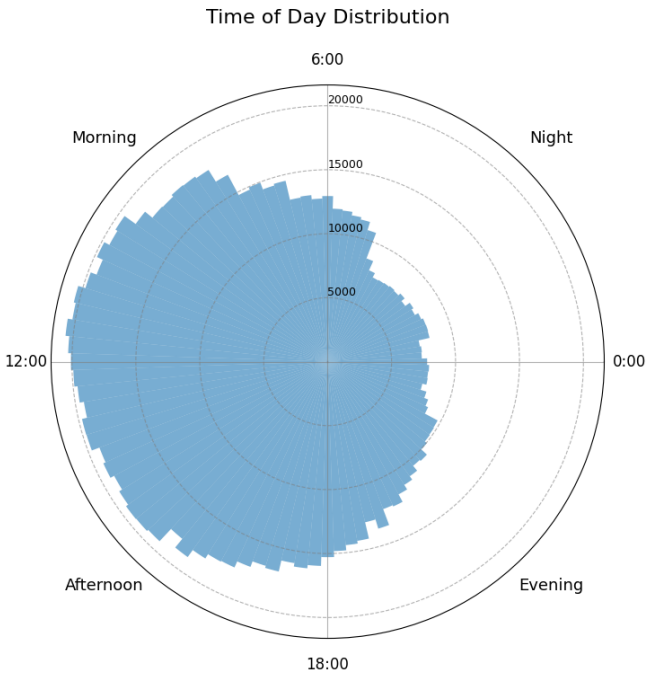


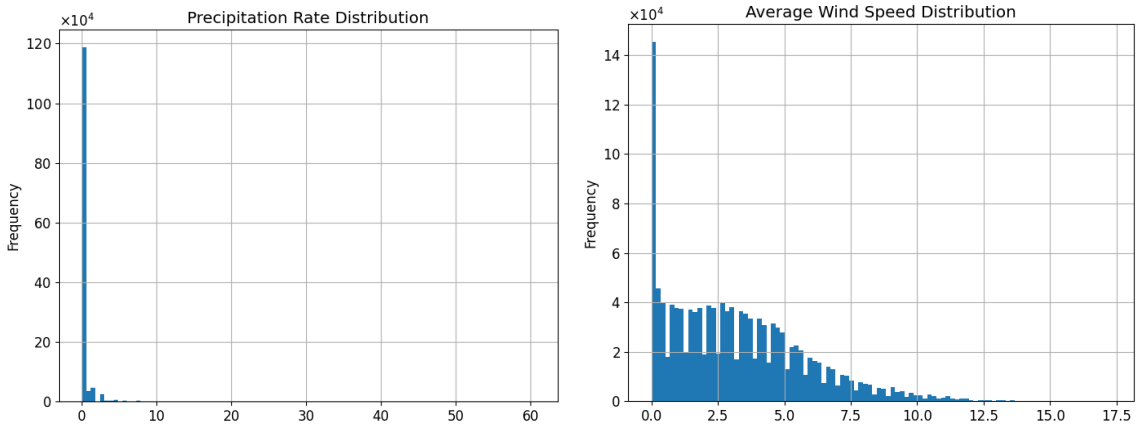
Figure 7.6: Time of day distribution.

To make this target suitable for training, we use the sine and cosine of the calculated angles.

Weather metadata. We choose two weather features to use as targets in our experiments: the precipitation rate and the average wind speed. The original distributions of these values are shown in Figure 7.7. We observe that these features are heavily skewed towards 0 and exhibit a long decaying tail with large range. Note that due to the extreme skewness of the data, it is not advisable to normalise using min-max scaling, as it would lead to most of the values being concentrated at the lower end of the range or to apply standardisation, as it would lead to large and infrequent outliers dominating the mean and standard deviation estimations. To avoid numerical instability during training and over-fitting to extreme values, we instead compress the true range by applying a logarithmic transformation:

$$x \mapsto \log(x + 1).$$

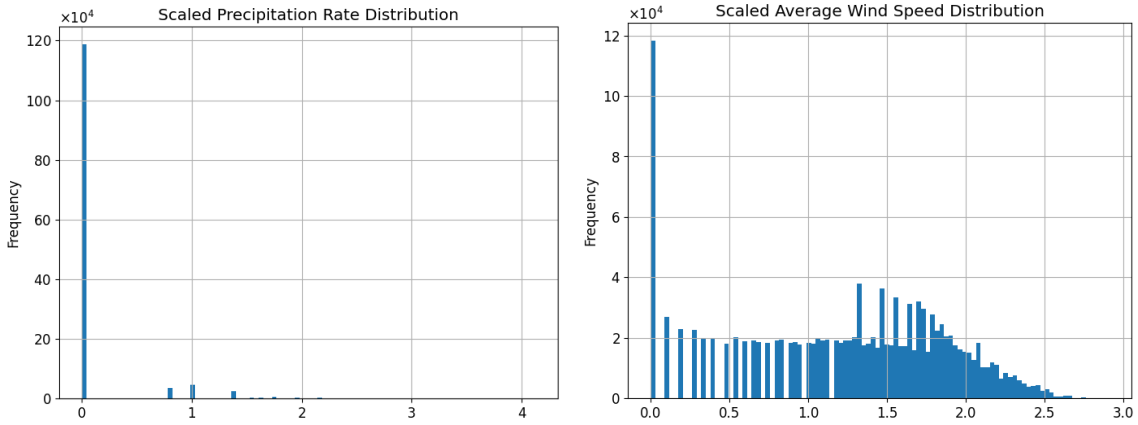
The resulting distributions are shown in Figure 7.8.



(a) Precipitation rate.

(b) Average wind speed.

Figure 7.7: Distribution of the chosen weather metadata features.



(a) Scaled Precipitation rate target.

(b) Scaled Average wind speed target.

Figure 7.8: Distribution of the scaled weather metadata targets.

Chapter 8

Methodology

In this chapter we describe and motivate the methodology used in this work. We first provide an overview of the entire training pipeline and then describe in detail the encoder architectures compared. Moreover, we summarise the self-supervised pre-training tasks used and the fine-tuning and evaluation procedures.

8.1 Overview

In this work we aim to create a foundation model suitable for feature extraction on environmental audio data. Following a typical foundation model training pipeline as described in chapter 3, we first pre-train our models on a large dataset using self-supervised tasks and then fine-tune them on subsets of the original dataset using supervised tasks. Our encoder architecture follows the SSAST paradigm, consisting of the following sequential steps:

1. Each spectrogram is split into non-overlapping patches, from which a sequence of tokens is created.
2. The sequence of tokens passes through an embedding layer.
3. A ratio of the input tokens is masked using a chosen masking strategy.
4. Positional encodings are added to the input tokens.
5. The desired features are extracted through a series of encoder blocks.
6. The extracted embeddings are processed by a task-specific head.

In this work we compare two architectures for the encoder blocks:

- **SSAST** [13]: a model based on the Self-Supervised Audio Spectrogram Transformer architecture. The model is built using standard ViT blocks.
- **Bi-directional Mamba** [53, 55]: a model which replaces the standard ViT blocks with bi-directional Mamba blocks.

We conduct experiments using two pre-training pipelines:

- **Masked reconstruction:** a self-supervised task where a ratio of the input patches is randomly masked and the model is trained to reconstruct the original input from the unmasked patches.
- **Contrastive learning:** a self-supervised task where the model is trained to distinguish between positive and negative pairs of samples. Instead of creating positive pairs by applying random augmentations on the sample, we use the left and right channels of our chosen dataset's stereo recordings. The two channels are randomly masked in a complementary manner, meaning that if a patch is masked in the left channel, it is unmasked in the right channel and vice versa.

Finally, we fine-tune our models and evaluate them on the following downstream tasks:

- **Bird detection:** a binary classification task where the model is trained to detect the presence of birds in a given audio clip.
- **Temporal metadata prediction:** a regression task where the model is trained to predict the time of day from a given audio clip, which is treated as a cyclical variable and encoded by sine and cosine pairs.
- **Weather metadata prediction:** a regression task in which the model is trained to predict weather metadata associated with each audio clip, including precipitation rate and average wind speed.

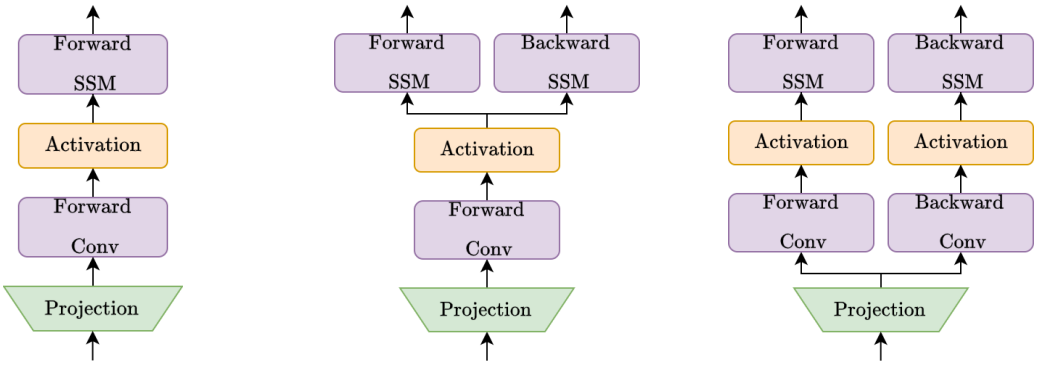
8.2 Model architectures

SSAST. The SSAST block is a standard ViT block, which we use as the state-of-the-art architecture which provides the baseline for the comparisons in our work. Staying close to the archetypical Transformer block architecture, the ViT block first applies normalisation on the input and then calculates the query, key and value matrices using linear projections. Multi-head self-attention is then calculated, followed by another normalisation layer and a feed-forward network. The block includes residual connections, combining the block input with the output of the attention mechanism and then the attention scores with the output of the feed-forward network. The SSAST block is shown in Figure 8.2a.

Bi-directional Mamba. The bi-directional Mamba block builds upon the standard Mamba block by processing the input sequence both in the forward and backward directions, aiming to capture the full context of the input. This is typically implemented in one of two ways:

- **Single forward convolution, bi-directional SSM**: The input sequence is passed through a single forward convolution, after which it is split into two separate paths, passing through two separate SSM layers, one for the forward and one for the backward direction, as shown in Figure 8.1b.
- **Bi-directional convolution, bi-directional SSM**: The input sequence is split directly after the first linear projection of the layer and each path is processed separately. The two sequences

each pass first through a forward/backward convolution and then through a forward/backward SSM layer, as shown in Figure 8.1c.



(a) Standard Mamba block with single forward convolution, single forward SSM.

(b) Bi-directional Mamba block with single forward convolution, bi-directional SSM.

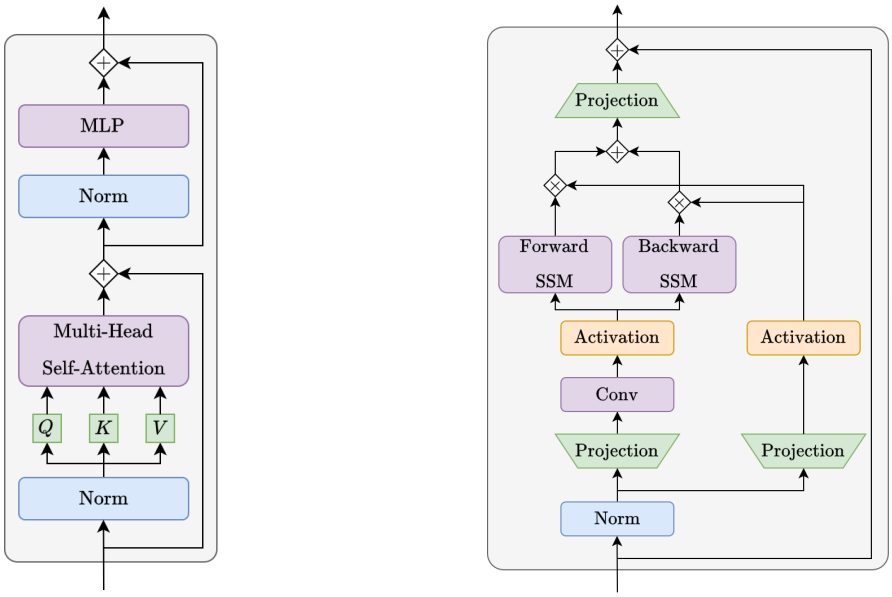
(c) Bi-directional Mamba block with bi-directional convolution, bi-directional SSM.

Figure 8.1: Comparison of a standard forward Mamba block with bi-directional Mamba block implementations.

Both of these implementations have been used in the literature. For example, Shams et al. [55] use the second implementation to train a bi-directional Mamba model in a self-supervised manner. On the other hand, Erol et al. [53] test both implementations in a supervised setting and find that the first implementation generally performs better in many of their experiments. Inspired by their results, we adopt a single forward convolution, bi-directional SSM implementation for our bi-directional Mamba architecture. The choice can be further justified as follows:

- This implementation is more efficient, as it requires fewer parameters per block.
- In agreement with the analysis of Erol et al., we hypothesise that the bi-directional SSM benefits from extracting features using two scans on the same input sequence and then combining the information, rather than acting on two separate sequences. From a theoretical point of view, the output y_t of the SSM at time t aims to summarise the cumulative history $x_{\leq t}$ of the input sequence up to time t. Therefore it seems natural to use the same input sequence for both forward and backward SSMs, as they are both trying to summarise the same information, with each sequence being biased towards a different view of the input. The final aggregated output sequence should then in theory at time t contain information about both $x_{\leq t}$ as well as $x_{\geq t}$.

The full bi-directional Mamba block is shown in Figure 8.2b. The block contains all the standard Mamba block components (cf. section 5.5), with the only difference being the addition of a backward SSM layer and a residual connection which bypasses the input normalisation layer and is added directly to the output of the Mamba pipeline.



(a) SSAST block.

(b) Bi-directional Mamba block.

Figure 8.2: Encoder blocks.

