UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques
i Informàtica

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Treball final de grau

F1 RACE SIMULATOR

Autor: David Díez Vidueira

Director: Dr. Santiago Seguí Mesquida
Realitzat a: Departament de Matemàtiques
i Informàtica
Barcelona, June 3, 2025

Faculty of Mathematics and Computer Science
University of Barcelona
Barcelona on June 3, 2025

To all the artists on Spotify,
singing to me 24/7.

Contents

Abstract

This thesis presents the design and implementation of a real-time Formula 1 strategy simulator, built on state-of-the-art Transformer architectures to enable dynamic, data-driven decision-making during a Grand Prix. The project explores how Transformer models, originally developed for language processing, can be adapted to predict and optimise race strategies using sequential motorsport data.

The simulator relies on two specialised models: the `PitStopTransformer`, which predicts the optimal lap to pit, and the `CompoundTransformer`, which selects the most appropriate tyre compound. Both models are based on the Transformer architecture, incorporating multi-head attention, positional encoding and feed-forward layers to capture complex temporal patterns and race dynamics.

Data is sourced from Fast F1 for historical records and Open F1 for real-time telemetry. Lap-by-lap features such as laps times, gaps, weather and strategy phase are processed through a `PostgreSQL` database and structured into sequences for `TensorFlow` pipelines.

Live deployment confirms the system's ability to generate accurate, low-latency predictions during evolving race scenarios. The simulator adapts to events like tyre degradation or Safety Cars, offering strategic insights as conditions change.

By combining mathematical rigour with cutting-edge architecture, this work delivers a scalable tool for real-time race strategy, bridging theoretical machine learning and applied motorsport analytics.

Resum

Aquesta tesi presenta el disseny i la implementació d'un simulador d'estratègia en temps real per a la Fórmula 1, construït sobre arquitectures Transformer d'última generació per permetre una presa de decisions dinàmica i basada en dades durant un Gran Premi. El projecte explora com els models Transformer, inicialment desenvolupats per al processament del llenguatge, poden adaptar-se per predir i optimitzar estratègies de cursa a partir de dades seqüencials del món del motor.

El simulador es basa en dos models especialitzats: el `PitStopTransformer`, que prediu la volta òptima per fer una parada, i el `CompoundTransformer`, que selecciona el compost de pneumàtic més adequat segons les condicions actuals. Ambdós models parteixen de l'arquitectura Transformer, incorporant mecanismes com l'atenció multi-capçal, les codificacions posicionals i les xarxes feed-forward per capturar patrons temporals complexos i la dinàmica estratègica de la cursa.

Les dades provenen de Fast F1, que ofereix registres històrics i d'Open F1, que aporta telemetria en temps real. Variables com els temps per volta i el clima s'emmagatzemen en una base de dades `PostgreSQL` i es transformen en seqüències per a `TensorFlow`.

Les proves en directe confirmen que el sistema pot generar prediccions precises i ràpides en contextos canviants. El simulador reacciona a situacions com degradació extrema o Safety Car, ajustant les decisions estratègiques en temps real amb un model sòlid i escalable.

Combinant les matemàtiques amb una arquitectura d'avantguarda, aquesta tesi ofereix una eina per a l'estratègia en temps real, a mig camí entre la teoria i la pràctica del motorsport.

Chapter 1

Introduction

Formula 1 represents the pinnacle of motorsport, where innovation, precision and competition intersect. In this arena, mere tenths of a second can separate a future Hall of Fame from a footnote in racing history. With team budgets capped at 145 million euros per season for development purposes [For21], every decision, whether technical or strategic, bears significant consequences.

The championship features ten teams, each fielding two drivers, competing across approximately 24 races held over nearly an entire calendar year. While the engineering challenge is immense, the role of race-day strategy continues to stand out as a decisive factor in overall performance.

Two distinct titles are awarded each season: the *Drivers' Championship* (WDC), based on individual driver results and the *Constructors' Championship* (WCC), awarded to the team whose drivers collectively score the most points. Although the two titles are interrelated, they are not always won simultaneously. For example, in the 2024 season, Max Verstappen clinched the WDC with Red Bull, while McLaren secured the WCC.

Formula 1 is as much a testament to human endurance as it is to technological excellence. Drivers are subjected to extreme physical forces, such as 5.6 g under braking at Monza [Soy19], conditions more severe than those experienced by astronauts. Reaction times as fast as 0.2 seconds [For23], coupled with the ability to suppress blinking at speeds exceed-

ing 350 km/h [Sci23], highlight the extraordinary physiological demands placed on them.

Strategic and engineering departments operate in parallel. Teams such as Mercedes and Ferrari employ over 1,000 professionals across a wide array of domains including aerodynamics, power-train design, logistics and race strategy. The goal is not merely to construct the fastest car in a vacuum, but to optimise performance across a full lap, tuning downforce and drag to suit the specific characteristics of each circuit.

A Grand Prix weekend is structured into four primary phases: car setup on Thursday, free practice sessions on Friday, qualifying on Saturday and the race on Sunday. Qualifying is organised into three knockout rounds: Q1, Q2 and Q3 with only the top ten drivers advancing to the final session. Sunday's race typically covers a total distance of 305 kilometres or the full lap count defined for each circuit. Since their introduction in 2021, sprint races have added another layer of complexity by altering the weekend structure and offering additional points.

Alongside the drivers, race strategists play a vital, yet often understated, role in determining race outcomes. Working in real time, they monitor hundreds of telemetry variables, simulate possible race scenarios and make critical decisions regarding pit stops, tyre selection and reaction to on-track incidents. These professionals must weigh tyre degradation, fuel load, weather evolution and the behaviour of rival teams to optimise both short-term race position and long-term championship points. In many cases, a single strategic call can be the difference between victory and defeat.

This thesis aims to simulate the role of a Formula 1 race strategist through the development of a real-time strategy engine. The proposed framework employs advanced machine learning techniques, specifically Transformer-based architectures, to predict both the optimal lap for a pit stop and the most suitable tyre compound under dynamic race conditions. Historical data is sourced from the Fast F1 API for model training, while live telemetry is obtained via the Open F1 API during inference. By combining mathematical modelling with modern deep learning methods, the resulting system aspires to become a robust, data-driven decision-support tool capable of delivering high-impact strategic insights in real time.

Chapter 2

Formula 1 strategy

Strategy is a core function within every Formula 1 team, responsible for determining when to pit and which tyres to use. These decisions must account for a range of dynamic variables, making race strategy one of the most nuanced areas in the sport.

To formalise the discussion, several key definitions are introduced

> **Definition 2.1: Pit stop.** A pit stop is a brief but crucial halt in the pit lane where a car receives service, primarily for tyre changes, adjustments or repairs, playing a crucial role in race strategy and performance.

A well-timed pit stop can offer a decisive advantage, while a poorly executed one may result in significant time loss, traffic, or a compromised race position. Including the time to enter and exit the pit lane, the total time loss is usually around 30 seconds.

> **Definition 2.2: Tyre compound.** A tyre compound refers to the rubber mixture used in an F1 tyre, which determines its grip, durability and overall performance characteristics.

Pirelli supplies all teams with both slick and wet-weather tyres. Slick tyres, used in dry conditions, are divided into five compounds (C1 to C5), with C1 being the hardest and C5 the softest [Pir25]. For each Grand Prix, three compounds are selected and labelled hard,

medium, and soft. Softer tyres provide superior grip but degrade quickly, whereas harder compounds offer endurance at the expense of outright performance.

Definition 2.3: Strategy. Strategy in Formula 1 refers to a team's race plan, including tyre choices, pit stop timing and fuel management, aimed at maximizing performance and race position.

To illustrate the essence of strategy, consider a simplified 20-lap race scenario with ideal conditions and linear tyre degradation. To further simplify, it is assumed that a lap time on this circuit with hard tyres is 110 seconds, with medium tyres is 105 seconds and with soft tyres is 100 seconds. Additionally, tyre degradation follows a linear model: the hard tyres degrade at a rate of 0.25 seconds per lap, meaning each subsequent lap is 0.25 seconds slower. The degradation rate for medium tyres is 0.75 seconds per lap, while for soft tyres it is 1.5 seconds per lap.

The total pit stop duration, denoted as t_p, is assumed to be 30 seconds. This includes the time required to enter the pit lane, change tyres and exit.

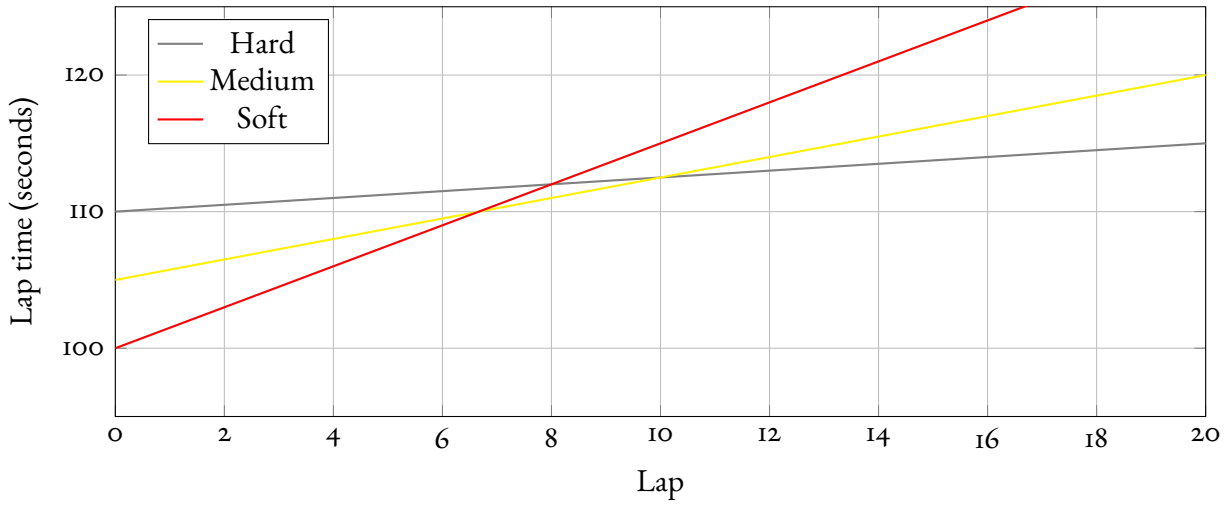

Figure 2.1: Lap times as a function of tyre compound and lap number.

In the early stages, the soft compound yields the fastest lap times. However, its steep degradation soon negates the initial advantage, allowing the medium and later the hard compound to become more effective across longer stints.

Consider a one-stop strategy in which soft tyres are used for the first six laps, followed by a switch to hard tyres. The total race time is computed as:

1. First six laps on soft tyres:

$$t_s = \sum_{l=0}^{5} (100 + 1.5l) = 600 + 1.5 \cdot \frac{6(6-1)}{2} = 622.5$$

2. Pit stop on lap 7 adds $t_p = 30$.

3. Final 14 laps on hard tyres:

$$t_h = \sum_{l=0}^{13} (110 + 0.25l) = 1540 + 0.25 \cdot \frac{14(13)}{2} = 1562.75$$

Total time is computed as $t = t_s + t_p + t_h = 622.5 + 30 + 1562.75 = 2215.25$ seconds.

Definition 2.0.1: Stint. A *stint* refers to the continuous period a driver spends on track between two pit stops while using the same set of tyres. The length of a stint is influenced by tyre degradation, race strategy and track conditions.

Now consider a second one-stop strategy, using medium tyres for the first 10 laps and softs for the final 10, this is, $t = 1083.75 + 30 + 1067.5 = 2181.25$ seconds.

This approach is 34 seconds faster. Finally, the optimal strategy in this simulation is a two-stop plan: softs for 7 laps, mediums for 6, and softs again for the last 7 laps:

$$t = 731.5 + 30 + 641.25 + 30 + 731.5 = 2164.25 \text{ seconds}$$

Although it requires two pit stops, the lower average lap time compensates for the added stationary time. This illustrates that the best race time is not necessarily achieved by minimising pit stops, but by striking the right balance between degradation and pit time.

In real Formula 1, tyre degradation is non-linear and typically accelerates over time. This is especially relevant when compounds approach their thermal or structural limits, leading to phenomena such as graining or blistering. Weather conditions, fuel loads and track evolution further complicate strategic decisions. For instance, heavier cars at the start of a race experience more degradation, favouring harder compounds early. As fuel burns off and the track rubberises, softer tyres become increasingly viable.

Rain and temperature shifts demand reactive strategy updates, particularly regarding the timing of switches between slicks and wet-weather compounds. Pit decisions must also account for traffic: rejoining behind slower cars or in dirty air can neutralise the benefit of a fresh tyre set. Additionally, neutralisation phases, such as Safety Cars or red flags, reduce pit lane time losses, often triggering opportunistic stops.

Chapter 3

Previous literature

The study of race strategy in Formula 1 has become an increasingly popular field for both academic and applied research. The intersection between motorsport, mathematics and computer science offers a fascinating environment for experimentation with optimisation algorithms, machine learning models and simulation techniques. In this context, a number of works have laid the groundwork by proposing models based on stochastic processes, Monte Carlo simulations, neural networks and reinforcement learning. While none of these fully capture the complexity and speed of real-time decision-making in F1, they each contribute valuable components for building a complete strategic simulator.

A significant portion of the literature focuses on using Monte Carlo Simulation (MCS) to evaluate thousands of possible race scenarios by simulating stochastic elements such as tyre degradation, fuel weight, pit stops and the appearance of safety cars. These methods provide solid estimations and are frequently combined with statistical validation metrics like hypothesis testing or Spearman's rank correlation [Jim22]. Some authors also introduce enhancements such as Latin Hypercube Sampling to reduce computational overhead and improve coverage across scenarios [Meh+21]. However, these models often rely on hardcoded strategies or last-year winner heuristics, which limit adaptability and novelty [Mur21].

Another widely explored avenue is the application of machine learning, especially Artificial Neural Networks (ANNs), to predict optimal strategies. These models are typically trained

on historical data to make decisions. Some works propose multi-network systems, assigning separate neural networks to each parameter to improve accuracy [Hei+20b]. Others experiment with architectures such as LSTMs to handle sequential decision-making under time dependency [LF20]. However, while useful for offline predictions, these models often fail to adapt dynamically during a live race [Ron22].

Several papers delve into Reinforcement Learning (RL) for simulating the strategic space of a race, often framing it as a Markov Decision Process (MDP). Proximal Policy Optimisation (PPO) and Advantage Actor-Critic (A2C) algorithms are used to learn policies that maximise rewards based on simulated race outcomes. These approaches show great promise, especially when trained using simplified track models, yet they tend to require a significant number of episodes and usually simulate only the early part of a race due to computational constraints [Jim23].

Other authors propose the use of Monte Carlo Tree Search (MCTS) to make branching decisions based on a simulation tree. MCTS is especially useful in dynamically changing environments like motorsport, as it allows reevaluating the best strategy based on real-time events such as the deployment of a safety car or weather changes [Pic+21]. Nevertheless, MCTS struggles with computational cost, limiting the number of simulations that can be feasibly executed in real time [Hei+20a].

What stands out in the literature is that few works address race strategy with a holistic and real-time adaptive perspective. Most approaches either focus on pre-race planning or lack the ability to adjust their predictions during the event. Furthermore, very few integrate a solid mathematical backbone with a modern software stack capable of handling data ingestion, transformation and prediction with low latency.

In light of these gaps, the use of a Transformer-based model for F1 strategy prediction introduces a novel and highly promising approach. Unlike traditional sequence models such as RNNs or LSTMs, Transformers are designed to handle long-range dependencies with greater efficiency and parallelism, making them ideal for modelling the sequential and conditional nature of race events. The self-attention mechanism enables the model to weigh different moments in the race contextually, allowing the system to reason about overtaking opportunities, tyre performance and timing in a more flexible manner.

By training the Transformer to predict the optimal lap to pit and the best tyre compound to use, it becomes possible to offer data-driven strategy suggestions both before and during the race. The model can be updated in real time as new telemetry or race conditions are fed into the system, overcoming the static nature of most previous works. Instead of simulating thousands of outcomes, the Transformer directly estimates the best action over the next lap, balancing speed with interpretability.

In summary, this thesis aims to build upon the foundations laid in previous research by combining the robustness of simulation, the adaptability of machine learning and the efficiency of Transformers. The resulting system is designed to operate entirely in real time, continuously generating and refining strategic recommendations based on live data streams throughout the race. This dual focus on mathematical rigour and software implementation reflects the interdisciplinary nature of the project, bridging the gap between academic insight and real-world applicability in motorsport engineering.

The structure of this thesis is as follows:

1. Chapter 1 provides a general introduction to the world of Formula 1. It outlines the structure of a race weekend, introduces the key stakeholders such as teams and drivers and explains how different sessions unfold and how teams typically operate.
2. Chapter 2 delves into the intricacies of race strategy in Formula 1. This section offers a comprehensive breakdown of strategic decision-making illustrated through a simple example, an analysis of the various factors that influence strategy and several real-world case studies.
3. Chapter 3 presents a review of relevant literature, including previous work and research conducted in this field. It highlights the contributions made by others and situates the current project within the broader academic and technical context.
4. Chapter 4 introduces the Transformer architecture in detail. This chapter is intended to familiarise the reader with the model's internal mechanisms including its structure, mathematical foundations and why it is particularly suited for sequential decision-making tasks such as race strategy prediction.
5. Chapter 5 outlines the development of the race strategy simulator, which is built upon the theoretical principles and machine learning techniques discussed in Chapter 4. It includes the engineering and software aspects required to bring the model into a functional system.
6. Chapter 6 presents the theoretical results from training the models. It highlights key design changes and evaluates performance using different metrics.
7. Chapter 7 evaluates the performance of the simulator using real race data. This evaluation demonstrates the applicability and potential of the proposed approach in real-world Formula 1 scenarios highlighting strengths
8. Finally Chapter 8 summarises the main findings and reflects on the outcomes of the project. It also considers future directions and possible improvements, both from a technical and strategic perspective.

Chapter 4

State of the art: Transformers

Transformers represent the current state of the art in machine learning. They were first introduced by Google in their seminal 2017 paper [Vas+17], where it was observed that traditional sequential models such as Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs) and Gated Recurrent Units (GRUs) suffered from limitations in training speed and memory usage due to their inherently sequential nature and lack of parallelisation.

To address these issues Google proposed a novel architecture built entirely around attention mechanisms. This innovation enabled significantly faster training and better performance on various tasks. For context and motivation it is worth noting that many of today's most powerful models, such as ChatGPT, are entirely based on transformers. In fact, the *T* in *GPT* stands for Transformer. Given their success and relevance this thesis will explore transformers in depth, both from a computer science and mathematical perspective.

The original paper [Vas+17], which primarily focuses on machine translation tasks, introduces the Transformer architecture and presents the following diagram to illustrate its structure:

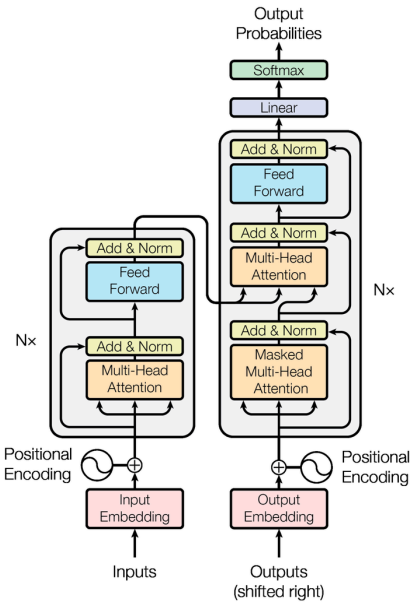

Figure 4.1: The Transformer architecture.

At first glance, the diagram may appear complex or overwhelming. However by breaking it down step by step it becomes much more approachable. The architecture is divided into two main components: the Encoder and the Decoder (the left and right blocks in Figure 4.1, respectively). Each component serves a distinct function:

1. **Encoder**: The encoder transforms an input sequence of tokens into a sequence of embedding vectors. As illustrated in Figure 4.1, the encoder consists of N_x stacked layers each containing two sub-layers. The first sub-layer is a multi-head self-attention mechanism and the second is a fully connected feed-forward neural network. Notably each sub-layer is equipped with residual connections followed by layer normalization which helps to stabilise and accelerate training.

2. **Decoder**: The decoder takes the encoder's output and generates the output sequence of tokens iteratively. Like the encoder, it comprises N_x layers, but with an additional key feature: masked multi-head attention. This masking, in conjunction with shifting the output sequence by one position, ensures that the prediction at position i depends only on the known outputs at positions less than i. Finally the output passes through a linear layer followed by a softmax function to produce a probability distribution.

Now that a general overview of the transformer architecture has been established, each component will be analysed in depth. The goal is to provide a thorough understanding, supported by rigorous mathematical formulation and intuitive explanation, of how each part contributes to the overall performance of the model.

4.1 Input embedding

The input embedding sublayer plays a fundamental role within the Transformer architecture. Its primary objective is to convert input tokens into numerical vectors of a predefined dimension through learned embeddings. The motivation behind this layer lies in enabling neural networks to capture the semantic meaning of each token. This semantic understanding allows the model to work with fixed-size vector representations that can be processed effectively in subsequent layers.

The overall procedure can be broken down into two steps: tokenisation and vector embedding.

Tokenisation is defined as a map

$$t : \mathcal{I} \longrightarrow \mathcal{P}(\mathcal{I})$$

where $\mathcal{I}$ denotes the input sequence and $\mathcal{P}(\mathcal{I})$ is the corresponding set of tokens after tokenisation.

The next step is embedding, defined by the map

$$E : \mathcal{P}(\mathcal{I}) \longrightarrow \mathbb{R}^{d_{model}}$$

which transforms each token t into a vector in $\mathbb{R}^{d_{model}}$, where d_{model} is the dimensionality of the model's internal representation space. It is essential that d_{model} is an even number as several operations and derivations presented in the following chapters rely on this assumption. If d_{model} were odd, many of the subsequent formulations would break or become ill-defined.

These vectors are then assembled into a matrix using the following transformation

$$\begin{array}{rccc} G: & \mathbb{R}^{d_{model} \times n} & \longrightarrow & \mathbb{R}^{d_{model} \times size} \\ & v_1, v_2, \ldots, v_n & \longrightarrow & (v_1, v_2, \ldots, v_n, \text{padding}) \end{array}$$

where $n \leqslant size$ is the number of tokens in the input and $size$ is the fixed maximum sequence length. Padding is applied when necessary to maintain consistent input dimensions.

Putting it all together, the full embedding pipeline can be expressed as the composition $M = G \circ E \circ t$, or explicitly

$$\begin{array}{rccc} M: & \mathcal{I} & \longrightarrow & \mathbb{R}^{d_{model} \times size} \\ & i & \longrightarrow & m \end{array}$$

where the input i is transformed into a matrix m of shape $\mathbb{R}^{d_{model} \times size}$, ready to be processed by the Transformer.

An illustrative example can help to make the process more tangible.

