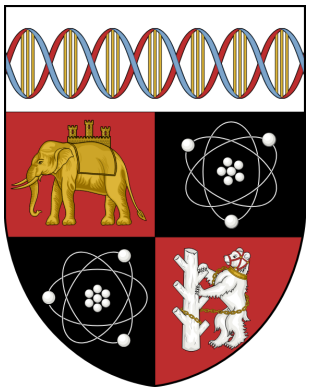


University of Warwick

Master's Thesis

Interpretability and Transformers

Author:
Hamza Sajjad

Supervisor:
Dr. Roman Kozhan
Dr. Dan Phillips

A thesis submitted in fulfilment of the requirements
for the degree of Master of Science

Declaration of Authorship

I declare that this work is entirely my own in accordance with the University's Regulation 11 and the WBS guidelines on plagiarism and collusion. All external references and sources are clearly acknowledged and identified within the contents. No substantial part(s) of the work submitted here has also been submitted by me in other assessments for accredited courses of study, and I acknowledge that if this has been done it may result in me being reported for self-plagiarism and an appropriate reduction in marks may be made when marking this piece of work.

Signed: Hamza Sajjad

Date: 30/08/2025

UNIVERSITY OF WARWICK

Abstract

This paper explores the use of Transformer models on bucketed financial ratios for multi-class prediction. We first assess the model's robustness by demonstrating its temporal stability and competitive performance relative to XGBoost. We then fix the seed and hyperparameter configuration and apply a suite of attention-based interpretability techniques, including attention matrix extraction, attention scoring, entropy-based layer importance, feature and head ablations, weighted head norms, and attention rollout. Across methods, the same financial signals consistently emerge, most notably *momentum*, *earnings-to-price*, and *book-to-market ratio*. Our findings suggest that Transformers offer a **robust** and **interpretable** alternative for financial modelling, with the attention mechanism providing valuable insights often overlooked in existing applications.

Acknowledgements

I would like to thank my supervisors, Dr Roman Kozhan and Dr Dan Phillips. Without their support, this project would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Literature Review

1.1 Introduction and Motivation

The influence of Artificial Intelligence (AI) has expanded substantially across multiple domains in recent years **(Maslej et al. 2025)**. New technologies are rapidly emerging across various sectors worldwide, with a shared goal of leveraging AI for practical gains. This growing trend has naturally caught the attention of portfolio managers and industry professionals, who are exploring how best to utilise various machine learning algorithms to generate positive return in various markets **(Mercer 2024)**.

However, in the pursuit of positive alpha, many professionals have encountered a key challenge: a lack of interpretability in many machine learning techniques. These algorithms, despite often achieving high performance, raise concerns due to their opaque nature. Industry professionals are often uncomfortable deploying such models because they lack transparency, trust, and accountability **(Arsenault et al. 2025)**. This makes it significantly harder to justify their use to key stakeholders such as regulators, clients, and internal decision makers, ultimately limiting their practical value. This challenge is not exclusive to finance. Other high stakes industries such as law and healthcare face similar issues, where many legal and medical professionals view the opacity and the lack of trust in AI as one of the key barriers for their widespread adoption **(Garrett and Rudin 2023; Ennab and Mcheick 2024; Mienye et al. 2024)**.

Fortunately, there has been notable progress in developing model agnostic techniques that can be applied to any machine learning model to improve interpretability. These methods are collectively referred to as Explainable Artificial Intelligence (XAI) **(Madathil et al. 2025)**.

Popular tools such as SHAP **(Lundberg and Lee 2017)** and LIME **(Ribeiro et al. 2016)** aim to make black box machine learning models more transparent and interpretable by using different mathematical techniques. SHAP assigns feature importance using game theoretic principles, while LIME approximates the model locally with simpler and more interpretable models.

While both techniques have their merits, they also come with a set of limitations. A key challenge is that as model complexity increases, applying these interpretability techniques becomes significantly more difficult and less effective. Moreover, even when applied correctly, they may not provide a comprehensive understanding of the model's decision making process, potentially producing misleading results **(Huang and Marques-Silva 2024)**.

This has led to a growing view among some experts that, rather than focusing on explaining black box models **after** they have been produced (*post-hoc*), we should instead build models that are inherently interpretable from the start **(Rudin 2019; Leslie 2019)**.

1.2 Transformer Model

Transformers belong to a class of deep learning models that are inherently complex. First introduced in the seminal paper Attention Is All You Need **(Vaswani et al. 2017)**, Transformers are designed to operate on sequential data by leveraging the concept of **attention** to capture dependencies and relationships across elements in a sequence. Their primary application has been in the domain of Natural Language Processing (NLP), where the objective is to interpret and understand human language. They have achieved notable success in tasks such as sentiment analysis, with models such as BERT demonstrating superior performance over previous approaches **(Devlin et al. 2019)**.

One of the fundamental aspects of the Transformer architecture is the concept of attention. This mechanism is widely regarded as the model's most critical component. At its core, attention allows the model to selectively focus on the most relevant elements of the input sequence when producing predictions. Notably, attention weights are inherently integrated into the model's structure; these weights can be extracted and interpreted, providing a level of interpretability that is generally absent in other high capacity neural architectures.

This built-in interpretability makes Transformers particularly appealing in contexts where transparency is required. While attention mechanisms already offer meaningful insights, their utility can be significantly expanded by combining them with additional techniques such as feature importance scoring or SHAP. Furthermore, the attention mechanism itself can be extended through methods such as attention rollout and attention flow, which can significantly improve the quality and depth of interpretability **(Abnar and Zuidema 2020)**.

While transformers have been widely used by researchers and practitioners to forecast returns, most of the emphasis has been placed on positive predictive performance rather than on understanding the model structure itself. For example, **(Ma et al. 2023)** applies a transformer to asset allocation in the Chinese stock market, demonstrating improved time series modelling over other non-linear methods. However, their interpretability analysis relies solely on SHAP

values to assess feature importance. Similarly, **(Korangi et al. 2023)** studies the application of transformers on mid-cap companies and shows superior performance compared to traditional models, yet again interprets the model using SHAP.

While SHAP is useful for identifying which features contribute most to the model's predictions, it does not explain what the model is actually doing. Thus, while one could argue that these approaches offer a degree of interpretability, they fall short if the objective is to genuinely understand the internal workings of the model. Some studies, such as **(Govindaraj et al. 2023)**, have attempted to combine SHAP values with attention matrices to provide a more comprehensive interpretability perspective. However, this work remains fairly superficial, discussing attention only at a surface level without delving into what makes attention mechanisms particularly insightful. All three examples rely on post-hoc SHAP or basic attention visualisations; none trace how specific features propagate through layers or attention heads in the model, nor do they use the attention matrices themselves as a basis for applying different interpretability methods. This study aims to **address that gap** by using ratio data to train a transformer model and then analysing the attention in depth. The goal is to develop a richer interpretability framework, making this an original contribution rather than an extension of prior work.

For this project, the native `PyTorch` implementation of the Transformer is employed, as it adheres closely to the original architecture proposed in the *Attention Is All You Need* paper **(Vaswani et al. 2017)**. Only the encoder component is utilised, given that the decoder is not required for the predictive task at hand.

Although several Transformer variants have been developed specifically for tabular or financial data such as TabTransformer **(Huang et al. 2020)**, FinBERT **(Araci 2019)**, and StockFormer **(Ma et al. 2024)**, the objective of this study is to assess the performance of the base Transformer architecture as demonstrating strong performance in this baseline setting may suggest the potential for further improvements through the use of more specialised architectures. We then proceed to *bucket* the data as various studies have shown that categorical data tends to perform well with Transformers, especially when compared to continuous data **(Rabanser et al. 2020)**. After bucketing, we train and validate the model on a fixed time period, followed by some testing. We then analyse the attention matrix and use various techniques to extract relevant insights into the Transformer's behaviour.

Primary Research Question: *To what extent can the Transformer architecture identify interpretable and explainable signals in financial data?*

The specific objectives of this study are outlined as follows:

- **Benchmarking:** To evaluate whether the Transformer model outperforms established baseline methods in out-of-sample classification tasks.

- **Interpretability:** To assess whether the attention matrix reveals economically meaningful signals that may be of practical relevance to portfolio managers.
- **Trade-off:** To determine whether the Transformer offers improved interpretability and explainability while maintaining strong predictive accuracy.

1.3 Emerging Market Data

For this project, we focus on yearly emerging market data and train our Transformer model using financial data sourced from the Compustat Global (IQ) database **(*Compustat Global database via WRDS* 2025)**.

The motivation for focusing on emerging market data is threefold. First, these markets are typically less saturated with advanced models and professional analysts **(Lang et al. 2003)**, creating opportunities to uncover overlooked or under-exploited alpha. Second, they are often more volatile and less regulated than developed markets, yet tend to exhibit higher average returns and more predictable patterns **(Bekaert and Harvey 1997)**. Demonstrating that a Transformer can learn and generalise in such noisy and unstable environments would indicate its robustness. Finally, emerging markets are shaped by a diverse set of economic, political, and structural factors, which increases data complexity. While this diversity provides rich signals for the model to capture, it also introduces noise that must be managed carefully.

1.4 Summary of Findings

Using bucketed financial ratios and emerging market data, the Transformer performs on **par** with a strong tabular baseline (XGBoost).

Temporal analysis shows that while model weights shift across regimes (e.g., 2017–2018 vs. 2020–2021), the same **core economic signals remain influential**. The model is also sensitive to objective weighting: overly prioritising class-2 F1 causes mode collapse, underscoring the need for balanced training objectives.

Across interpretability techniques (attention maps, rollout, feature attributions, and SHAP) the **same** families of signals consistently emerge: *momentum*, *earnings-to-price*, *free-cash-flow yield*, *past-12-month return*, and *book-to-market*.

Finally, feature ablations indicate that while removing any **single** top variable has little effect, **dropping all five** leads to a clear deterioration, confirming their joint importance.

Overall, the Transformer achieves performance **comparable** to a strong baseline (XGBoost), while additionally offering a suite of interpretability techniques based on the attention mechanism, making it substantially more **transparent and explainable**.

Chapter 2

Data (Methodology and Data)

In this chapter, we describe the data and the preprocessing steps taken to ensure it is suitable for use with the Transformer.

2.1 Data Description and Source

To begin this project, we obtain emerging market equity data from the Compustat Global IQ database **(*Compustat Global database via WRDS* 2025)**, covering the period from 2006 to 2025. We extract two key datasets: the Securities Dataset, which contains trading and security information (such as price, volume, and shares outstanding) recorded monthly for each asset; and the Fundamental Dataset, which contains annual, asset specific financial data (such as total assets, liabilities, and revenue). The full data extraction and processing workflow is available in the accompanying notebook.

2.2 Data Preprocessing

The data preprocessing involves standard cleaning, merging, feature engineering, and preprocessing steps. Each stage is designed to avoid data leakage, ensuring that only information available at the time of prediction is used during training.

2.2.1 Loading, Handling, and Filtering the Data

As an initial step, these two datasets are loaded and preprocessed to retain only the variables relevant for ratio construction and merging. Redundant fields that do not contribute to the calculation of financial ratios or model inputs in any way are removed. We observe a substantial difference in row counts between the two datasets, primarily due to their reporting frequencies: the Securities dataset is recorded monthly, while the Fundamentals dataset is reported annually.

Additionally, the Securities dataset covers a broader date range, naturally resulting in a higher number of rows. To address this, we process each dataset individually:

Fundamental Data:

To ensure data quality and minimise the risk of introducing bias during imputation, a filtering criterion is applied to the dataset to exclude excessively sparse records. Specifically, rows missing more than 50% of their values are removed, as they are unlikely to yield meaningful information. Additionally, records missing key entity or date identifiers are excluded, as they cannot be imputed and are essential in merging the dataset. Dates are then standardised to ensure consistency across records.

Securities Data:

The Securities data undergoes similar preprocessing, including standardising dates and aligning records to the same time frame. To ensure consistency across datasets, the data is filtered to retain only securities whose identifiers also appear in the Fundamentals dataset. Furthermore, due to missing observations in certain months for some securities, we also ensure that each SEDOL (Stock Exchange Daily Official List) has an entry for every month in the sample period. This complete structure allows for computations that rely on lagged or shifted observations.

2.2.2 Filling, Merging, and Ratio Calculations

To address gaps in static identifying fields, missing values are filled using both forward and backward propagation **(Mudadla 2024)**. These fields do not vary over time and are not used as predictive inputs, so this imputation does not introduce any risk of data leakage.

To account for the delay in the release of financial information, a publication lag of 3 months is applied to the trading data. This adjustment reflects the practical delay between the end of a reporting period and the point at which the data becomes publicly available.

We then proceed in calculating two key variables to support model training and evaluation. These are *past 12-month return* and *momentum*, both of which rely solely on historical data (see **Appendix A** for further detail). While not strictly ratios, both are stored as ratio-like metrics based on their calculation. These variables are among the most informative for predicting future returns, so excluding them would risk losing valuable predictive signals **(Goyal et al. 2024)**. In this step, we also compute the *12-month forward return*, which serves as the prediction label and forms the basis for evaluating model performance.

The Fundamental Dataset and Securities datasets are then merged based on aligned security identifiers and reporting dates, ensuring that each financial record is matched with the corresponding market data.

Following the merge, financial ratios are computed (see **Appendix A** for which ratios were used) and any resulting infinite values are treated as missing. Categorical variables required for modelling are also converted to a suitable numerical format to ensure compatibility with the Transformer architecture. At this stage, the core features are constructed, though the dataset still contains missing values that must be addressed through imputation.

2.2.3 Splitting, Imputing, Encoding and Bucketing

To address missing values, features are grouped into three categories: essential metadata (e.g., country, industry, year, company ID), the target variable (*12-month forward return*), and ratio-based predictive features. This separation ensures that imputations are performed only within appropriate feature groups, avoiding cases where fixed metadata could be imputed from variable predictive features, or vice versa. The data is then split into **training**, **validation**, and **test sets** using fixed thresholds to prevent any form of data leakage, with exact cut-off dates varying depending on the specific analysis and testing conducted (see **Chapter 4**).

All imputations are performed exclusively on the training set. For each ratio feature, median values are computed within industry–country groups and applied to the validation and test sets, with global medians used as a fallback. Median imputation is preferred over the mean because financial ratios are typically skewed **(Barnes 2006)** and prone to outliers, which can substantially distort the mean but have minimal effect on the median. More advanced methods, such as KNN (k-nearest neighbours), were also tested but offered negligible improvement while significantly increasing computation time.

Remaining incomplete rows are removed from each split to ensure a fully clean dataset for model input.

Categorical variables are label-encoded using mappings derived from the training set. Ratio features are discretised into ten decile bins (also referred to as buckets or classes), while the target variable is categorised into three classes, both via quantile-based binning ***(KBinsDiscretizer — scikit-learn 1.7.1 documentation*** **n.d.)** fitted exclusively on the training data.

For the label, the bucket size is set to three to strike a balance between interpretability and classification difficulty. Too few buckets (e.g., two) limit the model to simple categories such as “high returns” versus “low returns,” which restricts the insights we can draw. Three buckets, by contrast, allow separation into low, average, and high returns, offering a more detailed view of the model’s decision making. Increasing the number of buckets too far (e.g., five) makes the data overly granular, as quantile-based binning allocates fewer samples to each class. This increases task complexity and can reduce prediction confidence. Testing showed that three buckets provided a practical middle ground.

For features, the objective differs from that of the label. Since they are not interpreted directly, the interpretability–accuracy trade-off that guided label binning is less relevant. Instead, the priority is to expose the Transformer to richer, more detailed input patterns so it can capture complex relationships in the data. Increasing the number of bins achieves this, providing finer distinctions between values and enabling the model to learn more intricate dependencies. After testing, ten bins appeared to offer enough detail to capture complex patterns without introducing excessive noise or overfitting.

Classes	**Lower Bound**	**Upper Bound**
Class 0	-0.9992	-0.1921
Class 1	-0.1921	0.1374
Class 2	0.1374	1883.0580

TABLE 2.1: Quantile-based binning of the target variable `return_12m_forward` into three classes. The bin edges are fitted on the training set and then applied consistently to validation and test sets. Class 0 corresponds to negative returns (below the first tercile), Class 1 to neutral or uncertain outcomes (middle tercile), and Class 2 to positive returns (above the second tercile). This three-class design balances interpretability and predictive difficulty, contrasting with two-class (too coarse) or five-class (too granular) alternatives.

Finally, all features are cast to integer type to ensure compatibility with the Transformer model.

In the end, this process yields three cleaned and discretised datasets, one for training, one for validation, and one for testing, ready for modelling.

Chapter 3

Transformer Model (Methodology and Data)

In this chapter, we describe the Transformer architecture choices, their meanings, and the rationale behind their use.

3.1 Overview of Transformers

A standard Transformer consists of two components: an encoder and a decoder. The encoder maps inputs into contextual representations using dense vectors to capture both content and relationships while the decoder generates an output sequence, typically one token at a time. For example, in machine translation the decoder sequentially produces the translated sentence from the encoded input.

The encoder relies on self-attention (often simply referred to as attention), which models dependencies within a sequence. In the sentence “I eat food,” the model may learn that “eat” is more related to “food” than to “I,” and assign higher attention weights accordingly **(Lin et al. 2017)**.

In this study the decoder is unnecessary, as the task does not involve sequence generation. Instead, the aim is to encode financial ratios and their relationships to form coherent representations, reducing the architecture to an encoder-only configuration.

3.1.1 Embedding and Positional Encoding

Word embedding is a fundamental technique in natural language processing (NLP), where each token in a sequence is represented as a numerical vector. A simple example is one-hot encoding, which assigns each token a unique binary vector but fails to capture any semantic relationships between tokens.

To address this limitation, we use **dense embeddings**: low-dimensional, continuous vectors learned during training. Tokens with similar statistical or semantic properties are placed close in the embedding space. In our setting, this allows the model to learn meaningful representations of

financial ratios, capturing relationships and functional overlap, thereby improving interpretability and performance.

Let the feature vocabulary size (number of discrete categories/buckets for a feature) be V and the embedding dimension d_{model}. We learn an embedding matrix $E \in \mathbb{R}^{V \times d_{\text{model}}}$ and map each categorical index $x_i \in \{1, \ldots, V\}$ to

$$e_i \;=\; \mathrm{Emb}(x_i) \;=\; E_{x_i:} \;\in\; \mathbb{R}^{d_{\text{model}}}.$$

Stacking these L vectors forms the Transformer input:

$$X \;=\; \begin{bmatrix} e_1 \\ \vdots \\ e_L \end{bmatrix} \in \mathbb{R}^{L \times d_{\text{model}}}, \qquad \text{batched: } X \in \mathbb{R}^{B \times L \times d_{\text{model}}}.$$

Implementation note. In practice, we learn a separate embedding table $E^{(f)}$ for each feature group (ratio buckets, imputation flags, year, stock, industry, industry format, country). These embeddings are stacked to form the input sequence, whose length is

$$L \;=\; R + I + \mathbb{1}_{\text{time}} + \mathbb{1}_{\text{stock}} + \mathbb{1}_{\text{industry}} + \mathbb{1}_{\text{format}} + \mathbb{1}_{\text{country}},$$

directly reflecting the code implementation, though it is not required for the general explanation.

Another integral component of the standard Transformer architecture is positional encoding, which conveys information about the order of tokens in a sequence. In natural language processing (NLP), word order is critical to meaning, and positional encoding compensates for the lack of recurrence in Transformer models. However, in this context, the order of input features (financial ratios) does not carry inherent semantic value. For example, “ratio 1 > 3.5, ratio 2 < 4.2” is equivalent in meaning to “ratio 2 < 4.2, ratio 1 > 3.5.” Imposing an artificial sequence through positional encoding would therefore be inappropriate, and is omitted in this study.

3.2 Model Components

After the embedding stage, we construct the Transformer model using prebuilt components from the PyTorch library. As the focus of this study is not on implementing the architecture from first principles, we rely on PyTorch’s native modules. The model architecture consists of three main components: the Transformer encoder layer, which forms the core of the model and captures dependencies between tokens; the pooling strategy, which serves as a bridge between the encoder and the output stage by aggregating the sequence representations; and the classifier, which maps the pooled representation to the final prediction. These three components are

central to the subsequent interpretability analysis and are therefore directly linked to addressing the primary research question.

3.2.1 Transformer Encoder Layer

The core computation is performed within the `nn.TransformerEncoderLayer` module. This layer integrates the essential components of the Transformer architecture, including multi-head self-attention, layer normalisation, and a feedforward network all efficiently abstracted within the PyTorch framework.