8.3 Self-supervised pre-training stage

Input representation and embeddings. A spectrogram $s \in \mathbb{R}^{T \times F}$ is first split into l non-overlapping patches using a window of size $t \times f$, following a zig-zag pattern starting from top-left and moving towards the right. The patch sequence is then flattened to produce a representation

$$x = (x_1, \ldots, x_l) \in \mathbb{R}^{l \times d},$$

where $l = \frac{T \cdot F}{t \cdot f}$ is the number of tokens and $d = t \cdot f$ is the patch dimension. The sequence x is the input to the model. Each token x_i of the input sequence x is projected into a d_m-dimensional space through a linear embedding layer E. Finally, trainable positional encodings are added to the input patches.

8.3.1 Pre-training tasks

We train our models on two self-supervised tasks: masked reconstruction and contrastive learning.

Masked reconstruction. As detailed in chapter 3, masked reconstruction is a typical foundation model pre-training task and we use it to establish a baseline for the performance of our models. For this task, a ratio $r = 0.5$ of the input embeddings is randomly replaced by a trainable mask token $[M]$ before the addition of the positional encodings. The encoder produces a sequence of embeddings $h = (h_i, \ldots, h_l) \in \mathbb{R}^{l \times d_m}$ and an MLP layer is applied on each token in order to produce the final reconstructions. We train the model to minimise the reconstruction loss between

the masked tokens and the corresponding reconstructions, with no other tokens being included in the loss calculation. We experiment with two loss functions, namely mean squared error (3.1) and mean absolute error (3.2). This pipeline is illustrated in Figure 8.3.

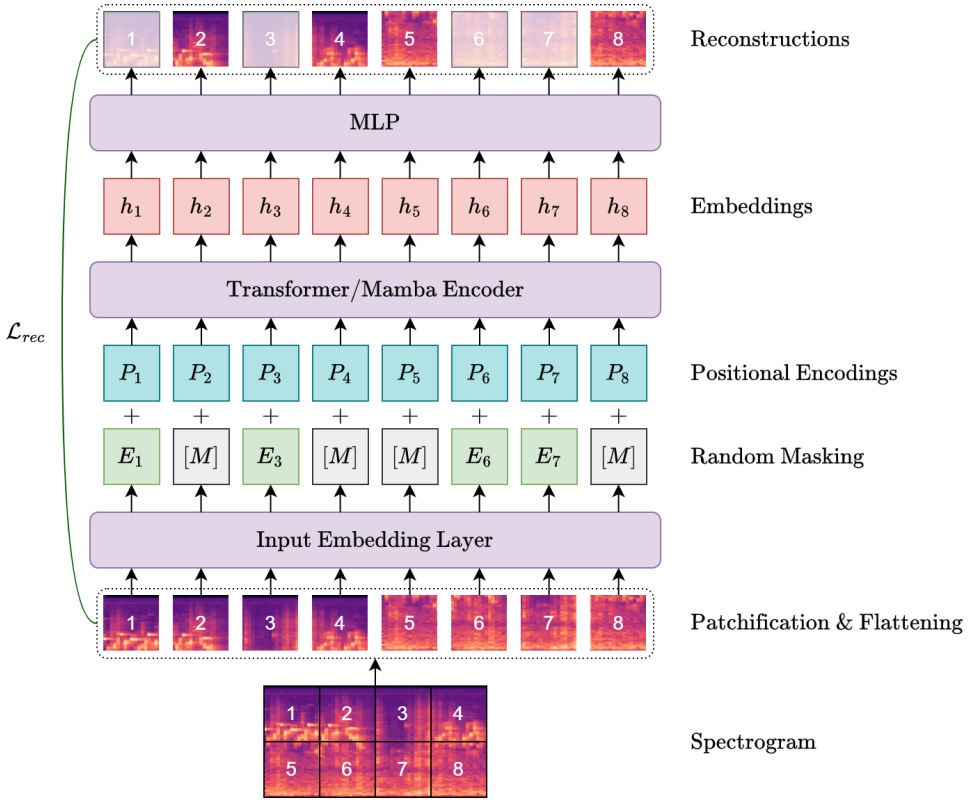


Figure 8.3: Reconstruction pre-training pipeline.

Contrastive learning. The contrastive learning task is a stereo contrastive task, where the model is trained to distinguish between positive and negative pairs of samples. Instead of creating positive pairs by applying random augmentations on the sample, we use the left and right channels of our chosen dataset's stereo recordings. The choice for this task is motivated as follows:

- The left and right channels of the stereo recordings are typically highly correlated in terms of the meaningful acoustic information they contain and therefore can be used to create positive pairs of samples.
- Given the location of the recording setup, the two channels often contain noise and other artifacts that are less correlated, which may be perceived as weak but naturally occurring augmentations applied to the same signal.

Due to the high correlation between the two channels, the two views are not expected to be different enough to be used as positive pairs without any further processing, as the contrastive task is

very easy. Therefore, we apply a masking strategy to the input patches, similar to the masked reconstruction task. The two channels are randomly masked by a ratio of 0.5 in a complementary manner, meaning that if a patch is masked in the left channel, it is unmasked in the right channel and vice versa. This strategy was chosen based on the observation that the models could typically achieve very low reconstruction losses on the masked patches even at high masking ratios and therefore they may be able to produce meaningful embeddings by learning to match two randomly segmented halves of the same weakly augmented signal.

In order to prevent the model from simply memorising the position of the masked token in the positive samples during training, we instead replace the masked patches by random noise sampled from a truncated normal distribution. After the tokens pass through the encoder, an average pooling layer is applied to the resulting embeddings. The two channel embeddings are then passed through an MLP layer and the InfoNCE loss (3.3) of the projected embeddings is used as a training objective. The contrastive learning pipeline is shown in Figure 8.4.

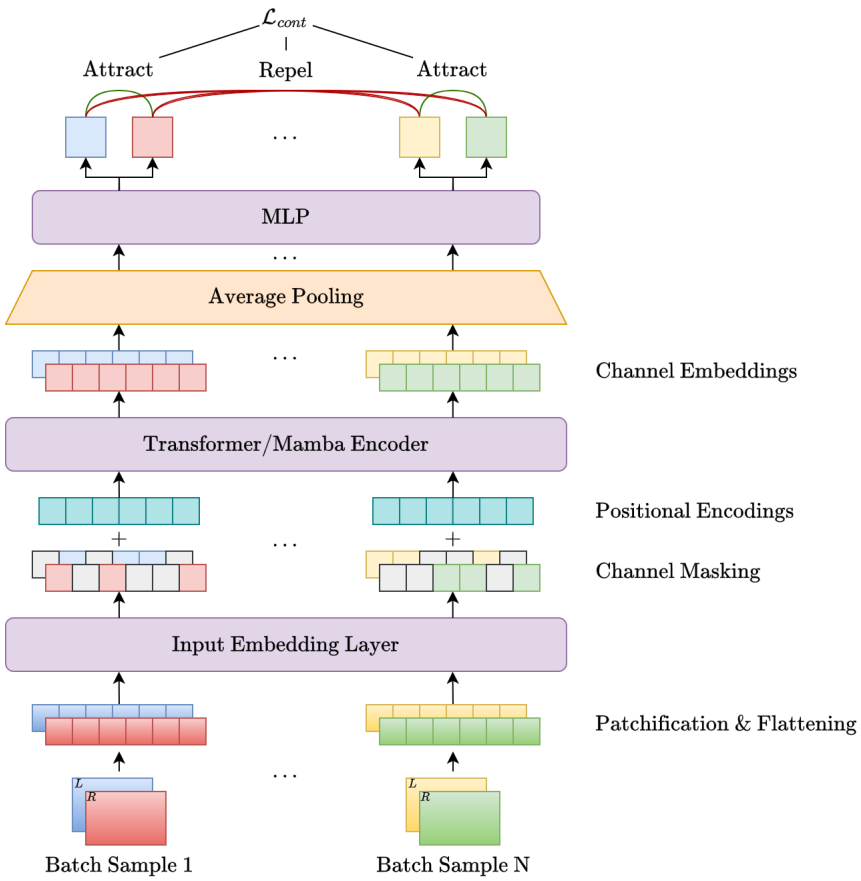


Figure 8.4: Contrastive pre-training pipeline.

8.4 Fine-tuning and evaluation

We fine-tune our models on three downstream tasks: bird detection, temporal metadata prediction and weather metadata prediction, as detailed in chapter 7. During fine-tuning the pre-trained encoder is frozen and no masking is applied to the input. An average pooling layer is applied to the output of the encoder, producing an aggregated embedding for each audio clip. The sample embedding is then passed through an MLP which is trained specifically for each of the three tasks. The fine-tuning pipeline is visualised in Figure 8.5.

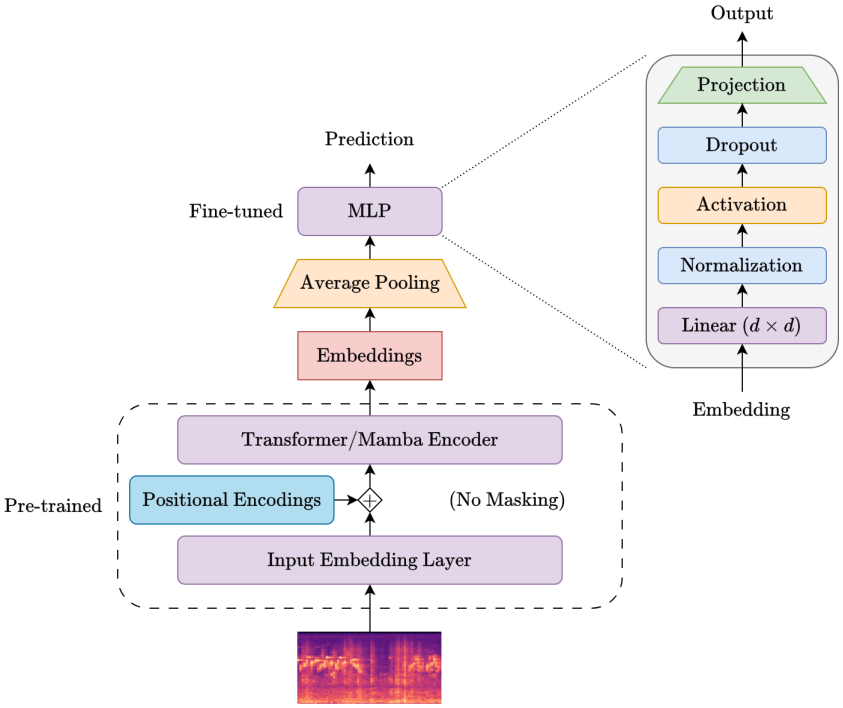


Figure 8.5: Downstream tasks.

For the bird detection task, a weighted binary cross-entropy loss is applied:

$$\mathrm{BCE}_{w_1,w_2}(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left(w_1 \cdot y_i \log(\hat{y}_i) + w_2 \cdot (1-y_i)\log(1-\hat{y}_i)\right), \tag{8.1}$$

where y is the ground truth label, $\hat{y}$ is the predicted probability and w_1 and w_2 are the weights for the positive and negative classes, respectively. The weights are calculated as the inverse of the class frequencies in the dataset (cf. Table 7.2) and are used to balance the loss function in order to account for the class imbalance in the dataset.

For the temporal metadata, which are encoded as sine-cosine pairs, we predict two values and train using the MSE loss. Finally, for the weather metadata we predict concurrently both the precipitation rate and the average wind speed. In order to assist the model to accurately predict

values larger than 1, which are very uncommon, without letting them dominate the loss and losing accuracy in the much more common values near 0, we use the Huber loss, which behaves like the MSE loss for small errors and like the MAE loss for large errors. The Huber loss is defined as follows:

$$\text{Huber}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| < 1, \\ |y - \hat{y}| - \frac{1}{2} & \text{otherwise.} \end{cases} \tag{8.2}$$

Chapter 9

Experiments and Results

In this chapter we provide the details of our realisation of the steps mentioned in chapter 8 and present their results. In particular, we describe the process used to generate our pre-training, fine-tuning and evaluation datasets, as well as the chosen implementation and hyperparameter settings for the models. We also provide quantitative and qualitative results for the best performing pre-training and fine-tuning experiments and discuss the performance of the models on the evaluation tasks.

9.1 Data splits

The dataset contains 1,321,378 samples in total, 744,288 out of which come from the original GardenFiles23 dataset [56]. We now describe the splits used for the pre-training and fine-tuning experiments, as well as the sources of each split. The splits are summarised in Tables 9.1 and 9.2.

Held-out test set. A subset of 100,000 examples is randomly drawn from the original dataset, which is used as a held-out test set. This set is not used in any of the training or validation procedures and is only used to evaluate the final performance of the models.

Masked reconstruction set. For the masked reconstruction experiments we use the remaining 1,221,378 samples of the expanded dataset, which are randomly split into training and validation sets, each containing 1,121,378 and 100,000 samples respectively. The validation set is used to monitor the training process and select the best model for downstream tasks based on minimum validation loss.

Contrastive set. For the contrastive experiments we use the remaining 644,288 samples of the original GardenFiles23 dataset. The training and validation sets are generated as the intersection between the original dataset with the masked reconstruction training and validation sets, respectively. The resulting training set contains 591,514 samples, while the validation set contains 52,774 samples. Once again, we use the validation loss as a criterion to select the best model for downstream tasks.

Split	Source	Train	Validation	Test	Total
Held-out test set	Original GardenFiles23	—	—	100,000	100,000
Masked. rec. set	Expanded dataset – Test set	1,121,378	100,000	100,000	1,321,378
Contrastive set	Original GardenFiles23 – Test set	591,514	52,774	100,000	744,288

Table 9.1: Pre-training dataset splits. The contrastive training and validation sets are obtained from the intersection of the original GardenFiles23 dataset with the masked reconstruction training and validation sets, respectively.

Fine-tuning set. For the fine-tuning experiments we construct a training set of 550,000 samples as follows:

- We randomly sample 400,000 samples from the contrastive training set.
- We randomly sample 100,000 samples not included in the contrastive set from the masked reconstruction training set.
- We randomly sample 50,000 samples from the masked reconstruction validation set.