Suppose the input sentence is: *Max won the 2024 Championship*. The first step is tokenisation, which involves splitting the input into individual components. While different models use different tokenisation algorithms (some subword-based, others character-level), for simplicity, let us assume a word-level tokenisation. The tokenised result becomes

$$t(\textit{Max won the 2024 Championship}) = [\text{"Max"}, \text{"won"}, \text{"the"}, \text{"2024"}, \text{"Championship"}]$$

Thus, the original sentence is transformed into the list

$$[\text{"Max"}, \text{"won"}, \text{"the"}, \text{"2024"}, \text{"Championship"}]$$

Additionally, it is common practice to include special tokens such as `<start>` and `<end>` to explicitly mark the boundaries of the sentence. The token list then becomes

```
[<start>, "Max", "won", "the", "2024", "Championship", <end>]
```

At this stage the input has been segmented and the next step consists of converting each token into a numerical vector, a format suitable for processing by neural networks. This transformation is performed by an embedding layer which is itself a neural network trained to assign a dense vector representation to each token. Further details on the internal mechanisms of neural networks, along with a justification for why this approach is appropriate for computing E, are provided in Section 4.4. An illustrative example of how E operates is as follows

$$E(\text{Max}) = \begin{pmatrix} 56.21 \\ 32.05 \\ -14.72 \\ \vdots \\ 70.89 \end{pmatrix}, \quad E(\text{won}) = \begin{pmatrix} 12.34 \\ -7.80 \\ 45.67 \\ \vdots \\ 22.13 \end{pmatrix}, \quad E(\text{the}) = \begin{pmatrix} -4.56 \\ 88.90 \\ -32.10 \\ \vdots \\ 14.75 \end{pmatrix},$$

$$E(2024) = \begin{pmatrix} 0.12 \\ -19.43 \\ 6.78 \\ \vdots \\ 33.33 \end{pmatrix}, \quad E(\text{Championship}) = \begin{pmatrix} 42.76 \\ -23.87 \\ 17.29 \\ \vdots \\ -9.01 \end{pmatrix}$$

Each vector has a fixed dimensionality d_{model}. The model can handle a maximum number of input tokens, defined by $size$. If the number of tokens is smaller than $size$, padding vectors are appended. If the input exceeds the limit the model may truncate the input or raise an exception, depending on the implementation.

These vectors are finally concatenated into a matrix in $\mathbb{R}^{d_{model} \times size}$

$$\begin{pmatrix} 56.21 & 12.34 & -4.56 & 0.12 & 42.76 & \text{padding} & \cdots \\ 32.05 & -7.80 & 88.90 & -19.43 & -23.87 & \text{padding} & \cdots \\ -14.72 & 45.67 & -32.10 & 6.78 & 17.29 & \text{padding} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \text{padding} & \cdots \\ 70.89 & 22.13 & 14.75 & 33.33 & -9.01 & \text{padding} & \cdots \end{pmatrix}$$

This matrix is the final output of the embedding layer.

It is important to highlight that this layer behaves deterministically, every token always maps to the same embedding. For instance, the word *bank* will yield the same vector representation whether it appears in *the bank of the river* or in *get money from the bank*, despite the difference in meaning. The disambiguation based on context is handled by deeper layers in the Transformer, particularly the positional encoding and attention mechanism.

Another key insight is the interpretation of these embedding vectors. Each of the d_{model} components can be considered as a latent feature of the token. These features are not explicitly defined but are learned through training and can represent properties such as syntactic roles, sentiment or topic.

Analysis of the embedding space reveals that directions within it have semantic meaning. For example the word *house* tends to lie close to vectors for *building*, *roof*, *home* and *construction*, reflecting their shared semantics.

Perhaps even more impressively, certain relationships between words manifest as vector arithmetic. A well-known and illustrative example is

$$E(\text{man}) - E(\text{female}) \approx E(\text{king}) - E(\text{queen})$$

Other similar analogies also emerge naturally from the learned embedding space like

$$\begin{aligned} E(\text{man}) - E(\text{female}) &\approx E(\text{uncle}) - E(\text{aunt}) \\ E(\text{man}) - E(\text{female}) &\approx E(\text{nephew}) - E(\text{niece}) \\ E(\text{man}) - E(\text{female}) &\approx E(\text{father}) - E(\text{mother}) \\ E(\text{man}) - E(\text{female}) &\approx E(\text{brother}) - E(\text{sister}) \end{aligned}$$

These relationships hint at the remarkable capacity of the embedding layer to capture underlying structures of the input, forming the foundation upon which the rest of the Transformer builds.

4.2 Positional encoding

This component represents a fundamental aspect of the transformer architecture. As previously discussed, one of the key advantages of transformers lies in their ability to process data in parallel. Unlike traditional sequential models, transformers can handle entire input sequences simultaneously, significantly accelerating both training and inference.

However this parallelism introduces a challenge: if all input tokens are processed at once, how does the model retain any information about the original order of the sequence? After all, time-series data are inherently ordered and ignoring that order would result in a loss of crucial context.

This is precisely where positional encoding becomes essential.

As introduced in Section 4.1, the input sequence is first transformed into a sequence of token embeddings, essentially numerical vector representations of each item. The goal now is to enrich these embeddings with information about their position in the original sequence. In other words, a mechanism must be added that allows the model to distinguish between tokens based on their relative or absolute position, even when processed in parallel.

To fully understand the concept behind positional encoding, it is beneficial to develop the idea step by step, starting from the most intuitive perspective.

Let us briefly recall the objective: to provide the model with information about the original position of each token within the input sequence. Ideally any positional encoding scheme should satisfy the following criteria:

1. The encoding process must be deterministic, ensuring consistent outputs for the same inputs.
2. It should produce a unique encoding for each time step.
3. The relative distance between any two positions should remain consistent across input sequences of varying lengths.
4. The model should be capable of generalising to longer sequences without requiring additional training.

To achieve this the goal is to define a function

$$PE : \{1, 2, \ldots, n\} \longrightarrow \mathbb{R}^{d_{model}}$$

where $\{1, 2, \ldots, n\}$ denotes the valid positions within an input sequence of length n and the output lies in $\mathbb{R}^{d_{model}}$, representing the positional vector associated with each token.

This mapping serves to enrich the embedding space with positional information aligning with the requirements outlined above.

A naive approach might involve adding a positional vector to each token embedding, this vector having the same dimensionality as the embedding vector: d_{model}. In its simplest form this positional vector could be a constant vector filled entirely with the token's position index i, that is, $[i, i, \ldots, i]$.

While this method encodes absolute position it leads to problems when dealing with longer sequences. In such cases, the magnitude of the positional vector can dominate that of the embedding vector, potentially overwhelming the semantic content encoded in the embeddings. As a result the model might lose valuable information about the token's meaning in favour of its position.

To mitigate this issue, one could normalise each positional vector by the total length of the input sequence, denoted as n (as previously defined in Section 4.1). This would yield vectors such as $[i/n, i/n, \ldots, i/n]$, ensuring that no single positional vector disproportionately outweighs the embedding it is added to. However this solution introduces ambiguity. Since the normalised position is relative different input lengths can produce identical vectors for different absolute positions. For example, in a sequence of length 4, the first token would receive a positional vector of $[0.25, 0.25, \ldots, 0.25]$. In a sequence of length 16, the fourth token would receive exactly the same vector. From the model's perspective these vectors are indistinguishable and the original position cannot be uniquely recovered.

To resolve this ambiguity one can attempt to combine both previous ideas: use absolute position values but restrict them within a fixed range. One simple way to achieve this is through binary encoding. Instead of assigning a vector of repeated decimal values (e.g., $[3, 3, \ldots, 3]$), one can encode the position using binary digits. For instance, the third position (which is decimal 3) can be represented by the binary vector $[0, \ldots, 0, 1, 1]$, which clearly captures the position without overwhelming the original embedding.

Interestingly, consecutive binary values exhibit a regular pattern of bit transitions,

Decimal	**Binary**			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0

Table 4.1: Binary representation of decimal numbers using 4 bits.

In Table 4.1, it becomes apparent that the least significant bit toggles at a higher frequency than the more significant bits. This observation leads to the next idea: if we view each bit as a signal that alternates between high and low states we can represent the full pattern of bit transitions as a matrix and then transpose it:

B4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
B3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
B2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
B1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Visualising these transitions as signals over time reveals a set of wave-like behaviours, as shown in the timing diagram in Figure 4.2.

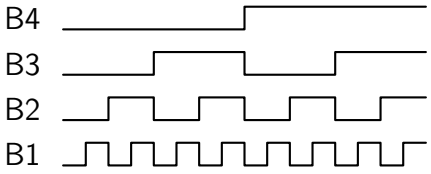

Figure 4.2: Bit transitions from decimal 0 to 15 represented as timing diagram.

Taking this idea further, these square waveforms can be approximated using sine functions of increasing frequency, producing a smoothly varying encoding that is both continuous and differentiable, ideal for use in neural networks.

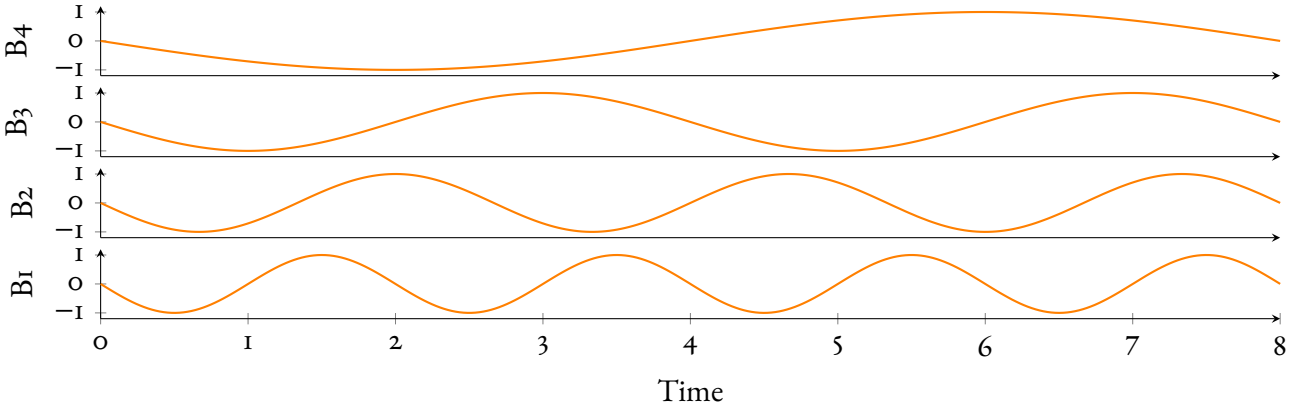

Figure 4.3: Stacked sine waves approximating digital bit transitions from B4 to B1.

Each wave corresponds to a frequency component of the signal and the values of these sine functions, evaluated at each token position, form the actual positional encoding.

In this simplified example the positional encoding is defined as

$$\sin\left(\frac{i}{n}\right)$$

where i is the token's position in the input and n is the total sequence length.

In the original Transformer architecture [Vas+17] a more sophisticated version was proposed using sine and cosine functions at multiple frequencies

$$PE(i)_t = \begin{cases} \sin\left(\dfrac{i}{10000^{\frac{2t}{d_{model}}}}\right) & \text{if } t \text{ is even} \\ \cos\left(\dfrac{i}{10000^{\frac{2t}{d_{model}}}}\right) & \text{if } t \text{ is odd} \end{cases}$$

In this formulation $PE(i)$ denotes the positional encoding vector associated with position i in the input sequence and t refers to a specific coordinate within this vector. Each pair of coordinates $(2t, 2t+1)$ shares the same frequency with one using a sine function and the other a cosine. This structure allows the model to encode both the magnitude and phase of the position using complementary signals. The use of sine for even indices and cosine for odd indices ensures that each position is mapped to a unique, continuous and differentiable encoding. Furthermore, the denominators form a geometric progression, allowing the model to capture both short and long-range dependencies in the sequence.

This mapping can be formally defined as follows

Definition 4.2.1: Positional encoding. Let $E \in \mathbb{R}^{n \times d_{model}}$ be a matrix that contains d_{model} dimensional column vectors $E_{i,}$ which encode the position i in an input sequence of length n. The mapping $PE : \{1, 2, \ldots, n\} \longrightarrow \mathbb{R}^{d_{model}}$ is referred to as *positional encoding*. It assigns to each position i a unique vector $PE(i) = E_{i,}$, defined as follows

$$PE(i) = E_{i,} = \begin{pmatrix} \sin(i\omega_1) \\ \cos(i\omega_1) \\ \sin(i\omega_2) \\ \cos(i\omega_2) \\ \vdots \\ \sin\left(i\omega_{\frac{d_{model}}{2}}\right) \\ \cos\left(i\omega_{\frac{d_{model}}{2}}\right) \end{pmatrix}$$

where $\omega_t = \dfrac{1}{10000^{\frac{2t}{d_{model}}}}$.

Regardless of the mathematical formulation, the core idea remains the same: positional encoding is a clever way to inject sequential order into a model that is otherwise completely

parallel.

The final output at the i-th position is a vector $v_i \in \mathbb{R}^{d_{model}}$ defined as

$$v_i = E(i) + PE(i)$$

where $E(i)$ represents the token embedding obtained through the mapping described in Section 4.1 and $PE(i)$ denotes the positional encoding corresponding to the i-th position in the input sequence. The addition of these two components allows the model to retain both the semantic meaning of the token and its position within the sequence.

Once these position-aware vectors are computed for the entire sequence, the grouping map G is applied to assemble them into a matrix in $\mathbb{R}^{d_{model} \times size}$, which is subsequently passed as input to the Transformer Encoder.

It is now necessary to verify that PE satisfies the four desired properties of a valid positional encoding function.

Firstly, PE is clearly well-defined, as its output depends solely on the position index i and the component index t. The function is entirely deterministic meaning that the same input will always yield the same output.

Secondly, to ensure that each time step receives a unique encoding, it must be shown that PE is an injective function, that is a one-to-one mapping where different inputs produce different outputs. This can be rigorously proven as follows

Proposition 4.2.1. Let PE be the positional encoding function. Then PE is an injective function.

As described in the original paper introducing the Transformer architecture [Vas+17], this specific choice of positional encoding was motivated by its compatibility with relative position reasoning. The authors claim that for any fixed offset k, the positional encoding $PE(i+k)$ can be expressed as a linear function of $PE(i)$. However, while the intuition is mentioned, no formal derivation or proof is provided in the original work. The following proposition presents a rigorous mathematical demonstration of this property.

Proposition 4.2.2. Let PE be the positional encoding function. There exists a linear transformation $T : \{1, 2, \ldots, n\} \longrightarrow \mathbb{R}^{d_{model} \times d_{model}}$ such that

$$T(k)PE(i) = PE(i+k)$$

holds for any positional offset $k \in \{1, 2, \ldots, n\}$ at any valid position $i \in \{1, 2, \ldots, n-k\}$ in the sequence.

This result highlights the stability and scalability of sinusoidal positional encoding as it enables efficient computation of future positions through a linear transformation. Once the transformation $T(k)$ is known predicting $PE(i + k)$ reduces to a simple matrix-vector multiplication avoiding the recomputation of trigonometric values.

More importantly, Proposition 4.2.2 shows that the encoding supports relative positioning. That is, the representation of $i + k$ can be derived from i without referring to the start of the sequence. This is especially useful in sequence modelling where relative distances often carry more meaning than absolute positions.

These results collectively demonstrate that the sinusoidal positional encoding used in Transformer models satisfies all four of the desired properties, making it a valid and effective choice.

To better understand the structure of these encodings, a visualisation of the orbits generated by PE is shown in Figure 4.4. This figure was generated using the script provided in Appendix B.

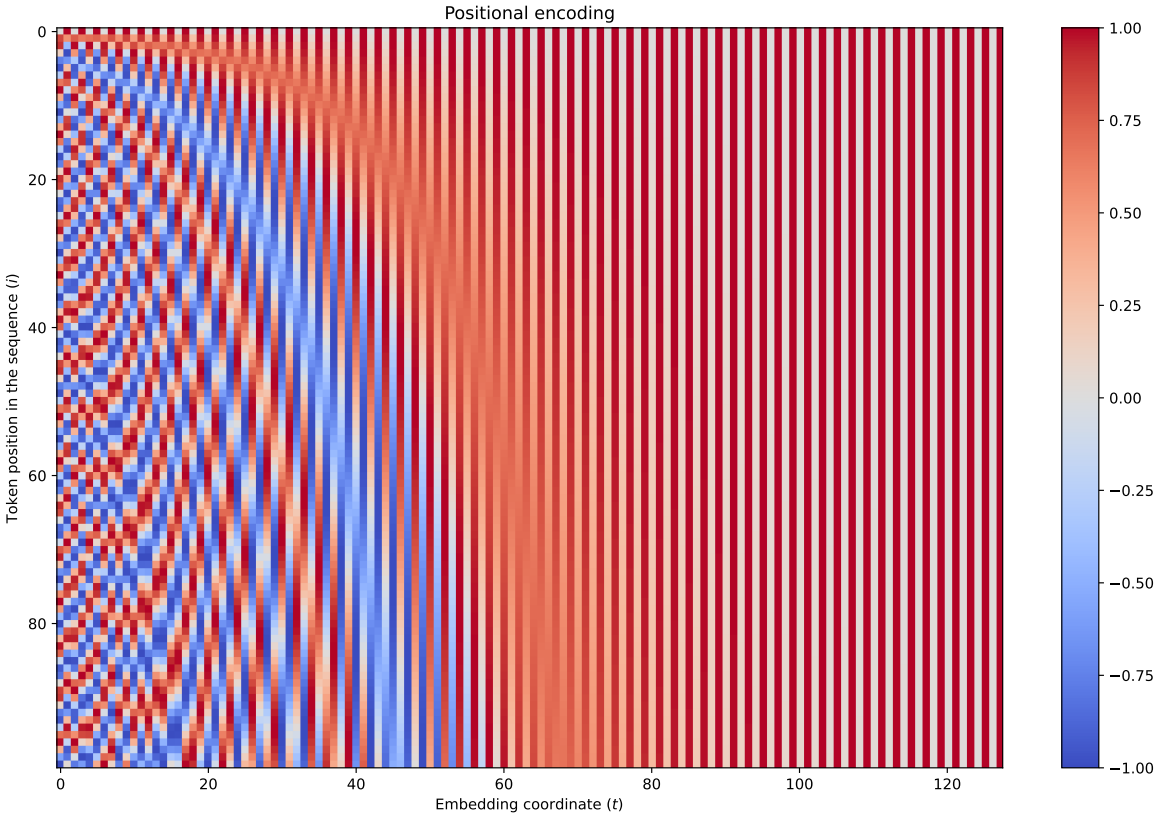

Figure 4.4: Heatmap visualisation of the sinusoidal positional encoding matrix. Rows represent sequence positions (i) and columns correspond to embedding coordinates (t).

Each row of the matrix corresponds to a token in the input sequence (in this case 100 tokens were considered, representing the total sequence length n), while each column represents

a specific coordinate of the embedding vector. The full range of embedding indices $t \in \{0, 1, 2, \ldots, d_{model}\}$ is considered, where d_{model} was set to 128.

The purpose of this heatmap is to provide a visual understanding of how each token in the sequence is assigned a distinct positional encoding. This allows the Transformer model to retain information about the original position of each token in the sequence which is essential for handling sequential data in a context where positional information is not inherently present, such as in self-attention mechanisms.

The values in the encoding matrix range from -1 to 1, with the colour gradient indicating these values, blue representing -1 and red representing 1, as shown in the colour bar on the right-hand side of the figure.

The first row, corresponding to position 0, can be intuitively compared to the binary number 000000. As one moves down the matrix, that is as the token position i increases, the patterns evolve in a manner reminiscent of binary counting. This analogy illustrates how lower positions exhibit faster oscillations in the encoding.

On the leftmost side of the heatmap, the lower embedding coordinates are shown. These exhibit high-frequency oscillations, analogous to the least significant bits in a binary representation. Conversely, the rightmost columns correspond to higher embedding coordinates, which vary more slowly, resembling the behaviour of the most significant bits. This hierarchical encoding of frequencies enables the model to capture both short-range and long-range dependencies within the input sequence.

4.3 Attention

Attention mechanisms are a fundamental component of the Transformer architecture. It is no coincidence that the seminal paper introducing Transformers, [Vas+17], is entitled *Attention is all you need*. This section provides a detailed explanation of how attention mechanisms operate, ensuring a comprehensive understanding of their role.

To begin, it is important to consider the motivation behind their development. As previously discussed, prior to the advent of Transformers, sequence-to-sequence models processed data in a strictly sequential manner, token by token. This approach proved inefficient, as a token at position i could only incorporate information from the token at position $i-1$. The introduction of attention mechanisms allowed models to process entire sequences simultaneously, leveraging the full context at once. In this manner, a token at position i gains access to information from all other tokens, enabling global context-awareness.

To formalise this process, [Vas+17] defined the attention mechanism as follows

$$\begin{aligned} \text{Attention}: \quad \mathbb{R}^{d_k \times d_k} \times \mathbb{R}^{d_k \times d_k} \times \mathbb{R}^{d_v \times d_v} &\longrightarrow \mathbb{R}^{d_v \times d_v} \\ Q, K, V &\longrightarrow \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \end{aligned}$$

Here, Q, K and V represent the query, key and value matrices, respectively. The term QK^T denotes the dot product between Q and the transpose of K, reflecting pairwise comparisons between the rows of Q and the columns of K. The division by $\sqrt{d_k}$ serves to scale the dot products, preventing them from becoming excessively large as the dimensionality d_k increases. Without this scaling, the softmax function could be driven into regions where its gradients become vanishingly small, impeding effective learning.

At this stage, it is natural to ask: what do these matrices Q, K and V represent intuitively? An analogy that proves particularly insightful, originally encountered in [Dot21], compares each token to a *Tinder profile*, where each token describes both what it is (the key) and what it is seeking (the query). This metaphor provides an accessible way to understand the interaction between tokens. Specifically, for any given token, the query vector encodes what the token is looking for, while the key vector expresses its characteristics. The attention mechanism computes the dot product between a token's query and the keys of all other tokens, effectively measuring *compatibility* between them. A higher dot product signifies a stronger match.

This process is performed for all tokens in parallel, which highlights one of the key advantages of attention mechanisms. Since the compatibility scores between any pair of tokens are independent of other pairs, these computations can be fully parallelised. For instance, the compatibility between the first and second tokens can be computed simultaneously

with the compatibility between the first and third tokens, as there are no sequential dependencies. This parallelisation enables significant computational efficiency.

With the fundamentals of attention mechanisms now established, it is worthwhile to explore further theoretical insights that have emerged in recent research. Several studies, including [VBC20], [FHP24] and [Ges+24] present alternative perspectives and formal interpretations of attention. These approaches remain fully aligned with, and mathematically equivalent to, the original self-attention framework introduced by [Vas+17], yet offer valuable viewpoints that deepen our understanding of its underlying principles.

To begin, it is essential to introduce the rigorous definition of *attention*, a concept that underpins numerous modern machine learning architectures.

