Enhancing Interpretability in Tabular Deep Learning: A Comparative Study of Sparsemax and Softmax in Feature Tokenizer Transformers

Aaron Fandrei a.fandrei@campus.lmu.de
Institute of Statistics
Ludwig-Maximilians Universität
Geschwister-Scholl-Platz 1, 80539 München

Supervisor: Prof. Dr. Matthias Feurer

Editor: -/-

Abstract

Despite the prevalence of tabular data in critical domains such as healthcare and finance, deep learning models for tabular data often lack the interpretability required for high-stakes decision making. This paper investigates the impact of replacing conventional softmax activation with sparsemax in the attention mechanism of Feature Tokenizer (FT) Transformers to enhance interpretability while maintaining competitive performance. We evaluate both standard and sparse attention variants of FT Transformer with linear and piecewise linear embeddings across five diverse datasets. Our experiments measure the rank correlation between attention-based feature importance and Permutation Feature Importance (PFI), a model-agnostic interpretability method. Results show that sparsemax significantly improves the correlation between attention weights and PFI scores across several datasets, particularly when combined with appropriate embedding techniques, without substantial performance degradation. This demonstrates that sparsemax attention mechanisms can serve as more reliable indicators of feature importance, addressing a key interpretability challenge in tabular deep learning. The source code has been made available at `https://github.com/aaronfdr/sparseFTTransformer`.

1 Introduction and Motivation

Tabular data, characterized by its structured format with rows representing instances and columns representing features, remains the most common data type in real-world applications of machine learning. It is ubiquitous across domains ranging from healthcare (electronic health records) to finance (transaction data, risk assessments) and beyond. The interpretation of models trained on such data is particularly critical in these high-stakes domains, where understanding the rationale behind predictions can be as important as the predictions themselves.

For nearly a decade, gradient-boosted decision trees (GBDTs) have dominated tabular data modeling, offering strong performance and a degree of interpretability through feature importance measures and partial dependence plots (Friedman, 2001; Chen and Guestrin, 2016). However, recent advances in deep learning architectures have begun to challenge this

dominance (Gorishniy et al., 2021; Arik and Pfister, 2021), offering comparable or superior performance on many tabular datasets.

Despite these advances, deep neural networks, particularly transformer-based architectures, suffer from interpretability challenges. Although attention mechanisms provide a window into model behavior, standard softmax attention distributes weights across all features, making it difficult to isolate the most influential inputs (Jain and Wallace, 2019). This limitation is particularly problematic in domains like healthcare and finance, where regulatory requirements and ethical considerations often necessitate model explanations (Rudin, 2019).

One of the first models to incorporate an attention mechanism for tabular data was TabNet (Arik and Pfister, 2021), which notably utilized the sparsemax operator instead of the traditional softmax. Sparsemax (Martins and Astudillo, 2016) produces sparse probability distributions, potentially aligning better with human intuition about feature relevance by emphasizing a smaller subset of important features.

In this work, we investigate whether replacing softmax with sparsemax in the attention mechanism of Feature Tokenizer (FT) Transformers improves the correlation between attention weights and feature importance, thus enhancing interpretability. We use attention scores as indicators of feature importance and measure their alignment with Permutation Feature Importance (PFI), a model-agnostic interpretation method widely accepted for its fidelity to model behavior (Fisher et al., 2019).

Our research builds on two key papers: "Revisiting Deep Learning Models for Tabular Data" (Gorishniy et al., 2021), which introduced the FT Transformer, and "On Embeddings for Numerical Features in Tabular Deep Learning" (Gorishniy et al., 2022), which proposed piecewise linear embeddings to improve the representation of numerical features. We extend this work by examining how sparsemax affects both model performance and interpretability in tabular deep learning.

2 Preliminaries

2.1 Notation

We denote tabular data as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where each instance $\mathbf{x}_i \in \mathbb{R}^d$ consists of d features (both numerical and categorical) and y_i represents the target variable. For classification tasks, $y_i \in \{1, 2, ..., C\}$ where C is the number of classes, and for regression tasks, $y_i \in \mathbb{R}$.