This strategy ensures that the fine-tuning data includes a subset of samples seen during pre-training (from the contrastive and masked reconstruction sets) and additional examples that the models have not yet encountered. The remaining 50,000 examples from the masked reconstruction validation set serve as the fine-tuning validation set and are used to select the best fine-tuned models, which are then evaluated on the held-out test set.

Split	Source	Size
Train	Contrastive train set	400,000
	Masked. rec. train set – Contrastive train set	100,000
	Masked. rec. validation set	50,000
		Total: 550,000
Validation	Masked. rec. validation set	50,000
Test	Held-out test set	100,000

Table 9.2: Fine-tuning set splits.

9.2 Implementation details

Both models are implemented in PyTorch and share the same input embedding, positional encoding and masking functionalities. The SSAST model is built from standard ViT components sourced from the timm library [57]. The bidirectional Mamba model is implemented based on the Vision

Mamba architecture [58] and associated code[1], which builds upon the official unidirectional Mamba release[2] [30]. Our codebase is provided as an installable Python package in `https://github.com/pk-470/master-thesis`.

Training was conducted on an internal computing unit provided by Maastricht University, which is equipped with NVIDIA GeForce RTX 2080 Ti and Quadro RTX 6000 GPUs, as well as on the Snellius[3] computing cluster provided by SURF[4], which includes NVIDIA A100 and NVIDIA H100 GPUs. We recommend training the Mamba models on NVIDIA GPUs, as the official implementation is optimised and supported mainly for these devices.

FlashAttention [31] was used to accelerate the training of the SSAST model, while the Mamba model was trained using its own optimised fused kernel implementation. Training time varies depending on the model, the type and number of GPUs used and the availability of computing units, but generally took between 2 to 4 days per experiment for the pre-training phase and around 1 day per experiment for the fine-tuning phase.

9.3 Pre-training

As described in chapter 7, the input to all models consists of normalised log-mel-spectrograms of size $T \times F = 65 \times 128$. We use a patch size of $t \times f = 5 \times 16$, which produces 13 patches in the time dimension and 8 patches in the frequency dimension, resulting in an input sequence x consisting of 104 patches, each of size $5 \times 16 = 80$.

Number of parameters per block. The original Mamba block applies a linear projection to the input sequence, which expands the input dimension by a factor of 2. Subsequently, it applies a convolutional layer with a kernel size of 4 on the projected input sequence and applies an SSM with a hidden state dimension of 16. On the other hand, the ViT model constructs the key, query and value matrices using linear projections with an expansion factor of 4. In practice, this results in a block with approximately half the number of parameters compared to a standard ViT block with the same input dimension and therefore typically Mamba models employ twice as many blocks as the Transformer architecture they are compared to (e.g., Shams et al. [55]). To ensure a fair comparison between the Mamba and ViT blocks, we follow the methodology of Yadav and Tan [50]. In particular, we choose the model hyper-parameters such that each block of the Mamba model has approximately the same number of parameters as the corresponding block of the SSAST model, thus removing the influence of depth as a hyperparameter affecting the model performance. Accordingly, we choose the hyper-parameters of the Mamba and SSAST models as shown in Tables 9.3 and 9.4. These choices result in a Bi-directional Mamba model with 405,120 parameters per block and a SSAST model with 444,480 parameters per block. The total number of parameters for the Bi-directional Mamba model is 4,897,730, while the SSAST model has 5,422,160 parameters.

[1] `https://github.com/hustvl/Vim`
[2] `https://github.com/state-spaces/mamba`
[3] `https://www.surf.nl/en/services/compute/snellius-the-national-supercomputer`
[4] `https://www.surf.nl/`

Bi-directional Mamba Model	
Embedding dimension	192
Expansion factor	3
Convolution kernel size	4
SSM hidden state dimension	24
Number of blocks	12

Table 9.3: Hyper-parameters of the SSM-based model.

SSAST Model	
Embedding dimension	192
Expansion factor	4
Number of heads	3
Number of blocks	12

Table 9.4: Hyper-parameters of the SSAST model.

9.3.1 Masked reconstruction

Each model is trained over 80 epochs with a batch size of 256. The AdamW optimiser [59] is used with a weight decay of 10^{-2}. The learning rate is set according to the OneCycle policy [60] using a warmup stage over the first 30% of the epochs, which anneals the learning rate from a small initial value to a maximum value of $5 \cdot 10^{-4}$ over the first 24 epochs and then decays it to zero over the remaining 56 epochs, using a cosine annealing schedule. The Early Stopping criterion is used to terminate the training process if the validation loss does not improve over 11 successive epochs, thus preventing over-fitting. The models are trained on the masked reconstruction pre-training dataset described in section 9.1.

The masked reconstruction pre-training involves 4 separate experiments, namely:

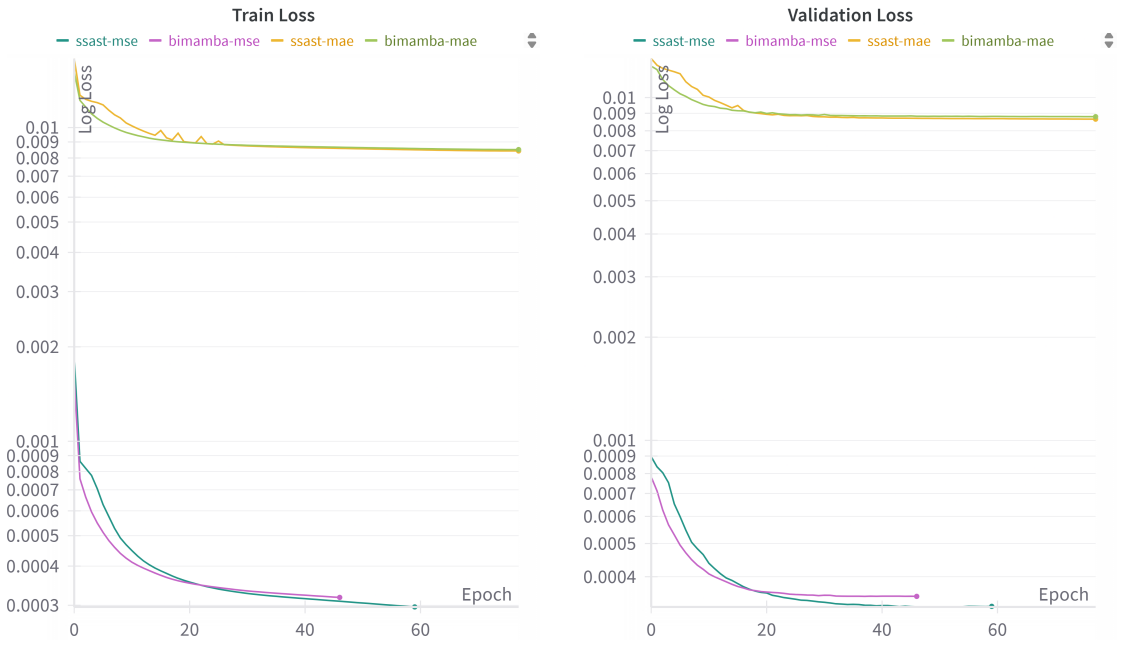
- `bimamba-mae`: Bi-directional Mamba model with masked MAE loss.
- `bimamba-mse`: Bi-directional Mamba model with masked MSE loss.
- `ssast-mae`: SSAST model with masked MAE loss.
- `ssast-mse`: SSAST model with masked MSE loss.

The training and validation losses for the masked reconstruction pre-training are shown in Figure 9.1. We observe that the models trained with the MSE loss converge faster to the point of over-fitting, thus triggering the Early Stopping criterion, unlike the models trained with MAE which continue to slowly improve over the entire training period.

9.3.2 Contrastive learning

For the contrastive pre-training, we train over 100 epochs with a batch size of 256 and use the AdamW optimiser, the OneCycle learning rate scheduler and the Early Stopping criterion as in the masked reconstruction pre-training. The maximum learning rate is set to $2 \cdot 10^{-4}$, while the temperature parameter in the InfoNCE loss is set to 0.01. Due to initial experiments that demonstrated instability in the learning process, we clip the gradients to a maximum of 1. The models are trained on the contrastive dataset described in section 9.1.

The contrastive pre-training consists of the following 2 experiments:



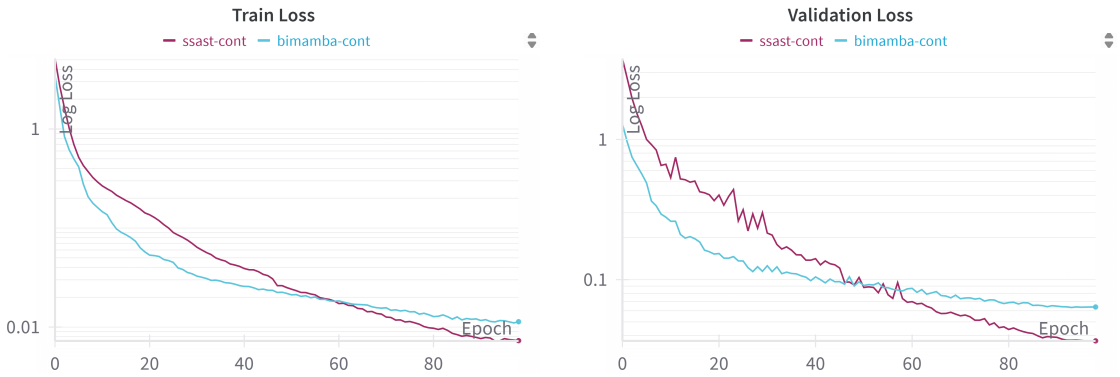
(a) Masked reconstruction training loss.

(b) Masked reconstruction validation loss.

Figure 9.1: Masked reconstruction training and validation losses for the pre-training experiments.

- `bimamba-cont`: Bi-directional Mamba model.
- `ssast-cont`: SSAST model.

Figure 9.2 shows the training and validation losses for the contrastive pre-training experiments.



(a) Contrastive training loss.

(b) Contrastive validation loss.

Figure 9.2: Contrastive training and validation losses for the pre-training experiments.

9.4 Fine-tuning

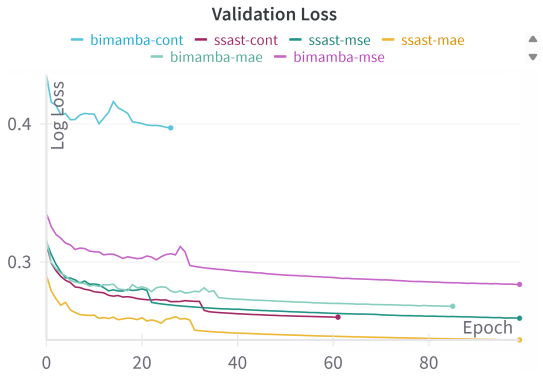
For the fine-tuning experiments we extract and freeze the backbone of the pre-trained models and train an MLP head consisting of two linear layers with ReLU activation and a dropout layer between them. The dropout rate is set to 0.3. The use of a single linear layer as the output head was also tested, but resulted in significantly lower performance across all models and tasks, so it is excluded from the final evaluation. For all experiments we use the AdamW optimiser and the Early Stopping criterion as in the pre-training phase. All models are trained on the fine-tuning dataset described in section 9.1.

9.4.1 Bird detection

For the bird detection task we train over 100 epochs with a batch size of 64. The learning rate is initially set to a value of 10^{-3} and is reduced using the ReduceOnPlateau strategy, which multiplies it by a factor of 0.1 if the validation loss does not improve over 5 successive epochs.

Train and validation losses for the bird detection task are shown in Figure 9.3. Sudden jumps in the losses are accounted for by the reduction of the learning rate. The training of the `bimamba-cont` experiment was cut short to save training time and computing budget, as it was observed across multiple similar experiments to struggle with convergence at high learning rates and to converge to higher validation losses compared to the other models at lower learning rates.

(a) Bird detection training loss.

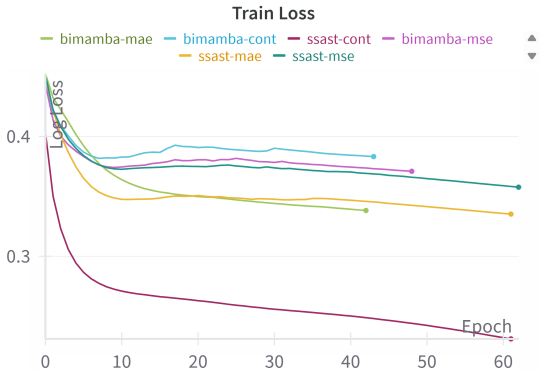


(b) Bird detection validation loss.

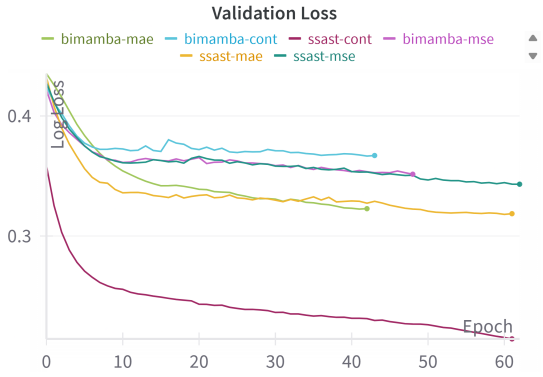
Figure 9.3: Bird detection training and validation losses during fine-tuning.

9.4.2 Temporal metadata

For the temporal metadata prediction task, which includes predicting two values corresponding to the sine and cosine of the time of day encoded as an angle, we fine-tune our models over 60 epochs using a batch size of 64. We use the OneCycle learning rate scheduler with a maximum learning rate of $5 \cdot 10^{-4}$. The training and validation losses for the temporal metadata prediction task are shown in Figure 9.4. We find that the `ssast-cont` model converges quickly to lower training and validation losses compared to the other models.



(a) Temporal metadata training loss.