Definition 4.3.1: Attention. Let $\mathcal{Q} \subseteq \mathbb{R}^{d_q}, \mathcal{K} \subseteq \mathbb{R}^{d_k}$ and $\mathcal{V} \subseteq \mathbb{R}^{d_v}$ represent the query space, key space and value space, respectively. Consider $K = \{k_1, k_2, \ldots, k_N\} \subset \mathcal{K}$ as a set of keys and $V = \{v_1, v_2, \ldots, v_N\} \subset \mathcal{V}$ as a corresponding set of values. Let $q \in \mathcal{Q}$ denote a query. Furthermore, let $a : \mathcal{Q} \times \mathcal{K} \longrightarrow \mathbb{R}$ be a similarity function that quantifies the relationship between a query and a key.

The *attention* mechanism is then defined as the mapping,

$$\text{Attention}(q, K, V) = \sum_{i=1}^{n} \text{softmatch}_a(q, K)_i \cdot v_i$$

where $\text{softmatch}_a(q, K)$ is a probability distribution over the set of keys K defined as

$$\text{softmatch}_a(q, K)_i = \frac{\exp(a(q, k_i))}{\sum_{j=1}^{N} \exp(a(q, k_j))} = \text{softmax}_j(\{a(q, k_j)\}_j)$$

While $\text{Attention}(\cdot, K, V)$ is formally defined for a single query, in practice it is typically applied to an entire set of queries $Q = \{q_1, q_2, \ldots, q_M\} \subset \mathcal{Q}$ in parallel. Therefore, it is customary to write

$$\text{Attention}(Q, K, V) = \left\{\text{Attention}(q_i, K, V)\right\}_{i=1}^{M}$$

It is important to note that, while the cardinalities of K and V must be equal (i.e., $|K| = |V| = N$), there is no such requirement for Q and in general $M \neq N$.

Definition 4.3.2: Self-attention. When the query set, key set and value set coincide, that is $Q = K = V$, the attention mechanism is referred to as *self-attention*

$$\text{SelfAttention} : Q \longrightarrow \text{Attention}(Q, Q, Q)$$

To proceed towards a more general mathematical framework, consider (E, ε) as a measurable subset of $\mathbb{R}^d$ endowed with its Borel σ-algebra and let $\mathcal{P}(E)$ denote the space of all probability measures on E. For any real-valued measurable function f, the expectation with respect to a probability measure μ is denoted as

$$\mu(f) = \int f(x)\mu(dx)$$

whenever the integral exists. In the case of a vector-valued function $F : E \longrightarrow \mathbb{R}^l$, where $F(x) = [F_1(x), F_2(x), \cdots, F_l(x)]^T$, the expectation is similarly defined component-wise as

$$\mu(F) = \left[\mu(F_1), \mu(F_2), \cdots, \mu(F_l)\right]^T$$

whenever all components exist.

Definition 4.3.3: Markov kernel. A *Markov kernel* is an E-indexed family of probability measures $M(x, dy) \in \mathcal{P}(E)$ such that, for every measurable set $A \in \varepsilon$, the function $x \mapsto M(x, A)$ is measurable.

A Markov kernel M defines a linear operator on the space of probability measures

$$\begin{aligned} \mu M(dy) : \ \mathcal{P}(E) &\longrightarrow \mathcal{P}(E) \\ x &\longrightarrow \int \mu(dx) M(x, dy) \end{aligned}$$

Additionally, it defines a linear operator acting on measurable functions f

$$M(f)(x) = \int f(y) M(x, dy)$$

Markov kernels can also be composed. Given two kernels, M and N, their composition is defined through integration as

$$MN(x, dz) = \int M(x, dy) N(y, dz)$$

We now proceed to develop a comprehensive construction of the attention mechanism, framed as a non-linear Markov transport on $\mathcal{P}(E)$, the space of probability measures on a measurable set E. This formulation reinterprets standard linear algebraic and pointwise operations within the attention model as operator-based transformations on $\mathcal{P}(E)$, thereby enriching the theoretical framework.

The key components of attention, as presented in Definition 4.3.1, are the *softmatch* operation, the key-value correspondence and the value aggregation according to the softmatch distribution. Each of these elements will be addressed in turn, beginning with the softmatch.

4.3.1 Softmatch

At the heart of the softmatch function, and indeed the attention mechanism as a whole, lie the interactions between queries and keys. These interactions can be viewed as a particular instance of a non-linear measure transformation known as the Boltzmann-Gibbs transformation.

Definition 4.3.4: Boltzmann-Gibbs transformation. Let $g : E \longrightarrow \mathbb{R}^+$ be a bounded measurable function. The *Boltzmann-Gibbs transformation* associated to g is the mapping $\psi_g : \mathcal{P}(E) \longrightarrow \mathcal{P}(E)$ defined by

$$\psi_g(\nu)(dx) = \frac{g(x)\nu(dx)}{\nu(g)}$$

where $\nu(g) = \int_E g(x)\nu(dx)$ denotes the expected value of g under the measure ν.

Definition 4.3.5: Interaction potential. An *interaction potential* is defined as the mapping

$$\begin{array}{rccc} G: & E \times E & \longrightarrow & \mathbb{R}^+ \\ & (x, y) & \longrightarrow & \exp(a(x, y)) \end{array}$$

where a is the similarity function introduced in Definition 4.3.1.

Definition 4.3.6: Softmatch kernel. Given an interaction potential G, the *softmatch kernel* is the family of Markov kernels $\{\psi_G(\nu)\}_{\nu \in \mathcal{P}(E)}$ indexed by $\nu \in \mathcal{P}(E)$. For any measurable set $A \in \varepsilon$, this kernel is defined as

$$\psi_G(\nu)(x, A) = \int_A \psi_{G(x,\cdot)}(\nu)(dy)$$

To illustrate how ψ_G models the softmatch operation, we introduce a fundamental construction from measure theory. Denote by $\mathcal{P}_\delta(E) = \{\delta_x : x \in E\}$ the subset of Dirac measures within $\mathcal{P}(E)$. There exists a natural bijection between the set E and $\mathcal{P}_\delta(E)$, given by $x \longleftrightarrow \delta_x$. This correspondence forms the primary entry point for embedding discrete structured data into the measure-theoretic framework.

For any realisation of structured data $X = \{x_1, x_2, \ldots, x_T\} \subset E$, an empirical measure can be associated as follows

$$X \mapsto m(X) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \delta_{x_t}$$

where $|\mathcal{T}|$ denotes the cardinality of the index set $\mathcal{T}$.

In the subsequent discussion, we shall interchangeably use X and $\{\delta_{x_1}, \delta_{x_2}, \ldots, \delta_{x_T}\}$ to represent the collection of individual vectors and $m(X)$ to denote their joint configuration as a probability measure. This representation will prove instrumental in modelling the behaviour of attention mechanisms.

Consider now a query representation δ_q and key representations $K = \{\delta_{k_1}, \delta_{k_2}, \ldots, \delta_{k_N}\}$, together with their empirical measure $m(K)$. The softmatch kernel models the interaction between the query q and the set of keys K via the left action of the Markov kernel $\psi_G(m(K))$ on the Dirac measure δ_q, defined by

$$\delta_q \psi_G(m(K)) = \int \delta_q(dq')\psi_{G(q',\cdot)}(m(K)) = \psi_{G(q,\cdot)}(m(K)) \sum_{s=1}^{N} \frac{G(q,k_s)}{\sum_{r=1}^{N} G(q,k_r)} \delta_{k_s}$$

Extending this framework to a set of queries $Q = \{\delta_{q_1}, \delta_{q_2}, \ldots, \delta_{q_M}\}$, the linearity of integration enables us to model the interaction between Q and K as follows

$$\begin{aligned} m(Q)\psi_G(m(K)) &= \frac{1}{M} \sum_{t=1}^{M} \int \delta_{q_t}(dq)\psi_{G(q,\cdot)}(m(K)) \\ &= \frac{1}{M} \sum_{t=1}^{M} \psi_{G(q_t,\cdot)}(m(K)) \\ &= \frac{1}{M} \sum_{t=1}^{M} \sum_{s=1}^{N} \frac{G(q_t,k_s)}{\sum_{r=1}^{N} G(q_t,k_r)} \delta_{k_s} \end{aligned}$$

This resulting measure represents the joint configuration of the set of queries Q after interacting with the keys K via the interaction potential G and the associated Boltzmann-Gibbs transformation. It forms a weighted combination of particle measures, providing a probabilistic interpretation of the softmatch operation introduced in Definition 4.3.1.

4.3.2 Key-value relationship

To generalise the relationship between keys and values within the attention mechanism, we introduce the concept of the *lookup kernel*.

Definition 4.3.7: Lookup kernel. Assume that the key and value spaces are measurable spaces $(\mathcal{K}, \mathbb{K})$ and $(\mathcal{V}, \mathbb{V})$, respectively. A *lookup kernel* is a Markov kernel $L : \mathcal{K} \times \mathbb{V} \longrightarrow [0, 1]$, also denoted as $L(k, dv)$, that maps each key $k \in \mathcal{K}$ to a probability distribution over the value space $\mathcal{V}$.

In the special case where this mapping corresponds to a deterministic lookup table, the

kernel reduces to

$$L(k, dv) = \sum_{i=1}^{N} \mathbb{I}_{\{k=k_i\}} \delta_{v_i}(dv)$$

where $\{(k_i, v_i)\}_{i=1}^{N}$ represents the finite set of key-value pairs. Here, each key k_i is deterministically associated with its corresponding value v_i via the Dirac measure δ_{v_i}.

This formulation of the key-value relationship allows us to represent both deterministic and probabilistic associations between keys and values, providing a flexible framework for modelling the lookup process in attention mechanisms.

4.3.3 Averaging

The final component required for the construction of attention is the averaging operation over the set of values being attended to. To formalise this we introduce the concept of *moment encoding* within measures and define a family of measures parametrised by their moment vectors.

Definition 4.3.8: Moment encoding. Let $F : E \longrightarrow E' \subset \mathbb{R}^l$ be a measurable map, referred to as a feature map, which maps elements of E to an l-dimensional feature space E'. A measure $\mu \in \mathcal{P}(E)$ is said to *encode a moment vector* $f \in E'$ with respect to F if

$$\mu(F) = f$$

where $\mu(F) = \int_E F(x)\mu(dx)$ denotes the expectation of F under the measure μ.

A particularly useful case arises when $E = E'$ and $F(x) = x$. In this setting, for a Dirac measure δ_x, it holds that $\delta_x(F_j) = x_j$, meaning that δ_x encodes the moment vector corresponding to x. This correspondence facilitates the translation between vector-space representations and measure-theoretic notions.

Definition 4.3.9: Moment subspace of $\mathcal{P}(E)$. Given a measurable map $F : E \longrightarrow E'$, suppose there exists an injective mapping $f \mapsto \nu_f \in \mathcal{P}(E)$ such that ν_f encodes the moment vector f with respect to F. The *moment subspace* associated with F is then defined as

$$\mathcal{F}_F = \{\nu_f : f \in E'\}$$

where, for convenience, the subscript F may be omitted when the context is clear.

Definition 4.3.10: Moment projection. The *moment projection* onto the moment subspace $\mathcal{F}$ is defined as the mapping $\prod : \mathcal{P}(E) \longrightarrow \mathcal{F}$ given by

$$\prod_{\mathcal{F}}(\mu) = \nu_{\mu(F)}$$

where $\prod_{\mathcal{F}}(\mu)$ is the unique measure in $\mathcal{F}$ that encodes the moment vector $f = \mu(F)$.

In the context of attention mechanisms, the averaging over the input values is achieved via the moment projection $\prod = \prod_{\mathcal{F}}$ described in Definition 4.3.10, where $\mathcal{F} = \mathcal{P}_{\delta}(E)$ and $F(x) = x$. This projection effectively computes the empirical average over the attended values, aligning with the conventional aggregation step in attention.

4.3.4 The attention kernel

The preceding sections have laid the groundwork for reinterpreting the attention mechanism in measure-theoretic terms. Bringing these components together, we now introduce the *attention kernel*, which formalises the composition of the softmatch, lookup and averaging operations into a unified framework.

Definition 4.3.11: Attention kernel. The *attention kernel* A is defined as the composition of the moment projection $\prod$, the softmatch kernel and the lookup kernel. For $x \in E$ and $\mu \in \mathcal{P}(E)$, the attention kernel is given by

$$A_{\mu}(x, dz) = \prod(\psi_{G(x,\cdot)}(\mu)L)(dz) = \prod\left(\int \psi_{G(x,\cdot)}(\mu)(dy)L(y,\cdot)\right)(dz)$$

This formulation of the attention kernel provides a general operator-based view of attention mechanisms, consistent with the classical definition of attention (Definition 4.3.1) for appropriate choices of the interaction potential G and lookup kernel L.

Proposition 4.3.1. Let $G(x, y) = \exp(a(x, y))$ and let the lookup kernel be given by $L(k, dv) = \sum_{i=1}^{N} \mathbb{I}_{\{k=k_i\}}\delta_{v_i}(dv)$. Consider Q, K and V as in Definition 4.3.1. Then, using the left action of kernels on measures, the mapping

$$(Q, K, V) \mapsto \left\{\delta_{q_1}A_{m(K)}, \delta_{q_2}A_{m(K)}, \ldots, \delta_{q_M}A_{m(K)}\right\}$$

recovers the standard attention mechanism as defined in Definition 4.3.1.

This proposition demonstrates that the measure-theoretic formulation of the attention kernel fully aligns with the classical vector-based formulation of attention, bridging the two perspectives seamlessly.

Proposition 4.3.2. Let $X = \{x_1, x_2, \ldots, x_N\} \subset \mathbb{R}^d$ be a collection of inputs. The non-linear Markov transport equation

$$\delta_{x_i} \mapsto \delta_{x_i} T_{m(X)}$$

where $T_{m(X)}$ is an appropriately defined transport kernel, implements the self-attention mechanism of the Transformer architecture.

This final result frames self-attention as a particular instance of non-linear Markov transport, offering a principled probabilistic interpretation of the Transformer model within the context of measure theory.

Now that the full construction of the attention mechanism has been presented and its equivalence with the standard formulation from Definition 4.3.1 established, it is natural to ask whether the traditional linear-algebraic definition of attention can be recovered directly from this measure-theoretic framework. The answer is affirmative.

Consider the case where E is a discrete set, specifically $E = \{1, 2, \ldots, N\}$ and let $Q = \{q_1, q_2, \ldots, q_N\}$, $K = \{k_1, k_2, \ldots, k_N\}$ and $V = \{v_1, v_2, \ldots, v_N\}$ be subsets of $\mathbb{R}^d$. Define the interaction potential $G : Q \times K \longrightarrow \mathbb{R}^+$ by

$$G(i, j) = \exp(Q[i] \cdot K[j])$$

where $Q[i]$ and $K[j]$ denote the i-th and j-th vectors in Q and K, respectively. The lookup kernel $L : K \times 2^V \longrightarrow [0, 1]$ is the discrete Markov kernel

$$L(i, dj) = \delta_i(dj)$$

Finally, define the moment projection $\prod : \mathcal{P}(V) \longrightarrow \mathbb{R}^d$ by

$$\prod(\mu) = \sum_{j=1}^{N} V[j]\mu_j$$

where μ_j represents the probability weight associated with v_j. This corresponds to the standard weighted sum used in linear attention.

Under these definitions, the analogue of the attention kernel becomes the *attention map*

$$A_{m(K)}(i) = \prod(\psi_{G(i,\cdot)}(m(K))L)$$

where $m(K)$ denotes the empirical measure over the keys K.

We can now make several observations that illustrate how this discrete framework recovers the classical attention mechanism:

1. The Dirac measure $\delta_{q_i}(dj)$ reduces to the Kronecker delta δ_i^j and the empirical measure $m(K)$ becomes the uniform distribution over K, represented as

$$m(K) = \frac{1}{N}[1, 1, \ldots, 1]$$

of dimension N.

2. The Boltzmann-Gibbs transformation $\psi_{G(i,\cdot)}(dj)$ simplifies to the standard softmax function

$$\psi_{G(i,\cdot)}(m(K))(dj) = \text{softmax}\left(Q[i]K^T\right)$$

The softmatch kernel in this setting corresponds to the stochastic matrix

$$\psi_G(m(K)) = \text{softmax}\left(QK^T\right)$$

3. The composition of kernels becomes standard matrix multiplication

$$\psi_G(m(K))L = \text{softmax}\left(QK^T\right)I = \text{softmax}\left(QK^T\right)$$

since L is the $N \times N$ identity matrix.

Finally, the attention map simplifies to the familiar matrix multiplication form

$$\begin{aligned}\delta_{Q[i]}A_{m(K)} &= \prod\left(\psi_{G(i,\cdot)}(m(K))L\right) = \sum_{j=1}^{n}\psi_{G(i,\cdot)}(m(K))(dj)V[j] \\ &= \sum_{j=1}^{n}\text{softmax}\left(Q[i]K^T\right)[j]V[j] = \text{softmax}\left(Q[i]K^T\right)V\end{aligned}$$

This precisely recovers the standard definition of attention given in Definition 4.3.1, widely used in machine learning.

Having established the equivalence between the probabilistic and algebraic perspectives of attention, we now explore an important property of attention mechanisms: their Lipschitz continuity. This property is crucial in ensuring the stability and robustness of attention-based models.

To formalise this we begin with several key definitions.

Definition 4.3.12: Lipschitz continuity. Let $f : \mathcal{S} \longrightarrow \mathcal{T}$ be a mapping between metric spaces $(\mathcal{S}, d_{\mathcal{S}})$ and $(\mathcal{T}, d_{\mathcal{T}})$. The function f is said to be *Lipschitz continuous* if there exists a constant $K > 0$ such that

$$d_{\mathcal{T}}(f(x), f(y)) \leqslant K d_{\mathcal{S}}(x, y)$$

for all $x, y \in \mathcal{S}$.

Definition 4.3.13: Wasserstein distance. Let $\mathcal{P}_1(E)$ denote the set of probability measures on E with finite first moments. The *1-Wasserstein distance* between two measures $\mu, \nu \in \mathcal{P}_1(E)$ is defined as

$$\mathbb{W}_1(\mu, \nu) = \inf_{\pi \in \mathcal{C}(\mu,\nu)} \int \int_{E \times E} \|x - y\|_1 \pi(dx, dy)$$

where $\mathcal{C}(\mu, \nu)$ is the set of joint distributions on $E \times E$ with marginals μ and ν.

Proposition 4.3.3. The 1-Wasserstein distance $\mathbb{W}_1$ defines a metric on the space $\mathcal{P}_1(E)$ of probability measures with finite first moments. Moreover, the metric space $\mathcal{W}_1 = (\mathcal{P}_1(E), \mathbb{W}_1)$ is complete and separable.

Definition 4.3.14: Wasserstein contraction coefficient. Given a mapping $\phi : \mathcal{P}_1(E) \longrightarrow \mathcal{P}_1(E)$, the *Wasserstein contraction coefficient* of ϕ is defined as

$$\tau(\phi) = \sup_{\mu \neq \nu} \frac{\mathbb{W}_1(\phi(\mu, \phi(\nu))}{\mathbb{W}_1(\mu, \nu)}$$

The following lemma establishes a bound on the contraction coefficient for the attention kernel.

Lemma 4.3.1. Let $E \subset \mathbb{R}^d$ be compact, $\mathcal{F} = \mathcal{P}_\delta(E)$ and $\prod = \prod_{\mathcal{F}}$. Let A be the attention kernel from Definition 4.3.11 with $L(x, dy) = \delta_{l(x)}(dy)$. Suppose the interaction potential G satisfies

$$G(x, y) \geqslant \varepsilon(G) > 0, \quad \|G\|_{Lip,\infty} < \infty \quad \text{and} \quad \|G\|_{\infty,Lip} < \infty$$

Then, the contraction coefficient $\tau(A)$ of A, considered as a mapping $A : \mu \longrightarrow \mu A_\mu$ on $\mathcal{P}(E)$, satisfies

$$\tau(A) \leqslant \tau(\prod)\tau(\psi_G)\tau(L)$$

where

$$\tau(\prod) = d, \quad \tau(\psi_G) = \frac{2(\|G\|_{Lip,\infty}+\|G\|_{\infty,Lip})\text{diam}(E)}{\varepsilon(G)}, \quad \tau(L) = K_l$$

This lemma leads to the following result on the Lipschitz continuity of the attention mechanism.

Theorem 4.3.1: Lipschitz continuity. Let $K = \{k_1, k_2, \ldots, k_N\} \subset E \subset \mathbb{R}^d$ and $V = \{v_1, v_2, \ldots, v_N\} \subset E \subset \mathbb{R}^d$. Assume that the attention function $\text{Att}(\cdot, K, V)$, as introduced in Definition 4.3.1, satisfies the conditions of Lemma 4.3.1. Then, the function

$$\begin{aligned} \text{Att}(\cdot, K, V): \ \mathbb{R}^d &\longrightarrow \mathbb{R}^d \\ q &\longrightarrow \text{Attention}(q, K, V) \end{aligned}$$

is Lipschitz continuous with respect to the Euclidean norm. Moreover, for all $q_1, q_2 \in \mathbb{R}^d$, the following inequality holds

$$\|\text{Att}(q_1, K, V) - \text{Att}(q_2, K, V)\|_2 \leqslant \sqrt{d^3}\|l\|_{Lip}\frac{2\|G\|_{Lip,\infty}\text{diam}(E)}{\varepsilon(G)}\|q_1 - q_2\|_2$$

Corollary 4.3.1. The mapping defined in Theorem 4.3.1 is continuous.

Attention is the core of the Transformer architecture, providing the mechanism that allows information to flow dynamically across sequences. Its ability to model dependencies, regardless of distance, is what makes Transformers so powerful. Without attention, the architecture would lack the flexibility to adapt to varying inputs. With this key mechanism now formalised and understood, we move on to the next fundamental component: the feed-forward network, which complements attention by enriching representations at each layer.

4.4 Feed-forward network

Neural networks have significantly evolved since the development of the earliest models. The initial concept aimed to construct an architecture capable of replicating human thought processes. From this foundation, researchers introduced the formal abstraction of a neuron, which has since become a fundamental building block in the field.

Definition 4.4.1: Neuron. An abstract *neuron* is defined as a quadruple (x, w, φ, y) where $x^T = (x_0, x_1, \ldots, x_n)$ denotes the extended input vector with $x_0 = -1$ used to incorporate the bias and $w^T = (w_0, w_1, \ldots, w_n)$ is the corresponding weight vector with $w_0 = -b$. The function φ is an activation function and the output of the neuron is computed as

$$y = \varphi\left(x^T w\right) = \varphi\left(\sum_{i=0}^{n} x_i w_i\right)$$

Using this formalism, a neuron can be understood as a unit designed to emulate the behaviour of its biological counterpart. The input vector x represents the incoming signals, the weights w simulate the synaptic strengths and the activation function φ mimics the firing mechanism of a real neuron.