2.2 Attention Mechanism

The attention mechanism, first introduced by Bahdanau et al. (2014) and later refined in the "Attention is All You Need" paper (Vaswani et al., 2017), has become a cornerstone of modern deep learning architectures. Attention allows models to focus on different parts of the input when producing outputs.

In the context of transformers, the attention mechanism involves three main components: queries ($\mathbf{Q}$), keys ($\mathbf{K}$), and values ($\mathbf{V}$). These are derived from the input embeddings through linear projections. The attention weights are computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{1}$$

where d_k is the dimensionality of the keys and serves as a scaling factor. The softmax operation normalizes the attention scores to form a probability distribution.

Multi-head attention extends this concept by applying multiple attention functions in parallel:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, ..., \text{head}_h)\mathbf{W}^O \tag{2}$$

where each head is computed as:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{3}$$

This allows the model to jointly attend to information from different representation subspaces.

2.3 Transformer Architecture

The transformer architecture (Vaswani et al., 2017) consists of two main components: encoders and decoders. For tabular data analysis, we primarily focus on the encoder component.

A transformer encoder consists of multiple identical layers, each containing two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Each sub-layer employs a residual connection followed by layer normalization:

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \tag{4}$$

The feed-forward network consists of two linear transformations with a ReLU activation in between:

$$\text{FFN}(x) = \max(0, x\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2 \tag{5}$$

In FT Transformer, the encoder architecture processes tokenized tabular features, with a special CLS token used to aggregate information for the final prediction.

2.4 Interpretable Machine Learning

Interpretable machine learning refers to methods and models that provide human-understandable explanations for their predictions (Molnar, 2020). Interpretability is crucial for several reasons:

1. **Trust and adoption**: Users are more likely to trust and adopt models whose decisions they understand.

2. **Regulatory compliance**: Domains like finance (GDPR, FCRA) and healthcare (HIPAA) increasingly require explainable AI systems.

3. **Scientific discovery**: Interpretable models can help identify new patterns and generate hypotheses.

4. **Bias detection**: Understanding model behavior helps identify and mitigate unfair biases.

Deep learning models are generally considered "black boxes" due to their complex, non-linear nature and high number of parameters, making their decisions difficult to interpret directly. Various post-hoc interpretation methods have been developed to address this limitation, including feature importance measures, partial dependence plots, and attention visualization (Guidotti et al., 2018).

2.5 Feature Importance

Feature importance quantifies the contribution of each input feature to a model's predictions. Various types of feature importance exist:

1. **Model-specific importance**: Derived directly from model parameters (e.g., coefficient magnitudes in linear models, split frequencies in tree-based models).

2. **Perturbation-based importance**: Measures the change in model performance when features are perturbed or removed.

3. **Attribution-based importance**: Allocates prediction contributions to individual features (e.g., SHAP values, integrated gradients).

4. **Attention-based importance**: Uses attention weights as proxies for feature importance.

2.6 Permutation Feature Importance

Permutation Feature Importance (PFI), introduced by Breiman (2001) for random forests and generalized by Fisher et al. (2019), is a model-agnostic approach that measures the decrease in model performance when a feature's values are randomly permuted.

The PFI algorithm works as follows:

Algorithm 1 Permutation Feature Importance

Input: Model f, dataset $D = \{(X_i, y_i)\}$, performance metric M
Output: Importance scores for each feature
Compute baseline performance: baseline $= M(f(X), y)$
for each feature j **do**
    Create permuted dataset by randomly shuffling feature j: $X' = X$ with $X[:, j]$ permuted
    Compute performance on permuted data: $\text{score}_j = M(f(X'), y)$
    Calculate importance: $\text{imp}_j = \text{baseline} - \text{score}_j$
end for
return importance scores for all features

PFI has several advantages: It is model-agnostic (applicable to any model), conceptually simple and intuitive, and captures feature interactions. However, it also has limitations: computationally expensive for large datasets and complex models, sensitive to feature correlations, and may provide misleading results if features are highly dependent.