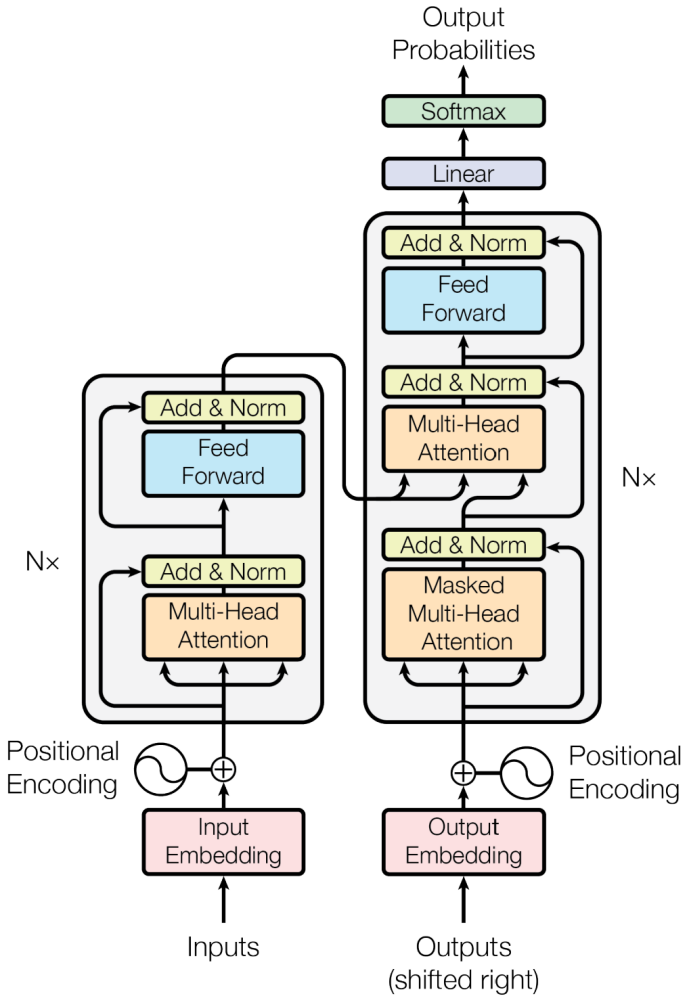


FIGURE 3.1: The overall Transformer architecture from the *Attention Is All You Need* paper **(Vaswani et al. 2017)**, showing both the encoder and decoder components. This figure provides useful context for the following section, where the encoder-only adaptation is described in detail.

For notational clarity, let $H^{(\ell)} \in \mathbb{R}^{B\times L\times d_{\text{model}}}$ denote the hidden representation after the ℓ-th encoder layer, where B is the batch size, L is the sequence length (i.e., the number of feature tokens), and d_{model} is the embedding dimension. The initial hidden state is given by

$$H^{(0)} = X,$$

the input embedding matrix previously introduced. At the token level, we write $H_t^{(\ell)} \in \mathbb{R}^{d_{\text{model}}}$ for the representation of token t after layer ℓ, with $H_t^{(0)} = e_t$ corresponding to its embedding.

The multi-head attention block is the core component of the Transformer architecture, designed to capture relationships between elements in a sequence through a learned Query-Key-Value (QKV) structure. Given $H^{(\ell-1)}$ as input to layer ℓ, the model computes projections:

$$Q = H^{(\ell-1)}W_Q, \qquad K = H^{(\ell-1)}W_K, \qquad V = H^{(\ell-1)}W_V,$$

with learnable weights

$$W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}}\times d_k}, \qquad d_k = \frac{d_{\text{model}}}{n_{\text{heads}}},$$

where n_{heads} is the number of parallel attention heads. This results in

$$Q, K, V \in \mathbb{R}^{B\times L\times d_k},$$

where B is the batch size and L the sequence length.

The attention mechanism is then defined as:

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

where

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{L} e^{z_j}}, \qquad i = 1, \ldots, L.$$

where the scaled dot-product operation computes similarity scores between sequence elements. The resulting attention matrix $A \in \mathbb{R}^{B\times L\times L}$ encodes how much focus each token places on others in the sequence and is central to our model interpretability analysis.

The formulation above describes the attention mechanism for a *single head*. In practice, the Transformer employs multiple such heads in parallel, with each head operating on a different learned projection of the input space. For head $h \in \{1, \ldots, n_{\text{heads}}\}$, the computation is given by

$$\text{head}^{(h)} = \text{Attention}\big(Q^{(h)}, K^{(h)}, V^{(h)}\big),$$

with

$$Q^{(h)} = H^{(\ell-1)} W_Q^{(h)}, \qquad K^{(h)} = H^{(\ell-1)} W_K^{(h)}, \qquad V^{(h)} = H^{(\ell-1)} W_V^{(h)}.$$

The outputs of all heads are concatenated and linearly transformed to form the **multi-head** attention output:

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}\big(\mathsf{head}^{(1)}, \ldots, \mathsf{head}^{(n_{\mathsf{heads}})}\big) \in \mathbb{R}^{B \times L \times (n_{\mathsf{heads}} \cdot d_k)},$$

$$Y = \mathrm{MultiHead}(Q, K, V)\, W_O, \qquad W_O \in \mathbb{R}^{(n_{\mathsf{heads}} \cdot d_k) \times d_{\mathsf{model}}}, \quad Y \in \mathbb{R}^{B \times L \times d_{\mathsf{model}}}.$$

To stabilise training, the model applies a residual connection followed by layer normalisation. Specifically, for each token vector $y_t \in Y$, the residual pathway adds the original input $H_t^{(\ell-1)}$, applies dropout with rate p_{tf}, and then normalises the result:

$$\tilde{H}_t^{(\ell)} = \mathrm{LN}\big(H_t^{(\ell-1)} + \mathcal{D}_{p_{\mathsf{tf}}}(y_t)\big),$$

where LN denotes Layer Normalisation and $\mathcal{D}_{p_{\mathsf{tf}}}$ denotes dropout with rate p_{tf}.

The intermediate representation $\tilde{H}^{(\ell)}$ is then passed through a position-wise feedforward network (FFN) applied independently to each sequence element:

$$\mathsf{FFN}(\tilde{H}_t^{(\ell)}) = W_2\, \sigma(W_1 \tilde{H}_t^{(\ell)} + b_1) + b_2,$$

where σ is typically a ReLU or GELU activation (see **Appendix B**). The weight matrices and biases are

$$W_1 \in \mathbb{R}^{d_{\mathsf{model}} \times d_{\mathsf{ff}}}, \quad b_1 \in \mathbb{R}^{d_{\mathsf{ff}}}, \quad W_2 \in \mathbb{R}^{d_{\mathsf{ff}} \times d_{\mathsf{model}}}, \quad b_2 \in \mathbb{R}^{d_{\mathsf{model}}},$$

with d_{ff} denoting the dimensionality of the hidden layer in the feedforward network.

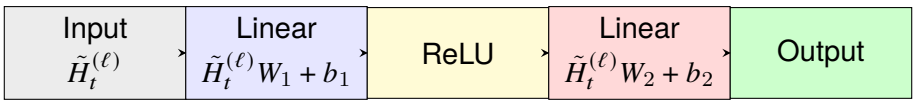


FIGURE 3.2: Feedforward network: $\mathsf{FFN}(\tilde{H}_t^{(\ell)}) = \mathsf{ReLU}(\tilde{H}_t^{(\ell)} W_1 + b_1) W_2 + b_2$

Finally, a second residual connection and layer normalisation are applied to yield the output of layer ℓ:

$$H^{(\ell)} = \mathrm{LN}\big(\tilde{H}^{(\ell)} + \mathcal{D}_{p_{\mathsf{tf}}}(\mathsf{FFN}(\tilde{H}^{(\ell)}))\big).$$

This sequence of operations (multi-head self-attention, residual connection with normalisation, feedforward transformation, and a second residual connection with normalisation) defines a **single Transformer encoder layer**. The complete encoder is obtained by stacking n_{layers} where

n_{layers} denotes the chosen model depth. Stacking increases the representational capacity of the model, allowing successive layers to capture higher-level and more abstract dependencies in the data.

Since each encoder layer preserves the dimensionality of its input, the final encoder output is:

$$H^{(n_{\text{layers}})} \in \mathbb{R}^{B\times L\times d_{\text{model}}}.$$

For each batch element, this corresponds to a matrix of size $L\times d_{\text{model}}$, where each row represents the contextualised embedding of a single feature token. This sequence-level representation forms the input to the subsequent pooling stage.

3.2.2 Pooling Strategy

Pooling serves as an intermediary between the Transformer encoder and the classification head, converting the sequence output into a fixed-size representation suitable for the classification head. While it does not enrich the underlying output, it reformats it for compatibility with the classifier.

Formally, let the final encoder output be

$$H^{(n_{\text{layers}})} \in \mathbb{R}^{B\times L\times d_{\text{model}}},$$

with $H_b^{(n_{\text{layers}})} \in \mathbb{R}^{L\times d_{\text{model}}}$ denoting the output for batch element b. Each row

$$H_{b,i}^{(n_{\text{layers}})} \in \mathbb{R}^{d_{\text{model}}}, \qquad i \in \{1,\dots,L\},$$

corresponds to the contextualised embedding of the i-th token position.

We introduce a learnable weight vector $w \in \mathbb{R}^L$, where each entry w_i represents the unnormalised importance score of the i-th token position. These weights are normalised with a softmax:

$$\alpha_i = \frac{e^{w_i}}{\sum_{j=1}^{L} e^{w_j}}, \qquad \alpha \in \mathbb{R}^L, \qquad \sum_{i=1}^{L} \alpha_i = 1.$$

The pooled representation for sample b is then computed as a weighted sum over the token embeddings:

$$\bar{h}_b = \sum_{i=1}^{L} \alpha_i \, H_{b,i,:}^{(n_{\text{layers}})} \in \mathbb{R}^{d_{\text{model}}},$$

or equivalently in matrix form,

$$\bar{h}_b = \alpha^\top H_b^{(n_{\text{layers}})}.$$

Stacking across the batch yields the pooled representation

$$\bar{H} \in \mathbb{R}^{B \times d_{\text{model}}},$$

which is passed to the classification head.

3.2.3 Classification Head

The final stage of the model architecture is the classification head, which receives the pooled Transformer output and produces the final predictions. This head consists of a lightweight feedforward neural network designed to balance model capacity and apply some regularisation.

Let the pooled batch representation be

$$\bar{H} \in \mathbb{R}^{B \times d_{\text{model}}},$$

with row $\bar{h}_b \in \mathbb{R}^{d_{\text{model}}}$ corresponding to batch element b. Let C denote the number of classes (in our case $C = 3$), and define the hidden dimension $d_h = \lfloor d_{\text{model}}/2 \rfloor$ and dropout rate p_{clf}.

The classification head is applied row-wise as follows:

1. Layer normalisation stabilises training by standardising activations, ensuring consistent distributional properties across tokens **(Ba et al. 2016)**:

$$\tilde{H} = \mathrm{LN}(\bar{H}).$$

2. Linear projection reduces the representation to d_h:

$$U = \tilde{H}W^{(1)} + b^{(1)}, \qquad W^{(1)} \in \mathbb{R}^{d_{\text{model}} \times d_h}, \;\; b^{(1)} \in \mathbb{R}^{d_h}.$$

3. Non-linear activation increases expressivity:

$$V = \mathrm{ReLU}(U).$$

4. Dropout further regularises the hidden activations by randomly deactivating units during training, reducing overfitting and enhancing generalisation **(Srivastava et al. 2014)**:

$$V' = \mathcal{D}_{p_{\text{clf}}}(V).$$

5. A final linear layer maps to the class logits:

$$\text{Logits} = V' W^{(2)} + b^{(2)}, \qquad W^{(2)} \in \mathbb{R}^{d_h \times C}, \;\; b^{(2)} \in \mathbb{R}^{C}.$$

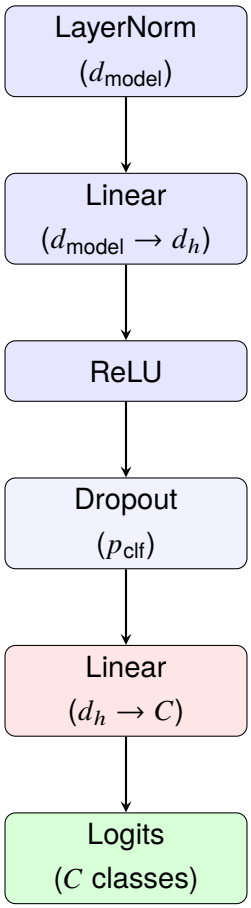


FIGURE 3.3: Classification head of the Transformer. The pooled vector passes through LayerNorm, a hidden projection to d_h, ReLU activation, dropout with rate p_{clf}, and a final linear layer producing C logits.

3.3 Training Loop and Validation

After the overall architecture of the transformer and classifier has been defined, we create a training function that trains and evaluates the model. The training function is designed to take various key input parameters that directly affect model performance.

3.3.1 Overall Training Implementation

The training procedure follows a standard supervised learning loop. The processed datasets (training, validation, and test splits) are loaded from the data pipeline into the training function.

3.3.2 Loss Function (Cross Entropy and Weights)

The loss function plays a central role in training by quantifying the discrepancy between predicted outputs and true labels. Specifically, the loss is computed at each *epoch*, which refers to one full pass through the training data.

Given that the objective is a multi-class classification task, the targets are constructed using quantile-based discretisation applied exclusively to the training data. The same bin edges are subsequently used to label the validation and test sets. As a result, the training set exhibits perfectly balanced class distributions by design, whereas the validation set and possibly the test set are only approximately balanced.

We can confirm this balance by examining the class counts:

Class Label	Train	Validation
0	37,754	5,264
1	37,754	8,874
2	37,754	8,592

TABLE 3.1: Sample distribution across discretised return buckets for both training and validation sets. Class labels $0, 1, 2$ correspond to the three quantile-based categories of forward returns. The training set is exactly balanced by construction through quantile discretisation, while the validation set reflects the natural distribution of the same bins applied to unseen data. This table illustrates how class balance is enforced during training but only approximately preserved in validation.

To account for the approximate class imbalance in the validation and test sets, the model employs standard **cross-entropy loss** with **class weights** and **label smoothing**. Class weights, when treated as a tunable hyperparameter, help mitigate bias towards majority classes and improve the reliability of model evaluation. Label smoothing further regularises training by preventing the model from becoming overconfident in its predictions, thereby improving generalisation **(Müller et al. 2020)**.

As previously discussed, the classifier produces raw logits:

$$\text{Logits} \in \mathbb{R}^{B \times C},$$

where B is the batch size and C the number of classes.

Applying the softmax function converts logits into probabilities:

$$\hat{p}_{i,c} = \frac{e^{\text{logits}_{i,c}}}{\sum_{j=1}^{C} e^{\text{logits}_{i,j}}}, \qquad \hat{\mathbf{p}}_i \in \mathbb{R}^C,$$

where $\hat{p}_{i,c}$ is the probability assigned to class c for batch element i.

Label smoothing. Instead of using a strict one-hot target distribution, label smoothing assigns the true class a probability of $1 - \epsilon$ and distributes the remaining ϵ uniformly across the other $C - 1$ classes:

$$q_{i,c} = \begin{cases} 1 - \epsilon & \text{if } c = y_i, \\ \frac{\epsilon}{C-1} & \text{otherwise,} \end{cases}$$

where ϵ is the smoothing parameter.

Weighted cross-entropy. To compensate for class imbalance, each class c is also assigned a weight $w_c > 0$. The weighted, smoothed cross-entropy loss is then:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} w_c \, q_{i,c} \, \log \hat{p}_{i,c},$$

where N is the number of samples.

3.3.3 Optimisation and Backpropagation

The optimiser is applied to the weights of the model during training. It determines how to update these weights in order to best minimise the overall loss. This is done using the gradients obtained via backpropagation, specifically through the `loss.backward()` call in PyTorch. The optimiser updates the weights in the direction opposite to the gradient.

In our base case, we use the **Adam** optimiser **(Kingma and Ba 2017)**, which is one of the most widely used optimisers in modern machine learning. For more information about Adam and its variations, refer to **Appendix B**

3.3.4 Adaptive Fine Tuning and Early Stopping

Transformers are considerably more data-intensive than traditional methods such as ensemble models or classical classifiers **(Kaplan et al. 2020)**. While capable of modelling complex relationships, this expressiveness increases the risk of overfitting, especially with the modest sample size of approximately 100,000 observations used in this study. Given the limited computational resources available, this constraint can negatively affect the performance and stability of the Transformer model.

To mitigate these issues, we employ an *adaptive fine-tuning* strategy with patience-based early stopping **(Stollenwerk 2022)**. Unlike the original formulation, which also adapts the learning rate and training horizon, our implementation retains a standard scheduler (e.g., cosine annealing, step decay) and halts training when no improvement is observed within a fixed patience window. This ensures computational efficiency by preventing unnecessary epochs while attempting to preserve the "best" performing model.

3.4 Hyperparameter Tuning

Given the modest dataset size, hyperparameters play a critical role in Transformer performance, with small changes often producing large variations in predictive behaviour. Careful and systematic tuning is therefore essential.

3.4.1 Optuna (Hyperparameter Search)

Traditional search methods such as grid and random search are widely used, with random search often outperforming grid search in high-dimensional spaces **(Bergstra and Bengio 2012)**. In this study, we employ the **Optuna** framework, which extends random search with adaptive sampling and pruning. These features enable more efficient exploration of the hyperparameter space, yielding faster convergence and more effective identification of high-performing configurations (see **Appendix C** for more details).

Chapter 4

Empirical Methods, Results, and Discussion

This chapter is divided into three parts, each dictating a portion of the overall analysis. **Section 4.1** outlines the *prerequisites* required before any further analysis, including the choice of a baseline comparison model (XGBoost) and a fixed set of hyperparameters obtained via Optuna. **Section 4.2** evaluates the *stability and robustness* of the Transformer model under both standard and temporal conditions. **Section 4.3** investigates the *interpretability* of a specific transformer instance. Together, these components provide a foundation for addressing the core research question. Results may vary with each run due to the inherent randomness built into the transformer itself (see **Seed Robustness** for more details).

Data Partitioning

It is also important to note that the time splits for training, validation, and testing vary depending on the specific analysis conducted. For the baseline configuration and most experiments, the following split is adopted:

$$t_1 = 2019, \quad t_2 = 2021, \quad t_3 = 2024$$

Which corresponds to:

- **Training years:** 2006–2019
- **Validation years:** 2020–2021
- **Test years:** 2022–2024

This configuration corresponds approximately to a 70/15/15 split between training, validation, and test sets, a convention widely adopted in machine learning research **(*Datasets: Dividing the original dataset* | *Machine Learning* | *Google for Developers* n.d.)**. Unless otherwise stated, this partitioning scheme is used as the default throughout all subsequent analyses. Any deviations will be explicitly specified.

4.1 Prerequisites of the Transformer Model

4.1.1 External Model Results: XGBoost

To contextualise the performance and interpretability of the Transformer model, we benchmark it against a widely used, conventional machine learning method: XGBoost (Results for a Random Forest baseline are also provided in **Appendix B**). This comparison serves two purposes. First, it establishes a performance baseline using a model known for its strong results on tabular data **(Shwartz-Ziv and Armon 2021)**. Second, it provides an alternate interpretability perspective when combined with SHAP analysis, allowing us to assess whether the added complexity of the Transformer model yields meaningful gains. For further details on XGBoost, see **Appendix B**.

This model is trained and evaluated using data split mentioned in **Data Partitioning**, and performance is assessed through the same classification metrics as the Transformer.

Each classification report summarises model performance across the three output classes defined in this study. For each class, standard evaluation metrics are reported: precision, recall, F1-score, and support. Additionally, the macro and weighted averages provide aggregate performance across all classes. For further details, see **Appendix B**.

Using a 24-fit cross validation setup, the XGBoost Model achieved a validation accuracy of 42.34% and a test accuracy of 42.1% (**Table 4.1**). Compared to the random baseline of selecting 3 classes (33.33%), this indicates that the model captures meaningful patterns in the data, providing confidence in its quality.

Metric	**Value**
Classes	3
Candidates Evaluated	8 (total of 24 fits)
Best Hyperparameters	`learning_rate` = 0.05, `max_depth` = 5, `n_estimators` = 500
Cross-Validated Accuracy	42.34%
Random Baseline (Uniform)	33.33%
Test Set Accuracy	42.1%

TABLE 4.1: Performance summary of the XGBoost baseline model. Results are based on 24 cross-validation fits across 8 hyperparameter candidates, with the best configuration shown. Accuracy is reported on both cross-validation folds and the held-out test set. A random baseline (uniformly guessing among three classes) achieves 33.33%, so the XGBoost model's ~42% accuracy represents a clear improvement and demonstrates its ability to capture non-trivial patterns in the data. These results serve as the benchmark against which the Transformer is compared in subsequent sections.

Class-wise metrics in **Table 4.2** reveal that XGBoost performs best on the middle class (Class 1), with F1-scores ranked as: Class 1 > Class 0 > Class 2. This indicates that the model is most confident in predicting the middle class, while being more hesitant with the high-return class (Class 2), reflected in its low recall of 0.259. Despite this, Class 2 exhibits a relatively high accuracy of 61.6%, suggesting that when the model does predict Class 2, it is often correct.

	Confusion counts				**Metrics**				
Class	**TP**	**FP**	**FN**	**TN**	**Prec.**	**Recall**	**F1**	**OvR Acc.**	**Support**
0	2855	4841	3939	13445	0.371	0.420	0.394	0.650	6794
1	5462	6431	4218	8969	0.459	0.564	0.506	0.576	9680
2	2232	3259	6374	13215	0.406	0.259	0.317	0.616	8606
Overall (micro) acc.	**Correct** = $10{,}549/25{,}080$						**0.421**		
Macro avg					0.412	0.415	0.406		
Weighted avg					0.417	0.421	0.411		

TABLE 4.2: XGBoost (Test-set) per-class confusion counts and derived metrics for a 3-class classifier. For each class k we view the problem in a one-vs-rest manner and define: TP_k (true positives) = # items of class k predicted as k; FP_k (false positives) = # items *not* in k predicted as k; FN_k (false negatives) = # items in k predicted as another class; TN_k (true negatives) = # items *not* in k predicted as not k. Metrics are then: $\mathrm{Precision}_k = \dfrac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FP}_k}$, $\mathrm{Recall}_k = \dfrac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FN}_k}$, $F1_k = \dfrac{2\,\mathrm{Precision}_k\,\mathrm{Recall}_k}{\mathrm{Precision}_k + \mathrm{Recall}_k}$, and one-vs-rest accuracy $\mathrm{OvRAcc}_k = \dfrac{\mathrm{TP}_k + \mathrm{TN}_k}{N}$ with total $N = 25{,}080$ test samples. "Support" is the number of true instances in class k. The "Overall (micro) acc" row reports the multiclass *micro-accuracy* $\sum_k \mathrm{TP}_k/N$ (= 0.421 here), which differs from the per-class one-vs-rest accuracies. "Macro avg" is the unweighted mean of per-class Precision/Recall/F1; "Weighted avg" uses class supports as weights. Counts are integers; metric entries are between $[0, 1]$. Overall, using the test data, the XGBoost model is most reliable on Class 1 (highest F1); Class 0 is precision-limited (many false positives) and Class 2 is recall-limited (many false negatives), suggesting boundary overlap and a tendency to default to the middle class.