In 1959, Frank Rosenblatt proposed the perceptron as an attempt to mathematically model the functioning of a biological neuron. The model was simple yet effective and can be formally described as follows:

Definition 4.4.2: Perceptron. A *perceptron* is a neuron whose input values are binary, that is $x_i \in \{0, 1\}$ and whose activation is defined by the Heaviside step function

$$\varphi(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

The output of the perceptron is computed using a threshold rule,

$$y = \varphi(x^T w - b) = \begin{cases} 0 & \text{if } x^T w - b < 0 \\ 1 & \text{otherwise} \end{cases}$$

Here w denotes the weight vector, x the binary input vector and b represents the threshold, which can be interpreted as a measure of inhibition.

Thus, a perceptron operates as a decision-making rule that weighs the evidence provided by its inputs. The threshold b determines the difficulty of producing an output of 1, acting

as a boundary that must be surpassed for the unit to activate.

The perceptron also admits a geometric interpretation. It defines an $(n-1)$-dimensional hyperplane in $\mathbb{R}^n$ given by

$$\mathcal{H} = \{x \in \mathbb{R}^n : x^T w = b\}$$

where $\mathcal{H}$ represents the decision boundary. The normal vector to this hyperplane is determined by the weights, namely $N^T = (w_1, w_2, \ldots, w_n)$. The hyperplane passes through a point p such that $b = p^T w$, where b corresponds to the bias. The output y of the perceptron assigns the value 1 to one of the half-spaces defined by $\mathcal{H}$ and 0 to the other.

Although simple and elegant, the perceptron suffers from a fundamental limitation when tackling more complex tasks. As a single neuron, its capacity for modelling non-linear relationships is severely restricted. A classic example illustrating this shortcoming is the inability of a perceptron to compute the `XOR` function.

To overcome these limitations more expressive architectures are required. This necessity gave rise to the concept of *neural networks*. These consist of multiple layers of interconnected neurons, where the outputs of one layer serve as the inputs for the next. The resulting structure resembles a web and is typically represented as follows

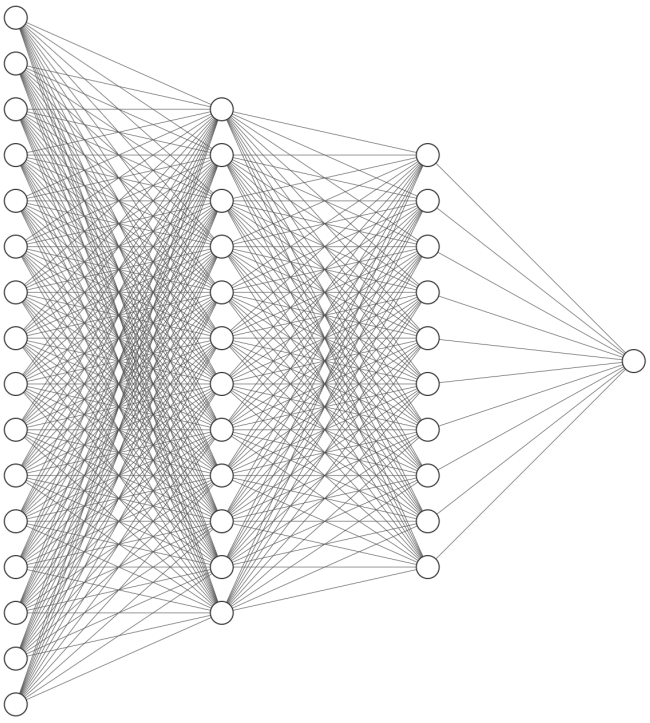

Figure 4.5: Visualization of a neural networks with two hidden layers.

The leftmost layer is known as the input layer, while the rightmost layer is referred to as the output layer. All layers positioned between these are called hidden layers. In the exam-

ple above there are two hidden layers, though in practice a network may include as many hidden layers as necessary to capture the complexity of the problem at the expense of increased computational resources. From this point forward, the total number of layers in the network will be denoted by L.

To formalise the notation, the synaptic connections between neurons are represented as edges in the network with each edge associated to a weight denoted by $w_{ij}^{(l)}$. Here the superscript l indicates the layer[1], the subscript i refers to the input neuron[2] and j corresponds to the output neuron. These three indices uniquely determine a specific connection within the network, specifying the layer it belongs to, its origin neuron and its destination neuron.

While the underlying computations can become intricate, the core principle is relatively intuitive: the output of each layer depends directly on the outputs of the previous one. This relationship can be concisely expressed using matrix notation

$$y^{(l)} = \varphi\left(W^{(l)} a^{(l-1)} + b^{(l)}\right)$$

where

1. $W^{(l)}$ is the weight matrix for layer l with dimensions $d^{(l)} \times d^{(l-1)}$.
2. Each row of $W^{(l)}$ corresponds to the connections from neurons in layer $l-1$ to a specific neuron in layer l.
3. $a^{(l-1)}$ is a column vector of size $d^{(l-1)} \times 1$ containing the activations from the previous layer.
4. $b^{(l)}$ is the bias vector for layer l of size $d^{(l)} \times 1$.
5. $y^{(l)}$ denotes the output of the current layer after applying the activation function φ.

This matrix formulation can be expanded into its element-wise form as

$$y_i^{(l)} = \varphi\left(\sum_{j=1}^{d^{(l-1)}} w_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)}\right)$$

where, again, i indexes the neurons in the current layer and j indexes those in the previous one. The activation of each neuron is thus a weighted sum of the activations from the previous layer, adjusted by a bias term and passed through a non-linear activation function φ.

The next step involves elevating the level of abstraction to describe neural networks in a more formal functional framework.

[1] The case $l = 0$ refers to the input layer.

[2] The case $i = 0$ typically represents the bias unit.

Definition 4.4.3: Real-valued functions. Let $\mathcal{F}(\mathcal{U}) = \{f : \mathcal{U} \longrightarrow \mathbb{R}\}$ denote the set of *real-valued functions* defined on the set $\mathcal{U}$.

Let $\mathcal{U}_l = \{1, 2, \ldots, d^{(l)}\}$ denote the index set corresponding to all neurons in layer l. Define the affine transformation

$$\begin{array}{rccc} \alpha_l : & \mathcal{F}(\mathcal{U}_{l-1}) & \longrightarrow & \mathcal{F}(\mathcal{U}_l) \\ & x^{(l-1)} & \longrightarrow & s^{(l)} \end{array}$$

which maps the output of layer $l-1$ to a new set of pre-activations $s^{(l)}$ at layer l.

The activations at layer l are then computed as

$$x^{(l)} = \left(\varphi^{(l)} \circ \alpha_l\right)\left(x^{(l-1)}\right)$$

where $\varphi^{(l)}$ is the non-linear activation function applied element-wise to the output of the affine transformation α_l.

With this setup, a feed-forward neural network can be formally defined as follows.

Definition 4.4.4: Feed-forward neural network. Let $\mathcal{U}_l = \{1, 2, \ldots, d^{(l)}\}$ for each $l \in \{0, 1, \ldots, L\}$. Consider a sequence of affine transformations $\alpha_1, \alpha_2, \ldots, \alpha_L$ such as $\alpha_l : \mathcal{F}(\mathcal{U}_{l-1}) \longrightarrow \mathcal{F}(\mathcal{U}_l)$ and a sequence of activation functions $\varphi^{(l)} : \mathbb{R} \longrightarrow \mathbb{R}$. Then, the corresponding *feed-forward neural network* is defined as a sequence of functions $f_0, f_1, \ldots, f_L$, where

$$f_l = \varphi^{(l)} \circ \alpha_l \circ f_{l-1}$$

for all $l \in \{1, 2, \ldots, L\}$, with f_0 given.

In this formulation, a deep feed-forward neural network constructs a sequence of progressively more abstract representations f_l by transforming the initial input f_0 through successive compositions of affine transformations α_l and non-linear functions $\varphi^{(l)}$.

The final output of the network is therefore given by the full composition

$$f_L = \varphi^{(L)} \circ \alpha_L \circ \varphi^{(L-1)} \circ \alpha_{L-1} \circ \cdots \circ \varphi^{(1)} \circ \alpha_1$$

A wide range of topics remain uncovered, as neural networks span an extensive landscape within machine learning. Subjects such as back-propagation, the mechanism through which gradients are propagated backwards through the network or gradient descent, the optimisation algorithm guiding the learning process, are fundamental to neural network training. Additional concepts such as weight initialisation, where naïve random strategies may not

always yield effective results, as well as activation functions, cost functions and advanced optimisers like Adam or AdaGrad are also critical components.

However, having now established a general understanding of how neural networks function, attention can be turned to a more profound and theoretical question: *why are these architectures capable of learning and making predictions at all?*

The answer lies in the Universal Approximation Theorem, first formally stated in [HSW90]. The purpose of the remainder of this section is to state and prove this remarkable result.

In general, when modelling a system, the goal is to find a function f that maps a set of inputs to an output. That is

$$f(x_1, x_2, \ldots, x_n) = y$$

where the x_i represent the n inputs and y is the corresponding output. The modelling problem is trivial when the function f is known, one can simply evaluate it for any input to obtain the desired output.

The difficulty arises when the function f is entirely unknown and only a collection of input-output pairs (x, y) is available. In this scenario neural networks become relevant. The Universal Approximation Theorem essentially states that any continuous function f can be approximated, to any degree of desired accuracy, by a neural network with a single hidden layer and a sufficiently large number of neurons.

The process of approximating this unknown function is referred to as the learning process. In what follows, the existence of such a learning process will be formally proved. However no explicit construction of the network's weights will be provided, as the goal here is to establish existence, not a specific implementation.

From this point forward the formal theory underlying this result will be introduced. While the notation may become more abstract, the core idea remains: neural networks are powerful because they can approximate arbitrary functions.

The first key result that must be stated, due to its fundamental role in what follows, is the Stone–Weierstrass theorem. Before stating the theorem, a few definitions are necessary.

Definition 4.4.5: Dense. Let $(\mathcal{S}, d)$ be a metric space and let $\mathcal{A} \subseteq \mathcal{S}$. The set $\mathcal{A}$ is said to be *dense* in $\mathcal{S}$ if for every element $g \in \mathcal{S}$ there exists a sequence $(f_n)_n$ of elements in $\mathcal{A}$ such that $\lim_{n\to\infty} d(f_n, g) = 0$.

Equivalently, this means that for every $\varepsilon > 0$ there exists an $N > 0$ such that

$$d(f_n, g) < \varepsilon$$

for all $n \geqslant N$.

Intuitively, the topological notion of density implies that the elements of the subset $\mathcal{A}$ can get arbitrarily close to any element in the space $\mathcal{S}$. In other words, the subset $\mathcal{A}$ is rich enough to approximate any function in $\mathcal{S}$ as closely as desired.

Definition 4.4.6: Algebra. A family $\mathcal{A}$ of real-valued functions defined on a set $\mathcal{S}$ is said to be an *algebra* if it is closed under addition, multiplication and scalar multiplication.

In other words, combining functions from the set $\mathcal{A}$ by adding them, multiplying them together or scaling them by real numbers will always produce a new function that also belongs to $\mathcal{A}$. This ensures that the set maintains a consistent algebraic structure under these standard operations.

Definition 4.4.7: Separated points. A family $\mathcal{A}$ of real-valued functions is said to *separate points* on $\mathcal{S}$ if for every pair of distinct elements $x, y \in \mathcal{S}$ there exists a function $f \in \mathcal{A}$ such that $f(x) \neq f(y)$.

This property implies that the family of functions is expressive enough to distinguish between any two different points in the domain. In a sense, if two inputs are not the same then the family always includes at least one function that can *notice* the difference.

Definition 4.4.8: Vanishes at no point. A family $\mathcal{A}$ of real-valued functions *vanishes at no point* of $\mathcal{S}$ if for every $x \in \mathcal{S}$ there exists a function $f \in \mathcal{A}$ such that $f(x) \neq 0$.

This means the set $\mathcal{A}$ never entirely goes quiet at any point in the domain. No matter which x is considered, at least one function in the family has a non-zero output there, making sure every point gets some attention.

Theorem 4.4.1: Stone-Weierstrass. Let $\mathcal{A}$ be an algebra of real-valued continuous functions defined on a compact set $\mathcal{K}$. If $\mathcal{A}$ separates points in $\mathcal{K}$ and vanishes at no point of $\mathcal{K}$ then $\mathcal{A}$ is dense in the space of all real-valued continuous functions on $\mathcal{K}$.

Essentially, this theorem states that under the given conditions the set $\mathcal{A}$ is rich enough to approximate any continuous function defined on the compact set $\mathcal{K}$ to any desired level of accuracy. This result provides a strong mathematical foundation for the idea that neural networks, with the right components, can learn to mimic virtually any behaviour or pattern.

The following definitions introduce mathematical objects and function classes that will be

useful for the upcoming results.

Definition 4.4.9: Set of affine functions. For any $n \in \mathbb{N}$ let $\mathcal{A}^n$ denote the *set of all affine functions* from $\mathbb{R}^n$ to $\mathbb{R}$. Explicitly,

$$\mathcal{A}^n = \{A : \mathbb{R}^n \longrightarrow \mathbb{R} : \ A(x) = w \cdot x + b\}$$

where $w, x \in \mathbb{R}^n$ are vectors, $\cdot$ denotes the dot product and $b \in \mathbb{R}$ is a scalar.

As introduced earlier, the vector x corresponds to the input of the network, w represents the weight vector connecting the input to the next layer and b is the bias term.

Definition 4.4.10: Borel map. A map $f : \mathcal{X} \longrightarrow \mathcal{Y}$ between two topological spaces is a *Borel map* if the pre-image $f^{-1}(\mathcal{I})$ is a Borel set for any open set $\mathcal{I} \subseteq \mathcal{Y}$.

Being Borel is an important property in real analysis, probability theory and integration theory as it guarantees that the function behaves well with respect to measurable sets. From now on only Borel functions will be considered. This is not a strong restriction since almost every function encountered in practical applications satisfies this condition. Non-Borel functions do exist, but they tend to be pathological.

Definition 4.4.11: $\sum$ class. For any measurable function $G : \mathbb{R} \longrightarrow \mathbb{R}$ and $n \in \mathbb{N}$ let

$$\sum^n(G) = \left\{ f : \mathbb{R}^n \longrightarrow \mathbb{R} : \ f(x) = \sum_{j=1}^{\infty} \beta_j G(A_j(x)) \right\}$$

where $x \in \mathbb{R}^n$, $\beta_j \in \mathbb{R}$ and $A_j \in \mathcal{A}^n$.

Definition 4.4.12: Squashing function. A function $\psi : \mathbb{R} \longrightarrow [0, 1]$ is a *squashing function* if it is non-decreasing and satisfies $\lim_{x \to \infty} \psi(x) = 1$ and $\lim_{x \to -\infty} \psi(x) = 0$

Definition 4.4.13: Class of network output functions. For any measurable function $G : \mathbb{R} \longrightarrow \mathbb{R}$ and $n \in \mathbb{N}$ let

$$\sum \prod^n(G) = \left\{ f : \mathbb{R}^n \longrightarrow \mathbb{R} : \ f(x) = \sum_{j=1}^{\infty} \beta_j \prod_{k=1}^{l_j} G(A_{jk}(x)) \right\}$$

where $x \in \mathbb{R}^n$, $\beta_j \in \mathbb{R}$, $A_j \in \mathcal{A}^n$ and $l_j \in \mathbb{N}$.

Definition 4.4.14: Set of continuous functions. Let $\mathcal{C}^n$ be the *set of continuous functions* from $\mathbb{R}^n$ to $\mathbb{R}$. Explicitly

$$\mathcal{C}^n = \left\{f : \mathbb{R}^n \longrightarrow \mathbb{R} : \ f \text{ is continuous}\right\}$$

Definition 4.4.15: Set of Borel measurable functions. Let $\mathcal{M}^n$ denote the *set of Borel measurable functions* from $\mathbb{R}^n$ to $\mathbb{R}$. Explicitly

$$\mathcal{M}^n = \left\{f : \mathbb{R}^n \longrightarrow \mathbb{R} : f \text{ is Borel measurable}\right\}$$

Definition 4.4.16: Borel σ-field. The *Borel σ-field* on $\mathbb{R}^n$ is denoted by $\mathcal{B}^n$.

The function classes $\sum^n(G)$ and $\sum\prod^n(G)$ are subsets of $\mathcal{M}^n$ for any Borel measurable function G. Furthermore, the set $\mathcal{C}^n$ is itself a subset of $\mathcal{M}^n$ and includes all functions of interest in the present work.

To quantify the similarity between functions f and g belonging to either $\mathcal{C}^n$ or $\mathcal{M}^n$, a metric ρ is used. The notion of one function class being able to approximate another is captured by the concept of denseness.

Definition 4.4.17: ρ-dense. A subset $\mathcal{S}$ of a metric space $(\mathcal{X}, \rho)$ is *ρ-dense* in a subset $\mathcal{T}$ if for every $\varepsilon > 0$ and every $t \in \mathcal{T}$ there exists $s \in \mathcal{S}$ such that $\rho(s, t) < \varepsilon$.

In simpler terms, elements of $\mathcal{S}$ can approximate any element of $\mathcal{T}$ to an arbitrary level of accuracy. From this point onwards, $\mathcal{T}$ and $\mathcal{X}$ are taken to be either $\mathcal{C}^n$ or $\mathcal{M}^n$, $\mathcal{S}$ represents either $\sum^n(G)$ or $\sum\prod^n(G)$ for specific choices of G and the metric ρ is selected accordingly.

Definition 4.4.18: Uniformly dense. A subset $\mathcal{S} \subset \mathcal{C}^n$ is said to be *uniformly dense* on compacta in $\mathcal{C}^n$ if for every compact subset $\mathcal{K} \subset \mathbb{R}^n$ the set $\mathcal{S}$ is $\rho_{\mathcal{K}}$-dense in $\mathcal{C}^n$, where, for $f, g \in \mathcal{C}^n$ we define

$$\rho_{\mathcal{K}}(f, g) = \sup_{x \in \mathcal{K}} |f(x) - g(x)|$$

Theorem 4.4.2: $\sum\prod^n(G)$ is uniformly dense. Let $G : \mathbb{R} \longrightarrow \mathbb{R}$ be any continuous non-constant function. Then $\sum\prod^n(G)$ is uniformly dense on compacta in $\mathcal{C}^n$.

In simpler terms, feed-forward networks of the form $\sum\prod$ are capable of approximating any real-valued continuous function on a compact set with arbitrary precision. This approximation property holds as long as the domain of the input variable x is bounded.

An especially appealing aspect of this result is that the activation function G can be any continuous non-constant function. It does not need to be a standard squashing function, although such functions are certainly admissible.

Lemma 4.4.1. The space $\mathcal{C}^n$ is ρ_μ-dense in $\mathcal{M}^n$ for any finite measure μ.

Lemma 4.4.2. Let $\{f_n\}$ be a sequence of functions in $\mathcal{M}^n$ that converges uniformly on compacta to a function f. Then $\rho_\mu(f_n, f) \to 0$.

Theorem 4.4.3: Universal approximation in $\sum\prod$. For every continuous function G, every $n \in \mathbb{N}$ and every probability measure μ on $(\mathbb{R}^n, \mathcal{B}^n)$, the class $\sum\prod^n(G)$ is ρ_μ-dense in $\mathcal{M}^n$.

In other words, single-hidden-layer $\sum\prod$ feed-forward networks can approximate any Borel measurable function to arbitrary precision. This holds irrespective of the choice of the continuous non-constant activation function G, the dimension r or the input distribution μ. In this precise and satisfying sense, $\sum\prod$ networks are universal approximators.

Lemma 4.4.3. Let F be a continuous squashing function and let ψ be any squashing function. For every $\varepsilon > 0$ there exists a function $H_\varepsilon \in \sum^n(\psi)$ such that

$$\sup_{x\in\mathbb{R}} |F(x) - H_\varepsilon(x)| < \varepsilon$$

Lemma 4.4.4. For every squashing function ψ, every $\varepsilon > 0$ and $M > 0$ there exists a function $\cos_M \in \sum^n(\psi)$ such that

$$\sup_{x\in[-M,M]} |\cos_M(x) - \cos(x)| < \varepsilon$$

Lemma 4.4.5. Let it be

$$g = \sum_{j=1}^{Q} \beta_j \cos(A_j)$$

where each $A_j \in \mathcal{A}^n$. For any squashing function ψ, compact set $\mathcal{K} \subset \mathbb{R}^n$ and $\varepsilon > 0$ there exists a function $f \in \sum^n(\psi)$ such that

$$\sup_{x\in\mathcal{K}} \left|g(x) - f(x)\right| < \varepsilon$$

Lemma 4.4.6. For every squashing function ψ the class $\sum^n(\psi)$ is uniformly dense on compacta in $\mathcal{C}^n$.

Theorem 4.4.4: Universal approximation in $\sum$. For every squashing function ψ, every $n \in \mathbb{N}$ and every probability measure μ on $(\mathbb{R}^n, \mathcal{B}^n)$ the class $\sum^n(\psi)$ is uniformly dense on compacta in $\mathcal{C}^n$ and ρ_μ-dense in $\mathcal{M}^n$.

In other words, standard feed-forward neural networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function arbitrarily well in the ρ_μ metric. This result holds regardless of the choice of squashing function ψ (as long as it is a squashing function), the dimension r of the input space or the underlying probability measure μ. Thus, the class $\sum$ is also a universal approximator.

Now that the central theorems have been established we can derive some corollaries that highlight more practical consequences of these results.

Corollary 4.4.1. For every function $g \in \mathcal{M}^n$ there exists a compact subset $\mathcal{K} \subset \mathbb{R}^n$ and a function $f \in \sum^n(\psi)$ such that for any $\varepsilon > 0$ we have $\mu(\mathcal{K}) < 1 - \varepsilon$ and for every $x \in \mathcal{K}$ we have $\left|f(x) - g(x)\right| < \varepsilon$ regardless of ψ, n or μ.

In other words, there exists a single hidden-layer feed-forward neural network capable of approximating any measurable function to any desired degree of accuracy on a compact subset $\mathcal{K}$ of input patterns, where this approximation holds on a set of measure arbitrarily close to 1.

Moreover, the Universal Approximation Theorem 4.4.4 and Corollary 4.4.1 extend naturally to the case of multi-output, multi-layer architectures $\sum_l^{r,s}(\psi)$, allowing approximation of functions in both $\mathcal{C}^n$ and $\mathcal{M}^{r,s}$. Therefore, neural networks of the class $\sum_l^{r,s}$ are universal approximators of vector-valued functions.

Having now established the theoretical foundations, we return to the motivation of this work. The original challenge arises in scenarios where the target function f is entirely unknown and only a collection of input-output pairs (x, y) is observed. The preceding results confirm that neural networks, as universal approximators, are well-suited to approximate such unknown functions. Hence, one can effectively model f using a neural network, rendering it no longer unknown in practice.

This perspective directly informed the construction of the embedding mapping E in Section 4.1. Recall that E was not defined analytically but rather learned—implicitly—through the training of a neural network. The approximation results above now justify this ap-

proach: the function E is well-defined in the functional sense as the setting satisfies all conditions required by the theorems.