3 Feature Tokenizer Transformer

The Feature Tokenizer (FT) Transformer was introduced by Gorishniy et al. (2021) as a specialized transformer architecture for tabular data. It was developed to address the limitations of earlier deep learning approaches to tabular data, which often failed to match the performance of gradient boosted decision trees (GBDTs). The FT Transformer demonstrated competitive performance against both traditional GBDTs and other deep learning architectures such as ResNet, achieving state-of-the-art results on many benchmark datasets. Its success challenged the notion that deep learning models inherently underperform on tabular data.

3.1 Architecture

The FT Transformer architecture consists of four main components, as illustrated in Figure 1:

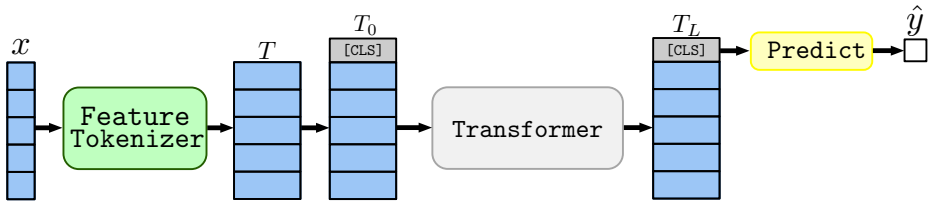

Figure 1: Complete FT Transformer architecture: Input features are processed by the Feature Tokenizer, a special CLS token is prepended, the sequence passes through transformer layers, and the final prediction is made based on the transformed CLS token representation. Figure reproduced from Gorishniy et al. (2021).

1. **Feature Tokenizer**: Converts each feature into a token embedding. Categorical features are embedded using standard embedding layers, while numerical features are embedded using either linear or piecewise linear embeddings. The detailed implementation of the Feature Tokenizer is shown in Figure 2(a).

2. **CLS Token**: A special learnable token is prepended to the sequence of feature embeddings, similar to BERT (Devlin et al., 2019). This token aggregates information from all features to make the final prediction.

3. **Transformer Encoder**: A stack of transformer encoder layers processes the token embeddings. Each layer applies multi-head self-attention followed by a feed-forward network with residual connections and layer normalization, as shown in Figure 2(b).

4. **Prediction Head**: The final output layer that takes the CLS token representation and maps it to the target output space (e.g., class probabilities for classification or real values for regression).

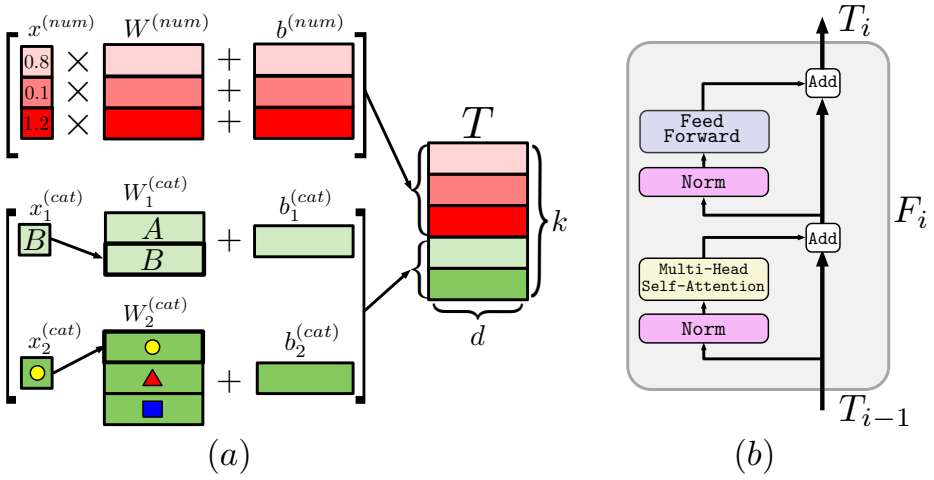

Figure 2: Feature Tokenizer architecture: (a) Feature tokenization process showing separate paths for numerical and categorical features. Numerical features are embedded via linear projections (shown) or piecewise linear embeddings. Categorical features are embedded using standard embedding layers. (b) Transformer encoder block with multi-head self-attention, normalization, and feed-forward layers. Figure reproduced from Gorishniy et al. (2021)