In terms of SHAP importance (see **Appendix B** for more details on SHAP) (**Figure 4.1**), the most influential features are the `book_to_market_buckets`, followed by the `momentum_buckets`. These results are consistent with established literature, where both book to market and momentum effects have been shown to be significant predictors of returns **(Fama and French 1993; Jegadeesh and Titman 2011)**.

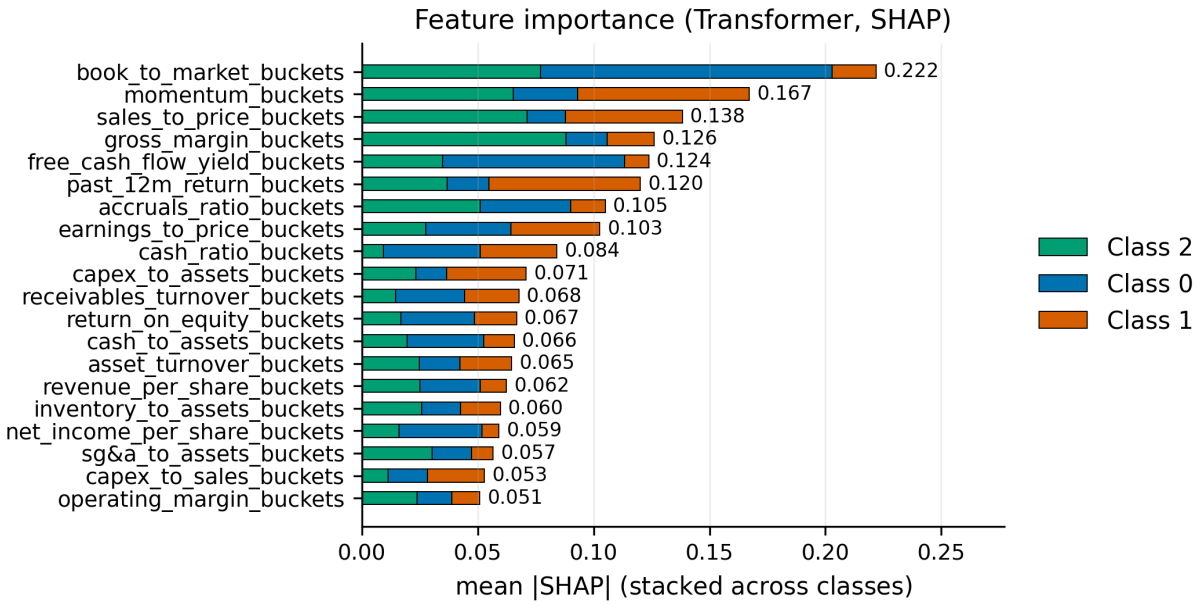


FIGURE 4.1: SHAP feature importance scores from the XGBoost model. Bars show the mean absolute SHAP value (average contribution magnitude) for each feature, separated by class. Values range from 0 (no influence on predictions) upwards, with higher scores indicating stronger average impact on the model's output. The most influential features are `book_to_market_buckets` and `momentum_buckets`, with contributions varying across classes.

4.1.2 Optuna Results

Given the relatively modest dataset size for a Transformer, hyperparameters play a critical role in model performance. Even small variations in their values can lead to significant changes in predictive behaviour, potentially resulting in unstable or misleading outcomes. Therefore, before analysing and interpreting the Transformer, careful and systematic hyperparameter tuning is essential to avoid any misleading outcomes.

To this end, we employ an effective search strategy to identify an optimal configuration. While traditional methods such as grid search and random search are widely used, empirical studies have demonstrated that random search often outperforms grid search, particularly in high-dimensional spaces **(Bergstra and Bengio 2012)**. In this study, we adopt a more advanced approach using the **Optuna** framework. While similar in spirit to random search, Optuna improves upon it through the use of adaptive sampling strategies and pruning mechanisms. These enhancements allow for more efficient exploration of the hyperparameter space, leading to faster convergence and more effective identification of high performing configurations.

Choice of Hyperparameter Ranges Our choice of hyperparameter ranges was informed by early testing. We found that "simpler" models consistently outperformed overly complex ones.

This result aligns with our intuition, especially given the limited dataset, and is consistent with prior work **(Gorishniy et al. 2023)**.

Hyperparameter	**Search Space / Values**
Embedding Dimension (d_{model})	{16, 32, 64}
Number of Layers (n_{layers})	{1, 2, 4, 6}
Number of Heads (n_{heads})	{1, 2, 4, 6}
Transformer Dropout (p_{tf})	Uniform in [0.2, 0.5]
Classifier Dropout (p_{cls})	Uniform in [0.1, 0.4]
Learning Rate (η)	Log-uniform in $[2 \times 10^{-5}, 3 \times 10^{-4}]$
Weight Decay (λ)	Log-uniform in $[10^{-6}, 10^{-4}]$
Class 1 Weight (w_{pos})	Uniform in [0.8, 1.3]
Class Weights	{1.0, w_{pos}, $1.2 \cdot w_{\text{pos}}$}
Optimiser (O)	{Adam, AdamW, SGD}
Scheduler ($\mathcal{S}$)	{CosineAnnealing, ReduceLROnPlateau, StepLR}
Scheduler-Specific Parameters	
CosineAnnealing (T_0)	Integer in [1, 5], $T_{\text{mult}} = 2$
StepLR (γ)	Fixed at 0.1
StepLR (τ)	Integer in [1, 5]
Plateau Patience (π)	Integer in [2, 4]
Plateau Factor (ϕ)	Uniform in [0.4, 0.8]

TABLE 4.3: Hyperparameter search space for tuning the Transformer model with Optuna. Discrete parameters such as the embedding dimension (d_{model}), number of layers (n_{layers}), and heads (n_{heads}) are tested over fixed sets. Continuous parameters including Transformer dropout (p_{tf}), classifier dropout (p_{cls}), learning rate (η), and weight decay (λ) are sampled from the specified intervals. Class weights are scaled relative to w_{pos}. Optimiser choices are denoted by O and scheduler choices by $\mathcal{S}$, with scheduler-specific parameters (T_0, τ, π, ϕ) listed separately. A detailed overview of the Optuna setup is provided in **Appendix C**.

Best Trial Summary The optimal hyperparameters were identified across five separate Optuna trials, each run independently (see **Table 4.4**). These trials took 5–6 hours in total. Trial numbers are not ordered chronologically; they are used solely for referencing. For a detailed overview of the Optuna model, including its configuration, search space, and interpretation of results, please refer to **Appendix C**.

Hyperparameter Trials												
Trial	$\mathcal{J}_{obj}$	d_{model}	n_{layers}	n_{heads}	p_{tf}	p_{cls}	η	λ	w_{pos}	O	$\mathcal{S}$	τ/π
1	-0.575	64	4	4	0.417	0.386	2.61e-4	2.49e-6	1.017	SGD	plateau	$\pi = 4$
2	-0.571	16	1	2	0.331	0.280	1.83e-4	8.38e-5	1.198	AdamW	step	$\tau = 3$
3	-0.568	64	2	2	0.401	0.114	6.59e-5	2.61e-5	1.012	AdamW	step	$\tau = 3$
4	-0.528	32	2	4	0.235	0.178	1.71e-4	3.90e-6	1.212	Adam	plateau	$\pi = 2$
5	-0.522	64	4	2	0.333	0.207	2.07e-5	3.77e-6	1.285	Adam	step	$\tau = 5$

TABLE 4.4: Top five hyperparameter trials ranked by the custom objective score $\mathcal{J}_{obj}$, where higher values indicate better performance. Each row corresponds to one Optuna trial with its Transformer architecture (d_{model}, n_{layers}, n_{heads}), regularisation settings (p_{tf}, p_{cls}, λ), optimisation settings (η, O), and learning schedule ($\mathcal{S}$ with parameters τ or π).

Summary of Findings The most promising configuration sets identified in **Table 4.4** shared several characteristics. Learning rates tended to be moderate to high, with batch sizes in the range of 16–32. Both cosine and plateau schedulers were effective, typically in combination with either the Adam or AdamW optimisers. Embedding dimensions of 32 or higher yielded strong results, while the number of layers was generally between one and four. Finally, the attention mechanism favoured a relatively small number of heads, most often between two and four.

And the best results (Trial 5):

Model Architecture	
Embedding dimension (d_{model})	64
Number of Transformer layers (n_{layers})	4
Number of Attention heads (n_{heads})	2
Dropout (Transformer, p_{tf})	0.333
Dropout (Classifier, p_{cls})	0.207
Activation function	ReLU
Training	
Epochs	15
Batch size	32
Learning rate (η)	2.07×10^{-5}
Weight decay (λ)	3.77×10^{-6}
Class weights	[1.0, $w_{\text{pos}} = 1.29$, $1.2 \cdot w_{\text{pos}} = 1.54$]
Optimiser (O)	Adam
Scheduler ($\mathcal{S}$)	StepLR ($\tau = 5$, $\gamma = 0.1$)
Early stopping patience (π_{es})	15
Label smoothing (σ_ℓ)	0.136

TABLE 4.5: Best hyperparameters from Optuna. Here d_{model}, n_{layers}, n_{heads}, p_{tf}, p_{cls}, η, λ, w_{pos}, O, $\mathcal{S}$, τ, γ, π_{es}, and σ_ℓ denote the embedding dimension, number of layers, number of heads, Transformer dropout, classifier dropout, learning rate, weight decay, positive class weight, optimiser, scheduler, StepLR step size, StepLR decay factor, early stopping patience, and label smoothing parameter, respectively.

Note: All further analyses and experiments are conducted using the hyperparameters in **Table 4.5**, unless stated otherwise.

4.2 Stability and Robustness of the Transformer Model

In this section, we evaluate the model's robustness through two approaches: overall performance metrics (e.g., F1-score) and temporal analysis across multiple time periods to assess consistency over time. In both cases, the Transformer is evaluated using a fixed set of hyperparameters derived from Optuna (**Table 4.5**), which should be kept in mind when interpreting the results.

4.2.1 Transformer Results (Using Optimal Results from Optuna)

The classification report for the **Validation Set** (**Table 4.6**) highlights several notable patterns. **Class 2** achieves the highest F1-score (0.48), driven by strong recall (0.53) and solid precision

(0.44). **Class 1** also performs well, with balanced precision and recall of 0.50 and 0.40, respectively. In contrast, **Class 0** shows weaker performance, with a precision of 0.32 and recall of 0.33.

	Confusion counts				**Metrics**				
Class	**TP**	**FP**	**FN**	**TN**	**Prec.**	**Recall**	**F1**	**OvR acc.**	**Support**
0	1754	3660	3510	13806	0.324	0.333	0.329	0.685	5264
1	3507	3501	5367	10355	0.500	0.395	0.442	0.610	8874
2	4569	5739	4023	8399	0.443	0.532	0.483	0.571	8592
Overall (micro) acc.	**Correct** = $9{,}830/22{,}730$				**0.433**				
Macro avg					0.423	0.420	0.418		
Weighted avg					0.438	0.432	0.431		

TABLE 4.6: Transformer (validation) per-class confusion counts and derived metrics for a 3-class classifier. Values come from the multiclass confusion matrix with *rows = true (actual)* and *columns = predicted*. For each class k (treated one-vs-rest) we define: TP_k (true positives) = # items of class k predicted as k; FP_k (false positives) = # items *not* in k predicted as k; FN_k (false negatives) = # items in k predicted as another class; TN_k (true negatives) = # items *not* in k predicted as not k. Metrics are: $\mathrm{Precision}_k = \dfrac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FP}_k}$, $\mathrm{Recall}_k = \dfrac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FN}_k}$, $F1_k = \dfrac{2\,\mathrm{Precision}_k\,\mathrm{Recall}_k}{\mathrm{Precision}_k + \mathrm{Recall}_k}$, and one-vs-rest accuracy $\mathrm{OvRAcc}_k = \dfrac{\mathrm{TP}_k + \mathrm{TN}_k}{N}$ with total $N = 22{,}730$ validation samples. "Support" is the number of true instances in class k. The "Overall (micro) acc." row reports multiclass micro-accuracy $\sum_k \mathrm{TP}_k/N = 0.433$ here (Correct = $9{,}830/22{,}730$), which differs from per-class OvR accuracies. "Macro avg" is the unweighted mean across classes; "Weighted avg" uses class supports as weights. Counts are integers; metric entries are in $[0, 1]$. On this validation set, Class 2 attains the highest $F_1 = 0.483$ (driven by higher recall 0.532); Class 1 is moderate ($F_1 = 0.442$) with higher precision (0.500) than recall (0.395); Class 0 is weakest ($F_1 = 0.329$) with both precision and recall near one-third.

This trend continues in the **Test Set** (**Table 4.7**), where Class 1 achieves the highest F1-score (0.43), followed by Class 2 (0.41), while Class 0 remains the weakest (0.34). The overall test accuracy is slightly lower, as expected due to generalisation effects.

When compared to the XGBoost baseline (**Subsection 4.1.1**), the Transformer shows a clear improvement for Class 2 (F1: 0.41 vs. 0.32), but lower performance for Class 1 (F1: 0.43 vs. 0.51) and Class 0 (F1: 0.34 vs. 0.39). Compared to the XGBoost baseline (Macro F1: 0.41, Weighted F1: 0.41), the Transformer achieves broadly similar aggregate performance (Macro F1: 0.39, Weighted F1: 0.40), indicating that while its per-class trade-offs differ, its overall effectiveness is on par with the benchmark.

	Confusion counts				Metrics				
Class	**TP**	**FP**	**FN**	**TN**	**Prec.**	**Recall**	**F1**	**OvR acc.**	**Support**
0	2183	3901	4611	14385	0.359	0.321	0.339	0.661	6794
1	3922	4547	5758	10853	0.463	0.405	0.432	0.589	9680
2	3937	6590	4669	9884	0.374	0.457	0.412	0.551	8606
Overall (micro) acc.	**Correct** = $10{,}042/25{,}080$						**0.400**		
Macro avg					0.399	0.395	0.394		
Weighted avg					0.404	0.400	0.400		

TABLE 4.7: Transformer (test) per-class confusion counts and derived metrics for a 3-class classifier. Values are computed from the multiclass confusion matrix with *rows* = *true (actual)* and *columns* = *predicted*. For each class k (treated one-vs-rest) we define: TP_k (true positives) = # items of class k predicted as k; FP_k (false positives) = # items *not* in k predicted as k; FN_k (false negatives) = # items in k predicted as another class; TN_k (true negatives) = # items *not* in k predicted as not k. Per-class metrics are: $\mathrm{Precision}_k = \dfrac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FP}_k}$, $\mathrm{Recall}_k = \dfrac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FN}_k}$, $F1_k = \dfrac{2\,\mathrm{Precision}_k\,\mathrm{Recall}_k}{\mathrm{Precision}_k + \mathrm{Recall}_k}$, and one-vs-rest accuracy $\mathrm{OvRAcc}_k = \dfrac{\mathrm{TP}_k + \mathrm{TN}_k}{N}$ with total $N = 25{,}080$ test samples. "Support" is the number of true instances in class k. The "Overall (micro) acc." row reports multiclass micro-accuracy $\sum_k \mathrm{TP}_k/N = 0.400$ here (Correct = $10{,}042/25{,}080$), which differs from per-class OvR accuracies. "Macro avg" is the unweighted mean across classes; "Weighted avg" uses class supports as weights. Counts are integers; metric entries are in $[0, 1]$. On this test set, the model is most reliable on Class 1 ($F_1 = 0.432$); Class 2 shows higher recall than precision (0.457 vs. 0.374), indicating many false positives, while Class 0 is weakest overall ($F_1 = 0.339$).

Plots of **validation loss** and **validation accuracy** are also included, showing how these metrics evolve over training epochs. *(An epoch denotes one full pass through the training data.)* **Figure 4.2** shows the training loss, validation loss, and validation accuracy over 15 epochs. Increasing the number of epochs beyond this point yielded minimal improvement. The training loss exhibits a clear downward trend, converging to approximately `1.055`, indicating that the Transformer is successfully learning patterns from the data and demonstrating good in-sample performance.

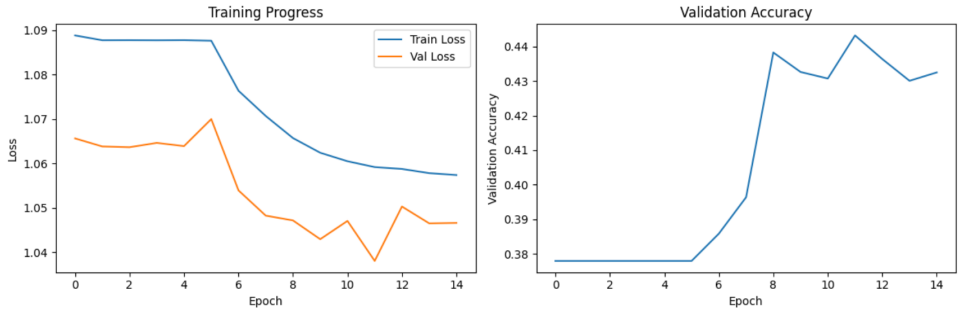


FIGURE 4.2: Training progress of the Transformer model across 15 epochs. The left panel shows training and validation loss, measured as categorical cross-entropy (minimum 0, with random guessing across three classes yielding a baseline of approximately 1.10 loss). Lower values indicate better fit. The right panel shows validation accuracy (ranging from 0 to 1), which reflects the proportion of correctly classified samples at each epoch. Together, these curves illustrate model convergence and the extent of generalisation to unseen data.

The validation loss follows a similar trend, following the train loss, which implies the model is generalising beyond the training set. However, it fluctuates more than the training loss which is expected given the noisy nature of financial data and the smaller size of the validation set (approximately 15% of the total data).

The validation accuracy curve converges quickly to approximately `0.42`, after which it fluctuates around 0.43–0.44. This behaviour is reasonable given the class imbalance, the inherent uncertainty of financial markets, and the relatively small validation set.

Overall, the Transformer model performs on par with XGBoost, demonstrating both resilience and strong predictive capability.

4.2.2 Expanding Window Results

To evaluate the impact of different time periods on model performance, we adopt an expanding window strategy to assess the model's temporal stability over time. This approach mirrors realistic deployment scenarios in financial settings, where models are periodically retrained as new data becomes available. Unlike fixed or sliding windows, the expanding window incrementally increases the training set over time, while advancing the validation and test periods forward. This enables us to assess both the stability and generalisability of the Transformer as it is exposed to progressively larger historical datasets.

Validation Accuracy (**Figure 4.3**): Over time, the model tends to stabilise within a certain performance range, oscillating around a consistent accuracy level. While increased training data does not guarantee improved validation accuracy, some patterns do emerge. A visual

inspection of the validation curves suggests an approximate ranking of years by performance, from highest to lowest: 2020, 2019, 2022, 2015, 2016, 2021, and 2018.

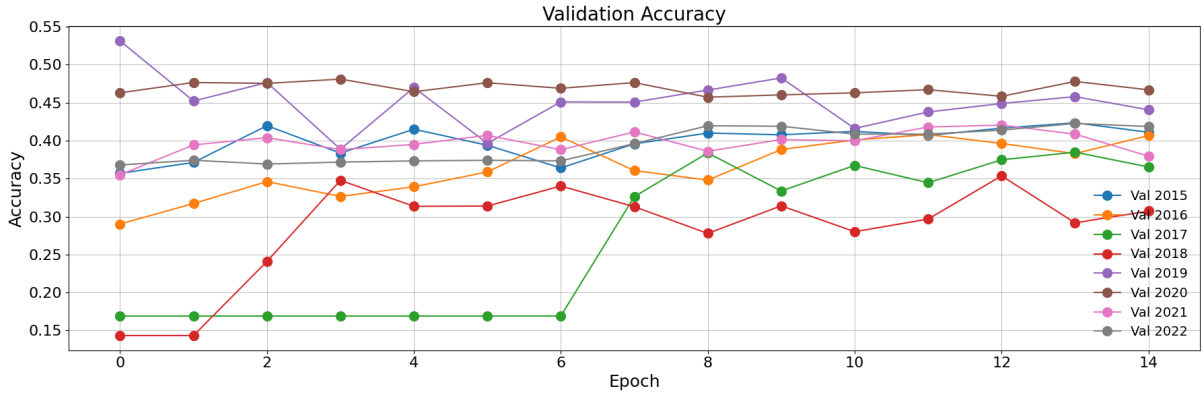


FIGURE 4.3: Validation accuracy trajectories during the expanding window analysis. Each line corresponds to the model trained on data up to the indicated year and evaluated on subsequent validation sets. Accuracy is reported on a 0–1 scale, representing the proportion of correctly classified samples. The plot shows that performance generally stabilises over epochs, with models trained up to 2019 and 2020 achieving the highest validation accuracy (0.45–0.50).