Having established the expressive capabilities of feed-forward networks through the Universal Approximation Theorem it becomes clear why they play a pivotal role in contemporary deep learning architectures. In particular, feed-forward networks appear recurrently within Transformer models where they are applied position-wise following the multi-head attention mechanism. These intermediate components, commonly referred to as position-wise feed-forward layers, enable the model to introduce non-linearity and to refine the latent representations generated by the attention mechanism.

By applying an identical feed-forward transformation independently to each position in the input sequence, the Transformer preserves permutation equivariance, which is essential for handling sequential data without introducing positional bias. This architectural choice significantly enhances the model's ability to capture complex, high-level patterns across different positions in the input.

Therefore, the theoretical insights discussed in this section not only provide justification for the integration of feed-forward layers within the Transformer framework but also emphasise their fundamental contribution to the model's capacity to approximate intricate functions during both training and inference phases.

4.5 Activation function

Activation functions play a fundamental role in neural networks, enabling models to learn non-linear mappings between inputs and outputs. Without them, the network would behave as a linear system, regardless of its depth. These functions introduce the non-linearity required to approximate complex relationships within data.

In the final layer of the Transformer architecture, as described in [Vas+17], a linear transformation followed by a softmax activation is used. The linear layer maps the model output to the target dimension, while the softmax function transforms the result into a probability distribution over classes.

This section presents the key activation functions used throughout this thesis. The selection includes classical and widely adopted functions that enable both binary and multi-class decision-making.

4.5.1 Threshold step function

Inspired by the all-or-nothing behaviour of biological neurons, the threshold step function, also known as the Heaviside function, is defined as

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geqslant 0 \end{cases}$$

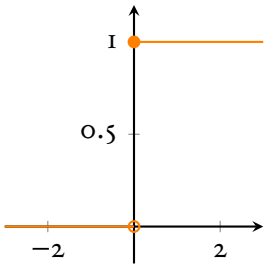

Figure 4.6: Plot of the Heaviside step function $H(x)$.

Although $H(x)$ is not differentiable at $x = 0$, it can be represented in the sense of distributions by the Dirac delta function, i.e., $H'(x) = \delta(x)$. Despite its simplicity, this function is rarely used in modern deep learning due to its lack of gradient information, which limits its applicability in gradient-based optimisation. However, it remains a theoretical foundation and is directly related to ReLU, as discussed below.

4.5.2 Rectified Linear Unit (ReLU)

One of the most prevalent activation functions in modern deep learning, ReLU is defined as

$$\mathrm{ReLU}(x) = \max\{x, 0\} = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geqslant 0 \end{cases}$$

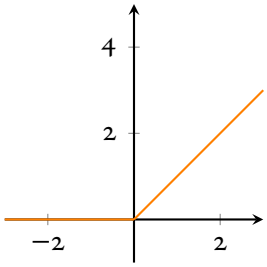

Figure 4.7: Plot of the ReLU function.

ReLU introduces non-linearity while preserving simplicity and computational efficiency. Its derivative is the Heaviside function, $\mathrm{ReLU}'(x) = H(x)$, which allows for efficient back propagation. Unlike sigmoid functions, ReLU does not saturate in the positive range, which mitigates the vanishing gradient problem and accelerates training.

4.5.3 Sigmoid function

The sigmoid function is a smooth, S-shaped curve mapping the real line to the interval $(0, 1)$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

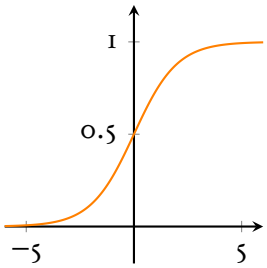

Figure 4.8: Plot of the sigmoid function $\sigma(x)$.

This function is commonly used in binary classification problems and models where outputs are interpreted probabilistically. The sigmoid satisfies the identity $\sigma(-x) = 1 - \sigma(x)$ and its derivative, useful in training, is $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

However, sigmoid activations can suffer from saturation and vanishing gradients for large input magnitudes, which has led to their reduced use in hidden layers of deep networks.

4.5.4 Hyperbolic tangent

The hyperbolic tangent is another sigmoid-like function, defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

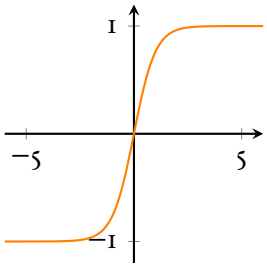

Figure 4.9: Plot of the hyperbolic tangent function $\tanh(x)$.

Unlike the sigmoid, $\tanh(x)$ maps inputs to the interval $(-1, 1)$ and is centred at zero, which can improve convergence in some architectures. The function is related to the sigmoid through the identity $\tanh(x) = 2\sigma(2x) - 1$. Its derivative is $\tanh'(x) = 1 - \tanh^2(x)$

4.5.5 Softmax function

Definition 4.5.1: Softmax. Let $x \in \mathbb{R}^n$ be a vector. The *softmax* function is defined as

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad \text{for } i = 1, \ldots, n$$

The softmax function maps a real-valued vector into a probability distribution over multiple classes. It ensures non-negative outputs that sum to one, making it ideal for multi-class classification tasks. In the Transformer architecture, it is used in the final layer to produce the model's class predictions.

Each of the activation functions described above plays a specific role depending on the task and location within the network. ReLU dominates modern deep learning architectures due to its simplicity and empirical performance, while sigmoid and tanh remain relevant in contexts where probabilistic interpretation or zero-centred outputs are beneficial. The softmax function is indispensable for classification, particularly in the output layer of models such as the Transformer.

Chapter 5

Race strategy simulator

With the theoretical framework now established, the focus shifts to the design and development of the race strategy simulator. This chapter aims to synthesise the content from the previous sections where an in-depth overview of Formula 1 strategy was provided in Chapter 2, the gap in the literature regarding real-time strategy simulation was discussed in Chapter 3 and the Transformer architecture was introduced and mathematically analysed in Chapter 4. The implementation presented here builds directly upon those foundations and it will often be necessary to refer back to the earlier chapters.

At a high level, the simulator is based on two specialised Transformer models: one designed to predict optimal pit stop timing (`PitStopTransformer`) and another dedicated to predict the most suitable tyre compound under given conditions (`CompoundTransformer`). Each model incorporates the architectural elements and mechanisms detailed in Chapter 4, tailored to the requirements of the strategy prediction task.

The model cannot be built in a vacuum. Therefore, this thesis relies on two key APIs to gather relevant data: Fast F1 and Open F1. The Fast F1 API provides comprehensive historical data, including lap-by-lap performance covering every race from 2018 to 2025. This rich dataset serves as a critical resource for training the models. The Open F1 API, while more limited in historical scope (available only since 2023), offers the advantage of real-time data access, which is essential for simulating live conditions and adapting strategies on the fly.

5.1 Data sources and variables

As previously mentioned, data for this project is collected using two primary sources: Fast F1 and Open F1. The dataset provided by these sources is extensive, though not exhaustive and the simulator must adapt to the type and quality of data available. This section outlines the variables employed by the model, which are selected based on the factors influencing race strategy as introduced in Section 1.

One of the most critical aspects when making strategic decisions in Formula 1 is the combination of lap time, tyre compound and tyre life. Lap time reflects the performance of a car per lap, the tyre compound indicates the type of tyres being used and tyre life represents the number of laps completed on the current set. These three variables form the cornerstone of any race strategy analysis and they are sufficient for constructing a basic simulator, especially when data availability is limited, as discussed in Chapter 2.

Tyre degradation is another fundamental parameter. Ideally one would utilise a direct measure of degradation, but this information is proprietary and not publicly released by teams. As a workaround, degradation is estimated indirectly through related variables such as telemetry data, lap times, weather conditions, tyre compound and tyre life. These proxies help model the wear experienced by tyres under varying race circumstances.

The car's fuel load would be highly beneficial for modelling degradation and performance, as it directly influences vehicle weight and lap times. Unfortunately, this information is also not publicly disclosed. Nevertheless, some inference can be made based on telemetry and lap data.

Telemetry is particularly useful for understanding whether a driver is pushing the car aggressively. Variables such as RPM, speed and throttle usage across a lap help infer driving style and, consequently, the stress placed on the tyres. Lap times are used as an auxiliary metric in the absence of detailed telemetry, providing a rough indication of performance intensity.

Weather conditions also play a vital role in tyre degradation. Factors such as air and track temperature, humidity, atmospheric pressure, rain and wind are all relevant. For instance, a hot day typically accelerates tyre wear, whereas lower track temperatures may prolong tyre life.

Strategic decisions are also influenced by the identity of the driver and the team. A highly skilled driver may execute complex or high-risk strategies more effectively than a less experienced one. Similarly, top-tier teams often have more refined decision-making tools and pit crew operations, which may allow for riskier but more rewarding strategies.

The lap number is another essential variable. Pitting at the beginning or very end of a race is generally suboptimal, so the strategy must consider the race phase. Therefore, lap count serves as a constraint when simulating or evaluating possible strategic choices.

Each Formula 1 circuit presents unique characteristics, from layout to typical weather patterns. However, beyond the circuit name, no specific technical data such as corner profiles, g-forces or downforce requirements is publicly available. Ideally, detailed circuit metadata would improve model accuracy, but in its absence, the simulator is designed to learn these nuances from historical data.

Qualifying results are another key input. By using the fastest laps from Q1, Q2 and Q3, the model gains an understanding of each car's peak performance under ideal conditions. Additionally, grid position is crucial, as it heavily influences race dynamics. A driver starting out of position may adopt an alternative strategy to gain track position, a common tactic for overtaking competitors.

As previously discussed, track status, especially during Safety Car and Virtual Safety Car periods, has a significant effect on pit stop timing. These scenarios often provide a cost-effective opportunity to change tyres and the model must take this increased pitting likelihood into account.

Time gaps to the leader and to adjacent cars offer valuable tactical information. When a driver is closely trailing a rival, strategic decisions such as the undercut or overcut may be viable. Consequently, gap data contributes directly to pit stop timing and compound selection.

Although the dataset is broad, several key parameters remain inaccessible due to confidentiality. These include tyre degradation metrics, fuel loads, engine modes and detailed car setups. This data scarcity introduces uncertainty, particularly when using Free Practice sessions to forecast race performance and generate pre-race strategies. In Practice sessions teams experiment with different setups, fuel levels and engine mappings, none of which are publicly disclosed. As a result, even if one driver appears significantly faster than another, it is unclear whether this is due to a lighter fuel load, a favourable setup or genuine performance superiority. This uncertainty complicates efforts to produce accurate predictions based solely on Free Practice data.

To conclude, it is essential to ensure that all data used for model training via the Fast F1 API is also available in real time through the Open F1 API during a race. Failing to do so would compromise the model's predictive capabilities when deployed in a live environment. This compatibility was thoroughly verified during development. Most variables used in the model meet this criterion. For those that do not, features were engineered from existing data, always within realistic constraints and avoiding speculation.

5.2 Data pipeline

As data is sourced from two distinct APIs and the ultimate goal is to feed this into the model, careful and precise data management is essential. This section outlines the complete data processing pipeline prior to inputting the data into the model.

Given that Transformer architectures process sequence data, race information is structured by laps. Specifically, each sequence is represented as an array of n values, where each value corresponds to a given attribute at a particular lap. Here n is set to a fixed value, with $n = 80$, accounting for the longest race on the calendar, Monaco, with 78 laps.

To illustrate, consider a driver whose lap times from lap 1 to 5 are: 91.220, 91.233, 90.956, 91.125 and 91.056 seconds and whose gap to the next driver in those same laps are: 0.529, 0.456, 0.320, 0.428 and 0.251 seconds. Each attribute is stored in a vector, where the value at position i corresponds to lap i. For example:

$$\begin{aligned}
\text{lap number} &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & \cdots \end{pmatrix} \\
\text{lap times} &= \begin{pmatrix} 91.220 & 91.233 & 90.956 & 91.125 & 91.056 & \cdots \end{pmatrix} \\
\text{gap to next} &= \begin{pmatrix} 0.529 & 0.456 & 0.320 & 0.428 & 0.251 & \cdots \end{pmatrix}
\end{aligned}$$

These are referred to as *lap-dependent features*, as their values vary across laps. In contrast, other features, such as driver, team, circuit, qualifying result, year and round number remain constant across the race. These are classified as *non lap-dependent features* and are processed as single, static values.

The full input to the Transformer is formed by concatenating lap-dependent and non lap-dependent features. Although the original Transformer architecture introduced by [Vas+17] was designed for text input, as discussed in Section 4.1, any structured input can be adapted. From this point onward, it is assumed that all input data is represented using integers. Categorical data such as driver names, teams, compounds or circuits are mapped to integers using lookup tables, which is a standard and straightforward transformation.

With the final data structure defined, the processing pipeline is split into two main stages:

1. **API to database**: In this step, raw data is retrieved from the respective APIs and stored in a `PostgreSQL` database for persistent and accessible storage.

   A dedicated script processes lap-by-lap race data, cleaning the dataset and applying necessary transformations, including type conversions, filtering irrelevant rows and computing derived values where appropriate.

Similarly, a separate script is used for telemetry data. Given that telemetry is recorded at high frequency (approximately every tenth of a second), the raw volume is considerable. To manage this, the mean value of each telemetry attribute is computed per lap and per driver. This ensures alignment between the telemetry and lap tables in terms of granularity.

A final script in this phase merges the processed lap and telemetry tables, carrying out post-processing tasks such as ordering columns, removing duplicates and performing integrity checks to ensure the dataset is consistent and ready for the next stage.

2. **Database to .tfrecord**: This stage retrieves the cleaned and merged data from the database and converts it into a `.tfrecord` file, the preferred format for `TensorFlow` pipelines. During this conversion both lap-dependent and non lap-dependent features are formatted according to the structure defined at the start of this section.

Each script in this pipeline presents specific challenges. The lap and telemetry scripts are responsible for managing API requests and thus must be robust and flexible enough to handle differences between the Fast F1 and Open F1 APIs. Data from both sources is standardised before being stored, ensuring that a user querying the `PostgreSQL` database cannot distinguish the origin of a given entry.

In contrast, the merging script is relatively straightforward, as it operates on data already stored locally in the database.

However, the final conversion script, from database to `.tfrecord`, is more complex. The structural differences between SQL and `.tfrecord` formats require careful manipulation to ensure data consistency and to avoid errors during training or inference phases.

During real-time prediction, two additional components become essential. The first is responsible for collecting the necessary information about the current lap and driver. Unlike offline data, live timing data is streamed in fragments as events occur, meaning that the full sequence is not available upfront. Therefore, before applying the complete data pipeline, a pre-processing step is required to gather and assemble relevant lap-level data. This file interacts with the Open F1 API, retrieves the real-time information and passes it along for further processing as previously described. Additional implementation details and examples can be found in Chapter 7.

The second component is in charge of retrieving data from the existing `.tfrecord` file and preparing it for use in inference. Specifically, it extracts the required features and formats them in the same way as was done during training, ensuring consistency between the model's training and deployment phases.

5.3 Transformer implementation

The model has been implemented entirely in Python, employing an object-oriented design to ensure modularity and clarity. This section outlines the implementation strategy, providing pseudocode for core components while intentionally omitting full code listings to preserve readability.

The architecture is structured into three principal scripts:

1. `transformer.py` contains the complete Transformer model which is subsequently used during testing to generate predictions.
2. `loss.py` defines a custom loss function to mitigate the class imbalance problem. Each class is weighted inversely to its frequency ensuring that less frequent events contribute more significantly to the loss.
3. `stopping.py` defines a custom callback that halts training once a specified accuracy threshold is reached, helping to optimise training time and resources.

The following subsections delve into the implementation details of each component.

5.3.1 Transformer

As mentioned previously, this class encapsulates the implementation of the Transformer architecture as introduced in [Vas+17]. The initial plan was to develop two separate classes for different predictions however, after extensive experimentation, it was determined that a single, deep Transformer capable of performing both tasks offered superior performance.

Earlier implementations reached sizes of approximately 600MB, rendering the model cumbersome, difficult to store and unsuitable for platforms with storage limitations (e.g., GitHub's 1GB limit). The current implementation has been significantly optimised, reducing the model size to approximately 40MB while simultaneously improving performance and portability.

For clarity and modularity each instance of the Transformer is assigned a distinct identifier: `PitStopTransformer` for pit stop prediction tasks and `CompoundTransformer` for tyre compound selection. Although both instances share an identical architecture, they differ in terms of internal weights, learned representations and task-specific performance metrics. As a result, it is both semantically and practically appropriate to distinguish them by name and treat them as separate entities.

The pseudocode for the Transformer model is presented below,

Transformer </>

```python
class Transformer(Layer):
    def call(self, inputs):
        lapDependent, mask, nonLapDependent = inputs

        # Input normalization
        lapDependent = self.lapDependentNormalizer(lapDependent)
        nonLapDependent = self.nonLapDependentNormalizer(nonLapDependent)

        # Input and positional encoding
        x = self.inputDense(lapDependent)
        pos = tf.range(self.maxSeqLength)[None, :]
        x  = x + self.positionalEncoding(pos)

        # Encoder stack
        for i in range(len(self.encoders)):
            # Multi-head attention
            attentionOut = self.encoders[i](x, x, mask)
            x = self.layerNorm1[i](x + attentionOut)

            # Feed-forward network
            ffnOut = self.ffns[i](x)
            x = self.layerNorm2[i](x + ffnOut)

        # Add non lap-dependent context
        nonLapDepenedentProjection = self.nonLapDependentProj(nonLapDependent)
        x = tf.concat([x, nonLapDepenedentProjection], axis=-1)
        x = self.finalDense(x)

        # BiLSTM and TimeDistributed for further processing
        x = self.biLSTM(x, mask)
        x = self.timeDense(x)

        # Output
        output = self.outDense(x, activation="softmax")

        return output
```

It is evident that this implementation closely resembles the architecture shown in Figure 4.1. The following list describes each of the components in detail:

1. Initially, the model receives its input and separates it into three components: lap-dependent data, a mask indicating which values should be considered and non-lap-dependent data. A critical first step involves normalising the inputs. Since training is conducted on normalised data to improve convergence, it is essential that the same normalisation be applied during inference to avoid invalid predictions.

   Early versions of the model introduced a standalone `Normalization` class to handle

this but the approach proved inefficient in terms of storage and practicality. The current design integrates the normalisation layers directly into the `Transformer` class allowing them to be recovered automatically during training. This not only simplifies the model usage but also enhances its flexibility and reusability as data can be directly fed into the model without requiring prior preprocessing.

2. After normalisation the input embedding and positional encoding are applied, as described in Section 4.1 and Section 4.2. These steps embed the input data into a richer representation space while encoding temporal structure via position.

3. The encoded data then passes through the encoder stack. This mirrors the standard Transformer encoder block. Each encoder layer comprises a multi-head attention mechanism followed by residual connections and layer normalisation. The result is then passed through a feed-forward neural network with a second residual connection and normalisation.

   Each of these components serves a specific role: attention mechanisms capture global dependencies across the sequence, feed-forward networks inject non-linearity and layer normalisation promotes numerical stability and accelerates convergence. This multi-layer design is central to the model's effectiveness on sequential data.

   The use of masking is fundamental in this context, as it prevents the attention mechanism from accessing future information. Without it, the model would be able to "see" data from upcoming laps when making predictions about the current one, which is unrealistic and violates the temporal nature of real-world racing scenarios, where future events are inherently unknown and cannot be used for decision-making.

   Initial versions of the model lacked proper masking and, although they yielded highly accurate results with full race data, these were misleading. The model exploited future information, leading to overfitting. When tested under real-time conditions predictions became erratic, underscoring the need for causal masking to ensure realistic and reliable outputs.

4. After the encoder stack, the non-lap-dependent data is reintegrated. It is projected to the appropriate shape and concatenated with the encoder output. This enriched representation is then processed using bidirectional Long Short-Term Memory (BiLSTM) layers followed by TimeDistributed layers, enabling further temporal refinement of the sequence.

5. Finally, the model output is produced via a dense layer with a softmax activation function (see Definition 4.5.1). This generates a probability distribution across the output classes, providing the final prediction.

5.3.2 Loss

The dataset exhibits a clear class imbalance. This is particularly evident when considering the number of laps remaining until the next pit stop. Intuitively, if at a given lap there are n laps left to pit then at the following lap there will be $n - 1$, and so forth. As a result, it is straightforward to observe that the value 0 appears most frequently, followed by 1, then 2 and so on. Figure 5.1 illustrates this distribution in detail, highlighting the skewed frequency of laps remaining until a pit stop. Similarly, the distribution of tyre compounds is also unbalanced, as shown in Figure 5.2.

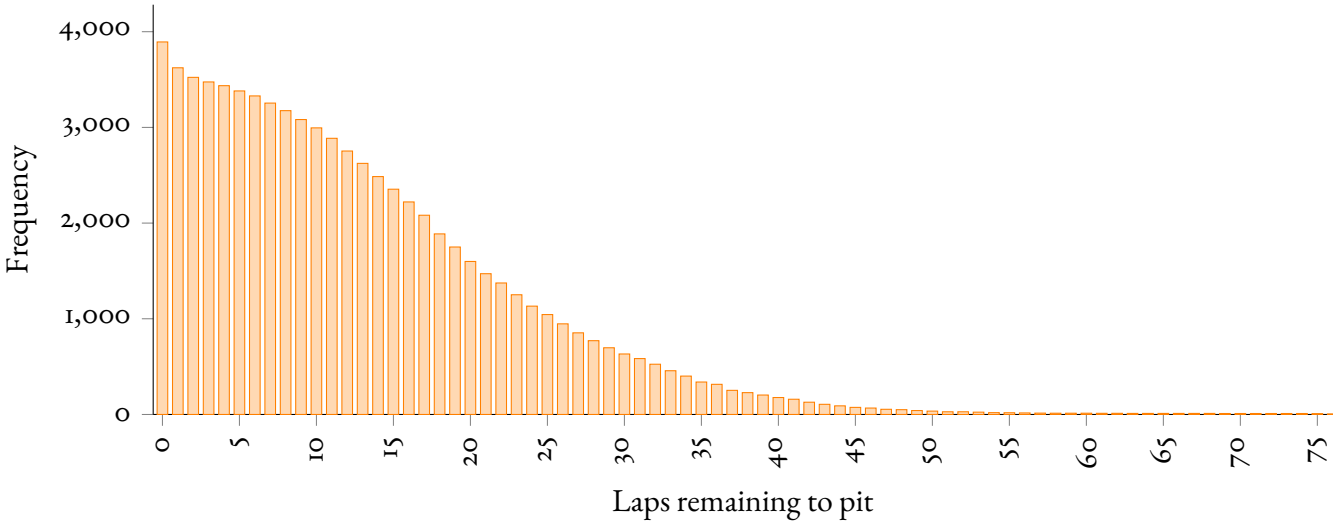

Figure 5.1: Frequency of laps until next pit

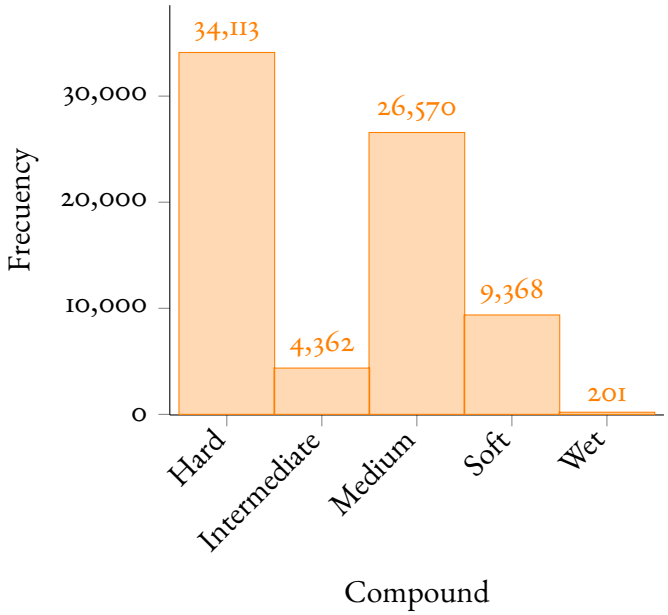

Figure 5.2: Frequency of compounds

To mitigate the effects of this imbalance, a custom loss function has been implemented via the `WeightedSequenceLoss` class. Its structure is as follows

WeightedSequenceLoss </>

```
class WeightedSequenceLoss(Loss):
    self.weights = tf.constant(weights, dtype = tf.float32)

    def call(self, yTrue, yPred):
        # Convert true labels to one-hot encoding
        yTrueOneHot = tf.one_hot(yTrue, depth = tf.shape(yPred)[-1])

        # Compute sample-specific weights
        sampleWeights = tf.reduce_sum(self.weights * yTrueOneHot, axis=-1)

        # Compute standard categorical crossentropy
        loss = sparse_categorical_crossentropy(yTrue, yPred)

        # Apply weights and return average loss
        return tf.reduce_mean(loss * sampleWeights)
```

This class receives a set of weights, typically calculated during the training phase, which assigns higher importance to under-represented classes. The true labels are one-hot encoded and the corresponding weight is extracted and applied to the loss values. The use of `sparse_categorical_crossentropy` allows efficient computation while preserving the original label format. By applying class-based weighting to the loss, the model is encouraged to learn under-represented patterns more effectively, thereby mitigating the effects of imbalance.