The model can be formalized as follows:

$$\hat{y} = \text{Head}(\text{Encoder}([\text{CLS}; \text{Tokenizer}(x_1); \text{Tokenizer}(x_2); ...; \text{Tokenizer}(x_d)])) \tag{6}$$

where $\hat{y}$ is the prediction, $x_1, x_2, ..., x_d$ are the features, and $[\cdot;\cdot]$ denotes concatenation.

3.2 Feature Tokenization Process

The Feature Tokenizer component, shown in detail in Figure 2(a), implements separate processing paths for numerical and categorical features:

- **Numerical features** ($x^{(num)}$): In the basic configuration, numerical features are transformed through a linear projection ($W^{(num)}$) with a bias term ($b^{(num)}$). In the enhanced version of the model, numerical features can instead be processed through piecewise linear embeddings, which provide more expressive representations.

- **Categorical features** ($x_1^{(cat)}, x_2^{(cat)}$, etc.): Each categorical feature is embedded using a separate embedding matrix ($W_1^{(cat)}, W_2^{(cat)}$, etc.) with corresponding bias terms. This is the standard approach for embedding categorical variables in deep learning.

The token embeddings produced by the Feature Tokenizer have a consistent dimensionality d, regardless of the original feature types. This allows them to be processed uniformly by the subsequent transformer layers.

3.3 Transformer Encoder

The transformer encoder component of the FT Transformer follows the original transformer architecture (Vaswani et al., 2017) with some adaptations for tabular data. As shown in Figure 2(b), each encoder layer consists of:

- **Multi-Head Self-Attention**: Allows the model to attend to different parts of the input sequence. In the context of tabular data, this means learning relationships between different features.
- **Layer Normalization**: Applied before the attention and feed-forward operations, improving training stability.
- **Feed-Forward Network**: A position-wise fully connected network applied to each position in the sequence independently.
- **Residual Connections**: Connecting the input of each sub-layer to its output, facilitating gradient flow during training.

The key innovation of the FT Transformer lies in its feature tokenization approach, which effectively captures both categorical and numerical feature relationships while enabling the transformer architecture to process tabular data efficiently. By representing each feature as a token, the model can leverage the powerful self-attention mechanism to capture complex interactions between features, potentially leading to more accurate predictions.

4 Embeddings for Numerical Features

4.1 What are Embeddings?

Embeddings are dense vector representations that capture semantic relationships between entities. Originally popularized in natural language processing for representing words (Mikolov et al., 2013), embeddings have been adapted for various data types, including tabular features. While embedding categorical features is straightforward (each category is mapped to a unique vector), embedding numerical features effectively is more challenging. Simple linear projections often fail to capture the complex patterns in numerical data, as shown in Figure 3.

4.2 Piecewise Linear Embeddings

Gorishniy et al. (2022) introduced piecewise linear embeddings (PLE) as a more expressive approach for numerical features. The method works by dividing the numerical range into bins and learning embeddings for each bin boundary. The embedding for a specific numerical value is then computed as a linear interpolation between the embeddings of the surrounding bin boundaries, as illustrated in Figure 4.