2020 ranks among the best performing years (**Figure 4.3**, **Table 4.8**), which is slightly expected given it was part of the hyperparameter tuning period (see **Subsection 4.1.2**). In contrast, 2021 performs surprisingly poorly despite also being part of the hyperparameter tuning period. A plausible explanation is the impact of COVID-19 **(Davis et al. 2021)** and associated market disruptions, which likely introduced instability and reduced model generalisability.

2019 began with high validation accuracy but failed to improve further (see **Figure 4.3**). Both 2017 and 2018 showed weak initial performance and limited improvement, with 2018 failing to surpass the random benchmark (approximately 33%). This pattern is supported by the yearly validation results in **Table 4.8**, where 2017 and 2018 exhibit particularly low precision for Class 2, leading to weak F1 scores. This likely reflects a lower prevalence of high-return cases in those years, which in turn caused poor generalisation for Class 2.

Year	Class	Validation					Test				
		Precision	Recall	F1	Support	Acc.	Precision	Recall	F1	Support	Acc.
2015	0	0.30	0.36	0.33	1850	0.41	*No test data*				
	1	0.52	0.35	0.41	4101						
	2	0.40	0.52	0.45	3304						
2016	0	0.48	0.23	0.31	3356	0.41	0.42	0.37	0.39	3186	0.41
	1	0.40	0.64	0.49	3411		0.45	0.49	0.47	3549	
	2	0.38	0.33	0.35	2768		0.34	0.34	0.34	2800	
2017	0	0.49	0.34	0.40	4009	0.37	0.53	0.23	0.32	4043	0.39
	1	0.54	0.33	0.41	4187		0.47	0.57	0.52	4193	
	2	0.19	0.52	0.28	1667		0.18	0.34	0.23	1627	
2018	0	0.56	0.18	0.27	5838	0.31	0.57	0.28	0.37	5954	0.32
	1	0.35	0.46	0.40	3192		0.37	0.33	0.35	3185	
	2	0.16	0.48	0.24	1511		0.15	0.52	0.23	1402	
2019	0	0.30	0.23	0.26	2185	0.44	0.32	0.21	0.25	2356	0.42
	1	0.32	0.41	0.36	2853		0.32	0.43	0.37	2939	
	2	0.57	0.53	0.55	5713		0.53	0.52	0.52	5456	
2020	0	0.30	0.31	0.30	2139	0.47	0.31	0.37	0.34	2035	0.47
	1	0.45	0.33	0.38	3812		0.43	0.45	0.44	3645	
	2	0.53	0.64	0.58	5132		0.58	0.53	0.55	5403	
2021	0	0.34	0.30	0.32	3299	0.38	0.31	0.30	0.31	3125	0.38
	1	0.54	0.34	0.42	5044		0.52	0.33	0.41	5062	
	2	0.31	0.52	0.39	3304		0.32	0.51	0.39	3460	
2022	0	0.50	0.31	0.38	4077	0.42	0.49	0.36	0.42	4018	0.41
	1	0.43	0.48	0.45	4440		0.46	0.33	0.38	4548	
	2	0.37	0.46	0.41	3716		0.35	0.57	0.44	3667	
2023	0	*No validation data*					0.31	0.23	0.26	3138	0.40
	1						0.46	0.49	0.47	4997	
	2						0.38	0.42	0.40	4692	

TABLE 4.8: Yearly classification performance of the Expanding Window model on both validation and test sets across the three classes (0, 1, 2). Reported metrics include precision, recall, and F1-score (all ranging from 0 = worst to 1 = best), along with class *support* (sample counts) and overall accuracy. Validation results are shown on the left, test results on the right. Certain years (e.g., 2015 test, 2023 validation) lack data due to the expanding window setup, shown as *No data*.

Similar patterns are observed in the validation loss (**Figure 4.4**) and Class 2 F1-score plots (**Figure 4.5**). Both metrics show notably poorer performance in 2017, 2018, and 2021, with elevated loss and particularly low F1 scores.

The model performs consistently across most years up to 2020, with 2020 showing the strongest results. Performance declines sharply in 2021, likely due to increased volatility and irregularities during that period. In 2022, the model appears to recover, potentially by treating 2021 as an anomaly and returning to more stable predictive behaviour.

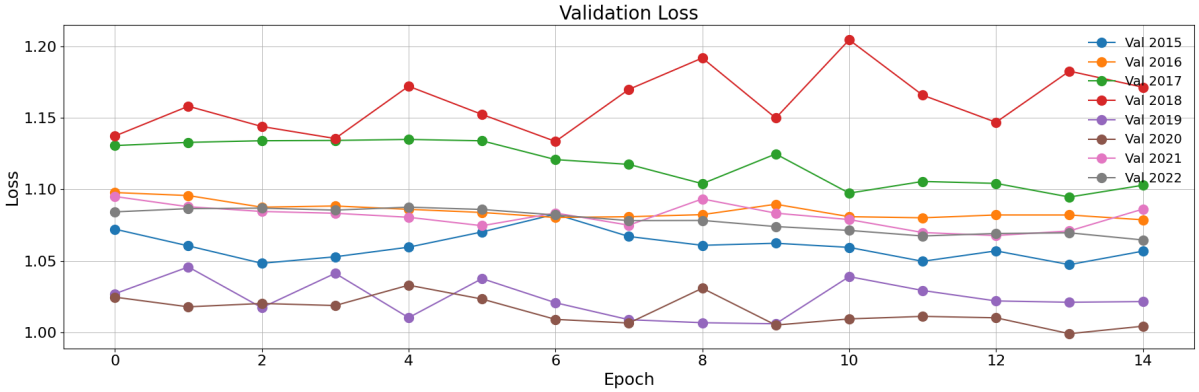


FIGURE 4.4: Validation loss trajectories across different training periods in the expanding window analysis. Loss is measured as categorical cross-entropy (minimum 0, with random guessing across three classes 1.10). Each line represents the model trained up to a specific year. Stable loss values are observed for most years, though 2021 shows noticeable degradation, consistent with market irregularities during that period. Notably, the 2018 model performs significantly worse than random, ending with a loss of about 1.17, whereas the 2017 model remains only slightly worse than random at approximately 1.11.

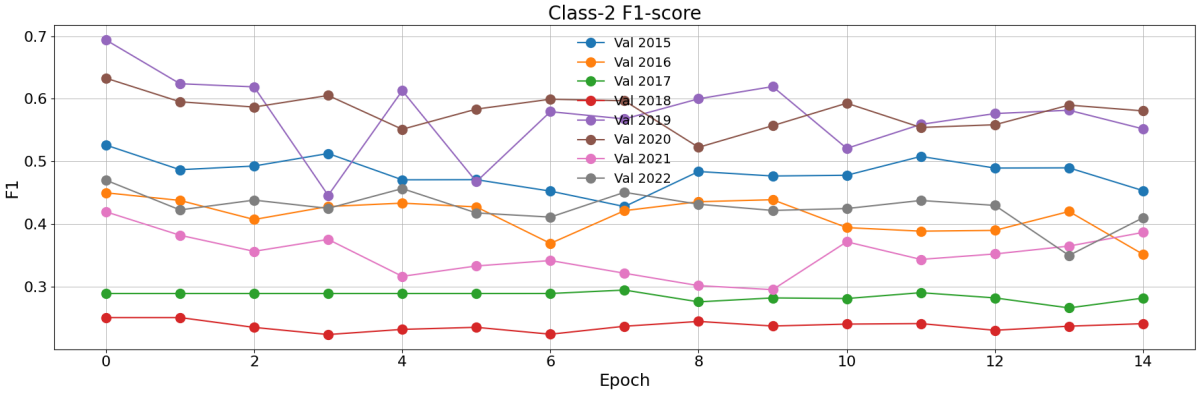


FIGURE 4.5: F1-score for Class 2 (high-return bucket) during the expanding window analysis. The F1-score combines precision and recall into a harmonic mean (0 = worst, 1 = best), reflecting overall classification quality. Each curve corresponds to training up to a given year. Performance peaks in 2019–2020, deteriorates in 2021, and partially recovers in 2022.

Overall, the expanding window analysis suggests that the Transformer model exhibits mild sensitivity to structural shifts in the data, particularly during periods like 2021. The decline in

performance between 2017 and 2018 may reflect both reduced data availability and a lack of exposure to similar market conditions. In contrast, the smaller performance drop in 2021, despite its volatility, may suggests that having access to a broader historical context enables the model to generalise more effectively.

4.3 Interpretability Analysis of the Transformer Model

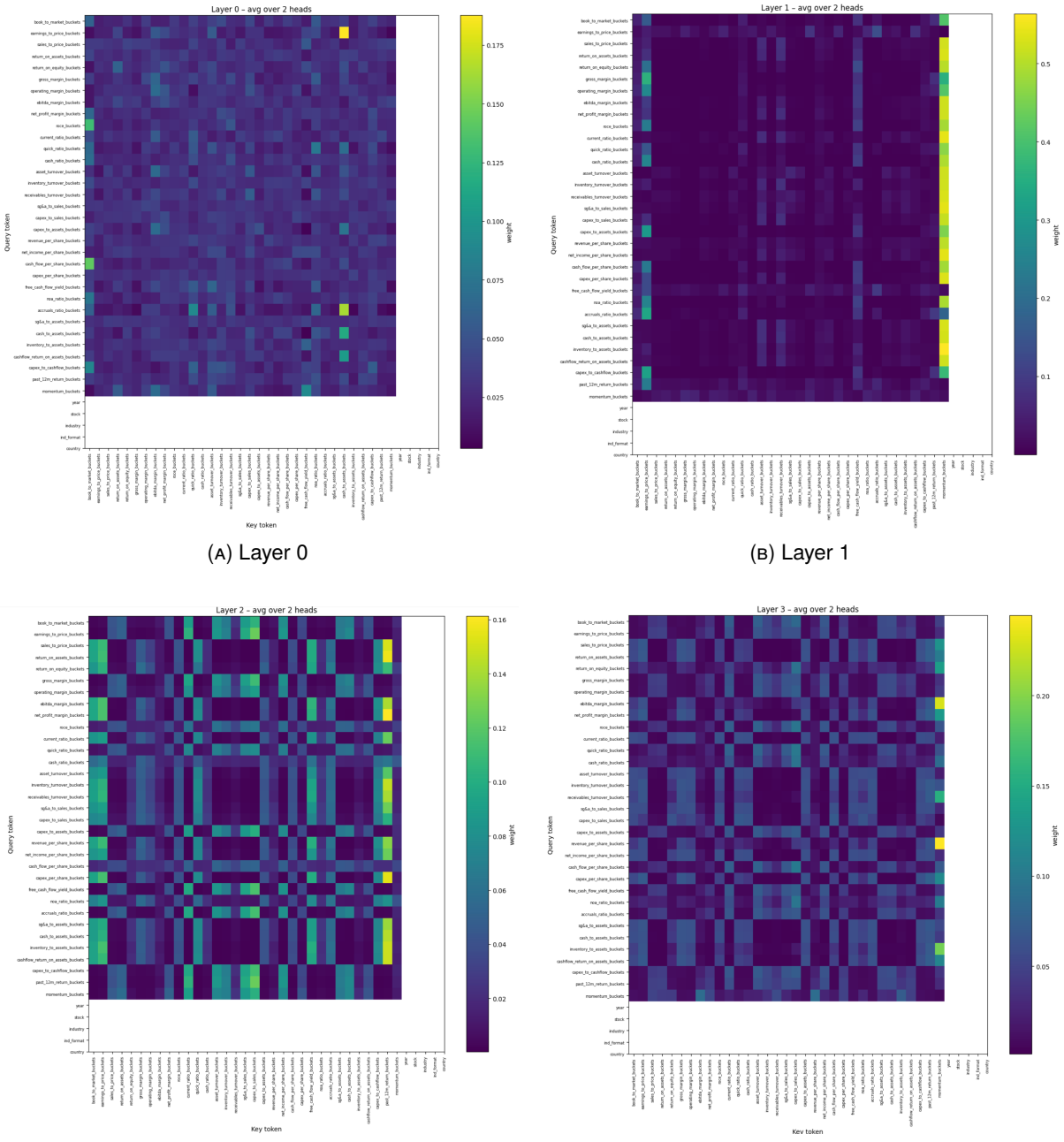
In this section, we assess the interpretability of a single Transformer model instance, using a fixed set of hyperparameters from **Table 4.5**. As previously noted, results vary across different random seeds; the analysis presented here reflects one specific instance. Nevertheless, the interpretability techniques and the manner in which the results are discussed are applicable across all seeds.

4.3.1 Attention Matrix Results

The attention mechanism is composed of two core components: keys and queries, which together determine how much focus the model allocates to **different input features**. By examining the resulting attention matrices, we can identify which features the model considers most relevant when making predictions (see **Appendix B** for more detail).

To interpret these matrices accurately, it is important to understand their structure. Each attention matrix captures the directional relationships between features, where higher weights indicate stronger relevance. These relationships are **not** symmetric, if feature A attends to feature B, the reverse does not necessarily hold. Conceptually, attention is computed as the "query attending to the key", which governs how information flows across features and provides the correct orientation for analysis.

In our setup, as specified by the hyperparameters in **Table 4.5**, the model comprises of **four** layers, each containing **two** attention heads. Within each layer, the attention matrix is calculated independently in each head, with each head offering a distinct perspective on the input features and capturing different relationships within the data. The attention weight matrices from the two heads are averaged, and these averaged matrices (multi-head matrices) are plotted to form the attention matrix plots. For further details on attention heads, layers, and the Transformer architecture, refer to **Appendix C Optuna Results** and **Chapter 3**.



(A) Layer 0

(B) Layer 1

(C) Layer 2

(D) Layer 3

FIGURE 4.6: Layer-wise self-attention heatmaps. For each layer l and a chosen validation sample $x^\star$, we plot the per-sample, head-averaged matrix $\widetilde{A}^{(l)}(x^\star) \in [0,1]^{N\times N}$ defined by $\widetilde{A}^{(l)}_{q,k}(x^\star) = \frac{1}{n_{heads}} \sum_{h=1}^{n_{heads}} A^{(l,h)}_{q,k}(x^\star)$, where $A^{(l,h)}_{q,k}(x^\star)$ is the softmax-normalised attention from query q to key k in layer l and head h (rows sum to 1 after masking) and n_{heads} is the number of heads ($n_{heads} = 2$). Rows are queries and columns are keys; colour encodes $\widetilde{A}^{(l)}_{q,k}(x^\star)$ (brighter = larger weight). Bright columns indicate keys broadly attended by many queries (globally influential tokens), while bright rows indicate low-entropy queries concentrating mass on few keys. **(a) Layer 0:** diffuse, high-entropy attention with little structure. **(b) Layer 1:** emergence of narrow vertical bands to the right of the heatmap, few keys attract attention from many queries. **(c) Layer 2:** emergence of symmetric/block structure, suggesting feature–feature interactions and more selective routing. **(d) Layer 3:** similar structure with slightly reduced intensity, suggesting a broader spread of focus.

Layer 0: For Layer 0 (**Figure 4.6a**), the attention is quite noisy, best described as roughly uniform across the matrix with a few standout activations. This is a common characteristic of an early Transformer layers, which often act as high entropy mixers that distribute attention broadly enabling later layers to identify more structured patterns **(Rogers et al. 2020)**.

We do however see subtle highlights on certain key tokens such as `book_to_market_bucket`, which is slightly more activated than others. This may imply that many different ratios already treat this feature as a weak anchor though the low levels of attention is too faint to draw any strong conclusions.

Layer 1: Notably, the most distinctive patterns begin to emerge in Layer 1 (**Figure 4.6b**). The earlier noise dissipates, and this layer becomes far more selective. In particular, a strikingly bright strip appears on the right edge of the matrix, corresponding to extremely high attention (up to 0.5), which is quite rare from previous testing. The dominant key here is `momentum`, with the majority of queries focusing on this key, except for a few notable exceptions: `earnings_to_price_buckets`, `price_to_yield`, `free_cashflow_yield_buckets`, and `past_12m_returns_buckets`, as well as the `momentum_buckets` token itself.

This is notable because momentum and these valuation based features usually capture different aspects of the data and do not tend to be positively correlated **(Asness 1997)**. Therefore, the few features **not** focused on momentum may be capturing alternative signals that provide useful additional information to the model.

Layer 2: Layer 2 (**Figure 4.6c**) differs from Layer 0 and Layer 1 in that a symmetric attention pattern begins to emerge, a uncommon characteristic. These patterns likely reflect pairwise interactions between specific query key tokens, suggesting that the model is capturing more structured dependencies and is generalising more effectively.

Notably, tokens such as `past_12m_return_buckets`, `earnings_to_price_buckets` and `book_to_market_buckets`, receive consistently high attention supporting the view that the model is beginning to combine related features, rather than evaluating them in isolation.

Layer 3: Layer 3 (**Figure 4.6d**) shows similar structural patterns to Layer 2, but significantly weaker in magnitude. A notable difference is the renewed emphasis on the `momentum_bucket`, which is now more heavily attended compared to other features.

This may suggest that by this stage, the model is refining its learned representations, with the renewed emphasis on momentum indicating its continued relevance as a key predictive feature.

Overall, by examining each attention matrix across the Transformer layers, we gain insight into the model's internal decision making, highlighting which features it attends to and how

these patterns evolve. This alone already offers a meaningful level of interpretability, helping us understand the model's behaviour beyond raw performance metrics.

4.3.2 Attention Matrix Scoring

Given that the Transformer allocates varying levels of attention to input features across layers and heads, aggregating these weights offers a principled way to assess feature importance (For more information on Attention Scoring refer to **Appendix C**).

As shown in **Table 4.9**, `momentum_buckets` stands out as the most dominant feature, with a total attention score (α_i) of 1.13. This is consistent with the strong emphasis on momentum seen in Layer 1 and Layer 3 of the attention matrices.

Rank	**Token**	**ID**	**Score** (α_i)
1	`momentum_buckets`	32	1.1315
2	`earnings_to_price_buckets`	1	0.5233
3	`free_cash_flow_yield_buckets`	23	0.3779
4	`past_12m_return_buckets`	31	0.3585
5	`book_to_market_buckets`	0	0.3407

TABLE 4.9: Top five tokens ranked by their inbound attention score α_i. Here $\alpha_i = \frac{1}{n_{\text{layers}}\, n_{\text{heads}}} \sum_{\ell=1}^{n_{\text{layers}}} \sum_{h=1}^{n_{\text{heads}}} \sum_{q=1}^{L} A_{q,i}^{(\ell,h)}$, the mean (over layers n_{layers} and heads n_{heads}, with L the sequence length) of the column-sum of the row-normalised attention matrices, i.e., the average probability mass that all queries allocate to key i. Larger α_i indicates a "hub" token. *ID* is the token's index in the model input. Because α_i sums over all queries, values can exceed 1 and should be compared relatively across tokens.

The other top ranked tokens also correspond closely with visual attention patterns, suggesting that the scoring method reliably captures the model's feature preferences.

Overall, this reinforces the insights from the attention matrices, while also providing an explicit ranking of the most attended features.

4.3.3 Individual Layer and Head Analysis

We now examine the Transformer model at a deeper level by analysing individual layers and attention heads. This allows us to move beyond aggregated attention matrices and gain more granular insight into how specific components capture different aspects of the input and contribute to the model's behaviour.

Entropy-Based Importance

Entropy ($\mathcal{H}_{b,h,q}$) measures the amount of randomness or disorder in the attention weights of each Transformer layer. A high entropy implies that the attention is more spread out (i.e., the model is less focused), while a low entropy suggests that the model is attending more "selectively" which is generally preferred **(Zhang et al. 2025)**. While that study also proposes techniques for actively reducing entropy in specific layers, such interventions are beyond the scope of this work (For more information on entropy, refer to **Appendix C**).

The entropy values for the attention layers are as follows: Layer 1 (1.6820), Layer 2 (2.6501), Layer 3 (2.7685), and Layer 0 (3.3953). Since lower entropy indicates more focused attention (i.e., concentration on a smaller subset of tokens), the ranking from most to least focused is Layer 1, Layer 2, Layer 3, and finally Layer 0. Notably, Layer 1 exhibits the lowest entropy, suggesting that it allocates attention in a particularly selective manner.

However, it's important to note that lower entropy does not necessarily mean better performance. A layer might focus strongly on features that are not helpful or even misleading. Therefore, entropy should be understood as a measure of how sharply focused the attention is, not as a direct indicator of feature quality or importance.

Ablation (Head)

In this ablation approach, individual attention heads are selectively muted, and the resulting change in validation loss is measured **(Michel et al. 2019)**. A larger increase in loss indicates a more important head. This method provides a fine grained view of which heads are most critical, enhancing our understanding of how attention is distributed.

As shown in **Table 4.10**, the most obvious and consistent impact comes from Layer 1, which is unsurprising given the strong momentum focus we previously observed. Dropping any head in Layer 1 leads to the highest increases in validation loss. Notably, muting Head 1 in this layer results in a loss increase of 0.0087, by far the largest among all heads. This may suggest that the key momentum signal may primarily flow through Head 1, while Head 0 contributes less of a signal.

Layer	Head	Δ_{loss}
0	0	-0.0021
0	1	-0.0027
1	0	$+0.0022$
1	1	$+0.0087$
2	0	0.0000
2	1	$+0.0009$
3	0	$+0.0014$
3	1	$+0.0046$

TABLE 4.10: Change in validation loss (Δ_{loss}) from muting individual attention heads. Positive values indicate a degradation in performance, hence a more important head. While most heads cause only minor changes (on the order of 10^{-3}), Layer 1 heads show the strongest impact, with Head 1 in particular causing the largest increase ($+0.0087$).