5.3.3 Stopping

The `EarlyStoppingByAccuracy` class implements a simple yet effective early stopping mechanism based on a user-defined accuracy threshold. Its implementation is as follows

EarlyStoppingByAccuracy </>

```
class EarlyStoppingByAccuracy(Callback):
    def on_epoch_end(self, logs = None):
        # Retrieve monitored accuracy metric
        accuracy = logs.get(self.monitor)

        # Stop training if threshold is reached
        if accuracy is not None and accuracy >= self.value:
            self.model.stop_training = True
```

During model training, it is often necessary to monitor specific performance metrics to de-

termine whether the learning process is progressing as expected. This callback is designed to track such a metric, most commonly `sparse_categorical_accuracy`, as defined by `self.monitor`. At the end of each epoch, the callback checks whether the current accuracy has surpassed a predefined threshold (`self.value`). If the condition is met, training is halted early. This approach not only saves computational resources but can also help prevent overfitting by stopping the learning process once satisfactory performance has been achieved.

A crucial aspect, not explicitly shown in the code above, is that every custom class must be decorated with `@register_keras_serializable()`. This decorator ensures that the model is fully serializable and can later be recovered using `load_model()`. Without this, although the model can be trained and saved without issues, it becomes irrecoverable when attempting to reload it for inference.

Moreover, it is essential that all model inputs are correctly defined and stored. Failure to do so can lead to inconsistencies during deserialization. To address this, each class should implement a `get_config()` method, which explicitly defines the model configuration and facilitates seamless saving and loading of models.

Using these custom classes, in conjunction with the data pipeline described in Section 5.2, enables efficient training of the models. The dataset used comprises lap-by-lap data for every driver across all Formula 1 races from 2022 through 2024. This dataset is split using a standard 70/15/15 ratio for training, validation and testing, respectively.

While the training process is conceptually straightforward once the codebase is properly configured, it posed significant practical challenges. The computational demands exceeded the capabilities of a standard personal computer.

Given the intensive nature of the computations, CPU-based training proved ineffective and the onboard GPU was insufficient for the task. The most viable solution was to offload training to Google Colab, which provides access to NVIDIA T4 GPUs. This significantly accelerated the training process compared to local resources.

However, the use of Google Colab came with its own limitations. The platform imposes usage caps on GPU access and sessions can be terminated unexpectedly without prior notice. To circumvent these restrictions and allow for extensive experimentation with different configurations and hyper-parameters, a practical workaround was employed: creating 15 separate Google accounts. These accounts shared a common Google Drive folder containing the complete project codebase. This setup provided a flexible and scalable environment for experimentation, effectively bypassing GPU usage constraints and enabling parallel testing.

Chapter 6

Results

The most effective way to assess the success of the training process is through a careful examination of the resulting metrics. This chapter presents the theoretical outcomes and provides a detailed walk-through of the entire training process highlighting the evolution of the model over time. It also addresses the main challenges encountered during development and concludes with an analysis of the final results.

It is important to emphasise that the model underwent continuous refinement throughout this process. Consequently, any variations in performance metrics reflect changes in the model's structure, data processing pipeline or training strategy.

Initially, model performance was primarily evaluated using accuracy. The early stages of development were thus centred around improving this metric. The evolution of the model and its corresponding metrics are summarised below:

1. The first training run yielded approximately 50% accuracy in both predictions, with a loss value of around 1.7. Although far from ideal, this result was acceptable for a first benchmark and provided a useful starting point.

2. By modifying the labelling approach and the way the data was presented, the model's accuracy in predicting the tyre compound significantly improved, reaching 88.89% with a corresponding loss of 0.4995. However this improvement came at the cost of

the pit stop prediction which dropped to 27.61% accuracy and a loss of 1.4516. This indicated that the model was learning to predict one feature effectively but was neglecting the other.

3. Further data preprocessing focused on the pit stop feature while maintaining the improvements made to the compound predictions. As a result the compound accuracy remained largely stable while pit stop accuracy increased to 44.26%, albeit with a higher loss of 1.8588. This was seen as a step forward as the compound accuracy remained high while the pit stop prediction improved.

4. Incorporating LSTM layers into the Transformer architecture provided a notable boost in pit stop prediction performance. While the compound accuracy remained consistent the pit stop accuracy increased significantly to 70.98% and the loss decreased to 0.7206. This configuration represented the best results obtained up to that point.

5. A key breakthrough occurred when the model architecture was modified to use two separate Transformers instead of a single shared one. Initially a single Transformer was responsible for predicting both outputs, meaning that both tasks shared the same weights and internal structure. By splitting the model into two distinct Transformers, one dedicated to compound prediction and the other to pit stop timing, results improved considerably. The compound prediction accuracy increased to 91.04% and the pit stop accuracy soared to an impressive 95.15%.

6. In an attempt to further refine the model's performance a custom training callback was developed. Its goal was to halt the training process once the model reached a predefined accuracy threshold. The training was allowed to run for up to 1000 epochs, ensuring that the model had sufficient time to reach these targets. The target accuracies were set at 99.1% for pit stop prediction and 96.2% for compound prediction. These values were inspired by the television series *Breaking Bad*, in which the character Walter White achieves a 99.1% purity in his product and Jesse Pinkman later reaches 96.2%, reflecting his growth and mastery. While anecdotal, this reference adds a personal and motivational touch to the experimentation process.

At this stage, the obtained metrics suggested that the model was performing exceptionally well in theory. However, real-world testing painted a different picture. When evaluated on pre-constructed datasets the model delivered outstanding results. Yet, when applied to live data streams, simulating the lap-by-lap progression of an actual race, the predictions deteriorated significantly.

Upon close inspection, the issue was traced back to the masking mechanism in the Transformer block. The model had been inadvertently allowed to access future information during training due to the absence of appropriate causal masking. This meant the Transformer

could "see" beyond the current time step, leading to artificially inflated performance on full datasets. However, during live data injection, where the future is unknown by design, the model's performance collapsed. The solution was straightforward: integrate causal masking into the multi-head attention layers and ensure proper sequence handling within the LSTM blocks.

Once this issue was resolved, a new problem emerged. The model began producing constant predictions, meaning that it always predicted the same outcome, irrespective of the race context. This behaviour was traced to the data normalisation step. During training, input data had been normalised but this preprocessing was not replicated during live inference. As a result the model received inconsistent inputs and, despite this, still attempted to deliver reasonable predictions. The fix involved embedding the normalisation logic directly within the Transformer model itself, as discussed in Section 5.3.

With these adjustments in place performance improved considerably. However, yet another issue surfaced during real-time testing with sliced, sequential input data. Two main anomalies were observed:

1. The model consistently predicted that a pit stop should occur in the next lap, i.e., the number of laps remaining until the next stop was always zero.

2. The model exhibited a lack of temporal consistency in its predictions. That is, predictions made at consecutive laps were not coherent with one another. For instance, a prediction at lap n might suggest that there are x laps remaining until the next pit stop, but the prediction at lap $n + 1$ would then indicate $x + 1$ laps remaining. This counter-intuitive behaviour implies that the model was not effectively integrating new information as the race progressed. Instead it appeared to shift its entire prediction horizon forward by one lap regardless of the updated context.

In summary, the model appeared to generate a reverse succession of values decreasing to zero. That is, if the model predicted 0 at lap n, then at lap $n - 1$ it would predict 1, at $n - 2$ it would predict 2 and so on, forming a decreasing sequence. Consequently, when using $n + 1$ laps as input instead of n, all values shifted upwards by one, thus introducing a systematic offset.

This challenge proved to be the most difficult to address. The root cause lay in the nature of the model's output. At the time the Transformer was designed to return a full sequence of prediction, one for each lap, indicating how many laps remained until the next pit stop. Since the model had access to the entire sequence during training, it learned to anchor the final value at 0 (representing the pit stop) and then inferred prior values in reverse. As a result, the output was essentially a descending line from $(0, n)$ to $(n, 0)$.

The key insight was to reformulate the problem. Rather than predicting the entire sequence, the model should instead output a single integer representing the number of laps remaining until the next pit stop based on the current state of the race. By focusing the output in this way, compressing the prediction into a single point rather than a sequence, the model became far more robust in live real-time scenarios.

After this modification, the model's behaviour improved significantly producing plausible and consistent predictions. From this point onward no further architectural changes were made to the Transformer model. The final version yielded the following performance metrics.

The overall training parameters and outcomes are summarised below

Prediction	Accuracy	Loss
Pit stop	0.9249	0.3325
Compound	0.9820	0.0856

Table 6.1: Final accuracy and loss for each prediction type.

For additional granularity, the performance across training, validation and test datasets is shown in Table 6.2

Prediction	Dataset	Accuracy	Loss
Pit stop	Train	0.9831	0.0762
	Validation	0.9255	0.3287
	Test	0.9249	0.3325
Compound	Train	0.9832	0.0715
	Validation	0.9823	0.0873
	Test	0.9820	0.0856

Table 6.2: Accuracy and loss for each prediction task across the training, validation and test splits.

These results reflect a high level of performance. The slightly better accuracy in compound prediction is due to the nature of the task: compound prediction is a 5-class classification problem (Hard, Medium, Soft, Intermediate and Wet), whereas pit stop prediction spans 31 discrete classes (corresponding to laps), making it inherently more challenging. Even so, achieving such metrics in a 31-class problem is particularly noteworthy.

Let us now evaluate the performance of the pit stop model in greater depth. A classification report was generated to summarise how well the model performs across each class.

Class	Precision	Recall	F1-score	Support
Class 0	0.94	0.92	0.93	604
Class 1	0.90	0.90	0.90	536
Class 2	0.91	0.92	0.92	556
Class 3	0.95	0.92	0.94	535
Class 4	0.95	0.94	0.94	516
Class 5	0.95	0.94	0.95	504
Class 6	0.96	0.95	0.95	468
Class 7	0.95	0.94	0.95	514
Class 8	0.92	0.94	0.93	492
Class 9	0.93	0.94	0.94	454
Class 10	0.93	0.95	0.94	437
Class 11	0.94	0.92	0.93	447
Class 12	0.90	0.94	0.92	397
Class 13	0.92	0.92	0.92	396
Class 14	0.94	0.93	0.93	403
Class 15	0.94	0.95	0.94	333
Class 16	0.93	0.94	0.94	315
Class 17	0.94	0.92	0.93	297
Class 18	0.93	0.92	0.93	309
Class 19	0.88	0.93	0.90	270
Class 20	0.87	0.86	0.87	241
Class 21	0.84	0.90	0.87	220
Class 22	0.91	0.89	0.90	206
Class 23	0.93	0.88	0.90	189
Class 24	0.88	0.94	0.91	158
Class 25	0.84	0.92	0.88	133
Class 26	0.92	0.87	0.90	148
Class 27	0.89	0.89	0.89	123
Class 28	0.84	0.88	0.86	105
Class 29	0.82	0.66	0.73	101
Class 30	0.95	0.96	0.95	784

Table 6.3: Classification metrics for the `PitStopTransformer` across all 31 classes.

In this table, precision represents the proportion of predicted instances for a given class that were correct, recall denotes the proportion of actual instances correctly predicted and the F1-score is the harmonic mean of both. The support indicates the number of true occurrences of each class in the test set.

Overall, the `PitStopTransformer` performs robustly across most classes, particularly those with higher support. Naturally, performance tends to decline slightly for less frequent classes, which is a common effect in multi-class classification due to imbalanced data.

In addition to classification metrics, other valuable indicators include the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). MAE captures the average abso-

lute difference between predicted and true class indices, while RMSE penalises larger errors more heavily. For this model, the MAE is 0.2535 and the RMSE is 1.8230.

Another insightful metric is the top-k accuracy which assesses whether the true label appears within the top k predictions. The top-2 accuracy is 0.9704 and the top-3 accuracy reaches 0.9808, indicating that even when the exact class is missed the model's alternative predictions remain highly relevant.

Calibration quality is assessed using the Expected Calibration Error (ECE) which evaluates how well the predicted probabilities reflect true likelihoods. Lower values indicate better calibration. The ECE for the pit stop model is 0.0122, suggesting that the model's confidence scores are highly reliable.

For visual inspection, the confusion matrix offers a detailed view of prediction versus ground truth

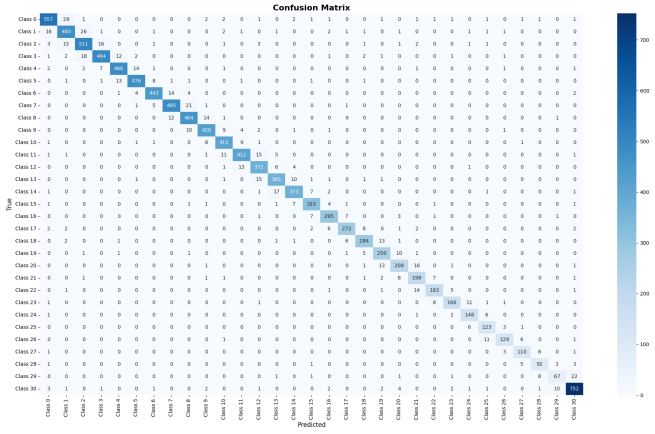

Figure 6.1: Confusion matrix for the `PitStopTransformer`.

The strong diagonal pattern indicates accurate predictions, where the predicted class matches the true class. Occasional off-by-one errors are visible and are naturally captured by the top-k metrics. In some cases, seemingly extreme mismatches such as predicting class 23 when the true value is 0 may appear, but these can often be explained contextually. For instance, such offsets may occur when the model anticipates a pit stop slightly earlier than it actually happens. Since the classes represent a circular concept (laps until stop) such cyclical behaviour may arise and still align with race dynamics.

A normalised version of the confusion matrix further highlights the strength of the diagonal

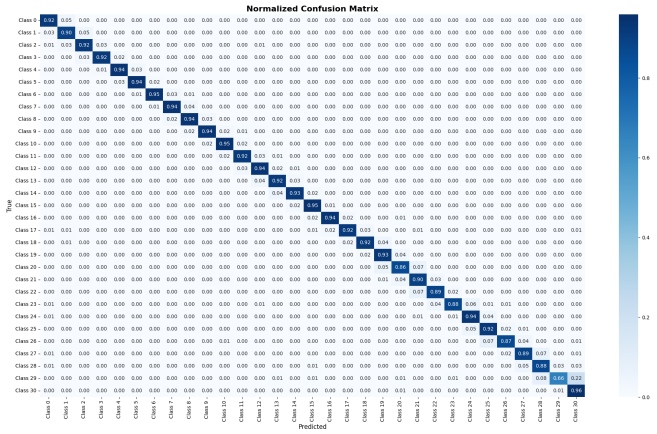

Figure 6.2: Normalised confusion matrix for the `PitStopTransformer`.

To conclude the pit stop evaluation, calibration curves are presented to visually assess how well the model's predicted probabilities align with true outcomes

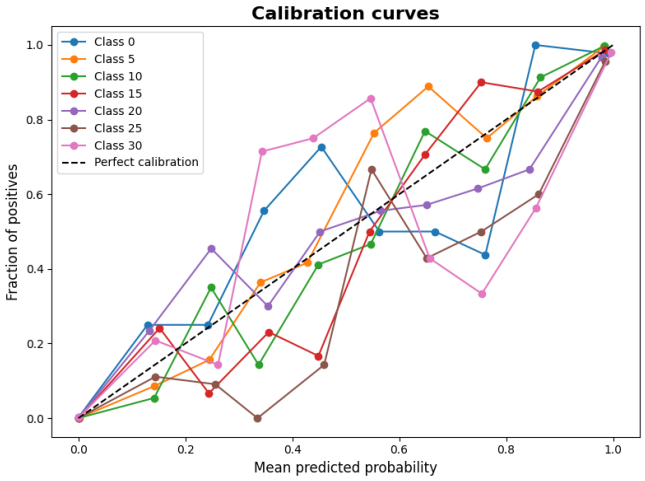

Figure 6.3: Calibration curves for the `PitStopTransformer`.

As expected, the closer the calibration line is to the diagonal, the better calibrated the model is, which is certainly the case here.

Turning now to the `CompoundTransformer`, we observe the following classification report

Class	Precision	Recall	F1-score	Support
Class 0	0.99	0.99	0.99	5086
Class 1	0.96	0.97	0.96	660
Class 2	0.98	0.98	0.98	4027
Class 3	0.97	0.98	0.98	1397
Class 4	0.43	0.90	0.58	21

Table 6.4: Classification metrics for the `CompoundTransformer` across all 5 classes.

The model demonstrates strong performance across all compound classes, with particularly high precision and recall for the most common tyre types. The exception is Class 4, which corresponds to the Wet compound. Due to its low support in the dataset (only 21 examples), performance on this class is notably lower, a typical outcome in the presence of data imbalance.

Additional metrics offer further insights into the model's reliability. The top-2 accuracy is 0.9985 and the top-3 accuracy reaches 0.9996, indicating that even when the model's top prediction is incorrect, the correct class is nearly always among its top alternatives.

In terms of calibration, the Expected Calibration Error (ECE) is 0.0039, suggesting a very strong alignment between the predicted probabilities and actual outcome frequencies.

The confusion matrix shown in Figure 6.4 provides a visual summary of how well the model differentiates between tyre compounds.

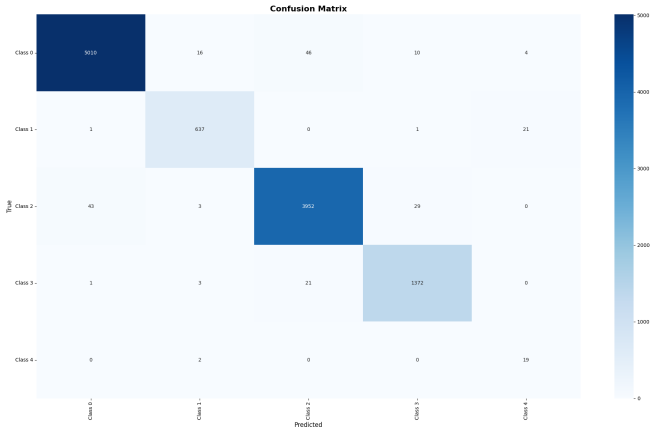

Figure 6.4: Confusion matrix for the `CompoundTransformer`.

It is instructive to examine the nature of the misclassification. These errors generally make sense within the domain context. For instance, Class 1 (Intermediate tyres) is occasionally misclassified as Class 4 (Wet tyres). This is reasonable given that both compounds are intended for wet weather conditions and the distinction can be subtle based on track humidity, rainfall intensity or temperature.

Similarly, Class 2 (Medium tyres) is sometimes confused with either Class 0 (Hard) or Class 3 (Soft), but not with the wet-weather compounds. A similar pattern is observed for Class 3 (Soft tyres), which may occasionally be predicted as Mediums, but is rarely, if ever, classified as a Wet or Hard compound. Likewise, Intermediate and Wet compounds are never confused with slick compounds, when misclassified, they tend to be mistaken for each other, which again reflects a reasonable error profile given the similarity in conditions under which they are used.

To further clarify the model's performance across all categories, the normalised confusion matrix is shown below

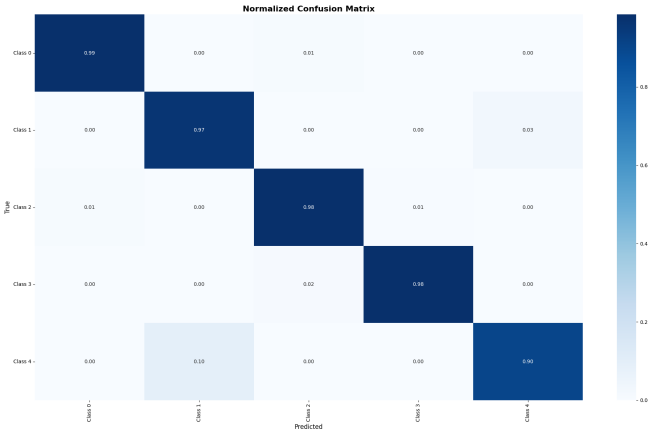

Figure 6.5: Normalised confusion matrix for the `CompoundTransformer`.

Finally, the calibration curves further confirm that the model's predicted probabilities are well-aligned with observed outcomes

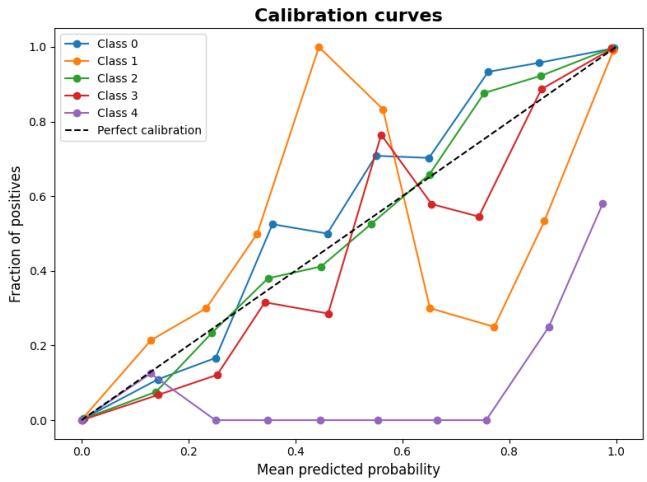

Figure 6.6: Calibration curves for the `CompoundTransformer`.