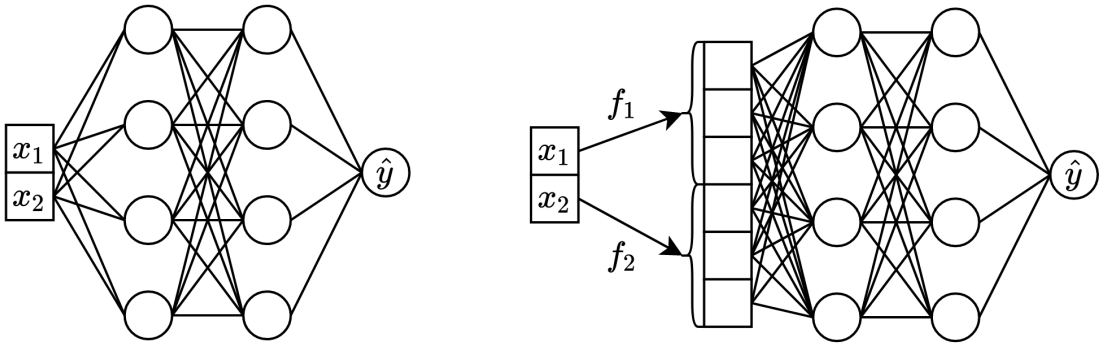

Figure 3: Comparison of neural network architectures: (Left) Standard MLP without feature transformation layers, directly connecting input features to hidden layers. (Right) MLP with feature embedding functions f_1 and f_2 to transform input features x_1 and x_2 before feeding them to the hidden layers. Figures reproduced from Gorishniy et al. (2022)

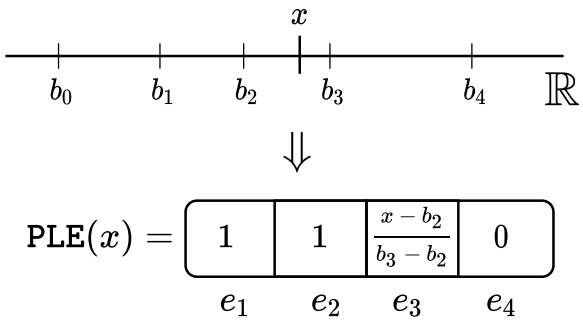

Figure 4: Piecewise Linear Embedding (PLE) representation. The real line is divided into bins with boundaries b_0, b_1, b_2, b_3, b_4. For a value x between bins b_2 and b_3, PLE is computed through linear interpolation, with the resulting one-hot-like encoding having non-zero values only at the relevant bin positions. Figure reproduced from Gorishniy et al. (2022)

For a numerical feature x with value v that falls between bin boundaries b_i and b_{i+1}, the piecewise linear embedding is computed as:

$$\text{PLE}(v) = (1 - \alpha) \cdot \mathbf{e}_i + \alpha \cdot \mathbf{e}_{i+1} \tag{7}$$

where $\mathbf{e}_i$ and $\mathbf{e}_{i+1}$ are the embeddings corresponding to boundaries b_i and b_{i+1}, and α is the interpolation coefficient determined by the position of v between b_i and b_{i+1}:

$$\alpha = \frac{v - b_i}{b_{i+1} - b_i} \tag{8}$$

As shown in Figure 4, this results in a sparse vector representation where most values are zero except for the entries corresponding to the surrounding bin boundaries.

4.2.1 Binning Strategies

The effectiveness of piecewise linear embeddings depends heavily on the binning strategy used to discretize the numerical features. Gorishniy et al. (2022) explored two main approaches:

Quantile-based bins: This approach divides the feature distribution into equal-frequency bins based on the training data. Specifically, the bin boundaries are set at the quantiles of the empirical distribution:

$$b_i = \text{Quantile}\left(\frac{i}{B}\right), \quad i = 0, 1, \ldots, B \tag{9}$$

where B is the number of bins, and the quantile function returns the value below which a given fraction of observations falls. This ensures a uniform distribution of data points across bins, which can help prevent issues with sparse bins in the tails of the distribution.

Target-aware bins: While quantile-based bins consider only the feature distribution, target-aware binning incorporates information about the relationship between features and the target variable. The goal is to create bins that maximize the correlation between bin indices and the target. One approach to target-aware binning is to use decision tree splits as bin boundaries, as illustrated in Figure 5. A shallow decision tree is trained to predict the target using the numerical feature, and the split points identified by the tree become the bin boundaries. This approach naturally prioritizes regions of the feature space where changes in the feature value correspond to significant changes in the target. Formally, a decision tree with B leaf nodes induces at most $B-1$ split points, which serve as the bin boundaries. These boundaries partition the feature space into regions that are maximally informative about the target variable according to the splitting criterion used (e.g., information gain, Gini impurity). The connection between decision trees and piecewise linear embeddings highlights an interesting relationship: PLEs can be viewed as a differentiable approximation of the feature transformations implicitly performed by tree-based models.