In Layer 3, a similar trend emerges. Muting Head 1 again leads to a relatively high loss increase. Looking back at the attention matrix in **Figure 4.6c**, we see momentum again appearing as a prominent attended feature. This reinforces the idea that Head 1 of Layer 3 also contributes significantly to momentum related attention; however, this remains speculative, and further analysis is required to draw a definitive conclusion.

Layer 0, on the other hand, behaves quite differently. Muting either head in this layer actually improves model performance, reducing validation loss. This suggests that both heads in Layer 0 may be contributing more noise than useful signal. As seen in **Figure 4.6a**, the attention map is quite chaotic and lacks strong structure as previously mentioned. While some attention is directed toward tokens like `book_to_market_bucket` and `cash_to_asset_buckets`, removing this layer does not appear to harm the final results.

Weight Norms

This analysis ranks attention heads based on the L2 norm ($\nu_{i,j}$) of their query, key, and value projection matrices. The assumption is that heads with larger norms are more actively contributing to the attention mechanism, as their learned parameters are more expressive or engaged during training **(Kobayashi et al. 2020)** (For more information about Weight Norms refer to **Appendix C**).

Note: A high weight norm does not necessarily indicate functional importance. To assess true relevance, weight norms should be interpreted in conjunction with other diagnostics, such as head ablation.

Layer	Head	Norm Score ($\nu_{i,j}$)
0	0	9.22
0	1	9.22
1	0	8.61
1	1	8.37
2	0	6.37
3	1	6.18
3	0	5.92
2	1	4.60

TABLE 4.11: Head weight-norm scores $\nu_{i,j}$, where i indexes the Transformer layer and j the attention head. For each head, $\nu_{i,j} = \|\mathbf{W}_q\|_2 + \|\mathbf{W}_k\|_2 + \|\mathbf{W}_v\|_2$, with $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ the learned query, key, and value projection matrices. Larger $\nu_{i,j}$ indicates a head with greater combined projection magnitude; see **Appendix C** for details. Here, Layer 0 heads attain the largest values (≈ 9.2), while Layer 2, Head 1 is smallest (≈ 4.6).

As shown in **Table 4.11**, Layer 0 exhibits the highest weight norms, followed by Layer 1, while Layers 2–3 display lower and more variable values.

The high weight norms indicate that Layer 0 carries a large volume of signals. When considered alongside the head ablation results (**Section 4.3.3**), this may suggest that the layer's activity is dominated by noise rather than useful information, potentially explaining why its heads appear highly active.

Individual Layer and Head Analysis Conclusion

Layer 1 clearly stands out as the key contributor. It has the lowest attention entropy ($\mathcal{H}_{b,h,q}$ = 1.6820) and the highest sensitivity in the ablation tests (Δ_{loss} = 0.0022 and 0.0087 for its two heads), showing that it routes information in a highly selective and performance critical way, exactly what we'd expect given its strong focus on momentum.

In contrast, Layer 0 shows the highest entropy ($\mathcal{H}_{b,h,q}$ = 3.3953) and actually improves performance when either head is removed (Δ_{loss} = –0.0021, –0.0027), possibly suggesting that its heads are either redundant or slightly harmful. What's interesting is that Layer 0 also has the largest weight norms ($\nu_{0,0}$ = 9.22, $\nu_{0,1}$ = 9.22), indicating that a substantial amount of information passes through it, despite its limited or even negative contribution to performance.

These results suggest that pruning or gating Layer 0 might help, but it's important to remember that attention head importance isn't simply additive. Furthermore, one could also argue that this specific attention layer (Layer 0) is capturing noise, thereby allowing subsequent layers to

be more focused and selective. From this perspective, the presence of such a noisy layer may be beneficial for overall model performance, and its removal could actually hinder predictive accuracy. Under this interpretation, the observed increase in loss following pruning and high weight norms may not indicate a lack of usefulness, but rather suggest that this layer plays a supportive role in feature disentanglement.

In terms of interpretability, examining each layer and head individually allowed us to identify which features the model attends to, how attention patterns evolve, and which components contribute most to the model's decision-making. This layered analysis helps uncover the inner workings of the Transformer, providing much more clearer insight of the model.

4.3.4 Attention Roll-Out

A key limitation of inspecting attention matrices layer by layer is that it only captures local interactions within individual layers, missing the cumulative effect of attention in-between the layers. Attention roll-out addresses this by **tracing** how information propagates through the entire stack of layers, offering more of a global perspective on token influence **(Abnar and Zuidema 2020)**.

Attention roll-out relies on several key assumptions. First, it assumes that all attention matrices are equally weighted and that each attention head contributes equally to the model's output. Second, roll-out does not yield a definitive attribution score; rather, it approximates the flow of information through the network by recursively multiplying attention matrices across layers. Importantly, the final prediction is also shaped by downstream components such as the classifier head, LayerNorm, and non-linear activations. As such, while attention roll-out provides useful high-level insights, its interpretability as a measure of "cumulative effect of attention across the network" should be approached with caution (For more information about Attention Rollout refer to **Appendix C**).

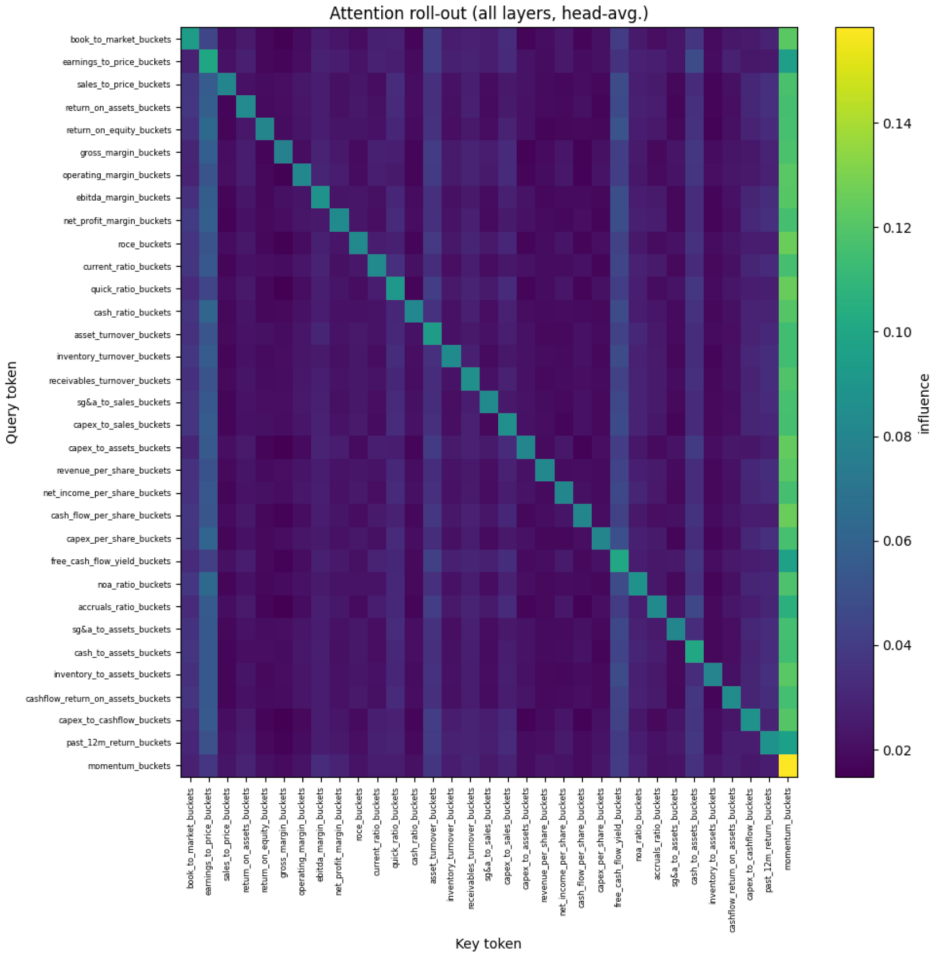


FIGURE 4.7: Attention roll-out matrix R (per-sample, head-averaged, residual-preserving). For a chosen validation sample $x^\star$ and each encoder layer ℓ, we form the head-averaged attention $A^{(\ell)}(x^\star) = \frac{1}{n_{\text{heads}}} \sum_{h=1}^{n_{\text{heads}}} A^{(\ell,h)}(x^\star)$. Residual paths are preserved by adding the identity and row-normalising, $\tilde{A}^{(\ell)} = \text{row-norm}\big(A^{(\ell)} + I\big)$. The roll-out is the ordered product $R = \tilde{A}^{(n_{\text{layers}})} \tilde{A}^{(n_{\text{layers}}-1)} \cdots \tilde{A}^{(1)} \in [0,1]^{L \times L}$, which is row-stochastic (each row sums to 1 after masking). Rows are query tokens i and columns are key tokens j; colour encodes R_{ij}, the cumulative influence of key j on query i aggregated across all layers. Bright vertical bands indicate globally influential keys (high column-wise influence), while strong diagonal intensity reflects self-retention across layers. In our model ($n_{\text{heads}} = 2$), the rightmost band corresponds to `momentum_buckets`, indicating it acts as a global hub.

In terms of results, attention rollout once again emphasizes momentum, it remains dominant even after being passed through all layers. The result is quite similar to what we saw in the scoring section (**Subsection 4.3.2**): `earnings_to_price_buckets`, `momentum_buckets` and `free_cash_flow_yield_buckets` are all prominent. Interestingly, `cash_to_asset_buckets` also appears, despite not being among the top-ranked features previously. This suggests that, although it may not attract substantial direct attention within any single layer, it contributes

meaningfully to the flow of information across the Transformer layers.

This makes sense if we consider the `cash_to_asset_buckets` as an indirect amplification. Some features (like momentum or valuation ratios) might consistently "route" through or "borrow" from `cash_to_asset_buckets`, effectively boosting its overall influence almost acting like a sort of bridge or hub of some kind even if its individual per-layer attention mass is not extreme.

Finally, the bright diagonal in the roll-out maps is expected, as each token retains some of its own information through row-normalised attention. This self-focus accumulates across layers, producing the diagonal pattern.

4.3.5 Ablation (Features)

Given the large number of input features used in this study, it is important to understand the relative contribution of each feature to the model's performance. One classic approach for this is LOFO (Leave One Feature Out) analysis, which involves systematically removing individual features from the input set and re-evaluating the Transformer model. The underlying assumption is that if a particular feature provides meaningful predictive information, its removal degrade degrade model performance. Conversely, if performance remains unchanged or improves, the feature may be redundant. This process allows us to rank features by importance and better understand the sources of predictive power within the model.

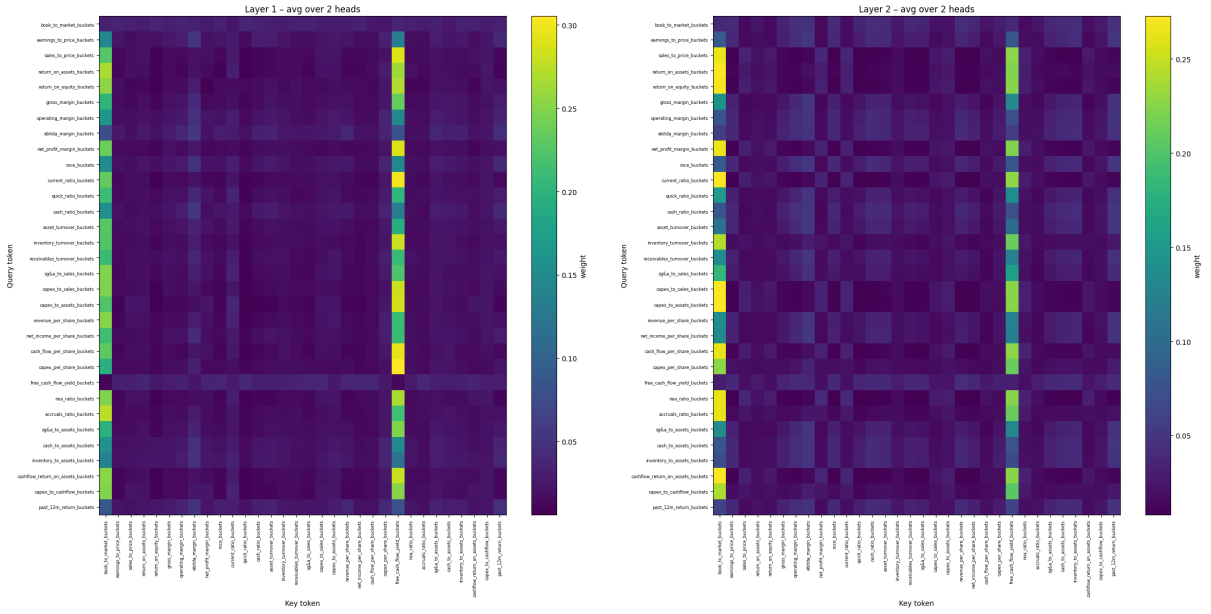
To make the analysis more realistic, particularly given the high dimensionality of the input space, we restrict ablation to the top five features identified as most influential based on prior interpretability techniques discussed earlier in this section. This ensures the analysis remains more focused and avoids unnecessary computational overhead.

A more comprehensive set of results is available in the accompanying notebook and in **Appendix C**. Here, we focus on the most notable and representative patterns. As shown in **Table 4.12**, removing individual features generally has minimal impact on validation accuracy and F1 score, with some cases even showing slight improvements, except when all five key ratios are removed, which leads to a noticeable drop.

Dropped	**Best Acc.**	**Final Acc.**	**Macro $\mathbf{P}$ (Precision)**	**Macro $\mathbf{R}$ (Recall)**	**Macro F_1**	**Weighted F_1**
None (baseline)	0.4432	0.4325	0.42	0.42	0.42	0.43
momentum_buckets	0.4382	0.4361	0.43	0.41	0.40	0.42
earnings_to_price_buckets	0.4421	0.4413	0.42	0.42	0.42	0.44
free_cash_flow_yield_buckets	0.4360	0.4275	0.42	0.42	0.42	0.43
past_12m_return_buckets	0.4410	0.4401	0.43	0.43	0.43	0.44
book_to_market_buckets	0.4396	0.4314	0.42	0.41	0.41	0.43
All Five	0.3893	0.3828	0.38	0.38	0.36	0.37

TABLE 4.12: Validation metrics after dropping individual features (LOFO analysis). "Macro $\mathbf{P}$" and "Macro $\mathbf{R}$" denote macro-averaged precision and recall, while "Macro F_1" and "Weighted F_1" correspond to macro- and weighted-averaged F1-scores. Dropping all five features together results in a substantial drop in performance (Final Accuracy 0.38 vs $\sim$0.43).

When `momentum_buckets` is removed and the model is retrained using the same hyperparameters, attention naturally shifts to alternative signals. As shown in **Figure 4.8**, there is increased focus on `book_to_market_buckets` and `free_cash_flow_yield_buckets`. This suggests that momentum may not have been strictly necessary and might have even introduced some noise. Alternatively, it demonstrates the model's robustness, as it successfully adapts by reallocating attention to other informative features. In either case, this highlights the flexibility of the Transformer when a dominant input is removed.



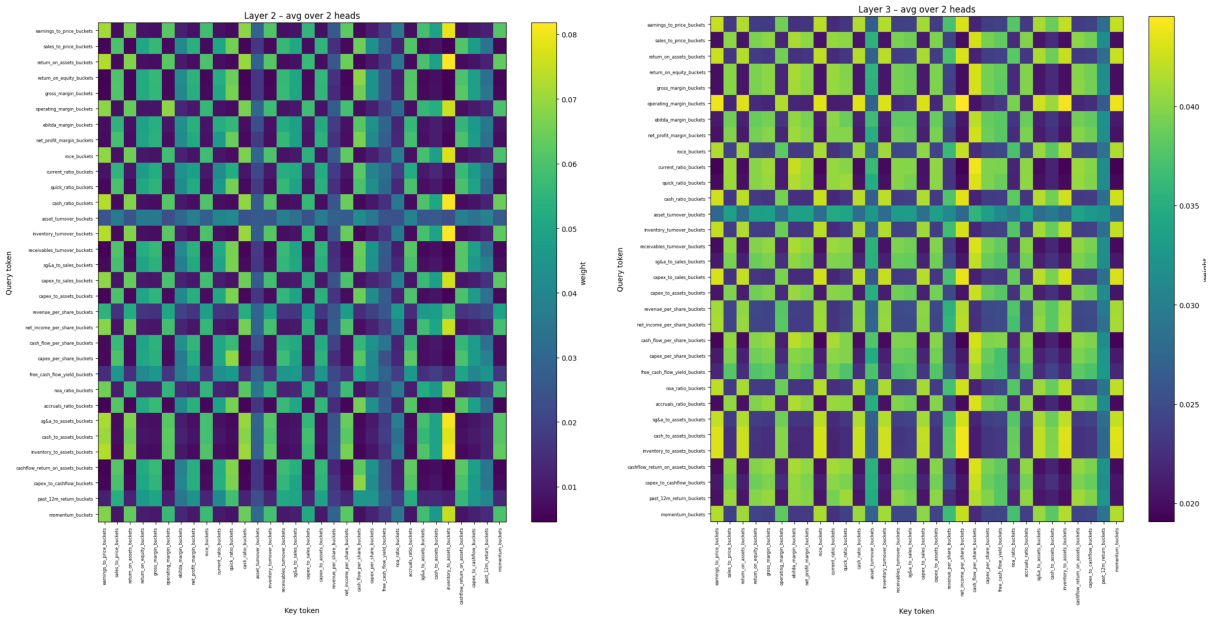
(A) Layer 1 after dropping momentum

(B) Layer 2 after dropping momentum

FIGURE 4.8: Self-attention heatmaps after ablating `momentum_buckets`. Each panel shows the *mean attention* for a Transformer layer, averaged over the validation set and over the two heads in that layer: $\widetilde{A}^{(l)}_{q,k}(x^\star) = \frac{1}{n_{heads}} \sum_{h=1}^{n_{heads}} A^{(l,h)}_{q,k}(x^\star)$. Rows are *query* tokens and columns are *key* tokens; colour encodes the average attention weight (darker → lower, brighter → higher; see the layer-specific colour bars at right). Because attention is row-normalised before averaging, bright *vertical* bands indicate tokens that are frequently attended to by many queries (globally influential keys), whereas bright *horizontal* bands indicate queries that concentrate their mass strongly. Layers 1 and 2 now focus on `free_cash_flow_yield_buckets` and `book_to_market_buckets`. The ablated feature is absent from both axes, so its attention mass is redistributed across the remaining tokens, most notably these two, which therefore emerge as relatively more influential.

Dropping other buckets shows similar behaviour. The model tends to redistribute attention to related signals often falling back on features like `book_to_market_buckets`, `earnings_to_price_buckets`, or occasionally new ones like `return_on_assets_buckets` (see **Appendix C**).

As noted earlier, features like `book_to_market_buckets` and `earnings_to_price_buckets` often act as fallback signals when more dominant inputs are removed. But what happens when one of these fallback options is also excluded? In **Figure 4.9**, we observe that removing `book_to_market_buckets` leads to a more subdued and evenly distributed attention pattern. The model's focus becomes more diffuse, spreading across many features, and the remaining layers (see **Appendix C**) appear nearly uniform. The slight symmetry in the heatmap suggests that some feature clusters weakly co-attend, but no strong structure dominates.



(A) Layer 2 after dropping `book_to_market_buckets`

(B) Layer 3 after dropping `book_to_market_buckets`

FIGURE 4.9: Self-attention heatmaps after ablating `book_to_market_buckets`. Each panel shows the head-averaged attention for layer l on a representative validation example $x^\star$: $\widetilde{A}^{(l)}_{q,k}(x^\star) = \frac{1}{n_{heads}} \sum_{h=1}^{n_{heads}} A^{(l,h)}_{q,k}(x^\star)$. Rows are *query* tokens and columns are *key* tokens; colour encodes $\widetilde{A}^{(l)}_{q,k}(x^\star)$ (darker → lower, brighter → higher). Because attention rows are normalised, bright *vertical* bands indicate keys that are widely attended by many queries, whereas bright *horizontal* bands indicate queries that concentrate their mass strongly. The ablated token is absent from both axes, so its probability mass is redistributed among the remaining tokens. Relative to Layer 2 **(a)**, Layer 3 **(b)** exhibits a clearer block/checkerboard structure with slightly lower peaks, consistent with more selective, yet broader, interactions at greater depth.

Finally, when all five key features are removed (**Figure 4.10**), model performance clearly deteriorates. Although the model attempts to adapt by reallocating attention, validation accuracy drops to 0.38 and the F1 score to 0.37. This demonstrates that while the Transformer is resilient to the loss of individual features, simultaneously removing all key inputs significantly harms performance, underscoring their collective importance.