To conclude this section, it is insightful to examine the attention matrices of the models.

In Section 4.3, the attention mechanism was explained in detail. Now, this mechanism can be visualised to better understand what each model "sees" during the prediction process.

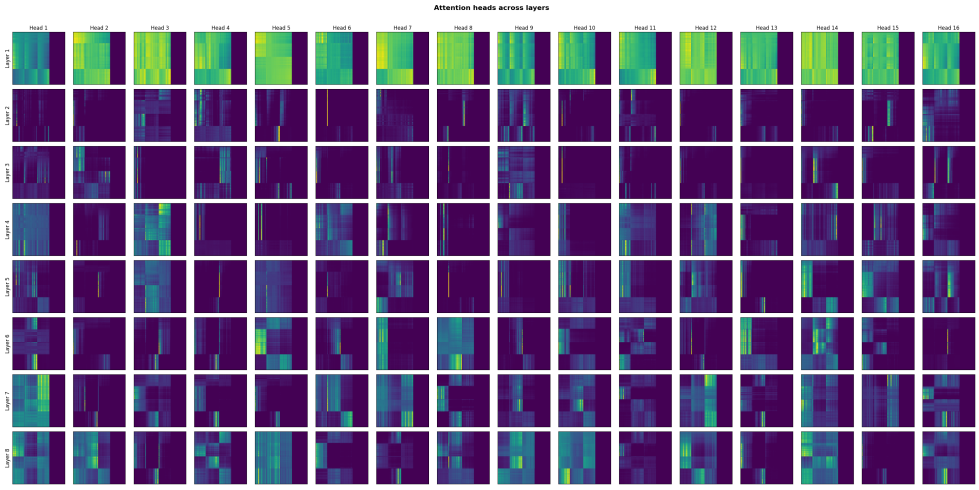

Figure 6.7: Attention matrices for the `CompoundTransformer`.

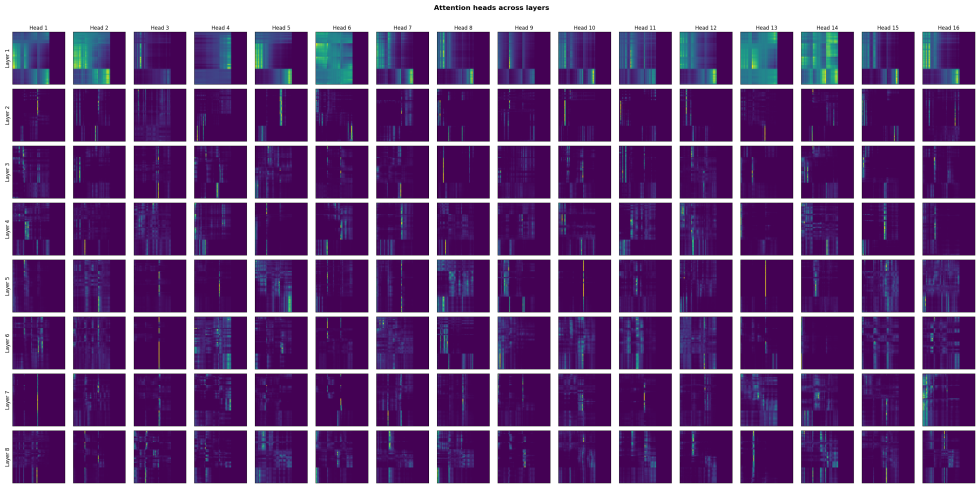

Figure 6.8: Attention matrices for the `PitStopTransformer`.

In these heat-maps, attention values are visualised across all 16 heads and 8 layers of the model architecture. Brighter areas indicate higher attention weights, while darker regions reflect lower attention. Each square in the grid represents an individual attention head at a specific layer, providing insight into how information flows through the Transformer.

A particularly interesting observation can be made in the first layer of the compound model, where many heads assign relatively high attention scores across a broad input range. This phenomenon is partly due to the nature of self-attention at the earliest stage of processing, these heads are still aggregating broad contextual information. Additionally, towards the lower right of each attention map, darker regions appear consistently. This is a direct consequence of the attention masking mechanism applied during sequence modelling. Once

the model reaches the lap currently being predicted, all subsequent inputs are masked out, resulting in zero attention beyond that point, effectively simulating future ignorance.

Another notable detail is how different heads appear to specialise in distinct patterns or structural behaviours. Some focus on local information (i.e., strong diagonal or near-diagonal attention), while others capture long-range dependencies or global interactions, such as sudden attention spikes to distant input positions. This diversity is a hallmark of multi-head attention and demonstrates how the model decomposes the task into subtasks distributed across layers and heads.

Altogether, these visualisations confirm that both models learn structured and interpretable attention patterns, with specific heads focusing on unique aspects of the input. This qualitative insight supports the theoretical architecture design and provides further confidence in the model's internal consistency.

With these results, the theoretical validation phase of the model is considered complete. The next step involves deploying the models in live scenarios to enable real-time inference and dynamic race strategy simulation.

Chapter 7

Real race simulation

The final stage of testing for both Transformer models focuses on real-time prediction during an actual Formula 1 race. This section aims to provide insight into the results obtained when deploying the models in a live environment.

As previously mentioned, the simulator outputs, for each lap, the number of laps left before an optimal pit stop and the recommended tyre compound. These results are visualised in a graph where the x-axis corresponds to the lap number, the y-axis shows the number of laps until the next stop and colour encodes the suggested tyre compound. Gradient lines are used to indicate transitions between compounds.

To assess model performance, a real-world scenario was evaluated using Max Verstappen's race at Suzuka, an iconic track and driven by arguably the most consistent performer on the grid. The predicted race strategy is shown in Figure 7.1

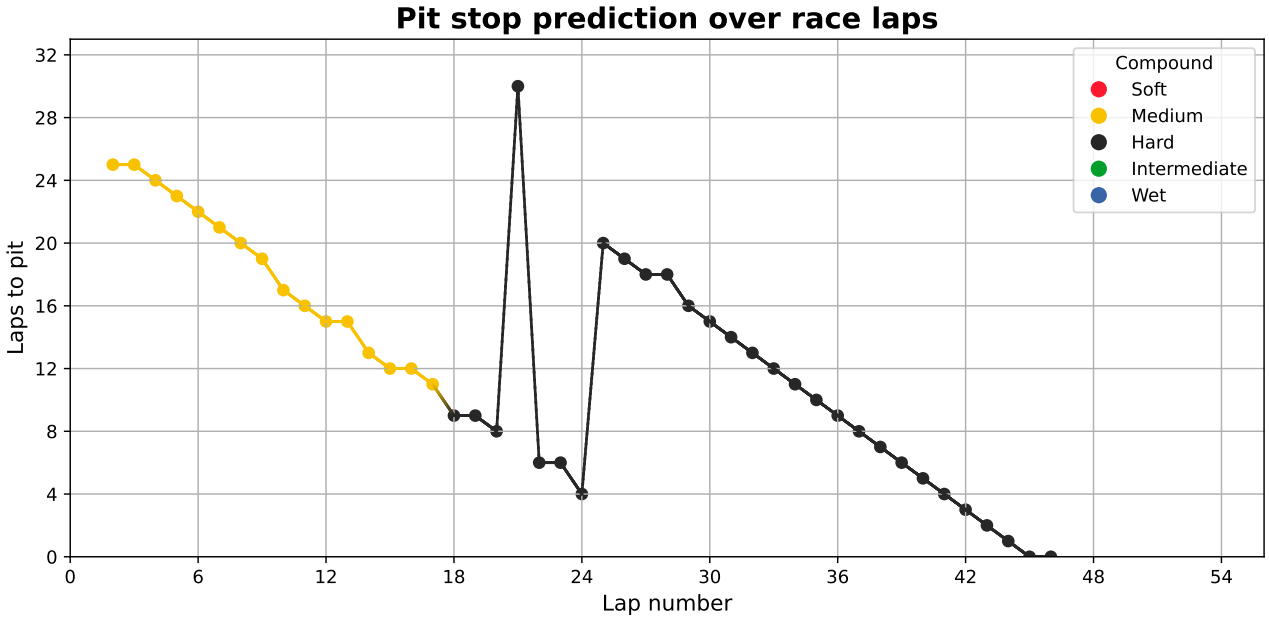

Figure 7.1: Max Verstappen's race strategy prediction for the 2025 Japanese Grand Prix.

Several key insights can be drawn from this visualisation:

1. The model suggests starting the race on medium tyres and switching to hard tyres at lap 18.

2. It correctly identifies lap 18 as the optimal point for the tyre change, accounting for all contextual input features and external variables.

3. The predicted laps-to-pit peak at lap 21, which aligns with the actual pit stop, suggesting a high level of predictive accuracy.

4. The plot exhibits the expected behaviour: a downward-sloping diagonal of constant colour, reflecting a consistent reduction in laps to pit over time.

5. Although the real pit stop occurred on lap 21, the model continued decreasing the laps to pit prediction until lap 25, where it reassessed and determined the current tyres could still be optimal.

6. The absence of alternative compounds (e.g. soft or intermediate) suggests that the model has filtered out suboptimal strategies, showing strong confidence in the selected compound.

7. The post-pit stint displays high compound stability, further validating the model's inference under typical dry conditions.

Ideally, the model would change compounds (i.e., colour change) exactly when the predicted laps to pit reach zero and then begin a new segment with a different colour. This ideal behaviour is better observed in the simulation for Oscar Piastri during the 2025 Italian Grand Prix at Imola (Figure 7.2)

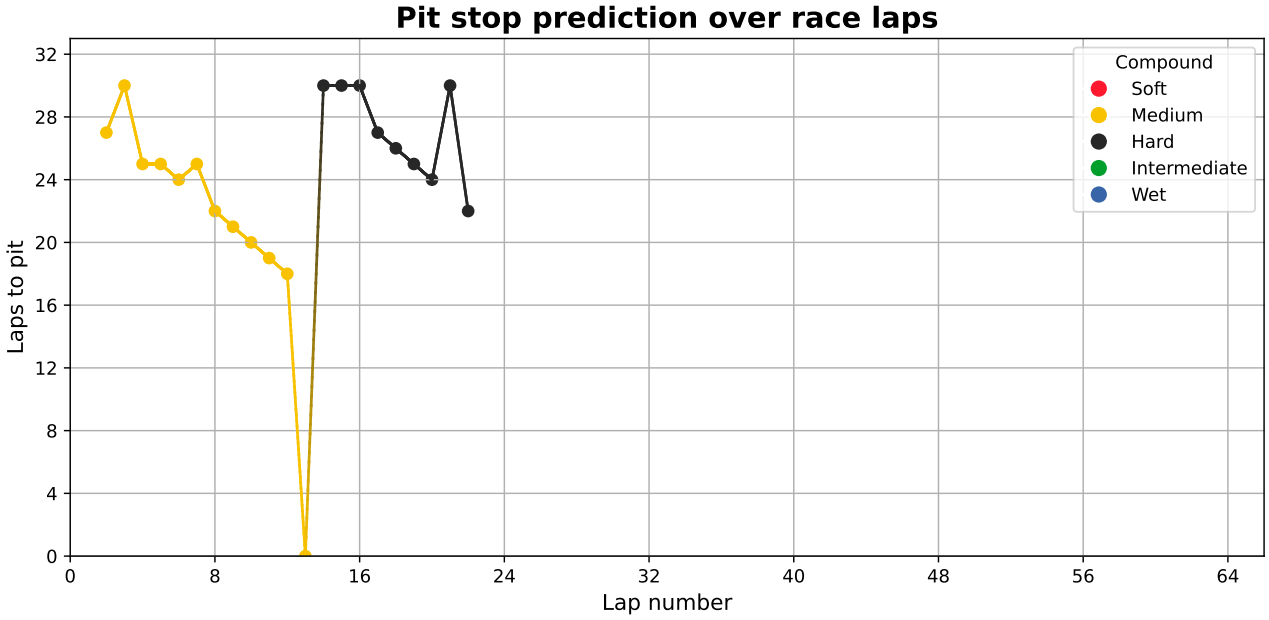

Figure 7.2: Oscar Piastri's race strategy prediction for the 2025 Italian Grand Prix.

Here, the tyre change prediction aligns perfectly with the zero-crossing of the laps to pit, suggesting an accurate timing decision. However, the expected clean diagonal is less evident due to some stochasticity in the predictions. Still, the underlying pattern remains visible. For example:

1. At lap 2 the model predicts 27 laps until the next stop. By lap 12 this value drops to 18 demonstrating a steady, albeit noisy, decline.
2. Between laps 14 and 22 a similar pattern appears. The model adjusts from 30 to 22 laps to pit, showcasing a consistent reduction over time.

Additional observations further validate model robustness:

1. The absence of mid-stint compound switches indicates low volatility in predictions and suggests that the model has learned track-specific strategy profiles, particularly relevant for circuits like Imola where undercuts and track position are critical.
2. The transition from prediction to action (i.e. compound change at zero-crossing) reflects a strong grasp of timing, arguably one of the most difficult aspects in F1 strategy planning.

Perhaps most striking is the model's ability to adapt dynamically. As race conditions evolve, it not only re-evaluates strategy but also advances the pit stop when beneficial. This highlights the model's capacity to interpret context and respond intelligently. In Piastri's case, it identifies the precise moment to switch compounds, reflecting real-time decision-making comparable to that of a seasoned race engineer.

Finally, another illustrative example of the model's overall behaviour can be observed in the case of Lando Norris during the 2025 Chinese Grand Prix

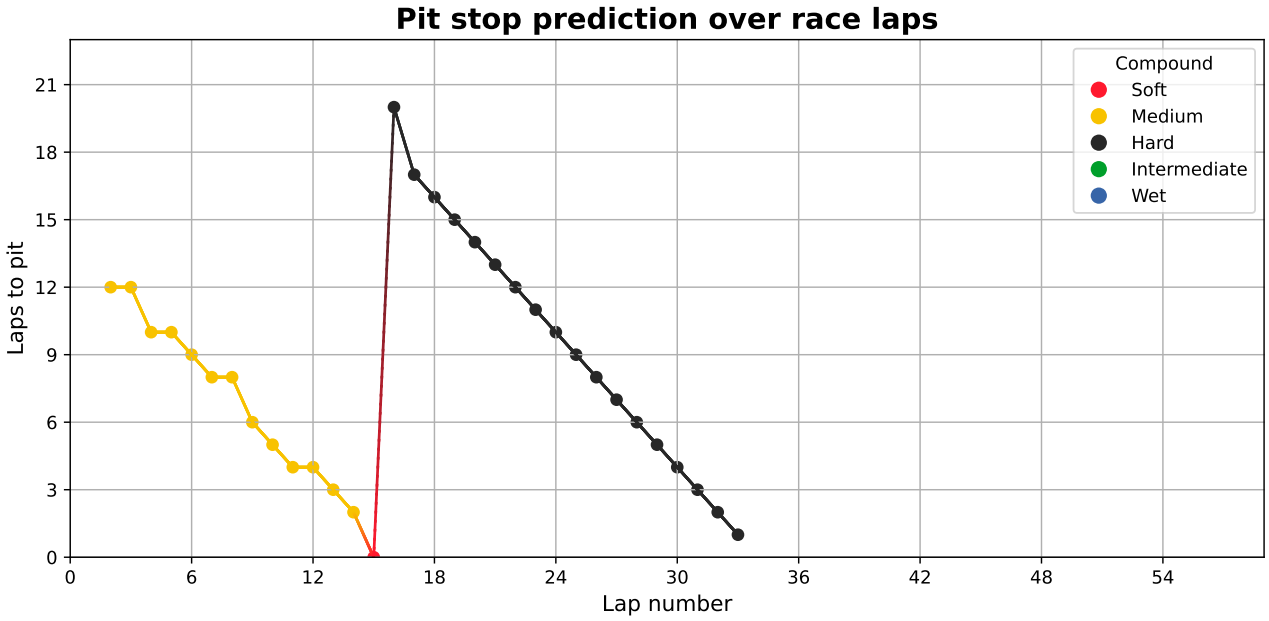

Figure 7.3: Lando Norris's race strategy prediction for the 2025 Chinese Grand Prix.

This graph exhibits the desirable diagonal structure, where the number of laps remaining before a pit stop steadily decreases. The tyre change aligns closely with the actual pit stop event. In the opening stint, the model opts for medium tyres. While the trend appears slightly irregular, the underlying diagonal behaviour remains evident. Notably, one lap before the predicted pit, the model momentarily suggests switching to soft tyres, likely the result of prediction noise or local uncertainty in compound suitability. However, at the actual pit stop, the model corrects its course and recommends transitioning to hard tyres.

Further insights include:

1. The model shows strong post-pit confidence by locking into the hard compound without subsequent fluctuation, confirming that the decision is both deliberate and stable.

2. No suggestion of intermediate or wet compounds appears throughout the race, indi-

cating that the system correctly identified the dry race conditions and excluded irrelevant alternatives.

3. The momentary soft compound suggestion could point to an interesting avenue for future probabilistic exploration strategies, where the model briefly entertains high-risk alternatives but then converges to optimal decisions.

This second stint showcases a clear diagonal pattern, confirming the model's intended behaviour and its ability to maintain strategic coherence across evolving race conditions. The combination of tactical stability, low compound variance and timing precision positions the simulator as a reliable tool for real-time race strategy analysis.

Across all three races, the simulator consistently demonstrates desirable characteristics: a descending diagonal pattern in laps-to-pit predictions, compound switches aligned with logical transition points and a coherent overall strategy. Minor fluctuations reflect race dynamics rather than model instability.

These findings indicate that the simulator can serve as a dependable decision-support system, capable of delivering realistic and timely race strategies with minimal manual intervention.

Chapter 8

Conclusions

This thesis has successfully demonstrated the feasibility of applying Transformer-based architectures to real-time race strategy simulation within the domain of Formula 1. The dual emphasis on mathematical foundations and computer science implementation has produced a robust and adaptable system capable of delivering strategic recommendations in live race scenarios.

Through the careful design and implementation of two specialised models, known respectively as the `PitStopTransformer` and the `CompoundTransformer`, this work integrates the mathematical depth of the Transformer, multi-head attention mechanisms, positional encodings and feed-forward network with the complexities of race strategy. These models process sequential race data, capturing temporal dependencies essential for making informed strategic decisions, while maintaining the flexibility required for real-time application.

A key contribution lies in the design of the data pipeline, which bridges historical and real-time data sources. The integration of Fast F1 and Open F1 APIs ensures that the simulator remains grounded in real-world constraints, processing variables such as lap times, tyre compounds and weather conditions with a consistent methodology. The simulator's ability to operate within strict API rate limits, leveraging timestamp-based querying and telemetry downsampling, highlights its engineering robustness.

Live testing validated the simulator's capability to provide actionable recommendations throughout a race. The system adapts dynamically, updating its predictions in response to unfolding events. This real-time adaptability distinguishes the simulator from traditional static models, offering a practical tool for motorsport engineers and strategists.

However, certain limitations persist. The unavailability of proprietary data like fuel loads, detailed car setups and tyre degradation metrics introduces uncertainty into the models' predictions. While the Transformer models infer these dynamics indirectly through observable variables, access to such data could significantly enhance accuracy. Computational constraints also posed challenges, with training and inference requiring GPU resources beyond standard personal hardware.

Several avenues for future work remain open, both in terms of model optimisation, data accuracy and mathematical exploration:

1. **Model optimisation:** Applying advanced memory management techniques, such as those used in the DeepSeek R1 architecture, could enhance training efficiency by reducing the memory footprint per token through optimised matrix storage. This would enable larger batch sizes or deeper models without exceeding hardware limits. Access to GPUs with greater memory and computational power would further support the training of larger architectures or datasets.

2. **Enhancing F1 accuracy:** Extending the current database by incorporating Free Practice sessions could provide valuable insights into car setups and performance variations. Although limited by the lack of setup and oil data, a more comprehensive dataset would refine model inputs and improve pre-race strategy predictions. This would help address some of the uncertainty stemming from undisclosed team parameters.

3. **Mathematical refinement:** Exploring the application of Topological Data Analysis to uncover latent structures within race data could offer a more nuanced understanding of performance dynamics. Further, introducing differential equations to model race state evolution as an initial value problem could provide an analytical framework to complement the data-driven models, blending classical mathematics with modern machine learning.

In summary, this project provides a solid foundation for further exploration at the intersection of machine learning, mathematics and motorsport strategy. It offers a clear pathway for future enhancements, whether through model refinement, expanded datasets or deeper mathematical analysis, all with the potential to deliver more sophisticated and accurate strategic tools for Formula 1 and beyond.

Bibliography

[Cyb89] G. Cybenko. "Approximation by Superpositions of a Sigmoidal Function". In: *Mathematics of Control, Signals, and Systems* 2 (1989), pp. 303–314. DOI: 10.1007/BF02551274.

Establishes that continuous functions can be approximated by neural networks with a single hidden layer and a sigmoidal activation function.

[HSW90] K. Hornik, M. Stinchcombe, and H. White. "Universal Approximation Using Neural Networks". In: *Neural Networks* 3 (1990), pp. 359–366.

Establishes that standard multilayer feedforward networks are universal approximators.

[Lon17] Imperial College London. "Improving the Aerodynamic Performance of Formula One Racing Cars". In: *Research Impact Case Study* (2017).

Study on improving the aerodynamic efficiency of Formula 1 cars.

[Vas+17] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NIPS)* 31 (2017).

Introduces the Transformer model, solely based on attention mechanisms.

[Sul18] Claudia Sulsters. "Simulating Formula One Race Strategies". In: *Vrije Universiteit Amsterdam* (2018).

Research paper on simulating race strategies in Formula 1 using mathematical models.

[BT19] Rory P. Bunker and Fadi Thabtah. "A Machine Learning Framework for Sport Result Prediction". In: *Applied Computing and Informatics* 15 (2019), pp. 27–33. DOI: 10.1016/j.aci.2017.09.005.

A study on machine learning techniques applied to predicting sports results.

[Rem+19] Adrian Remonda et al. "Formula RL: Deep Reinforcement Learning for Autonomous Racing Using Telemetry Data". In: *Preprint* (2019). DOI: 10.13140/RG.2.2.30678.09283.

Application of deep reinforcement learning for optimizing autonomous racing strategies.

[Soy19] SoyMotor. *Técnica: El funcionamiento de los frenos en Monza 2019*. 2019. URL: https://soymotor.com/f1/articulos/tecnica-el-funcionamiento-de-los-frenos-en-monza-2019.

Accessed: February 7, 2025.

[Hei+20a] Alexander Heilmeier et al. "Application of Monte Carlo Methods to Consider Probabilistic Effects in a Race Simulation for Circuit Motorsport". In: *Applied Sciences* 10 (4229 2020). DOI: 10.3390/app10124229.

Study on MCS methods for modeling probabilistic race effects in circuit motorsport.