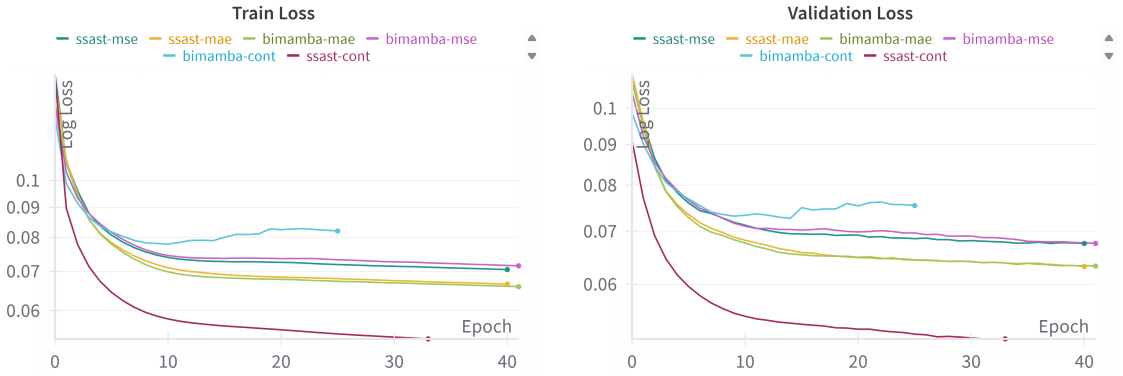


(b) Temporal metadata validation loss.

Figure 9.4: Temporal metadata training and validation losses during fine-tuning.

9.4.3 Weather metadata

Finally, for weather metadata prediction, which include precipitation rate and average wind speed, we train the models over 40 epochs using a batch size of 64. The OneCycle learning rate scheduler is used with a maximum learning rate of $5 \cdot 10^{-4}$. Figure 9.5 shows the training and validation losses for the weather metadata prediction task. The `bimamba-cont` model exhibits a similar instability during training as in the previous tasks, while the `ssast-cont` model is singled out as it achieves the lowest training and validation loss among all models.



(a) Weather metadata training loss.

(b) Weather metadata validation loss.

Figure 9.5: Weather metadata training and validation losses during fine-tuning.

9.5 Evaluation

We now present the results of the evaluation of the pre-trained and fine-tuned models on the held-out test set.

9.5.1 Embeddings

We qualitatively evaluate the normalised embeddings produced by the pre-trained models using alignment and uniformity metrics, as well as PCA analysis.

Alignment and uniformity. In the context of self-supervised learning, alignment and uniformity refer to two complementary metrics used to evaluate the robustness of embeddings to variations against the overall spread of the embeddings in the latent space. More precisely, given a feature extractor g with outputs on a high-dimensional unit sphere, a set of input samples X and a random transformation T, the alignment of g estimates the average distance between the embeddings of the original samples and their transformed versions as follows:

$$\text{Alignment}(g) = \mathbb{E}_{x \in X} \left[\|g(x) - g\left(T(x)\right)\|^2 \right].$$

Smaller alignment is preferable, as it indicates that the model is robust to variations and produces similar embeddings for similar samples. On the other hand, uniformity measures the overall spread

of embeddings in the unit sphere:

$$\text{Uniformity}_t(g) = \log \mathbb{E}_{x,y \in X} \left[\exp \left(-t \cdot \|g(x) - g(y)\|^2 \right) \right],$$

where t is a temperature parameter that controls the sensitivity of the metric, typically set to 2. Note that uniformity is a negative metric, so smaller values (i.e., larger in magnitude) are preferable, as they indicate that the produced embeddings are more evenly spread in the latent space, thus potentially leading to better generalisation performance. We estimate the alignment and uniformity metrics for all pre-trained models by randomly sampling $2^{13} = 8,192$ points from the test set. To produce positive pairs for the alignment metric we apply random transformations both on the waveform level, including:

- addition of Gaussian noise with a standard deviation of $0.1 \cdot \max |x|$ for each sample x,
- random volume gain between -5 and 5 dB,
- random pitch shift between -2 and 2 semitones,

as well as on the (normalised) spectrogram level, including:

- random Gaussian noise addition with a standard deviation of 0.01,
- random roll of the spectrogram along the time axis between -5 and 5 time steps,
- random masking of 5 time steps and 5 frequency bins.

Figure 9.6 shows the alignment and negative uniformity metrics for all pre-trained models. We observe distinct patterns in the results across model types and pre-training techniques. In particular, the Mamba models typically achieve lower alignment and lower absolute uniformity, meaning that the embeddings tend to cover a tightly clustered area of the unit sphere, while the SSAST models achieve higher alignment and higher absolute uniformity, indicating spread out embeddings regardless of similarity. Additionally, the MSE pre-training technique produces different results between the two model types, achieving the lowest alignment and absolute uniformity among the Mamba models and the highest alignment with the second highest absolute uniformity among the SSAST models. The masked MAE pre-training technique produces similar balanced results for both model types, whereas the contrastive models achieve the highest uniformity and the second lowest alignment within their model types, offering a good trade-off among their respective model types.

PCA analysis. We further evaluate the embeddings produced by the pre-trained models using PCA analysis. We apply PCA using 3 principal components to the 2^{13} sampled normalised embeddings of the test set and visualise the first two components across all models in Figure 9.7, with Figure 9.8 showing the PCA plots for each model separately for more clarity. The patterns observed by the alignment versus uniformity plot are further supported by these plots, with the Mamba models producing tightly clustered embeddings and the SSAST models producing more spread out embeddings. The `ssast-cont` model seems to cover the unit sphere almost uniformly, while the `bimamba-mse` model shows the smallest spread of embeddings.

Figure 9.9 shows the variance explained by the first 3 principal components for all pre-trained models, as measured by the magnitude of the corresponding eigenvalues. Large values in the first component followed by a steep drop in the second and third components indicate that most of

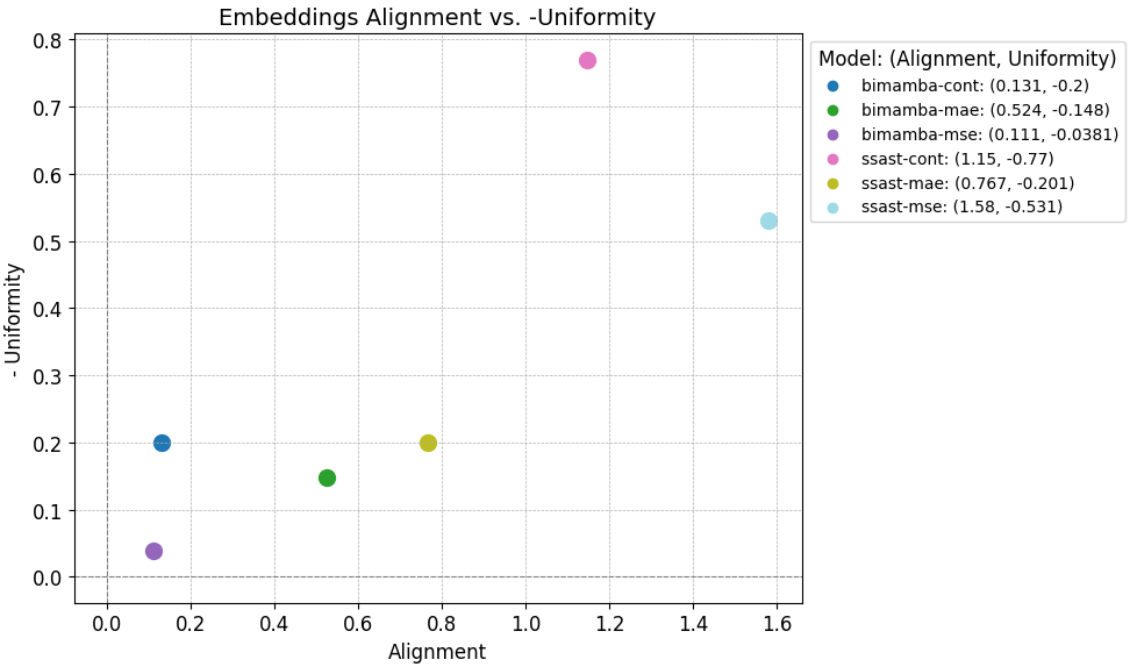


Figure 9.6: Alignment and absolute uniformity of the embeddings for all pre-trained models.

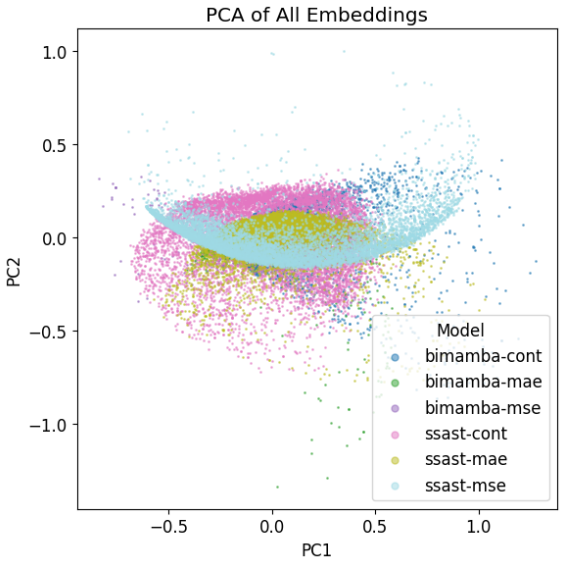


Figure 9.7: Visualisation of the first two PCA components of the embeddings for all pre-trained models.

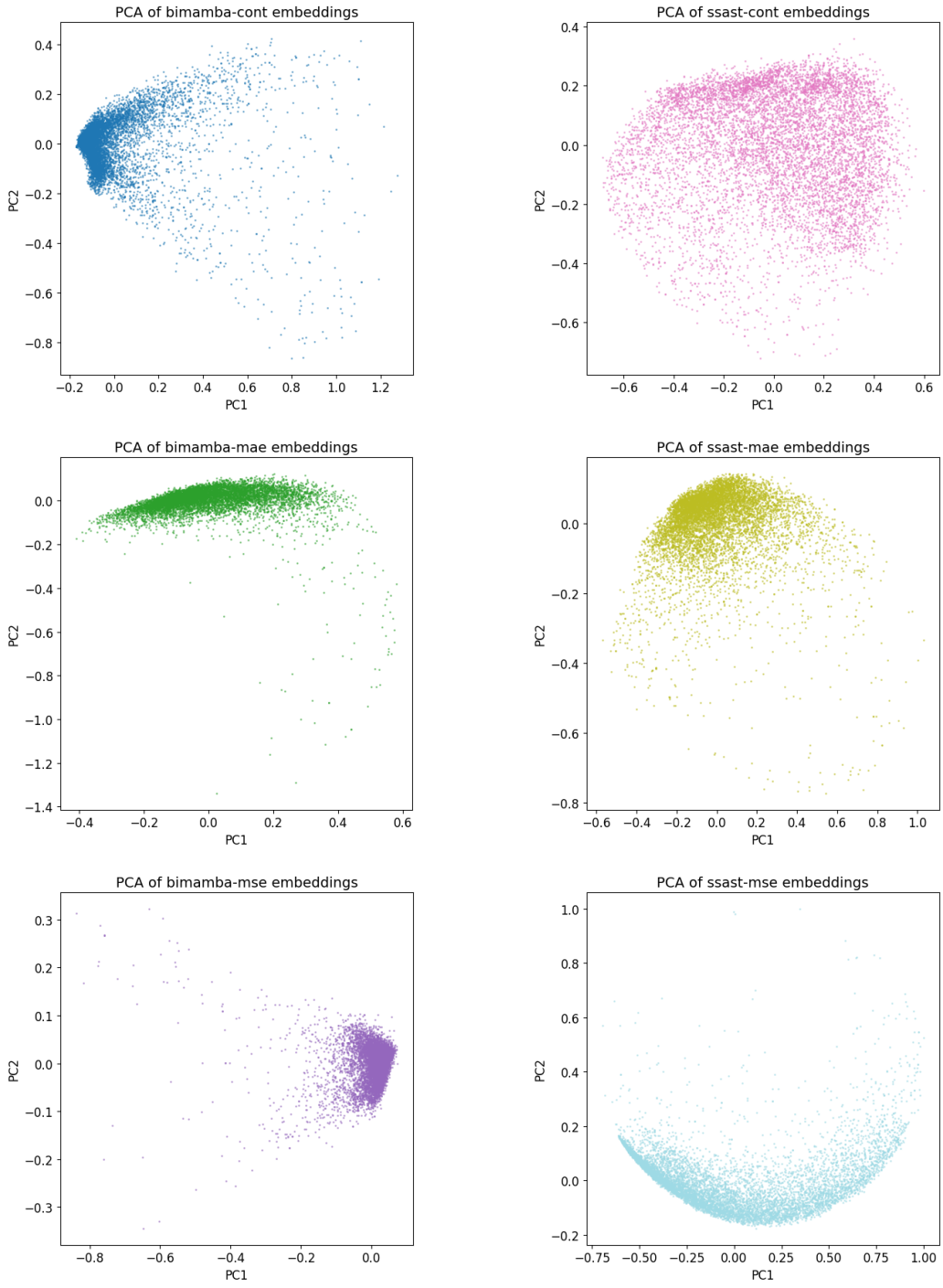


Figure 9.8: Visualisation of the first two PCA components of the embeddings per model.

the total variance in the model's embeddings lies along an one-dimensional manifold on the unit sphere, while flatter distributions indicate that the embeddings cover a higher-dimensional region. Among the Mamba family, `bimamba-mse` has the smallest PC1 value with respect to its PC2 and PC3 values among the Mamba family, indicating higher-dimensional structure within its tightly clustered embeddings. The SSAST models show the largest variations in structure across the pre-training techniques, with the `ssast-cont` model achieving the lowest drop in variance among the first three components and the `ssast-mse` model showing an almost one-dimensional structure, with the highest PC1 value. The latter observation is consistent with the corresponding PCA plot, in which the `ssast-mse` embeddings form an almost perfect circular arc with little spread in the PC2 direction.