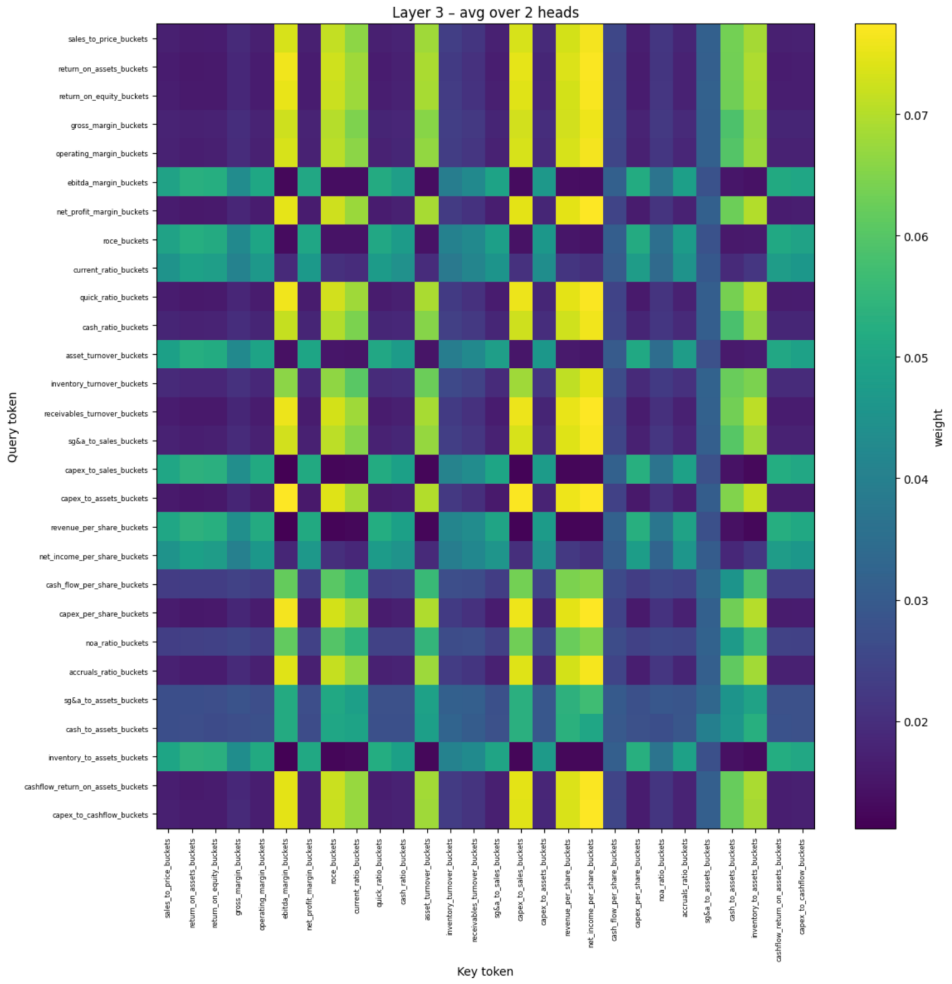


FIGURE 4.10: Self-attention heatmap for Layer 3 after ablating the five highest-impact bucketed features (`momentum_buckets`, `earnings_to_price_buckets`, `free_cash_flow_yield_buckets`, `past_12m_return_buckets`, `book_to_market_buckets`). The image shows head-averaged attention on a representative validation example $x^\star$: $\widetilde{A}^{(3)}_{q,k}(x^\star) = \frac{1}{n_{heads}} \sum_{h=1}^{n_{heads}} A^{(3,h)}_{q,k}(x^\star)$ (with row-wise softmax normalisation; H heads). Rows are *query* tokens and columns are *key* tokens; colour encodes $\widetilde{A}^{(3)}_{q,k}(x^\star)$ (darker → lower, brighter → higher). Because the ablated tokens are absent from both axes, their probability mass is redistributed across the remaining tokens. Relative to the non-ablated model, the pronounced vertical bands and checkerboard structure weaken and the map becomes more diffuse with lower peaks, indicating that these five features previously acted as globally influential keys and helped organise layer-level interactions.

4.3.6 SHAP Analysis of the Transformer

We have so far assessed which features the model focuses on using attention-based methods. We now turn to SHAP analysis to provide an alternative perspective that does not rely on attention matrices.

SHAP (SHapley Additive exPlanations) is a model interpretation technique grounded in *Shapley values* from cooperative game theory. It assigns an "importance" value to each feature, quantifying the extent to which that feature contributed to an individual prediction.

Unlike attention-based methods, which focus on the model's internal decision making process, SHAP takes a feature-centric approach. It attributes importance by quantifying how much each feature contributes to a specific prediction both individually and through interactions with other features **without using the attention matrix**. This allows SHAP to provide interpretability from a different perspective, offering insights into feature influence rather than model attention patterns (for more information about SHAP see **Appendix B**).

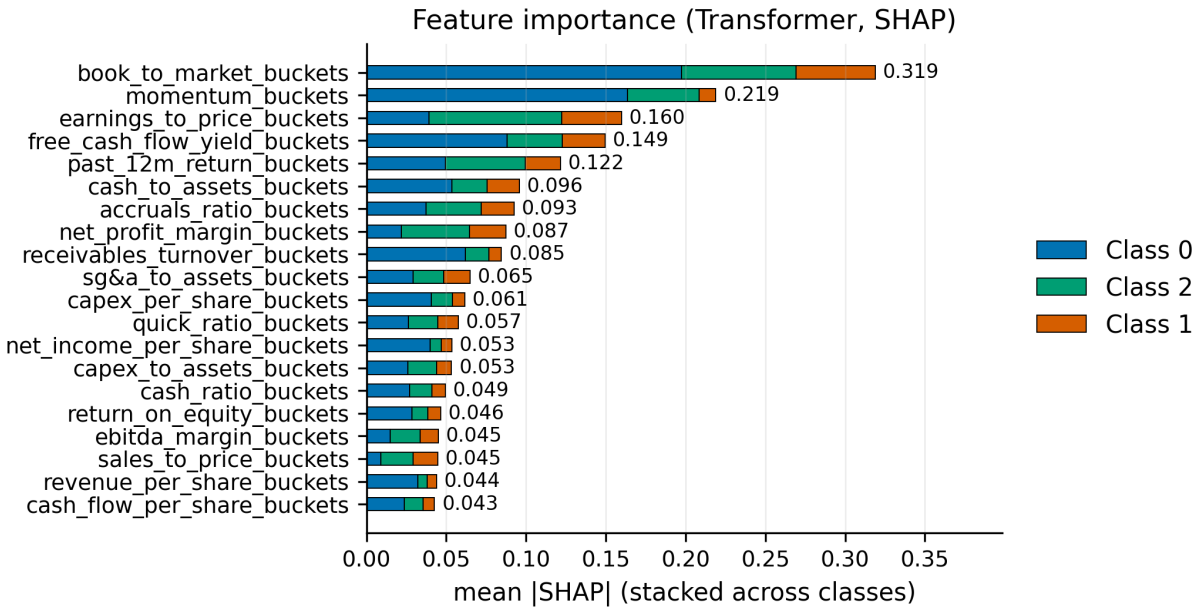


FIGURE 4.11: SHAP summary plot for the Transformer model on the test set. The x-axis shows SHAP values, quantifying each feature's marginal contribution to the predicted class (0 = no effect, positive values increase probability, negative values decrease it). Features are ordered on the y-axis by mean absolute SHAP value, so those at the top are most influential overall. Each point represents a single sample, with colour indicating the feature's actual value. This plot provides a global view of feature importance beyond attention-based measures.

Our earlier attention analysis highlighted a strong emphasis on `momentum_buckets`, whereas the SHAP analysis identifies `book_to_market_buckets` as the most influential feature **Figure 4.11**. This difference stems from the fact that SHAP values capture features that consistently cause large changes in the model's output, regardless of direction.

This distinction is meaningful. Rather than the high attention indicating that the `momentum` bucket is a core predictive feature, it may instead suggest that momentum serves as an internal reference or coordination signal across layers, while the SHAP scores highlight which features directly

influence the model's predictions. Features like `book_to_market_buckets` and `earnings_to_price_buckets` appear to influence decision boundaries more directly and decisively than `momentum`, which likely explains their higher SHAP scores. This does not reflect negatively on the Transformer; rather, it complements the attention based view by offering a more comprehensive understanding of the model's behaviour.

Finally, we briefly compare the analysis in **Subsection 4.1.1**. Both SHAP evaluations identify *book_to_market_bucket* and *momentum_buckets* as the most influential features, with the remaining top five also appearing but in a different order. This divergence is expected: although SHAP is feature-centric, the underlying model mechanics determine how contributions are attributed. Since Transformers and XGBoost leverage features differently, their SHAP explanations need not align exactly. The broad agreement between the two models, however, is reassuring; given XGBoost's robustness, it suggests that the Transformer has captured similarly meaningful structure in the data.

Chapter 5

Conclusion

5.1 Summary of Results

The Transformer achieves performance comparable to XGBoost, indicating that it learns meaningful structure in the data. This outcome is strongly influenced by hyperparameters: moderate learning rates, small–medium batch sizes, shallow depth, and Adam/AdamW with step or cosine scheduling produced the most stable results, consistent with prior evidence on tabular fine-tuning **(Gorishniy et al. 2023)**.

Across different time periods, the Transformer demonstrates temporary robustness and stability, with only a few outlier time periods (2017-2018 and 2021). This indicates that its performance remains relatively consistent over time, further supporting the model's reliability.

Across interpretability techniques (attention matrices, attention scoring, ablations, and rollout) a consistent set of highly attended features emerges:

- `momentum_buckets`
- `earnings_to_price_buckets`
- `free_cash_flow_yield_buckets`
- `past_12m_return_buckets`
- `book_to_market_buckets`

These signals are economically plausible and align with established literature **(Fama and French 1993; Jegadeesh and Titman 2011; Basu 1977; Davis 1994)**.

At the layer level, relative importance follows Layer 1 > Layer 3 > Layer 2 > Layer 0, with Layer 3 prioritised due to larger ablation losses. This accords with prior findings that mid–late layers in encoder-only models encode task-specific information **(Rogers et al. 2020)**. The most impactful heads appear in Layer 1, while Layer 0 contributes least, suggesting potential noise

or inefficiency. This aligns with earlier work **(Michel et al. 2019)**, though early layers may still provide auxiliary roles.

Feature ablations show robustness to single omissions: accuracy and F_1 remain stable (**Table 4.12**) as attention shifts to correlated proxies (e.g., `book_to_market_buckets` and `free_cash_flow_-yield_buckets` when momentum is removed; **Figure 4.8**). Dropping fallback features like `book_to_market_buckets` yields more diffuse attention (**Figure 4.9**), indicating redundancy across valuation/momentum signals. By contrast, removing all five ratios sharply reduces performance (Acc = 0.38, $F_1 = 0.37$), confirming they are individually substitutable but essential in aggregate.

Finally, SHAP values reinforce the significance of these features, though the precise ranking differs across methods.

5.1.1 Implications

This study carries several implications.

For **academia**, the results show that an encoder-only Transformer can perform competitively on financial tabular data while offering greater interpretability, advancing efforts toward more transparent deep learning in finance.

For **practitioners**, the model's reliance on economically plausible signals (e.g., momentum, earnings-to-price) reinforces its credibility and aligns with established literature, reducing concerns of "black-box" decision-making.

For **other stakeholders**, the findings show that advanced models can match trusted benchmarks such as XGBoost while providing transparent links between outputs and well-understood financial factors. This builds trust, supports compliance and audit requirements, and facilitates easier adoption across various industries.

5.2 Final Conclusion

Our results show that the Transformer performs **competitively in out-of-sample classification**, matching traditional models such as XGBoost, while also demonstrating **temporal robustness** with consistently strong results across different time periods.

Attention-based diagnostics, including attention matrices, rollout, scoring, entropy, and ablation, demonstrate that the model consistently emphasises **economically meaningful features**, particularly momentum and valuation ratios. These patterns are stable across layers and align with domain knowledge, suggesting genuine learning rather than noise.

Complementary methods provide additional perspective: SHAP often highlights different variables, offering a feature-level view, while attention reveals the **core signals** directly driving predictions. This divergence shows that relying on SHAP alone may be **misleading**, underscoring the value of combining interpretability tools.

To summarise, the Transformer achieves a balance between predictive performance and interpretability, thereby addressing the **primary research question** and supporting its potential as a transparent machine learning tool in finance.

Possible Extensions:

- **Deeper Analysis:** Extend SHAP analysis and compare with attention flow to clarify feature interactions.
- **Model Comparison:** Benchmark against RNNs and LSTMs.
- **Ensembles:** Combine Transformers with other models to enhance performance and explainability.

Bibliography

Abnar, S. and W. Zuidema (2020). *Quantifying Attention Flow in Transformers*. arXiv: 2005.00928 [cs.LG]. URL: https://arxiv.org/abs/2005.00928.

airalcorn2 (n.d.). *A simple script for extracting the attention weights from a PyTorch Transformer. · GitHub*. [Online; accessed 2025-07-28]. URL: https://gist.github.com/airalcorn2/50ec06517ce96ecc143503e21fa6cb91.

Araci, D. (2019). *FinBERT: Financial Sentiment Analysis with Pre-trained Language Models*. arXiv: 1908.10063 [cs.CL]. URL: https://arxiv.org/abs/1908.10063.

Arsenault, P.-D., S. Wang, and J.-M. Patenaude (May 2025). "A Survey of Explainable Artificial Intelligence (XAI) in Financial Time Series Forecasting". In: *ACM Computing Surveys* 57.10, pp. 1–37. 10.1145/3729531.

Asness, C. S. (1997). "The Interaction of Value and Momentum Strategies". In: *Financial Analysts Journal* 53.2. Available at SSRN: https://ssrn.com/abstract=7687, pp. 29–36. 10.2469/faj.v53.n2.2070.

Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). *Layer Normalization*. arXiv: 1607.06450 [stat.ML]. URL: https://arxiv.org/abs/1607.06450.

Barnes, P. (Dec. 2006). "Methodological Implications of Non-Normally Distributed Financial Ratios: A Reply". In: *Journal of Business Finance Accounting* 9, pp. 51–62. 10.1111/j.1468-5957.1982.tb00972.x.

Basu, S. (1977). "INVESTMENT PERFORMANCE OF COMMON STOCKS IN RELATION TO THEIR PRICE-EARNINGS RATIOS: A TEST OF THE EFFICIENT MARKET HYPOTHESIS". In: *The Journal of Finance* 32.3, pp. 663–682. 10.1111/j.1540-6261.1977.tb01979.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1977.tb01979.x.

Bekaert, G. and C. R. Harvey (1997). "Emerging equity market volatility". In: *Journal of Financial Economics* 43.1, pp. 29–77. 10.1016/S0304-405X(96)00889-6.

Bergstra, J. and Y. Bengio (Mar. 2012). "Random Search for Hyper-Parameter Optimization". In: *The Journal of Machine Learning Research* 13, pp. 281–305.

Compustat Global database via WRDS (2025). Wharton Research Data Services (WRDS), S&P Global Market Intelligence. Comprehensive global fundamentals and market data for active and inactive publicly traded companies; coverage spans over 80 countries and over 90% of world market capitalisation. URL: https://wrds-www.wharton.upenn.edu/.

Datasets: Dividing the original dataset | Machine Learning | Google for Developers (n.d.). [Online; accessed 2025-08-13]. URL: https://developers.google.com/machine-learning/crash-course/overfitting/dividing-datasets.

Davis, J. L. (1994). "The Cross-Section of Realized Stock Returns: The Pre-COMPUSTAT Evidence". In: *The Journal of Finance* 49.5, pp. 1579–1593. 10.1111/j.1540-6261.1994.tb04773.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1994.tb04773.x.

Davis, S. J., D. Liu, and X. S. Sheng (Jan. 2021). *Stock Prices and Economic Activity in the Time of Coronavirus*. Working Paper 28320. National Bureau of Economic Research. 10.3386/w28320.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019). "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186.

Ennab, M. and H. Mcheick (Nov. 2024). "Enhancing interpretability and accuracy of AI models in healthcare: a comprehensive review on challenges and future directions". In: *Frontiers in Robotics and AI* 11, p. 1444763. 10.3389/frobt.2024.1444763.

Fama, E. F. and K. R. French (1993). "Common risk factors in the returns on stocks and bonds". In: *Journal of Financial Economics* 33.1, pp. 3–56. 10.1016/0304-405X(93)90023-5.

Garrett, B. L. and C. Rudin (2023). "The Right to a Glass Box: Rethinking the Use of Artificial Intelligence in Criminal Justice". In: *Cornell Law Review, Forthcoming*. Duke Law School Public Law & Legal Theory Series No. 2023-03. 10.2139/ssrn.4275661.

Gorishniy, Y., I. Rubachev, V. Khrulkov, and A. Babenko (2023). *Revisiting Deep Learning Models for Tabular Data*. arXiv: 2106.11959 [cs.LG]. URL: https://arxiv.org/abs/2106.11959.

Govindaraj, V., H. Vellathur Jaganathan, and P. Periyasamy (Nov. 2023). "Explainable transformers in financial forecasting". In: *World Journal of Advanced Research and Reviews* 20, pp. 1434–1441. 10.30574/wjarr.2023.20.2.1956.

Goyal, A., N. Jegadeesh, and A. Subrahmanyam (Sept. 2024). "Empirical determinants of momentum: a perspective using international data". In: *Review of Finance* 29.1, pp. 241–273. 10.1093/rof/rfae038. eprint: https://academic.oup.com/rof/article-pdf/29/1/241/59835469/rfae038.pdf.

Huang, X., A. Khetan, M. Cvitkovic, and Z. Karnin (2020). *TabTransformer: Tabular Data Modeling Using Contextual Embeddings*. arXiv: 2012.06678 [cs.LG]. URL: https://arxiv.org/abs/2012.06678.

Huang, X. and J. Marques-Silva (2024). "On the failings of Shapley values for explainability". In: *International Journal of Approximate Reasoning* 171. Synergies between Machine Learning and Reasoning, p. 109112. 10.1016/j.ijar.2023.109112.

Jegadeesh, N. and S. Titman (2011). “Momentum”. In: *SSRN Electronic Journal*. Available at SSRN: https://ssrn.com/abstract=1919226 or http://dx.doi.org/10.2139/ssrn.1919226. 10.2139/ssrn.1919226.

Kaplan, J., S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei (2020). *Scaling Laws for Neural Language Models*. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

KBinsDiscretizer — scikit-learn 1.7.1 documentation (n.d.). [Online; accessed 2025-08-07]. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.KBinsDiscretizer.html.

Kingma, D. P. and J. Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

Kobayashi, G., T. Kuribayashi, S. Yokoi, and K. Inui (2020). *Attention is Not Only a Weight: Analyzing Transformers with Vector Norms*. arXiv: 2004.10102 [cs.CL]. URL: https://arxiv.org/abs/2004.10102.

Korangi, K., C. Mues, and C. Bravo (2023). “A transformer-based model for default prediction in mid-cap corporate markets”. In: *European Journal of Operational Research* 308.1, pp. 306–320. 10.1016/j.ejor.2022.10.032.

Lang, M. H., K. Lins, and D. P. Miller (2003). “ADRs, Analysts, and Accuracy: Does Cross Listing in the United States Improve a Firm’s Information Environment and Increase Market Value?” In: *Journal of Accounting Research* 41.2, pp. 317–345. URL: https://EconPapers.repec.org/RePEc:bla:joares:v:41:y:2003:i:2:p:317-345.

Leslie, D. (June 2019). *Understanding artificial intelligence ethics and safety: A guide for the responsible design and implementation of AI systems in the public sector*. [Online; accessed 2025-07-23]. URL: https://zenodo.org/records/3240529.

Lin, Z., M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio (2017). *A Structured Self-attentive Sentence Embedding*. arXiv: 1703.03130 [cs.CL]. URL: https://arxiv.org/abs/1703.03130.

Loshchilov, I. and F. Hutter (2019). *Decoupled Weight Decay Regularization*. arXiv: 1711.05101 [cs.LG]. URL: https://arxiv.org/abs/1711.05101.

Lundberg, S. and S.-I. Lee (2017). *A Unified Approach to Interpreting Model Predictions*. arXiv: 1705.07874 [cs.AI]. URL: https://arxiv.org/abs/1705.07874.

Ma, B., Y. Xue, Y. Lu, and J. Chen (2024). *Stockformer: A Price-Volume Factor Stock Selection Model Based on Wavelet Transform and Multi-Task Self-Attention Networks*. arXiv: 2401.06139 [q-fin.TR]. URL: https://arxiv.org/abs/2401.06139.

Ma, T., W. Wang, and Y. Chen (2023). “Attention is all you need: An interpretable transformer-based asset allocation approach”. In: *International Review of Financial Analysis* 90, p. 102876. 10.1016/j.irfa.2023.102876.

Madathil, A. P., X. Luo, Q. Liu, C. Walker, R. Madarkar, and Y. Qin (2025). “A Review of Explainable Artificial Intelligence in Smart Manufacturing”. In: *International Journal of Production Research* 0.0, pp. 1–44. 10.1080/00207543.2025.2513574. eprint: https://doi.org/10.1080/00207543.2025.2513574.

Maslej, N., L. Fattorini, R. Perrault, Y. Gil, V. Parli, N. Kariuki, E. Capstick, A. Reuel, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, J. C. Niebles, Y. Shoham, R. Wald, T. Walsh, A. Hamrah, L. Santarlasci, J. B. Lotufo, A. Rome, A. Shi, and S. Oak (2025). *Artificial Intelligence Index Report 2025*. arXiv: 2504.07139 [cs.AI]. URL: https://arxiv.org/abs/2504.07139.

Mercer (2024). *AI in Investment Management: Survey of Global Managers*. https://www.mercer.com/insights/investments/portfolio-strategies/ai-in-investment-management-survey/. Accessed: 2025-07-22.

Michel, P., O. Levy, and G. Neubig (2019). *Are Sixteen Heads Really Better than One?* arXiv: 1905.10650 [cs.CL]. URL: https://arxiv.org/abs/1905.10650.

Mienye, I. D., G. Obaido, N. Jere, E. Mienye, K. Aruleba, I. D. Emmanuel, and B. Ogbuokiri (2024). “A survey of explainable artificial intelligence in healthcare: Concepts, applications, and challenges”. In: *Informatics in Medicine Unlocked* 51, p. 101587. 10.1016/j.imu.2024.101587.