[Hei+20b] Alexander Heilmeier et al. "Virtual Strategy Engineer: Using Artificial Neural Networks for Making Race Strategy Decisions in Circuit Motorsport". In: *Applied Sciences* 10 (7805 2020). DOI: 10.3390/app10217805.

Application of neural networks for automated race strategy decision-making in motorsport.

[Kap+20] Jared Kaplan et al. "Scaling Laws for Neural Language Models". In: *arXiv preprint arXiv:2001.08361* (2020).

Study on empirical scaling laws for language model performance.

[LF20] Xuze Liu and Abbas Fotouhi. "Formula-E Race Strategy Development Using Artificial Neural Networks and MCTS". In: *Neural Computing and Applications* 32 (2020), pp. 15191–15207. doi: 10.1007/s00521-020-04871-1.

Study on race strategy optimization for Formula-E using AI-based decision models.

[Pic20] Diego Piccinotti. "Open Loop Planning for Formula 1 Race Strategy Identification". Politecnico di Milano, 2020.

MSc thesis on race strategy identification using open-loop planning.

[VBC20] James Vuckovic, Aristide Baratin, and Remi Tachet des Combes. "A Mathematical Theory of Attention". In: *arXiv preprint arXiv:2007.02876* (2020).

Framework to analyze attention using measure theory.

[Dot21] DotCSV. *¿Qué es un Transformer? La Red Neuronal que lo cambió todo!* 2021. url: https://www.youtube.com/watch?v=aL-EmKuB078.

Accessed: February 20, 2025.

[For21] Formula 1. *The 2021 F1 Cost Cap Explained: What Has Changed and Why*. 2021. url: https://www.formula1.com/en/latest/article/the-2021-f1-cost-cap-explained-what-has-changed-and-why.501Te8udKLmkUl4PyVZtUJ.

Accessed: February 7, 2025.

[Meh+21] Aditi Mehta et al. "Applications of Operations Research in Formula One". In: *International Journal of Engineering Applied Sciences and Technology* 6 (6 2021), pp. 270–275.

Study on the use of operations research in Formula 1 for optimization and simulation.

[Mur21] James Murkin. "Faster than Hamilton! - Optimising F1 Strategies". In: *University of Leeds - Mathematics Research Paper* (2021).

Optimization study on Formula 1 race strategies using mathematical modeling techniques.

[Pic+21] Diego Piccinotti et al. "Online Planning for F1 Race Strategy Identification". In: *Proceedings of the AAAI Conference* (2021).

Application of online planning and MCTS to optimize Formula 1 race strategy decisions.

[Hei22] Alexander Maximilian Heilmeier. "Simulation of Circuit Races for the Objective Evaluation of Race Strategy Decisions". PhD thesis. Technische Universität München, 2022.

PhD dissertation on evaluating race strategy decisions using circuit race simulation.

[Jim22] Miguel Jiménez Tardós. "Real Time Formula 1 Strategy Decision Problem Modeled as a Discrete Event Stochastic Process". Cranfield University, School of Aerospace, Transport and Manufacturing, 2022.

MSc thesis on modeling F1 strategy decisions using stochastic processes.

[Ron22] Massimo Rondelli. "The Future of Formula 1 Racing: Neural Networks to Predict Tyre Strategy". Alma Mater Studiorum - Università di Bologna, 2022.

Explores the use of neural networks for predicting optimal tire strategies in Formula 1 racing.

[Sic22] Horatiu Sicoie. "Machine Learning Framework for Formula 1 Race Winner and Championship Standings Predictor". Tilburg University, 2022.

Ppredicting Formula 1 race winners using supervised machine learning models.

[Thi22] John Thickstun. "The Transformer Model in Equations". In: *arXiv preprint arXiv:2022.00000* (2022).

Mathematical definitions and interpretations of the Transformer model.

[TWW22] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media, 2022.

Comprehensive guide on using transformers for NLP tasks with the Hugging Face library.

[Alb+23] Silas Alberti et al. "Sumformer: Universal Approximation for Efficient Transformers". In: *Proceedings of the 2nd Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML)* (2023).

Introduces Sumformer, demonstrating universal approximation for efficient transformers.

[For23] Formula 1. *Rapid Decisions, Driver Skills and Intricate Procedures: Why Race Starts Are So Important*. 2023. URL: https://www.formula1.com/en/latest/article/rapid-decisions-driver-skills-and-intricate-procedures-why-race-starts-are.3uziERC5iPE5EYOPRmerXU.

Accessed: February 7, 2025.

[Jim23] Miguel Jiménez Tardós. "Design of Formula 1 Race Strategies using Reinforcement Learning". Escuela de Ingeniería y Arquitectura, 2023.

Master's thesis on using reinforcement learning for optimizing Formula 1 race strategies.

[Sci23] Science News Explores. *Race Car Drivers Blink Strategically to Stay Focused*. 2023. URL: https://www.snexplores.org/article/race-car-drivers-blink-strategy#:~:text=The%20drivers'%20blinking%20was%20surprisingly,while%20changing%20speed%20or%20direction..

Accessed: February 7, 2025.

[Su+23] Jianlin Su et al. "RoFormer: Enhanced Transformer with Rotary Position Embedding". In: *arXiv preprint arXiv:2104.09864* (2023).

Proposes Rotary Position Embedding (RoPE) for transformer models.

[Bea24] Cédric M. L. Beaume. "Formula 1 Strategy Competition". In: *University of Leeds* (2024).

Competition framework for optimizing Formula 1 strategies using mathematical models.

[FHP24] Takashi Furuya, Maarten V. de Hoop, and Gabriel Peyré. "Transformers are Universal In-context Learners". In: *arXiv preprint arXiv:2408.01367* (2024).

Study on the universality of transformers in handling an arbitrary number of tokens.

[Ges+24] Borjan Geshkovski et al. "A Mathematical Perspective on Transformers". In: *arXiv preprint arXiv:2312.10794* (2024).

Mathematical framework analyzing transformers as interacting particle systems.

[Pir25] Pirelli. *Formula 1: Technology, Performance, and Innovation by Pirelli*. 2025. URL: https://www.pirelli.com/tyres/en-ww/motorsport/car/formula-1.

Accessed: February 8, 2025.

A: Mathematical proofs

Proof of Proposition 4.2.1.

To prove that PE is injective it must be shown that for all $x, y \in \{1, 2, \ldots, n\}$,

$$x \neq y \implies PE(x) \neq PE(y)$$

This can be more easily approached by proving the contrapositive

$$PE(x) = PE(y) \implies x = y$$

Assume $PE(x) = PE(y)$. Then, by the definition of PE, for every frequency component ω_t both the sine and cosine terms must satisfy

$$\begin{cases} \sin(x\omega_t) = \sin(y\omega_t) \\ \cos(x\omega_t) = \cos(y\omega_t) \end{cases}$$

for all $t \in \left\{1, 2, \ldots, \frac{d_{model}}{2}\right\}$.

Since sine and cosine are 2π-periodic functions, this implies that for each t there exists some integer $k \in \mathbb{Z}$ such that

$$x\omega_t = y\omega_t + 2\pi k$$

Rearranging gives

$$x - y = \frac{2\pi k}{\omega_t}$$

However, since the ω_t values are distinct for each t, the right-hand side differs across components. Therefore it is not possible for the same integer $x - y$ to satisfy all these equations simultaneously unless $x - y = 0$.

Hence, $x = y$, which completes the proof that PE is injective.

■

Proof of Proposition 4.2.2.

The objective is to construct a definition of $T(k)$ that is independent of the specific position i. Define $T(k)$ as a block-diagonal matrix composed of $\frac{d_{model}}{2}$ rotation matrices $\phi_t(k)$

acting on each sinusoidal component pair

$$T(k) = \begin{pmatrix} \phi_1(k) & 0 & \cdots & 0 \\ 0 & \phi_2(k) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \phi_{\frac{d_{model}}{2}}(k) \end{pmatrix}$$

where each 0 denotes a 2×2 zero matrix and each block $\phi_t(k)$ is defined as a rotation matrix

$$\phi_t(k) = \begin{pmatrix} \cos(r_t k) & -\sin(r_t k) \\ \sin(r_t k) & \cos(r_t k) \end{pmatrix}$$

with r_t denoting the angular frequency associated with the t-th sinusoidal component. Note that r_t is distinct from the previously defined ω_t used in positional encoding.

The goal is to find $T(k)$ such that $T(k)PE(i) = PE(i+k)$. Focusing on a single pair of sinusoidal components for coordinate t, this condition becomes

$$\underbrace{\begin{pmatrix} \cos(r_t k) & -\sin(r_t k) \\ \sin(r_t k) & \cos(r_t k) \end{pmatrix}}_{\phi_t(k)} \begin{pmatrix} \sin(\omega_t k) \\ \cos(\omega_t k) \end{pmatrix} = \begin{pmatrix} \sin(\omega(i+k)) \\ \cos(\omega(i+k)) \end{pmatrix}$$

where ω_t is the frequency defined by the positional encoding scheme.

This expression strongly resembles the trigonometric angle addition formulas

$$\sin(\omega k + \omega i) = \sin(rk)\cos(\omega i) + \cos(rk)\sin(\omega i)$$
$$\cos(\omega k + \omega i) = \cos(rk)\cos(\omega i) - \sin(rk)\sin(\omega i)$$

These identities confirm that a rotation matrix with angle $\omega_t k$ applied to $PE(i)$ yields $PE(i+k)$. Therefore one can identify $r_t = \omega_t$.

By applying this result and recalling the definition of the encoding frequency the rotation matrix becomes

$$\phi_t(k) = \begin{pmatrix} \cos(\omega_t k) & -\sin(\omega_t k) \\ \sin(\omega_t k) & \cos(\omega_t k) \end{pmatrix}$$

Hence, the transformation $T(k)$ is fully defined by the model dimension d_{model}, the position offset k and the component index t, without any dependence on the actual position i.

This completes the proof.

■

Proof of Proposition 4.3.1.

By the construction outlined previously, for a given query $q \in \mathcal{Q}$, we have

$$\begin{aligned}(\psi_{G(q,\cdot)}(m(K))L)(dv) &= \sum_{j=1}^{N} \int \frac{G(q,k_j)}{\sum_{p=1}^{N} G(q,k_p)} \delta_{k_j}(dk) L(k,dv) \\ &= \sum_{j=1}^{N} \frac{G(q,k_j)}{\sum_{p=1}^{N} G(q,k_p)} \delta_{v_j}(dv)\end{aligned}$$

Applying the moment projection $\prod$ to this distribution yields

$$A_{m(K)}(q,dv) = \delta_{\sum_{j=1}^{N} \frac{G(q,k_j)}{\sum_{p=1}^{N} G(q,k_p)} v_j(dv)}$$

Finally, applying the left action of this kernel on δ_{q_t} gives

$$\delta_{q_t} A_{m(K)}(dv) = \int \delta_{q_t}(dq) A_{m(K)}(q,dv) = \delta_{\sum_{j=1}^{N} \frac{G(q,k_j)}{\sum_{p=1}^{N} G(q,k_p)} v_j(dv)}$$

which corresponds exactly to the attention output from Definition 4.3.1, with $G(x, y) = \exp(a(x, y))$. Using the canonical bijection $\delta_x \longleftrightarrow x$ between Dirac measures and points in E completes the proof.

■

Proof of Theorem 4.3.1.

By definition of the Wasserstein distance and using the contraction properties from Lemma 4.3.1, we have

$$\begin{aligned}\|\mathrm{Att}(q_1,K,V) - \mathrm{Att}(q_2,K,V)\|_2 &\leqslant \|\mathrm{Att}(q_1,K,V) - \mathrm{Att}(q_2,K,V)\|_1 \\ &= \mathbb{W}_1(\delta_{q_1} A_{m(K)}, \delta_{q_2} A_{m(K)}) \\ &\leqslant d\|l\|_{Lip} \frac{2\|G\|_{Lip,\infty}\mathrm{diam}(E)}{\varepsilon(G)} \mathbb{W}_1(\delta_{q_1}, \delta_{q_2}) \\ &= d\|l\|_{Lip} \frac{2\|G\|_{Lip,\infty}\mathrm{diam}(E)}{\varepsilon(G)} \|q_1 - q_2\|_1 \\ &= \sqrt{d^3}\|l\|_{Lip} \frac{2\|G\|_{Lip,\infty}\mathrm{diam}(E)}{\varepsilon(G)} \|q_1 - q_2\|_2\end{aligned}$$

where the last step uses the norm equivalence $\|x\|_1 \leqslant \sqrt{d}\|x\|_2$ for vectors in $\mathbb{R}^d$.

■

Proof of Lemma 4.4.1.

Let $f \in \mathcal{M}^n$ be arbitrary and let $\varepsilon > 0$. The goal is to find a $g \in \mathcal{C}^n$ such that $\rho_\mu(f, g) < \varepsilon$. For sufficiently large M we have

$$\int \min\left\{\left|f\mathbb{I}_{\{|f|<M\}} - f\right|, 1\right\}\, d\mu < \frac{\varepsilon}{2}$$

There then exists a continuous function g such that

$$\int \left|f\mathbb{I}_{\{|f|<M\}} - g\right|\, d\mu < \frac{\varepsilon}{2}$$

Combining both inequalities yields

$$\int \min\left\{|f - g|, 1\right\}\, d\mu < \varepsilon$$

which proves the claim.

■

Proof of Lemma 4.4.2.

Pick $\varepsilon > 0$. It suffices to find $N \in \mathbb{N}$ such that for all $n \geqslant N$ we have

$$\int \min\left\{|f_n(x) - f(x)|, 1\right\}\, \mu(dx) < \varepsilon$$

Without loss of generality assume that $\mu(\mathbb{R}^n) = 1$. Since $\mathbb{R}^n$ is a locally compact metric space the measure μ is regular. Therefore, there exists a compact set $\mathcal{K} \subset \mathbb{R}^n$ such that $\mu(\mathcal{K}) > 1 - \frac{\varepsilon}{2}$.

Because $f_n \longrightarrow f$ uniformly on $\mathcal{K}$ there exists N such that for all $n \geqslant N$

$$\sup_{x \in \mathcal{K}} \left|f_n(x) - f(x)\right| < \frac{\varepsilon}{2}$$

The result then follows.

■

Proof of Theorem 4.4.3.

Given any Borel measurable, continuous and non-constant function G, it follows from Proposition 4.4.2 and Lemma 4.4.2 that $\sum \prod^n(G)$ is ρ_μ-dense in $\mathcal{C}^n$. Since $\mathcal{C}^n$ is itself ρ_μ-dense in $\mathcal{M}^n$ by Lemma 4.4.1, the result follows by transitivity of denseness.

■

Proof of Theorem 4.4.2.

The proof follows from the Stone–Weierstrass Theorem. Let $\mathcal{K} \subset \mathbb{R}^n$ be any compact set. For any G the class $\sum \prod^n(G)$ forms an algebra on $\mathcal{K}$.

If $x, y \in \mathcal{K}$ are distinct then there exists an $A \in \mathcal{A}^n$ such that $G(A(x)) \neq G(A(y))$. To see this, take $a, b \in \mathbb{R}$ with $a \neq b$ and $G(a) \neq G(b)$. Choose A such that $A(x) = a$ and $A(y) = b$. Then clearly $G(A(x)) \neq G(A(y))$. This shows that $\sum \prod^n(G)$ separates points on $\mathcal{K}$.

Additionally, there exists $G(A)$ that is constant and non-zero. For example, select $b \in \mathbb{R}$ such that $G(b) \neq 0$ and set $A(x) = 0x + b$. Then for all $x \in \mathcal{K}$, we have $G(A(x)) = G(b)$. Hence, the set $\sum \prod^n(G)$ does not vanish on $\mathcal{K}$.

By the Stone–Weierstrass Theorem, it follows that $\sum \prod^n(G)$ is $\rho_{\mathcal{K}}$-dense in the space of real-valued continuous functions on $\mathcal{K}$. Since $\mathcal{K}$ was arbitrary, the result holds for all compact subsets.

■

Proof of Lemma 4.4.3.

Fix an arbitrary $\varepsilon > 0$ and without loss of generality, assume $\varepsilon < 1$. The goal is to construct a finite collection of constants β_j and affine functions A_j such that

$$\sup_{x \in \mathbb{R}} \left| F(x) - \sum_{j=1}^{Q-1} \beta_j \psi(A_j(x)) \right| < \varepsilon$$

Choose Q so that $\frac{1}{Q} < \frac{\varepsilon}{2}$ and define $\beta_j = \frac{1}{Q}$ for $j \in \{1, \ldots, Q-1\}$. Let $M > 0$ be such that $\psi(-M) < \frac{\varepsilon}{2Q}$ and $\psi(M) > 1 - \frac{\varepsilon}{2Q}$. Since ψ is a squashing function, such an M exists.

For each $j \in \{1, \ldots, Q-1\}$, define

$$r_j = \sup\left\{x : \ F(x) = \frac{j}{Q}\right\}$$
$$r_Q = \sup\left\{x : \ F(x) = 1 - \frac{1}{2Q}\right\}$$

These values exist due to the continuity of F.

For any $r < s$, let $A_{r,s} \in \mathcal{A}^n$ be the unique affine function such that $A_{r,s}(r) = M$ and $A_{r,s}(s) = -M$. Define the approximating function as:

$$H_\varepsilon(x) = \sum_{j=1}^{Q-1} \beta_j \psi(A_{r_j, r_{j+1}}(x))$$

It can be checked that on each interval

$$(-\infty, r_1], \ (r_1, r_2], \ \ldots, \ (r_{Q-1}, r_Q], \ (r_Q, \infty)$$

the inequality $|F(x) - H_\varepsilon(x)| < \varepsilon$ holds.

■

Proof of Lemma 4.4.4.

Let F be the cosine squashing function. By constructing linear combinations of affinally shifted copies of F it is possible to approximate the cosine function over any compact interval $[-M, M]$. The result then follows from Lemma 4.4.3 and the triangle inequality.

■

Proof of Lemma 4.4.5.

Choose $M > 0$ such that for all $j \in \{1, \ldots, Q-1\}$, the image $A_j(\mathcal{K}) \subset [-M, M]$. Since $\mathcal{K}$ is compact and each A_j is continuous, such an M exists.

Define

$$Q' = Q \sum_{j=1}^{Q} |\beta_j|.$$

By Lemma 4.4.4, for all $x \in \mathcal{K}$ we have

$$\left| \sum_{j=1}^{Q} \beta_j \cos(A_j(x)) - g(x) \right| < \varepsilon$$

Since each $\cos(A_j(x))$ can be approximated by elements in $\sum^n(\psi)$, it follows that

$$f = \sum_{j=1}^{Q} \cos(A_j) \in \sum^n(\psi)$$

■

Proof of Lemma 4.4.6.

By Proposition 4.4.2, the class of trigonometric polynomials

$$\left\{ \sum_{j=1}^{Q} \beta_j \prod_{k=1}^{l_j} \cos(A_{jk}) : \ Q, l_j \in \mathbb{N}, \ \beta_j \in \mathbb{R}, \ A_{jk} \in \mathcal{A}^n \right\}$$

is uniformly dense on compacta in $\mathcal{C}^n$. Using the trigonometric identity

$$\cos(\alpha)\cos(\beta) = \frac{1}{2}\left(\cos(\alpha+\beta) + \cos(\alpha-\beta)\right)$$

every such polynomial can be rewritten as

$$\sum_{i=1}^{T} \alpha_i \cos(A_i)$$

where $\alpha_i \in \mathbb{R}$ and $A_i \in \mathcal{A}^n$. The result now follows directly from Lemma 4.4.5.

■

Proof of Theorem 4.4.4.

By Lemma 4.4.6 it is known that $\sum^n(\psi)$ is uniformly dense on compacta in $\mathcal{C}^n$. Then Lemma 4.4.1 implies that this class is ρ_μ-dense in $\mathcal{C}^n$. Finally, by applying the triangle inequality and Lemma 4.4.2 it follows that $\sum^n(\psi)$ is ρ_μ-dense in $\mathcal{M}^n$.

■

Proof of Corollary 4.4.1.

Fix $\varepsilon > 0$. From regularity of the measure μ, there exists a compact set $\mathcal{K}_1$ such that $\mu(\mathcal{K}_1) > 1 - \frac{\varepsilon}{2}$ and the restriction $g_{|\mathcal{K}_1}$ is continuous on $\mathcal{K}_1$. By the Tietze extension theorem there exists a function $g' \in \mathcal{C}^n$ such that $g'_{|\mathcal{K}_1} = g_{|\mathcal{K}_1}$ and $\sup_{x \in \mathbb{R}^n} g'(x) = \sup_{x \in \mathcal{K}_1} g_{|\mathcal{K}_1}(x)$.

By Lemma 4.4.6, we know that $\sum^n(\psi)$ is uniformly dense on compacta in $\mathcal{C}^n$. Therefore, there exists a compact set $\mathcal{K}_2$ with $\mu(\mathcal{K}_2) > 1 - \frac{\varepsilon}{2}$ and a function $f \in \sum^n(\psi)$ such that

$$\sup_{x \in \mathcal{K}_2} \left| f(x) - g(x) \right| < \varepsilon$$

Then $\sup_{x \in \mathcal{K}_1 \cap \mathcal{K}_2} \left| f(x) - g(x) \right| < \varepsilon$ and $\mu(\mathcal{K}_1 \cap \mathcal{K}_2) > 1 - \varepsilon$.

■

B: Positional encoding heatmap script

The following code generates the positional encoding heatmap shown in Figure 4.4. The implementation is written in Python, utilising the `numpy` and `matplotlib` libraries. The script builds the sinusoidal encoding matrix by computing the appropriate angles for each token position and then applies sine and cosine functions to even and odd dimensions, respectively.

positionalEncoding.py

```python
import numpy as np
import matplotlib.pyplot as plt

def getAngles(pos, i, d_model):
    """
    Computes the angle rates used in sinusoidal positional encoding.
    """
    angle_rates = 1 / np.power(10000.0, (2 * (i // 2)) / d_model)
    return pos * angle_rates


def getPositionalEncoding(n, d_model):
    """
    Generates a sinusoidal positional encoding matrix.
    """
    pos = np.arange(n)[:, np.newaxis]                # Shape: (n, 1)
    i = np.arange(d_model)[np.newaxis, :]            # Shape: (1, d_model)

    angles = getAngles(pos, i, d_model)
    angles[:, 0::2] = np.sin(angles[:, 0::2])        # Apply sin to even indices
    angles[:, 1::2] = np.cos(angles[:, 1::2])        # Apply cos to odd indices
    return angles

n = 100
d_model = 128
pe = getPositionalEncoding(n, d_model)

plt.figure(figsize=(12, 8))
plt.imshow(pe, cmap="coolwarm", aspect="auto", vmin=-1, vmax=1)
plt.colorbar()
plt.title("Positional encoding")
plt.xlabel(r"Embedding coordinate ($t$)")
plt.ylabel(r"Token position in the sequence ($i$)")
plt.tight_layout()

plt.savefig("positionalEncoding.pdf", bbox_inches="tight")
plt.show()
```