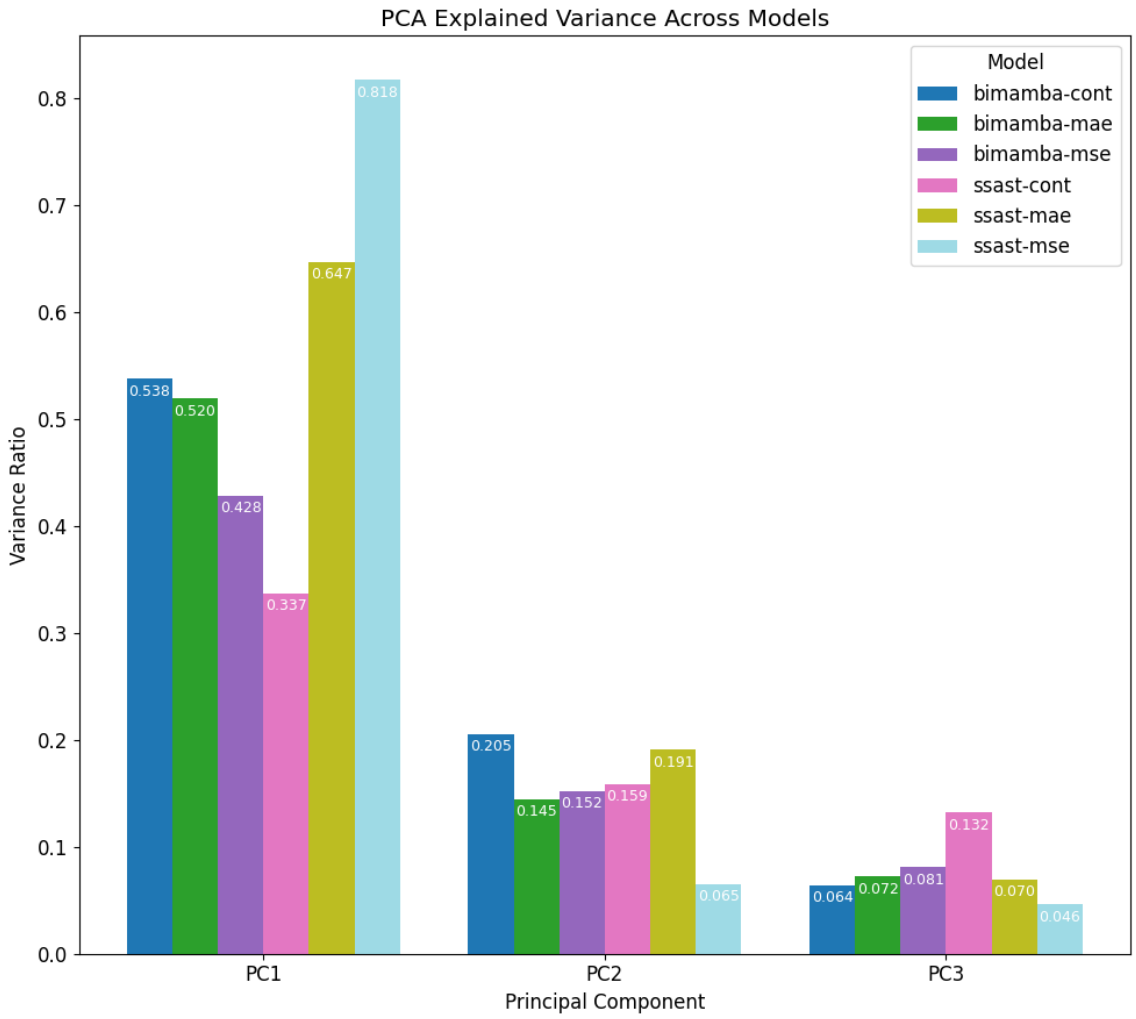


Figure 9.9: Variance explained by the first 3 principal components for all pre-trained models.

9.5.2 Bird detection

We evaluate the performance of the pre-trained and fine-tuned models on the bird detection task using the standard accuracy, precision, recall and F1 score metrics. Confidence intervals at a level of 95% are computed using the bootstrap method, which involves resampling the test set 1,000 times with replacement and computing the metrics for each resampled set. The results are shown in Table 9.5. The results suggest `ssast-mae` as the best performing model overall, achieving the highest accuracy, recall and F1 score. We observe that the Mamba models generally perform worse than their corresponding SSAST models for each pre-training technique and that the masked MAE pre-training generally outperforms the other pre-training techniques in terms of accuracy and F1 score, with the MSE and contrastive models following in that order within their respective model types. The `ssast-cont` model achieves the highest precision, but at the cost of lower recall and F1 score, indicating that it is the most conservative in its predictions.

Model	Accuracy	Precision	Recall	F1 Score
`bimamba-cont`	0.823 (0.821, 0.825)	0.759 (0.754, 0.764)	0.699 (0.694, 0.704)	0.728 (0.724, 0.732)
`bimamba-mae`	0.884 (0.882, 0.886)	0.783 (0.779, 0.787)	0.908 (0.905, 0.911)	0.841 (0.838, 0.844)
`bimamba-mse`	0.875 (0.872, 0.877)	0.765 (0.761, 0.769)	0.910 (0.907, 0.913)	0.831 (0.828, 0.834)
`ssast-cont`	0.845 (0.843, 0.847)	**0.863 (0.859, 0.867)**	0.645 (0.640, 0.650)	0.738 (0.734, 0.742)
`ssast-mae`	**0.885 (0.883, 0.887)**	0.778 (0.774, 0.782)	**0.926 (0.924, 0.929)**	**0.846 (0.843, 0.849)**
`ssast-mse`	0.880 (0.878, 0.882)	0.769 (0.764, 0.773)	0.922 (0.919, 0.925)	0.838 (0.836, 0.841)

Table 9.5: Performance metrics with 95% confidence intervals.

9.5.3 Temporal metadata

We evaluate the performance of the fine-tuned models on the time of day prediction task by converting the normalised output of the MLP head to an angle in radians, normalising it along with the true labels from $[0, 2\pi)$ to $[0, 1)$ and computing the mean absolute error and standard deviation between the predicted and true values. The results are shown in Table 9.6. We observe that the `ssast-mae` model achieves the lowest mean absolute error, along with the lowest standard deviation.

We are interested in the distribution of the predicted values over the course of the full 24 hours. Figure 9.10 shows the predicted versus true distributions for all pre-trained models. We find that all models apart from `bimamba-mae` and `ssast-cont` produce bimodal distributions, with a large peak around the mean of the true distribution and a smaller peak in the nighttime hours, almost directly opposite to the first peak. We hypothesise that the models struggle to learn the true distribution and collapse the task to almost a binary classification problem, with one class corresponding to "active" hours, which are predicted near the mean of the true distribution and another class corresponding to "inactive" hours, which are predicted during night-time. Night-time being the easiest direction to learn apart from the mean is expected, as presumably those samples are more quiet and contain mostly background outdoor sounds, while the daytime samples are significantly more diverse. In contrast, the `bimamba-mae` model output distribution lacks a night-time peak, although it appears to be slightly more spread out than that of the other models.

Model	Mean Absolute Error	Absolute Error STD
`bimamba-cont`	0.322	0.225
`bimamba-mae`	0.381	0.259
`bimamba-mse`	0.284	0.219
`ssast-cont`	0.304	0.250
`ssast-mae`	**0.257**	0.204
`ssast-mse`	0.278	0.217

Table 9.6: Time of day prediction mean and standard deviation of the absolute error by model.

In contrast to the above, the `ssast-cont` model produces a distribution that is much closer to the true values, making predictions over the entire 24-hour period.

While predicting the exact time of day is an almost impossible task, we are interested in the ability of the models to make predictions that indicate insights about the the acoustic patterns that may appear throughout a 24-hour period. For example, we expect that the night hours will be the easiest to recognise, while the morning hours might often be confused with the afternoon hours, as they are both characterised by increased human and animal activity. To this end, we bin the true and predicted values into 6-hour intervals as follows:

- 00:00 - 06:00: Night,
- 06:00 - 12:00: Morning,
- 12:00 - 18:00: Afternoon,
- 18:00 - 00:00: Evening.

We then compute the confusion matrix for the true and predicted values, which is shown for each model in Figure 9.11. As expected by the distribution plots, most models achieve high recall for the afternoon and night hours, since most of the predictions are concentrated in their quadrants. The `bimamba-mae` model on the other hand predicts the afternoon hours almost exclusively. The `ssast-cont` model presents the most interesting predictions. Indeed, most of the misclassified samples for the night hours are predicted as morning hours, which is entirely reasonable considering the similar acoustic patterns of the night with early morning. Similarly, most morning hours are correctly classified, with a smaller percentage of the morning samples being misclassified as afternoon hours and vice versa, which is also reasonable given that these are the most active hours of the day. Finally, the predictions for the evening hours are relatively uniform over all intervals, with their majority being misclassified as afternoon hours. This may be attributed once again to the model predicting the mean when it is uncertain, as the evening hours are typically characterised by a gradual decrease in activity which may not necessarily be as distinguishable as the most active or most quiet hours of the day.

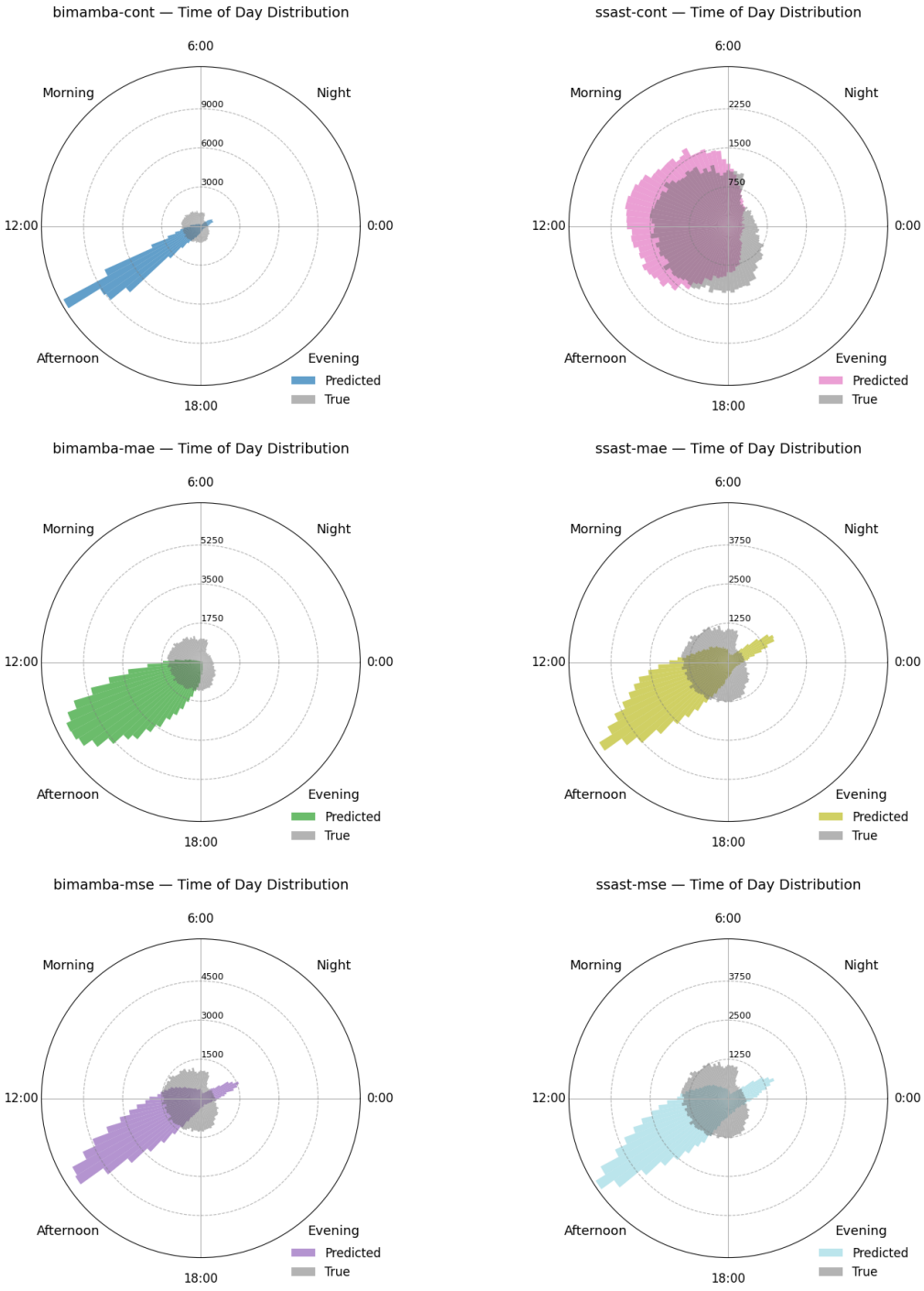


Figure 9.10: Predicted versus true precipitation rate distributions for all pre-trained models.