Mudadla, S. (May 2024). *What is Forward and Backward fillling of Imputation in Exploratory Data Analysis (EDA)? | by Sujatha Mudadla | Medium*. [Online; accessed 2025-08-07]. URL: https://medium.com/@sujathamudadla1213/what-is-forward-and-backward-fillling-of-imputation-in-exploratory-data-analysis-eda-74ec33aca4bd.

Müller, R., S. Kornblith, and G. Hinton (2020). *When Does Label Smoothing Help?* arXiv: 1906.02629 [cs.LG]. URL: https://arxiv.org/abs/1906.02629.

Rabanser, S., T. Januschowski, V. Flunkert, D. Salinas, and J. Gasthaus (2020). *The Effectiveness of Discretization in Forecasting: An Empirical Study on Neural Time Series Models*. arXiv: 2005.10111 [cs.LG]. URL: https://arxiv.org/abs/2005.10111.

Ribeiro, M. T., S. Singh, and C. Guestrin (2016). *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. arXiv: 1602.04938 [cs.LG]. URL: https://arxiv.org/abs/1602.04938.

Rogers, A., O. Kovaleva, and A. Rumshisky (2020). *A Primer in BERTology: What we know about how BERT works*. arXiv: 2002.12327 [cs.CL]. URL: https://arxiv.org/abs/2002.12327.

Rudin, C. (2019). “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”. In: *Nature Machine Intelligence* 1.5, pp. 206–215. 10.1038/s42256-019-0048-x.

Shwartz-Ziv, R. and A. Armon (2021). *Tabular Data: Deep Learning is Not All You Need*. arXiv: 2106.03253 [cs.LG]. URL: https://arxiv.org/abs/2106.03253.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Stollenwerk, F. (2022). *Adaptive Fine-Tuning of Transformer-Based Language Models for Named Entity Recognition*. arXiv: 2202.02617 [cs.CL]. URL: https://arxiv.org/abs/2202.02617.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). *Attention Is All You Need*. Revised 2023; arXiv version consulted. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

Watanabe, S. (2023). *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. arXiv: 2304.11127 [cs.LG]. URL: https://arxiv.org/abs/2304.11127.

Zhang, Z., Y. Wang, X. Huang, T. Fang, H. Zhang, C. Deng, S. Li, and D. Yu (2025). *Attention Entropy is a Key Factor: An Analysis of Parallel Context Encoding with Full-attention-based Pre-trained Language Models*. arXiv: 2412.16545 [cs.CL]. URL: https://arxiv.org/abs/2412.16545.

Appendix A

Appendix Variables

TABLE A.1: Descriptions of Raw Financial Variables

Variable	Description
act	Total current assets
lct	Total current liabilities
invt	Inventory levels
che	Cash holdings
oancf	Operating cash flow
revt	Net sales or revenues
cogs	Cost of goods sold
oiadp	Operating income after depreciation
nicon	Net income
at	Total assets
ceq	Common equity
ebit	Earnings before interest and taxes
ebitda	EBIT plus depreciation
rect	Accounts receivable
xrd	Research and development expense
xsga	Selling, general, and administrative expense
capx	Capital expenditures
cshoi	Common shares outstanding
cshtrm	Treasury stock
prccm	Monthly closing price
dvpsxm	Dividends per share
gsector	Industry classification
sedol	Security identifier
tpci	Security type code
isalrt	Alert indicator
fic	Country code
gvkey	Company global identifier

Variable	Description
`datadate`	Fiscal period end date
`iid`	Issue identifier
`exchg`	Exchange code
`secstat`	Security status
`costat`	Company status
`datafmt`	Data format
`indfmt`	Industry format
`consol`	Consolidation level
P_t	Price at time t

1. Valuation & Yield Ratios

Book-to-Market. Measures the ratio of book equity to market value.

$$\text{Book-to-Market} = \frac{\texttt{ceq}}{\texttt{market\_cap}}$$

Earnings-to-Price. A proxy for the earnings yield of a firm.

$$\text{Earnings-to-Price} = \frac{\texttt{nicon}}{\texttt{market\_cap}}$$

Sales-to-Price. Indicates revenue generated per unit of market cap.

$$\text{Sales-to-Price} = \frac{\texttt{revt}}{\texttt{market\_cap}}$$

Free Cash Flow Yield. Expresses free cash flow as a proportion of market value.

$$\text{FCF Yield} = \frac{\texttt{oancf} - \texttt{capx}}{\texttt{market\_cap}}$$

2. Profitability & Margin Ratios

Return on Assets (ROA). Net income as a share of total assets.

$$\text{ROA} = \frac{\texttt{nicon}}{\texttt{at}}$$

Return on Equity (ROE). Profitability relative to shareholder equity.

$$\text{ROE} = \frac{\texttt{nicon}}{\texttt{ceq}}$$

Gross Margin. Measures production efficiency.

$$\text{Gross Margin} = \frac{\texttt{revt} - \texttt{cogs}}{\texttt{revt}}$$

Operating Margin. Operating profit relative to revenue.

$$\text{Operating Margin} = \frac{\texttt{oiadp}}{\texttt{revt}}$$

EBITDA Margin. EBITDA over total revenue.

$$\text{EBITDA Margin} = \frac{\texttt{ebitda}}{\texttt{revt}}$$

Net Profit Margin. Final net income as a proportion of revenue.

$$\text{Net Profit Margin} = \frac{\texttt{nicon}}{\texttt{revt}}$$

Return on Capital Employed (ROCE). Profitability relative to capital employed.

$$\text{ROCE} = \frac{\texttt{ebit}}{\texttt{at} - \texttt{lct}}$$

Cashflow ROA. Cash flow from operations over total assets.

$$\text{Cashflow ROA} = \frac{\texttt{oancf}}{\texttt{at}}$$

3. Liquidity Ratios

Current Ratio. Measures ability to meet short-term obligations.

$$\text{Current Ratio} = \frac{\texttt{act}}{\texttt{lct}}$$

Quick Ratio. Excludes inventory to provide a conservative liquidity view.

$$\text{Quick Ratio} = \frac{\texttt{act} - \texttt{invt}}{\texttt{lct}}$$

Cash Ratio. Measures cash-only coverage of current liabilities.

$$\text{Cash Ratio} = \frac{\texttt{che}}{\texttt{lct}}$$

4. Efficiency / Activity Ratios

Asset Turnover. Sales generated per unit of total assets.

$$\text{Asset Turnover} = \frac{\texttt{revt}}{\texttt{at}}$$

Inventory Turnover. Measures how quickly inventory is sold.

$$\text{Inventory Turnover} = \frac{\texttt{cogs}}{\texttt{invt}}$$

Receivables Turnover. Indicates efficiency of collections.

$$\text{Receivables Turnover} = \frac{\texttt{revt}}{\texttt{rect}}$$

5. Balance Sheet Composition & Accounting Quality

NOA Ratio. Net operating assets as a share of total assets.

$$\text{NOA Ratio} = \frac{\texttt{at} - \texttt{che} - (\texttt{at} - \texttt{ceq})}{\texttt{at}}$$

Accruals Ratio. Non-cash component of earnings.

$$\text{Accruals Ratio} = \frac{\texttt{nicon} - \texttt{oancf}}{\texttt{at}}$$

SG&A to Assets. Overhead expenses relative to total assets.

$$\text{SG\&A to Assets} = \frac{\texttt{xsga}}{\texttt{at}}$$

Cash to Assets. Cash holdings relative to asset base.

$$\text{Cash to Assets} = \frac{\texttt{che}}{\texttt{at}}$$

Inventory to Assets. Inventory intensity in asset structure.

$$\text{Inventory to Assets} = \frac{\texttt{invt}}{\texttt{at}}$$

SG&A to Sales. Overhead costs relative to revenue.

$$\text{SG\&A to Sales} = \frac{\texttt{xsga}}{\texttt{revt}}$$

Capex to Assets. Investment intensity relative to assets.

$$\text{Capex to Assets} = \frac{\texttt{capx}}{\texttt{at}}$$

Capex to Sales. Capital expenditures as a share of revenue.

$$\text{Capex to Sales} = \frac{\texttt{capx}}{\texttt{revt}}$$

Capex to Cashflow. Proportion of operating cash flow used for capital investment.

$$\text{Capex to Cashflow} = \frac{\texttt{capx}}{\texttt{oancf}}$$

6. Return-Based Ratios

Forward 12-Month Return. Measures future return based on price 12 months ahead.

$$\text{return\_12m\_forward} = \frac{P_{t+12} - P_t}{P_t}$$

Past 12-Month Return. Realised return over the previous 12 months.

$$\text{past\_12m\_return} = \frac{P_t - P_{t-12}}{P_{t-12}}$$

Momentum. Price momentum based on change from 12 to 1 months ago.

$$\text{momentum} = \frac{P_{t-1} - P_{t-12}}{P_{t-12}}$$

Appendix B

Appendix Testing Analysis

B.1 Evaluation/Classification Metrics Overview

Precision Precision measures how accurate the model's positive predictions are. It is defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

A high precision score means that when the model predicts a certain class, it's often correct. In most cases, we aim for this to be as high as possible.

Recall Recall measures how well the model captures all the actual instances of a class. It is defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

High recall means the model is good at detecting members of a class, though it may come at the cost of precision. There's often a trade-off between precision and recall.

F1 Score The F1 score is the harmonic mean of precision and recall, providing a single metric that balances the two:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

It is the most commonly used performance measure in imbalanced settings, and we typically want this to be high.

Support Support refers to the number of true instances of each class in the dataset. It is not a performance metric but rather a helpful indicator of class frequency.

Accuracy Accuracy measures the overall percentage of correct predictions. While widely used, it may not be reliable in imbalanced datasets:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

Macro Average The macro average calculates the metric (e.g., precision, recall, F1) for each class independently and then takes the unweighted mean. This treats all classes equally regardless of their size.

Weighted Average The weighted average calculates the metric for each class and takes the mean weighted by the support of each class. This accounts for class imbalance and gives more influence to larger classes.

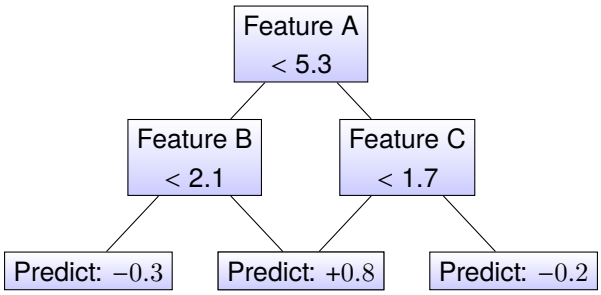
B.2 XGBoost (Extreme Gradient Boosting)

Gradient Boosting is an ensemble technique where decision trees are built sequentially, with each new tree correcting the errors of the previous ones. It optimizes a loss function using gradient descent in function space.

XGBoost (Extreme Gradient Boosting) is a specific, highly optimized implementation of gradient boosting. It introduces enhancements such as regularization (L1 and L2), efficient handling of missing values, and parallel processing. While it follows the same core idea as gradient boosting, XGBoost is far quicker, more robust to overfitting, and easier to tune in practice.

B.2.1 Illustration: Weak Learner in Gradient Boosting

Below is a simplified decision tree often used as a weak learner in gradient boosting:



Unlike standard classification trees, each leaf in this tree outputs a correction value (not a class). Gradient Boosting combines the outputs of many such trees to iteratively reduce prediction error.

B.3 Random Forest

A tuned Random Forest (3-fold CV) selected $n_{\text{estimators}}$=300, max_depth=10, min_samples_split=5 (CV accuracy = 0.4161). On the held-out test set: accuracy = 41.2% (random baseline = 33.3%). The high-return bucket (class 2) achieved precision = 39.94%, recall = 21.69%, accuracy = 61.94%.

Per-class test metrics

Class	Precision	Recall	F1	Support
0	0.363	0.491	0.417	6794
1	0.457	0.529	0.490	9680
2	0.399	0.217	0.281	8606
Macro avg	0.406	0.412	0.396	25080
Weighted avg	0.412	0.412	0.399	25080

Confusion matrix (rows: true, cols: predicted)

	0	1	2
0	3338	2314	1142
1	2895	5120	1665
2	2970	3769	1867

B.4 SHAP

What is SHAP?

SHAP (SHapley Additive exPlanations) is a game theoretic approach to try and explain the output of any machine learning model. It attributes each feature a contribution value for a particular prediction, based on Shapley values from cooperative game theory.

In addition, the SHAP framework includes a suite of diagnostic tools and visualisations that facilitate both local (per-prediction) and global (dataset-level) analysis of feature importance. A key strength of SHAP lies in its **model-agnostic design**, which enables consistent interpretation across a variety of machine learning models.

Given a model prediction, SHAP decomposes it as:

$$f(x) = \phi_0 + \sum_{i=1}^{n} \phi_i$$

where: - $f(x)$ is the model output, - ϕ_0 is the expected prediction, - ϕ_i is the SHAP value for feature i.

Advantages

- **Model-agnostic**: Can be applied to any black box model, including tree based methods and neural networks.
- **Feature explanations**: Shows how much each feature contributed to an individual prediction.
- **Global insight**: Aggregating SHAP values across many predictions can provide an overall sense of feature importance.

Limitations

- **Computational cost**: SHAP is resource intensive, especially for complex models or large feature sets. Deep models can also be harder to set up and interpret with SHAP.
- **Correlated features**: SHAP assumes features are independent, which may distort importance when features are correlated.
- **Misuse risk**: Over reliance on SHAP values can lead to misleading conclusions, especially if the underlying model is not well understood or validated.

Interpretability note: While SHAP is useful for understanding which features influence predictions, it **does not** explain how the model works internally. It reveals what the model focuses on, not how it processes information or makes decisions. In other words, SHAP gives insight into the **outputs**, not the **mechanism** behind them.

B.5 Adam and AdamW

Adam combines two key ideas: the momentum term from classical stochastic gradient descent (SGD), and adaptive learning rates. This combination generally results in more stable and faster convergence.

The update rule for Adam is given by:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
\theta_{t+1} &= \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}
$$

Here, θ_t are the weights at step t, g_t is the gradient, α is the learning rate, and β_1, β_2 are decay rates for the first and second moments.

Adam also allows input for learning rate and weight decay, which we tune through hyperparameter search.

AdamW (Loshchilov and Hutter 2019) is a variation of Adam that improves generalisation by decoupling the weight decay from the gradient update. In Adam, weight decay is applied implicitly through the gradient. In AdamW, it is applied directly to the weights, resulting in a cleaner separation of concerns.

The AdamW update rule is:

$$
\theta_{t+1} = \theta_t - \alpha \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right)
$$

B.6 Attention Matrix Extraction and Hooks

In the context of attention mechanisms, a query can be likened to a search request, such as entering a term into a video-sharing platform. The keys represent descriptive information associated with each item in the database, enabling the system to determine how relevant each item is to the query. The attention mechanism compares the query with all keys to produce relevance scores, which are then used to weight the values, representing the actual content to be retrieved.

The default implementation of `nn.MultiheadAttention` in PyTorch does not store attention weights by default. To extract them, we modify the internal behaviour of the model and apply **forward hooks** during inference. These hooks capture intermediate outputs without altering the trained model. The implementation is based on a modification strategy adapted from an open-source contribution by a GitHub user **(airalcorn2 n.d.)**, who addressed this limitation in a similar context.

Once enabled, attention weights are extracted by running the model in evaluation mode, using `torch.no_grad()` to avoid unnecessary gradient computations. These weights are subsequently visualised using heat maps.

B.7 Seed Robustness

Random initialisation and stochastic training (e.g., minibatch order) can affect both metrics and attention patterns. To avoid conditioning results on a single realisation, we fix the data splits and hyperparameters and repeat training $S = 4$ times, differing only in random initialisation. We report mean $\pm$ sd for validation/test Accuracy and Macro-F1 (**Table B.1**). All interpretability figures are shown for one representative run (the model with the highest validation Macro-F1); we do not average attention maps across runs.

Metric	Validation	Test
Accuracy	0.413 ± 0.019	0.385 ± 0.017
Macro-F1	0.393 ± 0.026	0.373 ± 0.021

TABLE B.1: Seed robustness (mean $\pm$ sd over $S = 4$ independent runs with identical splits/hyperparameters).

Objective sensitivity. Placing heavy weight on Class-2 F1 during selection caused mode collapse (predicting Class 2 almost exclusively). Accordingly, robustness statistics are reported under the standard, balanced objective.

B.8 ReLU and GELU

ReLU (Rectified Linear Unit). Defined as

$$\mathrm{ReLU}(x) = \max(0, x),$$

ReLU is simple, computationally efficient, and mitigates vanishing gradients by passing positive inputs unchanged.

GELU (Gaussian Error Linear Unit). Defined as

$$\mathrm{GELU}(x) = x\,\Phi(x),$$

where $\Phi(x)$ is the standard normal CDF. GELU smoothly weights inputs by their probability of being positive, combining properties of ReLU and sigmoid.

Comparison. Although GELU is often used in Transformer architectures, this study employs ReLU for its efficiency, ease of implementation, and stable performance on tabular financial data.

Appendix C

Results and Discussion Appendix

C.1 Optuna

C.1.1 Optuna Advantages and Disadvantages

Optuna offers several advantages that make it well suited to our problem:

- **Efficient Search:** Unlike standard random search, Optuna uses a *Tree-structured Parzen Estimator (TPE)* to "guide" the random searches using results from previous trials **(Watanabe 2023)**. This leads to faster and more efficient convergence.
- **Scalability and Flexibility:** Optuna can handle a wide and dynamic range of hyperparameters, including architectural and training-related ones.
- **Pruning Support:** Although our model includes early stopping via a patience parameter, Optuna adds another layer of efficiency by pruning unpromising trials early.
- **Parallel Execution:** It supports distributed and parallel training, enabling the use of multiple CPU cores and GPUs.

Despite its advantages, Optuna also has some drawbacks:

- **Non-Deterministic Results:** The TPE algorithm is non deterministic, meaning different runs can yield different outcomes.
- **Lack of Transparency:** As a black-box optimisation method, TPE offers limited interpretability, which presents a minor concern given the explainability objectives of this study.

Nevertheless, the primary aim is to interpret the behaviour of the Transformer model itself, rather than the hyperparameter search process. As such, the reduced transparency of the tuning procedure represents an acceptable trade-off in light of the efficiency and performance benefits it provides.

C.1.2 Optuna Results

Before proceeding, note that obtaining these hyperparameters is computationally intensive. To manage this, we utilised **Blythe**, a high-performance computing (HPC) cluster provided by the University of

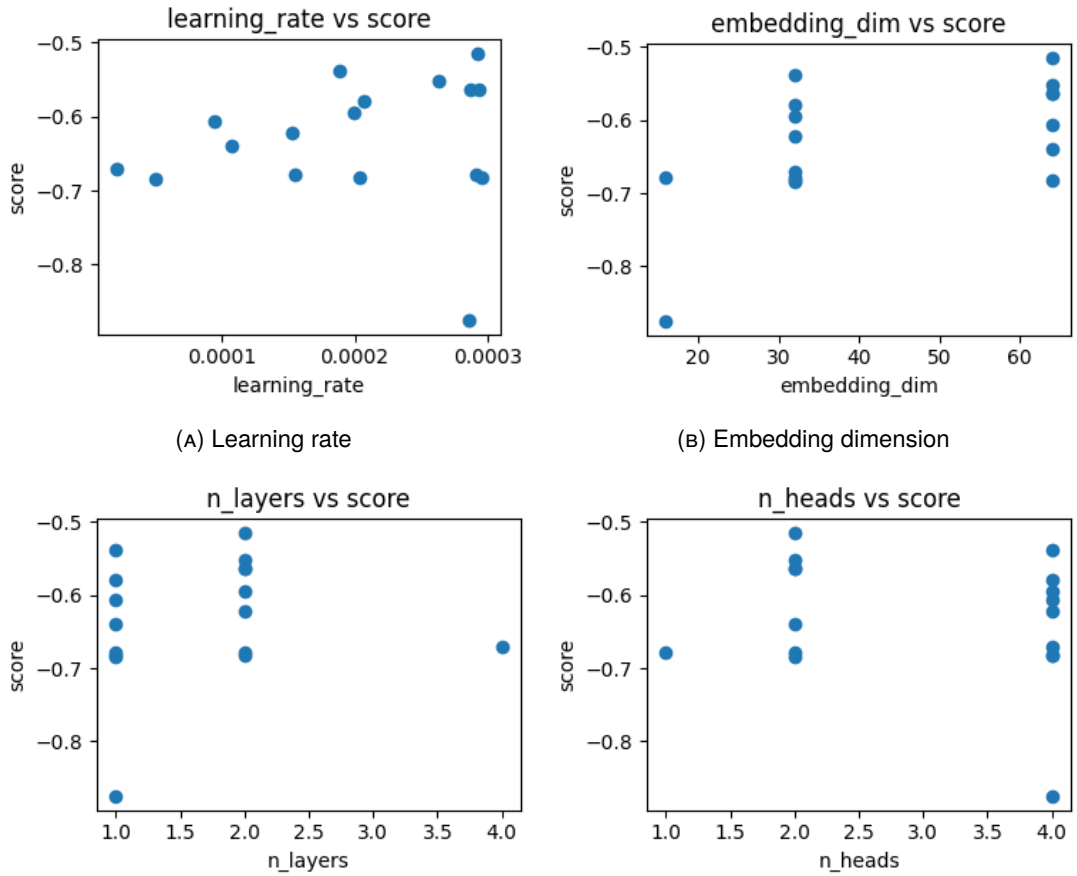
Warwick. Despite this computational support, runtime constraints persisted: a single Optuna run of approximately 25 trials required around one hour to complete. This limitation should be kept in mind when interpreting the hyperparameter results.