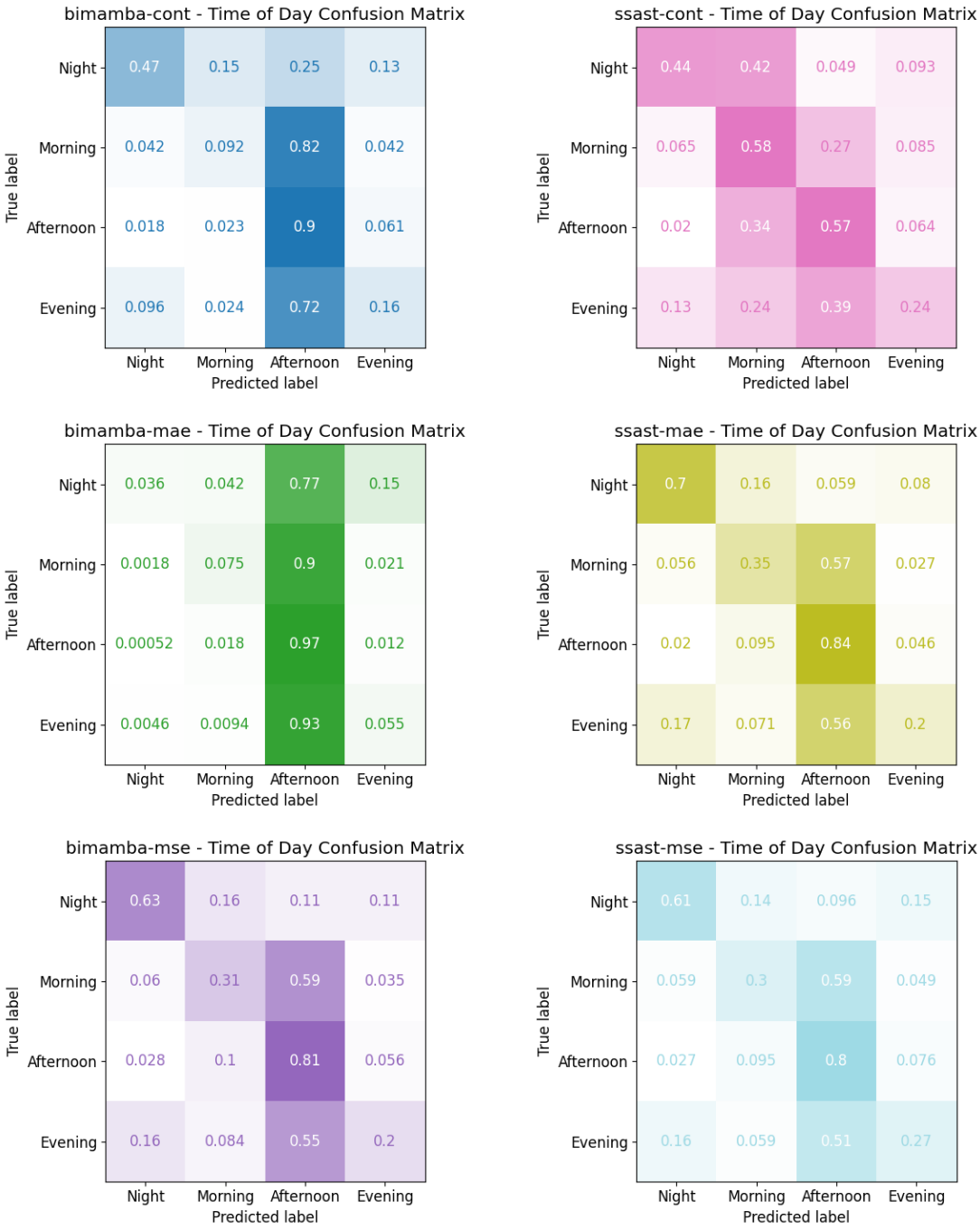


Figure 9.11: Confusion matrices for the true and predicted time of day values for all pre-trained models.

9.5.4 Weather metadata

We evaluate the performance of the fine-tuned models qualitatively by plotting the predicted versus true weather condition distributions, as well as quantitatively by computing the mean absolute error and standard deviation of the absolute error for each model. The results for the precipitation rate are shown in Figure 9.12 and Table 9.7, respectively, while the results for the average wind speed are shown in Figure 9.13 and Table 9.8 respectively.

Regarding the precipitation rate, we observe that most models produce distributions that resemble the true distribution towards 0, which is expected given the skewness of true values. However, they fail to capture small peaks that appear for higher values. The `ssast-cont` model on the other hand produces a more balanced distribution, with a smaller peak near 0 and a smoother drop towards higher values, better capturing events of increased precipitation at the loss of accurate prediction near 0. The lowest overall mean absolute error is achieved by the `ssast-mae` model, which also has the lowest standard deviation of the absolute error, indicating that it produces more consistent predictions across samples.

In contrast, most models produce an almost bell-shaped distribution of average wind speed that is centred around the median of the true distribution, failing to match the true distribution's skewness towards 0. Once again, the `ssast-cont` model stands out as its predictions match the true distribution quite well. The `bimamba-mae` model achieves the lowest mean absolute error and the second lowest standard deviation, indicating relatively accurate predictions overall.

Model	Mean Absolute Error	Absolute Error STD
`bimamba-cont`	0.193	0.282
`bimamba-mae`	0.113	0.25
`bimamba-mse`	0.121	0.25
`ssast-cont`	0.546	0.485
`ssast-mae`	**0.109**	0.249
`ssast-mse`	0.116	0.254

Table 9.7: Precipitation rate mean and standard deviation of the absolute error by model.

Finally, we are interested not only on the aggregated error and the shape of the predictions, but also on the distribution of the absolute error across the range of true precipitation rate and average wind speed values. For this purpose, we split the range of true values for each label into 11 equally spaced thresholds. We then calculate and plot the mean absolute error for the values that lie in the 10 intervals defined pairs of consecutive thresholds. The results for the precipitation rate and for the average wind speed are shown in Figure 9.14 and Figure 9.15 respectively. Each figure also shows the average of the binned mean absolute error across all thresholds, which differs from the overall estimations due to the uneven distribution of the true values.

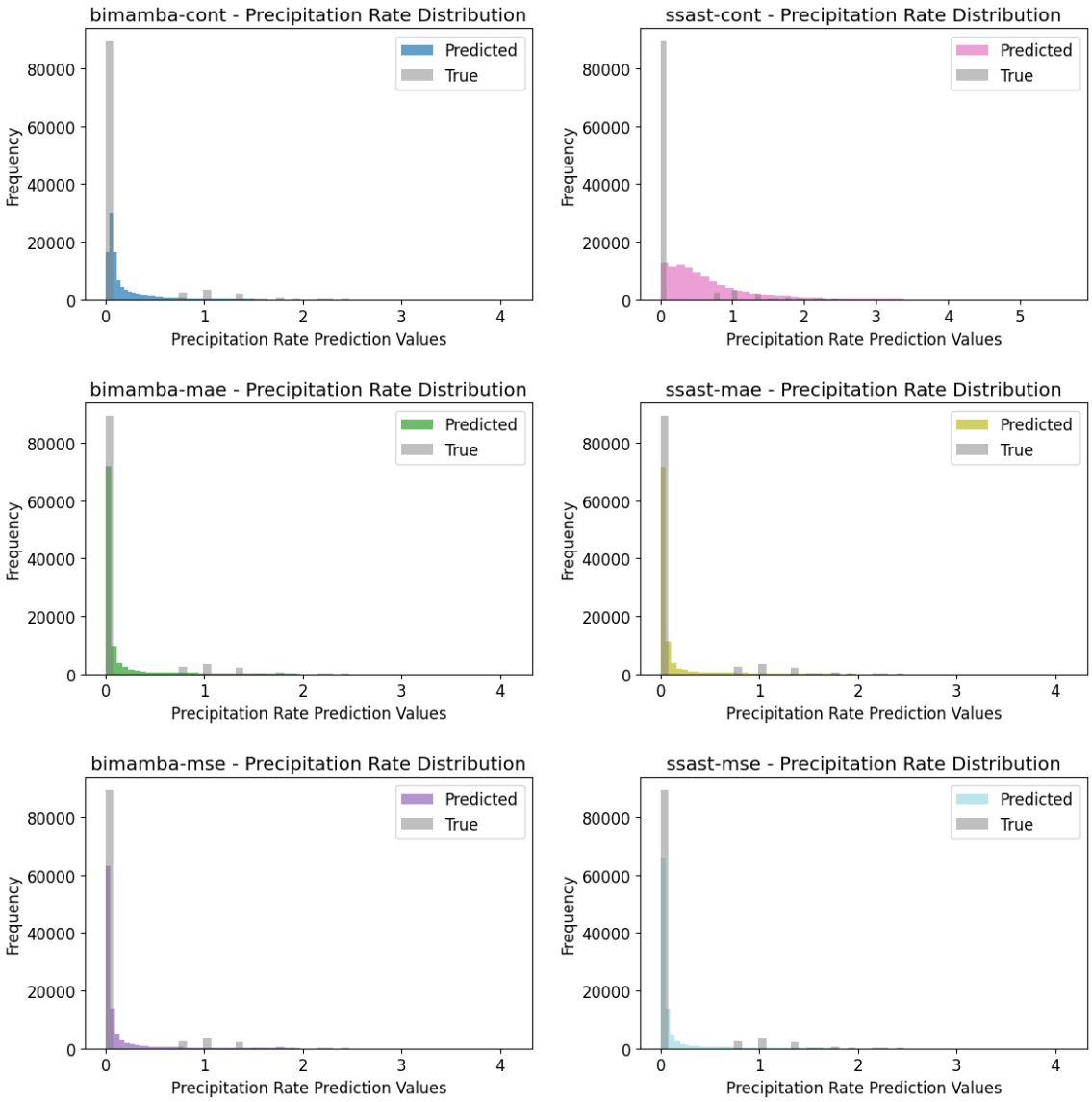


Figure 9.12: Predicted versus true precipitation rate distributions for all pre-trained models.

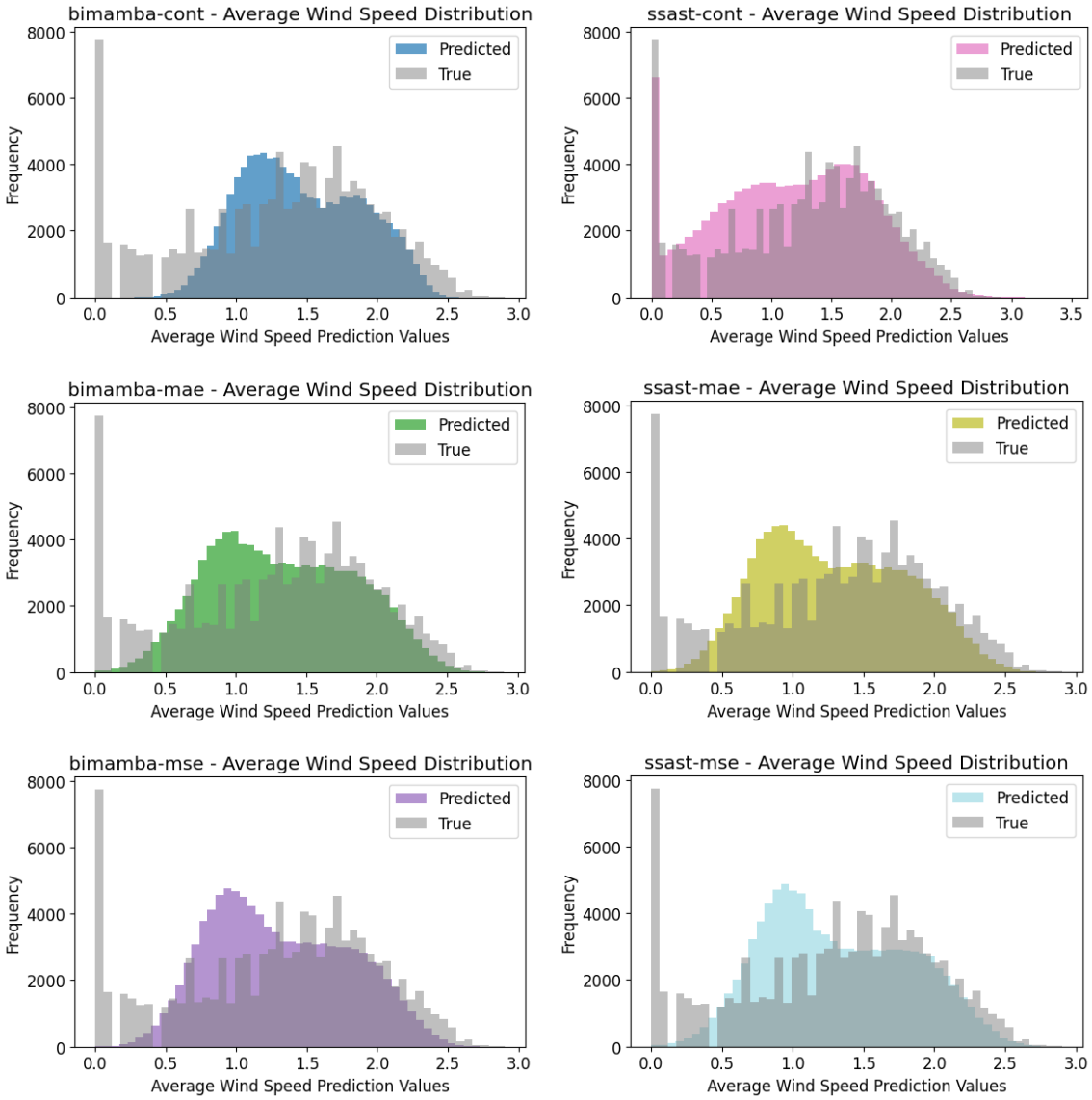


Figure 9.13: Predicted versus true average wind speed distributions for all pre-trained models.

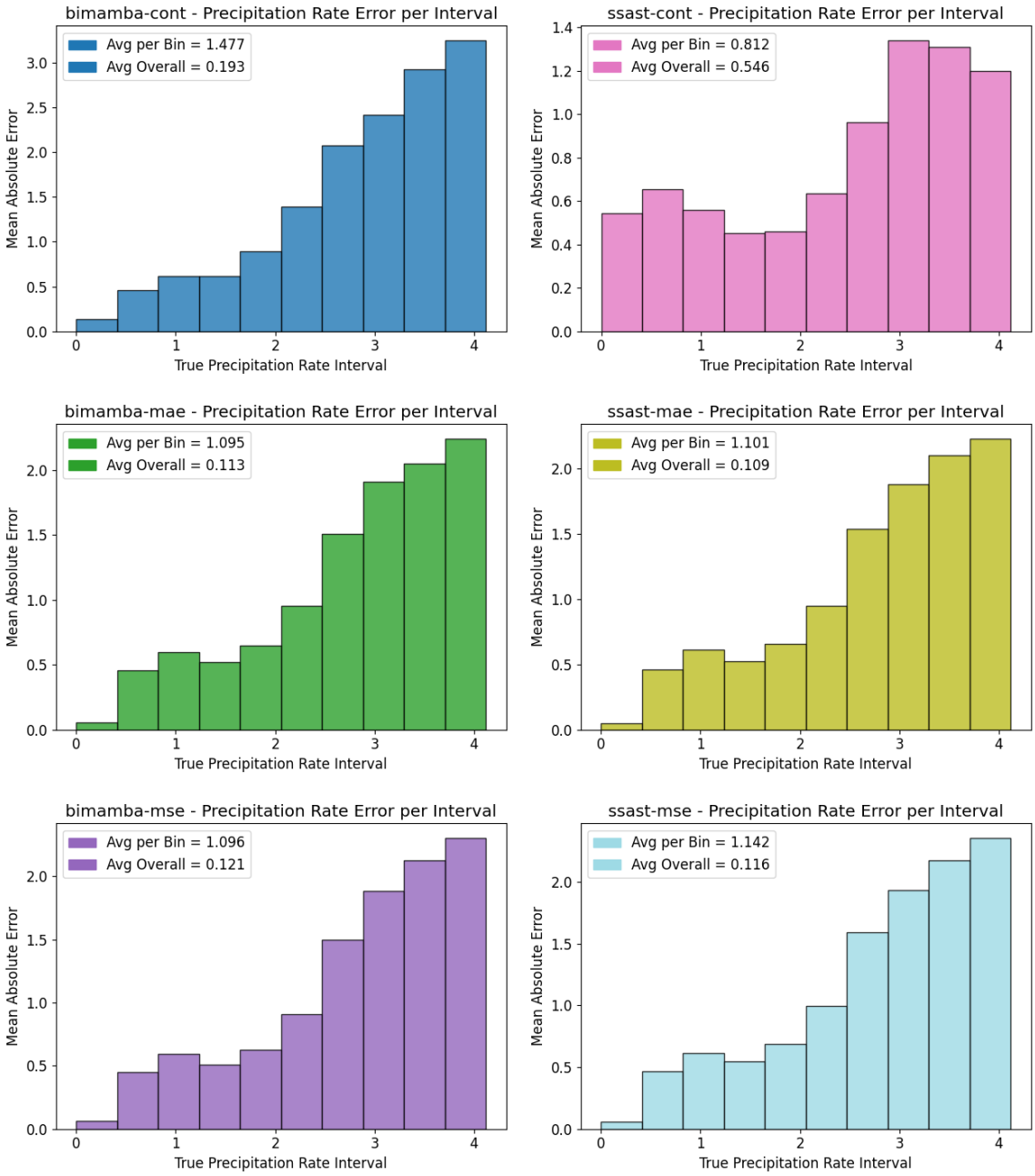


Figure 9.14: Thresholded mean absolute error for the precipitation rate for all pre-trained models.

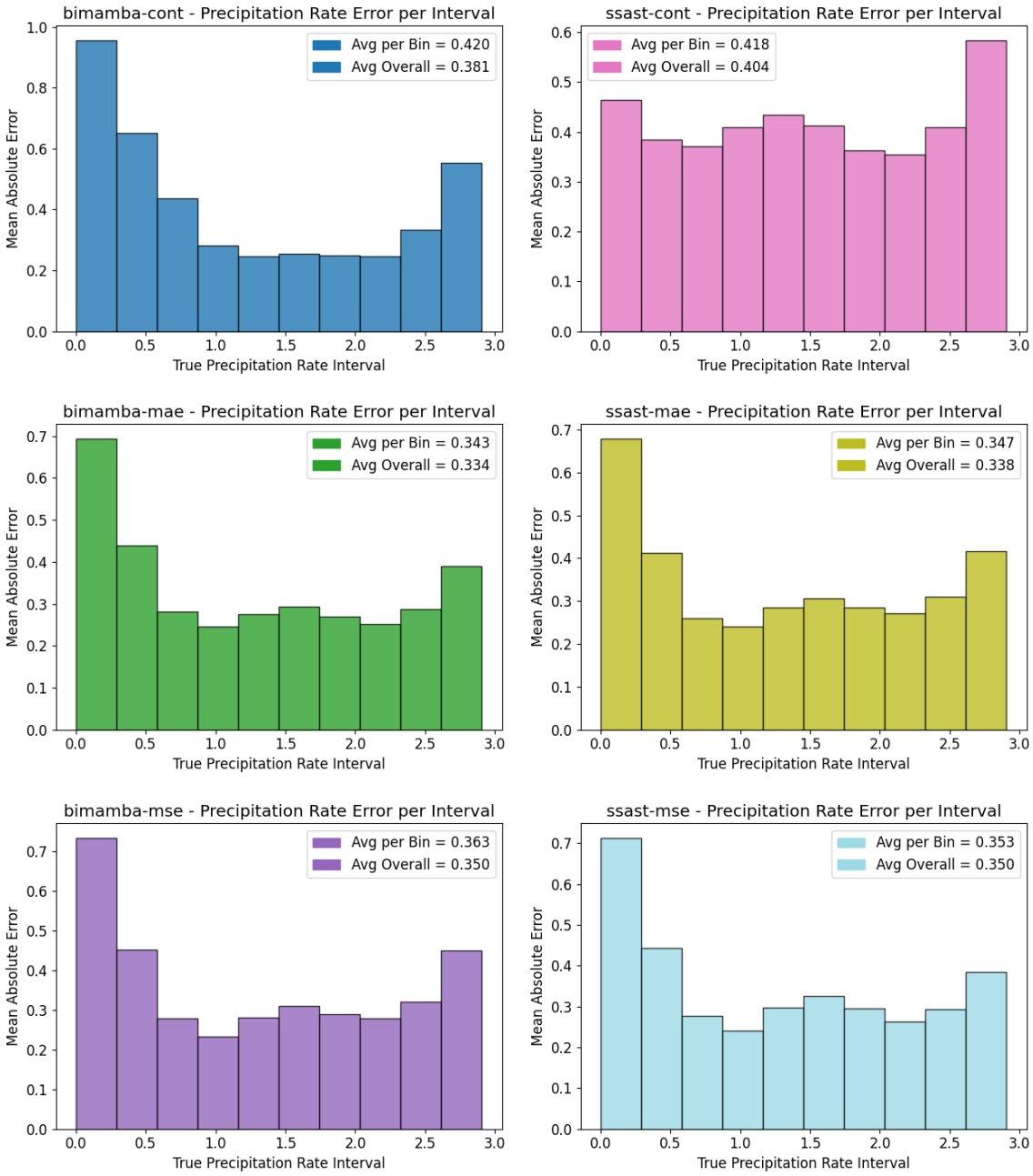


Figure 9.15: Thresholded mean absolute error for the average wind speed for all pre-trained models.

Model	Mean Absolute Error	Absolute Error STD
bimamba-cont	0.381	0.318
bimamba-mae	**0.334**	0.268
bimamba-mse	0.35	0.273
ssast-cont	0.404	0.326
ssast-mae	0.338	0.263
ssast-mse	0.35	0.274

Table 9.8: Average wind speed mean and standard deviation of the absolute error by model.

Chapter 10

Discussion

In this chapter we reflect on the results of our experiments, relating them to the research questions and contributions outlined in Chapter 1. We also discuss the limitations of our work.

Effects of the pre-training tasks on performance (RQ1). Our results show that masked reconstruction and stereo contrastive learning are both valid techniques for self-supervised pre-training of foundation models on environmental audio data, each of which presents different advantages and disadvantages. Our qualitative analysis of the learned embeddings shows that the contrastive pre-training task allows the models to learn features that cover a wide range of the unit sphere in the latent space, achieving lower uniformity than masked reconstruction, at the expense of worse alignment between similar samples. This is likely due to the fact that contrastive learning directly encourages the model to learn features that are discriminative between different audio samples. The importance the model places on discriminating distinct samples versus aligning similar samples is controlled by the temperature hyperparameter of the InfoNCE loss. Recall that the InfoNCE loss is defined as follows for a single positive pair (z_i, z_j):

$$\mathcal{L}_{i,j} = -\log\left(\frac{\exp\left(\text{sim}\left(z_i, z_j\right)/\tau\right)}{\sum_{k=1}^{2N} \mathbb{I}_{[k\neq i]} \exp\left(\text{sim}\left(z_i, z_k\right)/\tau\right)}\right).$$

For small values of τ, any positive similarity between the positive pair (z_i, z_j) is greatly amplified, leading to a large numerator in the calculated loss after exponentiation is applied. On the other hand, any negative similarity between the negative pairs is also amplified, leading to large negative values that are converted to small values in the denominator. This results in a probability distribution over the pairs which is highly peaked around the positive pair, which typically pushes positive pairs to be very close to each other in the latent space (low alignment). On the other hand, this practice may sometimes result in complete collapse of the latent space, where all samples are mapped to very similar representations (high uniformity). In our experiments we use a particularly small value of $\tau = 0.01$ compared to more common practices which set the temperature between 0.1 and 1 and yet we observe the opposite effect (high alignment, low uniformity). We attribute this to the complementary masking strategy used in the contrastive task, which forces the model to align half of the left channel of the stereo recording with the other half of the right channel, with

the masked tokens being replaced by random noise. This task might be particularly difficult for the models and thus such low values of τ may be required for effective training. This observation is supported by empirical evidence collected during early experimentation, in which higher temperature values were tried with less success. Note that small temperature values are not unheard of in particularly difficult contrastive tasks. For example, the CLIP model, which attempts to learn a joint embedding space for images and associated captions in a contrastive manner, applies a learned temperature parameter which is initialised at $\tau = 0.07$ [61].

The difficulty of the contrastive task was managed differently by the SSM and the Transformer models. While the `ssast-cont` model underperformed in the bird detection task, it surpassed all the other models in the temporal and weather condition prediction tasks. On the other hand, the `bimamba-cont` model consistently reached the worst performance in all tasks, while also exhibiting instability during fine-tuning. While it can not be ruled out that better performance could be achieved by careful hyper-parameter tuning, the rest of the models consistently achieved better performance without such extensive search.

The variance in performance of the `ssast-cont` model across the three tasks is of particular interest. Indeed, the model achieved the best performance in the temporal and weather condition prediction tasks and the worst overall recall in the bird detection task. Label quality cannot be ruled out entirely as a factor for this performance difference, as the bird detection labels were produced by other pre-trained models which are not fine-tuned for the local environment and at no point were the original labels or the labels produced as explained in Chapter 7 manually verified. In contrast, the other metadata were collected using precise time and environmental measurements. However we do not consider possible label discrepancies to be sufficient for explaining these results. We further hypothesise that the contrastive task allowed the model to learn particularly low level correlations between the spectra, which may be essential for difficult tasks such as accurate weather prediction or relation between the spectra and the phases of the day. On the other hand, bird detection relies on more high-level features, such as the pitch of the frequencies, the presence of harmonics and the overall time-frequency patterns present in the spectra. We expect that masked reconstruction models are more likely to learn such high-level features, since reconstructing those first is the best way to minimise the reconstruction loss. Moreover, the fact that the contrastive dataset consists of spectrograms generated from the original 48,000 Hz audio recordings, while the masked reconstruction dataset is generated from resampled 24,000 Hz recordings may also have played a role in allowing the `ssast-cont` model to learn more complex features. Finally, we note that the large spread of embeddings in the latent space may allow the model to explore more options during fine-tuning, thus avoiding the collapse to the mean that regression models often suffer from, including in our experiments.

Regarding the masked reconstruction task, we observe that using the MAE loss generally leads to slower training but better performance than using the MSE loss when compared within models of the same architecture, with the exception of the `bimamba-mae` model on the temporal prediction task. This is demonstrated by the generally better performance of the `bimamba-mae` and `ssast-mae` models in the bird detection task, in which the latter achieved the best performance overall. Moreover, the thresholded absolute error plots in Figures 9.14 and 9.15 show that the MAE-trained models generally achieve smaller MAE when taking the average across all intervals. We attribute this to the fact that the MAE is more robust to outliers as it is not dominated by high-energy pixels, pushing the models to learn better reconstructions throughout the entire spectra, which translates to finer feature extraction.

Comparison of SSMs and Transformers (RQ2). Our results show that the SSM-based models generally achieve comparable but consistently worse performance against the Transformer-based models, when compared within the same pre-training task. This is particularly true for the `bimamba-cont` model, which struggled to converge during most fine-tuning experiments and typically reached higher losses and worse performance metrics when it did. We attribute this to the fact that the core SSM mechanism is by design more suited for remembering long-range dependencies in sequential data, while the attention mechanism is generally better at discovering complex relationships between different parts of the input concurrently. It is entirely possible that using finer time-step and frequency bin resolutions could produce different results, making the SSMs more competitive with the Transformers.

Due to the training setup it was not practical to accurately compare the two architectures in terms of training and inference time, as the models were trained and evaluated on various machines using shared resources, the availability of which greatly influenced the processing time. In this setup in particular, I/O and CPU use were the main bottlenecks, completely dominating the actual GPU processing time. However, over multiple experiments we empirically observed that the SSM-based models were consistently slower than the Transformer architectures when both models were trained using hardware-aware optimisations. This observation does not contradict the asymptotic complexity analysis of the two architectures. Indeed, depending on the hidden constants in the complexity estimation, it is entirely possible that a linear time algorithm is slower than a quadratic time algorithm for small sequence lengths, as is the case in our experiments. However, we expect that for longer sequences the SSMs will eventually outperform the Transformers.

Downstream tasks and evaluation (RQ3). In this work we evaluated the learned representations on three varied downstream tasks: bird detection, temporal prediction and weather condition prediction. The first task is a traditional task in environmental audio processing, which has also been extensively researched on its own, while the latter two are novel tasks that we introduced to evaluate the models' ability to extract complex and informative features in a self-supervised manner. Comparison across these tasks demonstrated the different strengths and weaknesses of our chosen pre-training tasks and model architectures. In particular, the bird detection task provides a good indication of the models' ability to extract high-level features, while also providing a metric that is easy to interpret and compare with existing work. On the other hand, the temporal and weather condition prediction tasks are more complex, as they require the models to learn correlations between the spectra and the time of day or weather conditions, which may not be directly related to the audio content. The results of these tasks suggest contrastive learning as an effective pre-training task for learning low-level features which may facilitate more intricate analysis of the weather conditions or the changes in the environment throughout the day and possibly through longer periods of time. Moreover, extracting such metadata is often more practical than manually annotating the data, as it can be performed using a relatively simple passive acoustic monitoring setup, accompanied by a weather station and a clock.

Limitations. While this work attempts to advance the foundation model approaches for environmental audio analysis, several limitations should be noted.