We begin by presenting results obtained from the hyperparameter optimisation process, carried out using the Optuna framework. The search explored a wide variation of models and training parameters, including architectural settings (e.g., number of attention heads, embedding dimensions), as well as training level configurations such as learning rate schedules, optimiser types, and class weighting schemes. By treating nearly every component of the modelling pipeline as a tunable parameter, the optimisation process was able to identify a well-performing configuration suitable for deployment.

Each candidate configuration was trained and evaluated using a custom scoring metric designed to reflect the primary classification objective. This custom scoring allowed the tuning process to account for both predictive accuracy and class specific performance. This metric integrates three key components. First, the final validation accuracy is included, reflecting the model's ultimate predictive performance. Second, a stability term is added, defined as the cumulative change in validation accuracy across training epochs. This term encourages smoother and more stable convergence. Third, an extra stability term is computed but for the validation loss instead of the validation accuracy, promoting consistent reduction in error over time.

By combining these elements, the scoring function rewards both high accuracy and training stability, ensuring that the selected model performs reliably rather than relying on isolated improvements. The objective of the optimisation is to maximise this score.

Figures for each hyperparameter (e.g., **Figure C.1b**, **Figure C.1c**) correspond to Trial 5, which produced the "best" results and included 25 non-pruned trials. While the plots assume independence between hyperparameters (which is not strictly true), we consider this a reasonable simplification for some much needed exploratory analysis.



(A) Learning rate

(B) Embedding dimension

(C) Number of layers

(D) Attention heads

FIGURE C.1: Impact of model architecture on performance.

Note: Due to the limited number of data points available, the results being presented should be interpreted with caution. While further testing is necessary to draw definitive conclusions, the findings serve as a useful foundation for identifying promising directions for future exploration.

Embedding Dimension: A higher embedding dimension enables the model to capture richer representations but increases complexity. As seen in **Figure C.1b**, there is a general trend where higher dimensions yield better scores.

Number of Layers (n_layers): More layers increase depth and learning capacity. **Figure C.1c** shows that 1–2 layers perform best. Although Optuna explored up to 5 layers, results worsened, likely due to overfitting again, consistent with expectations.

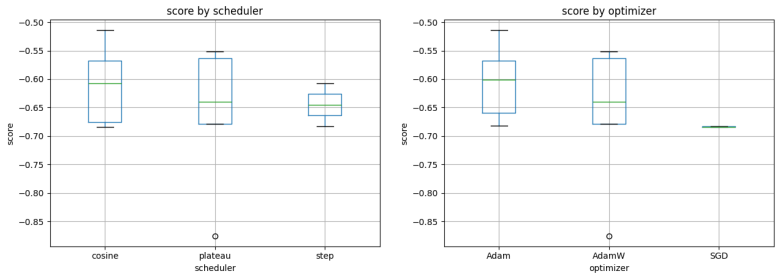
Number of Heads (n_heads): This controls the number of attention heads in the Transformer. While **Figure C.1d** suggests 2 heads often lead to better scores, Optuna leaned toward selecting 4 heads. Further investigation is warranted, as the optimal number of heads may depend on interactions with other parameters.

Dropout (Transformer and Classifier): Dropout was applied both to the Transformer and final classifier. These are standard mechanisms to reduce overfitting. Values were kept moderate to maintain model capacity.

Weight Decay: Used as L2 regularization to penalize large weights and encourage simplicity. Optuna consistently selected small values, which makes sense—given the model's complexity and limited data, regularization helps avoid overfitting.

Learning Rate: As shown in **Figure C.1a**, higher learning rates appeared to correlate with better scores, although the correlation is weak (see **Table C.1**). This trend was particularly pronounced when paired with the cosine scheduler (**Figure C.3**). Cosine annealing gave a wide distribution of scores depending on the learning rate, while step and plateau schedulers were rarely selected. Excessively high learning rates tended to degrade performance, but overall, moderately high values worked well.

Optimiser: Figure C.2b shows that both `Adam` and `AdamW` performed similarly, with `Adam` slightly outperforming `AdamW` on average. This may be due to confusion between weight decay as a separate hyperparameter versus its internal role in `AdamW`. `SGD`, on the other hand, consistently underperformed—except in one case (Trial 1), where it was selected alongside a deep architecture (4 heads and 4 layers). This might indicate that in some scenarios, a simpler optimiser can act as an implicit regulariser, balancing out increased model complexity.



(A) Scheduler type (B) Optimiser choice

FIGURE C.2: Effect of learning dynamics on validation score.

Scheduler: Cosine annealing was frequently selected and yielded relatively strong results, as seen in Figures **Figure C.2a** and **Figure C.3**. Step-based scheduling was rarely favored by Optuna, and its performance was generally lower. However, due to limited samples, strong conclusions should not be drawn.

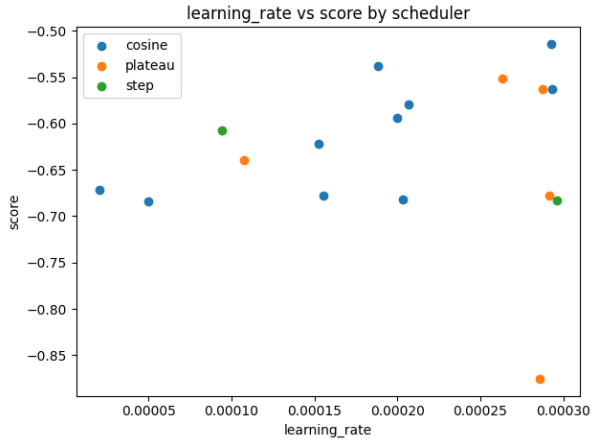


FIGURE C.3: Joint effect of learning rate and scheduler on validation score.

Other Notes

Table C.1 shows basic correlation values between hyperparameters and performance. Some features like `step_size` appear highly correlated but this is misleading due to the small number of trials involving them. One interesting insight is the negative correlation between `batch_size` and score, smaller batch sizes (16–32) consistently perform better, likely due to more frequent updates and regularization effects.

TABLE C.1: Correlation with `score` (excluding `score` itself)

Hyperparameter	**Correlation with Score**
step_size	1.000
plateau_factor	0.897
embedding_dim	0.556
w_pos	0.294
dropout_transformer	0.252
n_layers	0.145
learning_rate	0.096
patience	-0.031
T_0	-0.072
ls	-0.085
n_heads	-0.199
dropout_classifier	-0.299
weight_decay	-0.372
batch_size	-0.577
plateau_patience	-0.705

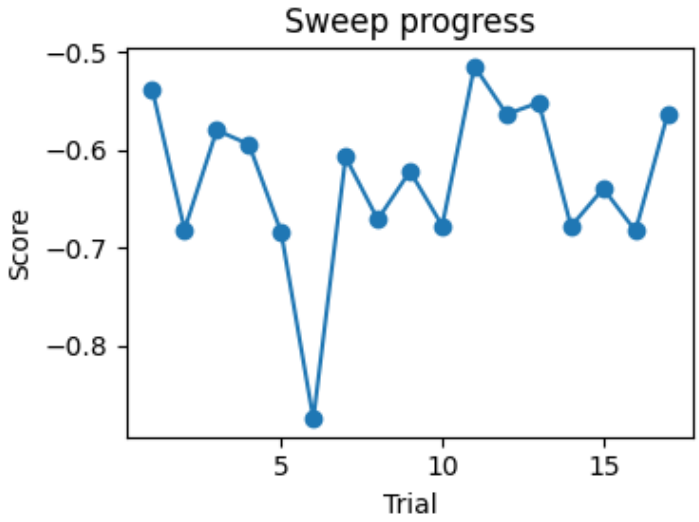


FIGURE C.4: Optuna hyperparameter sweep vs. custom score.

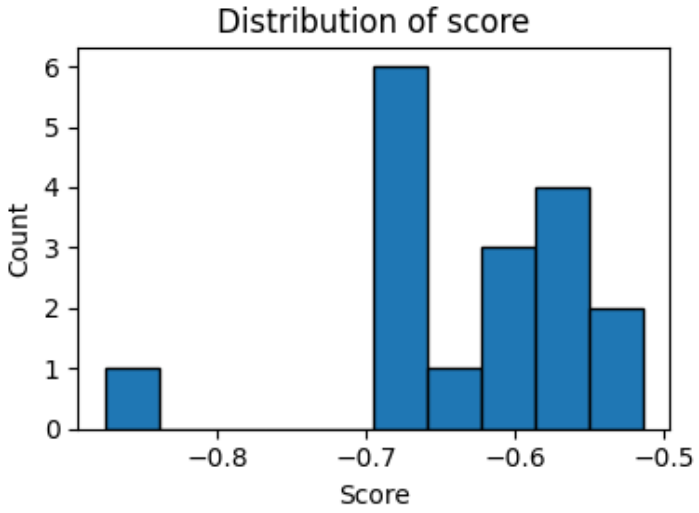


FIGURE C.5: Distribution of custom scores across all Optuna trials.

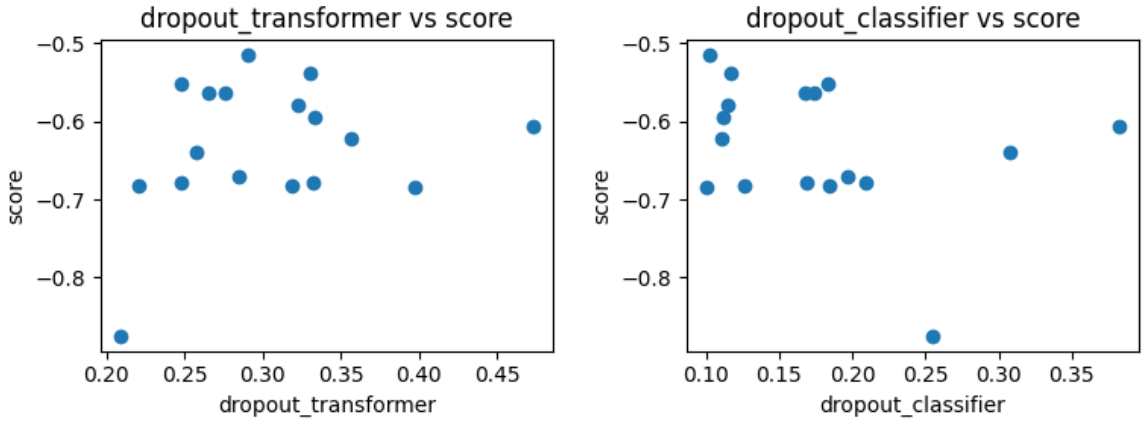


FIGURE C.6: Effect of dropout rates on transformer and classifier performance.

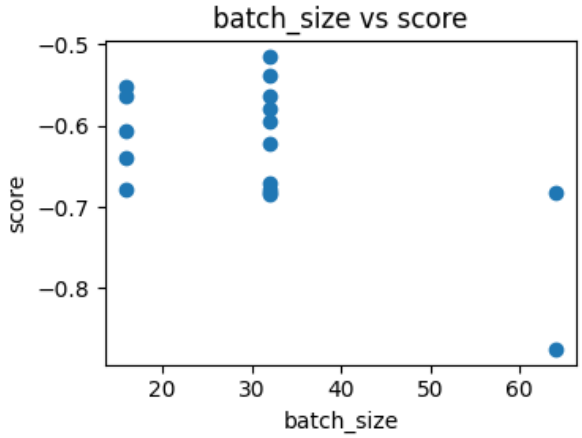


FIGURE C.7: Impact of batch size on validation score.

C.2 Attention Matrix Scoring

We compute a cumulative attention score for each token (input) by summing the attention it receives across all layers and heads. This total score serves as a proxy for importance, enabling us to rank features accordingly. The corresponding formulation is given below:

$$\alpha_i = \frac{1}{L} \sum_{l=1}^{L} \left(\sum_{q=1}^{N} \frac{1}{H} \sum_{h=1}^{H} A_{q,k}^{(l,h)} \right),$$

where:

- α_i is the average attention received by token i across all layers, heads, and queries,
- L is the total number of layers,

- H is the number of attention heads per layer,
- N is the total number of tokens in the input sequence,
- $A_{q,k}^{(l,h)}$ is the attention weight from query token q to key token i in layer l and head h.

C.3 Individual Layer and Head Analysis

C.3.1 Entropy-Based Importance

The entropy for each layer is calculated using the formula:

$$\mathcal{H}_{b,h,q} = -\sum_{k} A \cdot \log A$$

where:

- $\mathcal{H}_{b,h,q}$ is the entropy of the attention distribution for batch index b, head h, and query token q,
- $A = A_{b,h,q,k}$ denotes the attention weight assigned from query q to key k, for head h and batch b,
- k indexes over all key tokens,
- a small constant ε is added to A to avoid taking the logarithm of zero.

We first compute the entropy per query, then average over the batch and query dimensions to get **per head** entropy, and finally average across heads to obtain a scalar entropy value for each layer.

Layers are then ranked from lowest to highest entropy, indicating how "focused" each layer is.

C.3.2 Weight Norms

The weighted norm score ($\nu_{i,j}$) is computed as:

$$\nu_{i,j} = \|\mathbf{W}_q\|_2 + \|\mathbf{W}_k\|_2 + \|\mathbf{W}_v\|_2$$

where $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ are the learned weight matrices for the query, key, and value projections, respectively, and $\nu_{i,j}$ denotes the weight norm score for layer i and head j.

C.4 Attention Rollout

The formulation is based on the work of **(Abnar and Zuidema 2020)**, where the attention matrix for each layer is augmented with the identity matrix to preserve residual information, normalized, and then propagated:

$$\tilde{A}^{(l)} = \frac{A^{(l)} + I}{\sum_j (A^{(l)} + I)_{ij}}, \quad R = \tilde{A}^{(L)} \cdot \tilde{A}^{(L-1)} \cdot \ldots \cdot \tilde{A}^{(1)}$$

where:

- $A^{(l)} \in \mathbb{R}^{N \times N}$ is the average attention matrix from layer l,
- I is the identity matrix (residual connection),
- $\tilde{A}^{(l)}$ is the normalized matrix after adding the residual,
- $R \in \mathbb{R}^{N \times N}$ is the final roll-out matrix capturing end-to-end token influence.

Each element R_{ij} indicates the cumulative influence of token j (as a key) on token i (as a query), aggregated through the layers.

C.5 Ablation Results

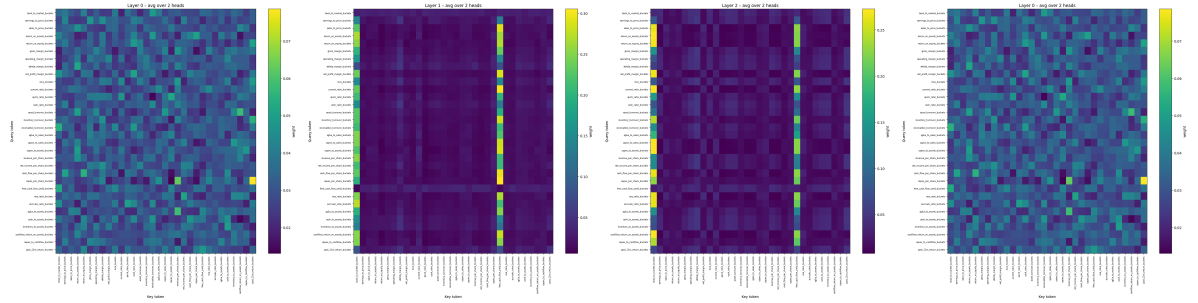


FIGURE C.8: Attention maps when dropping `momentum_buckets`.

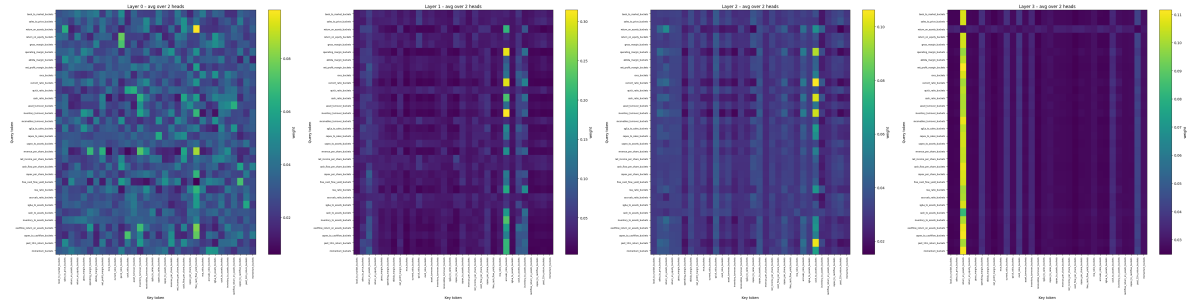


FIGURE C.9: Attention maps when dropping `earnings_to_price_buckets`.

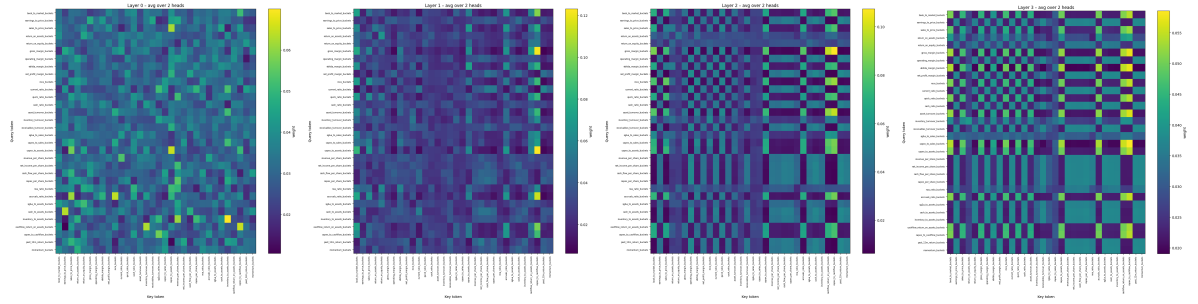


FIGURE C.10: Attention maps when dropping `free_cash_flow_yield_buckets`.

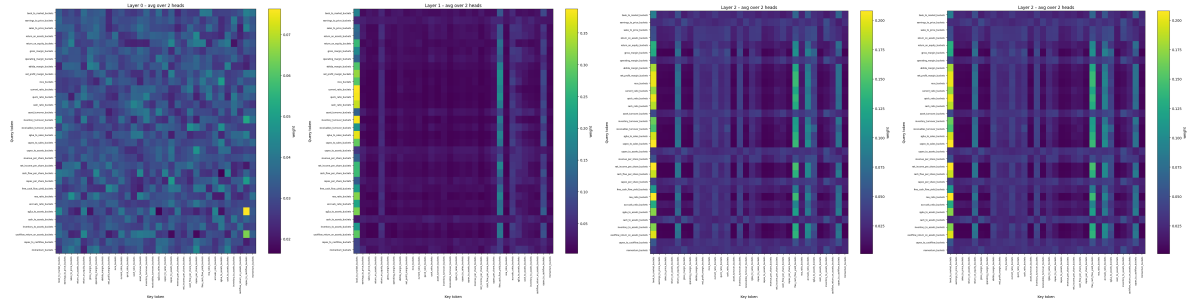


FIGURE C.11: Attention maps when dropping `past_12m_return_buckets`.

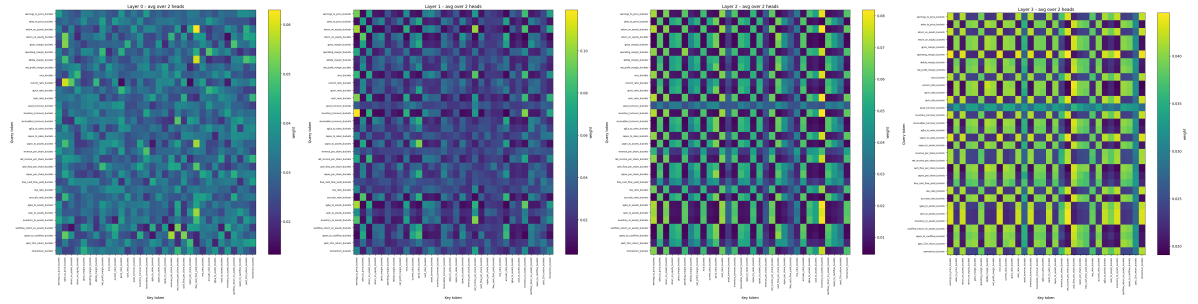


FIGURE C.12: Attention maps when dropping `book_to_market_buckets`.

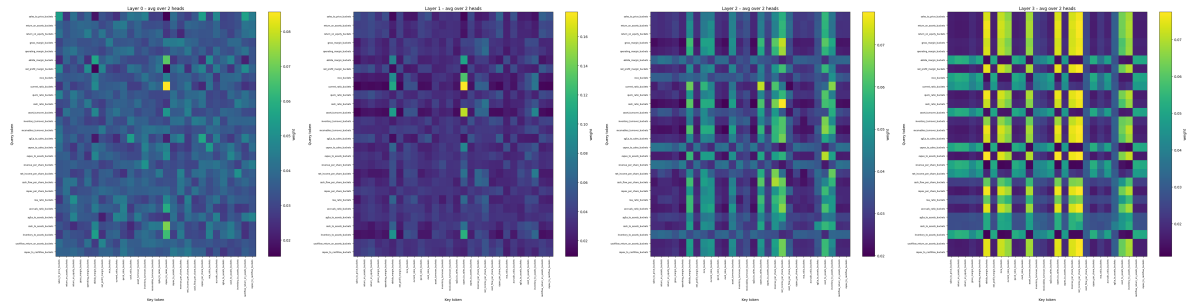


FIGURE C.13: Attention maps when dropping all top-5 ranked features.