- The GardenFiles23 dataset originates from a single location and thus is highly biased towards the acoustic characteristics of an urban garden in the Netherlands. As a result, generalisation of the models to other environments is not guaranteed. Moreover, no testing was performed

on other datasets.

- While the BirdNet and AST models achieve state-of-the-art performance, the quality of the bird detection labels is not guaranteed and no manual verification was performed.
- Our work focuses on simple pre-training tasks in order to establish a baseline. However, more complex pre-training tasks are often used in the literature. For example, the typical SSAST pre-training pipeline involves a combination of masked reconstruction with contrastive learning.
- Our fine-tuning tasks are also simplified. In particular, bird detection is only treated as a simple binary classification problem, while modern research typically focuses on more complex problems such as bird species classification. Moreover, the temporal and weather condition prediction tasks are treated as regression problems during training. In the case of the former, this ignores the fact that many parts of the day may be indistinguishable from each other. For example, anthropogenic noise may be present throughout the day, peaking in the morning and evening, while the weather conditions may not change significantly throughout the day. In the case of the latter, learning to predict the exact weather conditions from the acoustic data may not be practical, as these events may present differently in the spectrograms depending on wind direction or the presence of other sounds such as birds or human activity.
- The model comparison is limited to short sequence lengths and small models due to hardware and time constraints. Further experimentation with various lengths and model sizes is required to perform a more thorough comparison of the two architectures.

Chapter 11

Future Work

We conclude this thesis with a short discussion of potential future work in the development of foundation models for environmental audio data analysis.

As discussed in the limitations section of Chapter 10, the simple setup we have explored in our experiments can be greatly expanded upon. In particular, we expect that the model size and small sequence lengths have had a non-negligible impact both on the overall performance of our models, as well as the comparison between Transformers and SSMs. Further work could explore different sequence lengths, sample rates and time-frequency bin sizes. Moreover, it is typical for state-of-the-art models to be larger than our architectures.

Additionally, there are many ways in which the GardenFiles23 dataset can be further explored. The associated metadata can be used not only to generate interesting downstream tasks for evaluation, but also to improve the pre-training stage itself, for example by being added as multi-modal context during the embedding extraction. Another idea that seems interesting is to use the metadata in order to produce positive and negative pairs for contrastive learning. In particular, one could treat samples recorded under the same weather condition or time window as positive pairs and samples recorded under different conditions as negative pairs. This could potentially allow for more robust and context-aware representation learning, allowing a foundation model to encode information that is essential for environmental analysis. Further work may also refine the use of stereo recordings during pre-training.

Bibliography

[1] Aki Härmä and Evgeny Nazarenko. "Bird Activities in a Residential Back Garden". English. In: *32nd European Signal Processing Conference, EUSIPCO 2024 - Proceedings*. European Signal Processing Conference. United States: IEEE, 2024, pp. 1262–1266. ISBN: 9798331519773. DOI: 10.23919/EUSIPCO63174.2024.10715147. URL: https://eusipcolyon.sciencesconf.org/.

[2] Jort F. Gemmeke et al. "Audio Set: An ontology and human-labeled dataset for audio events". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017, pp. 776–780. DOI: 10.1109/ICASSP.2017.7952261.

[3] Karol J. Piczak. "ESC: Dataset for Environmental Sound Classification". In: *Proceedings of the 23rd Annual ACM Conference on Multimedia*. Brisbane, Australia: ACM Press, Oct. 13, 2015, pp. 1015–1018. ISBN: 978-1-4503-3459-4. DOI: 10.1145/2733373.2806390. URL: http://dl.acm.org/citation.cfm?doid=2733373.2806390.

[4] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. "A Dataset and Taxonomy for Urban Sound Research". In: *Proceedings of the 22nd ACM International Conference on Multimedia*. MM '14. Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 1041–1044. ISBN: 9781450330633. DOI: 10.1145/2647868.2655045. URL: https://doi.org/10.1145/2647868.2655045.

[5] Mark Cartwright et al. "SONYC-UST-V2: An Urban Sound Tagging Dataset with Spatiotemporal Context". In: *CoRR* abs/2009.05188 (2020). arXiv: 2009.05188. URL: https://arxiv.org/abs/2009.05188.

[6] Stefan Kahl et al. "BirdNET: A deep learning solution for avian diversity monitoring". In: *Ecological Informatics* 61 (2021), p. 101236. ISSN: 1574-9541. DOI: https://doi.org/10.1016/j.ecoinf.2021.101236. URL: https://www.sciencedirect.com/science/article/pii/S1574954121000273.

[7] Burooj Ghani et al. "Global birdsong embeddings enable superior transfer learning for bioacoustic classification". In: *Scientific Reports* 13.1 (Dec. 2023). ISSN: 2045-2322. DOI: 10.1038/s41598-023-49989-z. URL: http://dx.doi.org/10.1038/s41598-023-49989-z.

[8] George Vengrovski et al. "TweetyBERT: Automated parsing of birdsong through self-supervised machine learning". In: *bioRxiv* (2025). DOI: 10.1101/2025.04.09.648029. eprint: https://www.biorxiv.org/content/early/2025/04/10/2025.04.09.648029.full.pdf. URL: https://www.biorxiv.org/content/early/2025/04/10/2025.04.09.648029.

[9] Roberta Avanzato and Francesco Beritelli. "An Innovative Acoustic Rain Gauge Based on Convolutional Neural Networks". In: *Information* 11.4 (2020). ISSN: 2078-2489. DOI: 10.3390/info11040183. URL: https://www.mdpi.com/2078-2489/11/4/183.

[10] Meizhen Wang et al. "Estimating rainfall intensity based on surveillance audio and deep-learning". In: *Environmental Science and Ecotechnology* 22 (2024), p. 100450. ISSN: 2666-4984. DOI: https://doi.org/10.1016/j.ese.2024.100450. URL: https://www.sciencedirect.com/science/article/pii/S2666498424000644.

[11] Yuan Gong, Yu-An Chung, and James Glass. "AST: Audio Spectrogram Transformer". In: *Proc. Interspeech 2021*. 2021, pp. 571–575. DOI: 10.21437/Interspeech.2021-698.

[12] Yuan Gong, Yu-An Chung, and James Glass. "PSLA: Improving Audio Tagging with Pretraining, Sampling, Labeling, and Aggregation". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* (2021). DOI: 10.1109/TASLP.2021.3120633.

[13] Yuan Gong et al. "SSAST: Self-Supervised Audio Spectrogram Transformer". In: *CoRR* abs/2110.09784 (2021). arXiv: 2110.09784. URL: https://arxiv.org/abs/2110.09784.

[14] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

[15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[16] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[17] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[18] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[19] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV]. URL: https://arxiv.org/abs/2304.02643.

[20] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *CoRR* abs/2002.05709 (2020). arXiv: 2002.05709. URL: https://arxiv.org/abs/2002.05709.

[21] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation Learning with Contrastive Predictive Coding". In: *CoRR* abs/1807.03748 (2018). arXiv: 1807.03748. URL: http://arxiv.org/abs/1807.03748.

[22] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[23] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[24] Maxime Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. arXiv: 2304.07193 [cs.CV]. URL: https://arxiv.org/abs/2304.07193.

[25] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[26] R. Gray. “Vector quantization”. In: *IEEE ASSP Magazine* 1.2 (1984), pp. 4–29. DOI: 10.1109/MASSP.1984.1162229.

[27] Herve Jégou, Matthijs Douze, and Cordelia Schmid. “Product Quantization for Nearest Neighbor Search”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (2011), pp. 117–128. DOI: 10.1109/TPAMI.2010.57.

[28] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. “Neural Discrete Representation Learning”. In: *CoRR* abs/1711.00937 (2017). arXiv: 1711.00937. URL: http://arxiv.org/abs/1711.00937.

[29] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. “Self-Attention with Relative Position Representations”. In: *CoRR* abs/1803.02155 (2018). arXiv: 1803.02155. URL: http://arxiv.org/abs/1803.02155.

[30] Albert Gu and Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. 2024. arXiv: 2312.00752 [cs.LG]. URL: https://arxiv.org/abs/2312.00752.

[31] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. 2022. arXiv: 2205.14135 [cs.LG]. URL: https://arxiv.org/abs/2205.14135.

[32] Albert Gu et al. *Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers*. 2021. arXiv: 2110.13985 [cs.LG]. URL: https://arxiv.org/abs/2110.13985.

[33] Albert Gu et al. *HiPPO: Recurrent Memory with Optimal Polynomial Projections*. 2020. arXiv: 2008.07669 [cs.LG]. URL: https://arxiv.org/abs/2008.07669.

[34] Victor Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. New York: Springer Science & Business Media, 2001. ISBN: 978-0-8176-4204-5.

[35] Albert Gu, Karan Goel, and Christopher Ré. “Efficiently Modeling Long Sequences with Structured State Spaces”. In: *CoRR* abs/2111.00396 (2021). arXiv: 2111.00396. URL: https://arxiv.org/abs/2111.00396.

[36] Ankit Gupta, Albert Gu, and Jonathan Berant. *Diagonal State Spaces are as Effective as Structured State Spaces*. 2022. arXiv: 2203.14343 [cs.LG]. URL: https://arxiv.org/abs/2203.14343.

[37] Albert Gu et al. *On the Parameterization and Initialization of Diagonal State Space Models*. 2022. arXiv: 2206.11893 [cs.LG]. URL: https://arxiv.org/abs/2206.11893.

[38] Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. *Simplified State Space Layers for Sequence Modeling*. 2023. arXiv: 2208.04933 [cs.LG]. URL: https://arxiv.org/abs/2208.04933.

[39] Guy E. Blelloch. *Prefix Sums and Their Applications*. Technical Report. Pittsburgh, PA, USA, 1990.

[40] Sander Dieleman and Benjamin Schrauwen. “End-to-end learning for music audio”. In: May 2014, pp. 6964–6968. ISBN: 978-1-4799-2893-4. DOI: 10.1109/ICASSP.2014.6854950.

[41] Tara Sainath et al. “Learning the Speech Front-end with Raw Waveform CLDNNs”. In: *Interspeech*. 2015.

[42] Wei Dai et al. “Very Deep Convolutional Neural Networks for Raw Waveforms”. In: *CoRR* abs/1610.00087 (2016). arXiv: 1610.00087. URL: http://arxiv.org/abs/1610.00087.

[43] Aäron van den Oord et al. “WaveNet: A Generative Model for Raw Audio”. In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[44] Steffen Schneider et al. “wav2vec: Unsupervised Pre-training for Speech Recognition”. In: *CoRR* abs/1904.05862 (2019). arXiv: 1904.05862. URL: http://arxiv.org/abs/1904.05862.

[45] Alexei Baevski et al. “wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations”. In: *CoRR* abs/2006.11477 (2020). arXiv: 2006.11477. URL: https://arxiv.org/abs/2006.11477.

[46] Wei-Ning Hsu et al. “HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units”. In: *CoRR* abs/2106.07447 (2021). arXiv: 2106.07447. URL: https://arxiv.org/abs/2106.07447.

[47] Kannan Venkataramanan and Haresh Rengaraj Rajamohan. “Emotion Recognition from Speech”. In: *CoRR* abs/1912.10458 (2019). arXiv: 1912.10458. URL: http://arxiv.org/abs/1912.10458.

[48] Friedrich Wolf-Monheim. *Spectral and Rhythm Features for Audio Classification with Deep Convolutional Neural Networks*. 2024. arXiv: 2410.06927 [cs.SD]. URL: https://arxiv.org/abs/2410.06927.

[49] Hugo Touvron et al. “Training data-efficient image transformers & distillation through attention”. In: *CoRR* abs/2012.12877 (2020). arXiv: 2012.12877. URL: https://arxiv.org/abs/2012.12877.

[50] Sarthak Yadav and Zheng-Hua Tan. *Audio Mamba: Selective State Spaces for Self-Supervised Audio Representations*. 2024. arXiv: 2406.02178 [cs.SD]. URL: https://arxiv.org/abs/2406.02178.

[51] Yue Liu et al. *VMamba: Visual State Space Model*. 2024. arXiv: 2401.10166 [cs.CV]. URL: https://arxiv.org/abs/2401.10166.

[52] Jiaju Lin and Haoxuan Hu. *Audio Mamba: Pretrained Audio State Space Model For Audio Tagging*. 2024. arXiv: 2405.13636 [cs.SD]. URL: https://arxiv.org/abs/2405.13636.

[53] Mehmet Hamza Erol et al. *Audio Mamba: Bidirectional State Space Model for Audio Representation Learning*. 2024. arXiv: 2406.03344 [cs.SD]. URL: https://arxiv.org/abs/2406.03344.

[54] Kai Li et al. *SPMamba: State-space model is all you need in speech separation*. 2024. arXiv: 2404.02063 [cs.SD]. URL: https://arxiv.org/abs/2404.02063.

[55] Siavash Shams et al. “SSAMBA: Self-Supervised Audio Representation Learning With Mamba State Space Model”. In: *2024 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, Dec. 2024, pp. 1053–1059. DOI: 10.1109/slt61566.2024.10832304. URL: http://dx.doi.org/10.1109/SLT61566.2024.10832304.

[56] Aki Härmä. *GardenFiles23*. Version V2. 2024. DOI: 10.34894/HPLUCH. URL: https://doi.org/10.34894/HPLUCH.

[57] Ross Wightman. *PyTorch Image Models*. https://github.com/rwightman/pytorch-image-models. 2019. DOI: 10.5281/zenodo.4414861.

[58] Lianghui Zhu et al. “Vision Mamba: Efficient Visual Representation Learning with Bidirectional State Space Model”. In: *Forty-first International Conference on Machine Learning*.

[59] Ilya Loshchilov and Frank Hutter. “Fixing Weight Decay Regularization in Adam”. In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.

[60] Leslie N. Smith and Nicholay Topin. “Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates”. In: *CoRR* abs/1708.07120 (2017). arXiv: 1708.07120. URL: http://arxiv.org/abs/1708.07120.

[61] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *CoRR* abs/2103.00020 (2021). arXiv: 2103.00020. URL: https://arxiv.org/abs/2103.00020.