4.2.2 Periodic Activation Functions

In addition to piecewise linear embeddings, Gorishniy et al. (2022) introduced periodic activation functions to capture cyclical patterns in numerical features. Many real-world numerical features exhibit periodicity (e.g., time of day, day of week, month), and standard neural networks often struggle to efficiently learn such patterns. The periodic activation approach involves transforming the original feature x using sine and cosine functions with different frequencies:

$$\text{PeriodicActivation}(x) = [\sin(2\pi\omega_1 x), \cos(2\pi\omega_1 x), \sin(2\pi\omega_2 x), \cos(2\pi\omega_2 x), \ldots] \tag{10}$$

where $\omega_1, \omega_2, \ldots$ are different frequency parameters. In practice, a geometric progression of frequencies is often used:

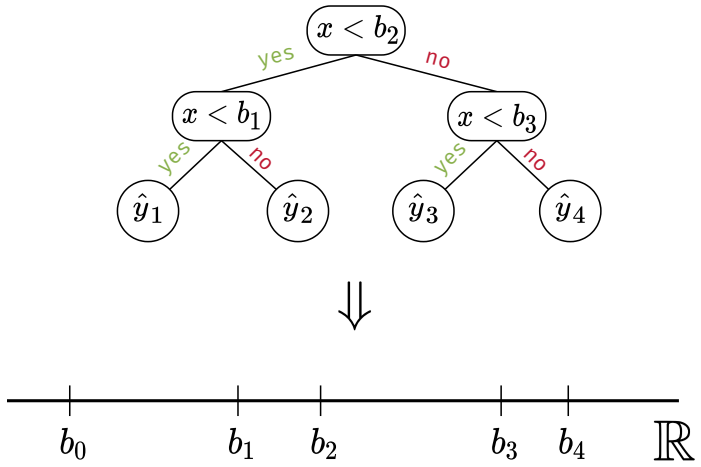

Figure 5: Connection between decision trees and binning strategies. Top: A decision tree partitioning the feature space. Bottom: The equivalent binning representation showing how tree splits correspond to bin boundaries on the real line. Figure by Gorishniy et al. (2022)

$$\omega_j = \omega_0 \cdot r^j, \quad j = 0, 1, 2, \ldots \tag{11}$$

where ω_0 is the base frequency and $r > 1$ is the ratio between consecutive frequencies. This multi-frequency approach enables the model to capture patterns at different scales simultaneously. The periodic activations can be applied either to the raw numerical features or to the interpolation coefficients (α) in the piecewise linear embedding, resulting in:

$$\text{PeriodicPLE}(v) = (1 - \text{PeriodicActivation}(\alpha)) \cdot \mathbf{e}i + \text{PeriodicActivation}(\alpha) \cdot \mathbf{e}i + 1 \tag{12}$$

This combination of piecewise linear embeddings with periodic activations provides a powerful and flexible way to represent numerical features, capturing both the general value range (through binning) and cyclical patterns (through periodic activations).

4.3 Benefits of Advanced Embedding Techniques

The advanced embedding techniques described above offer several advantages over simple linear projections:

1. **Non-linear feature transformations**: PLEs enable the model to learn non-linear transformations of numerical features without requiring deep networks.

2. **Interpretability**: The bin boundaries provide insights into how the model partitions the feature space, similar to decision tree splits.

3. **Robustness to outliers**: By focusing on the relative position within bins rather than absolute values, PLEs can be more robust to outliers and unusual feature distributions.

4. **Efficient representation of periodic patterns**: Periodic activations enable compact representation of cyclical patterns that would otherwise require many parameters in standard neural networks.

5. **Enhanced feature interactions**: These embeddings provide the transformer's self-attention mechanism with more informative feature representations, potentially leading to better modeling of feature interactions.

Experimental results from Gorishniy et al. (2022) demonstrated that these embedding techniques significantly boost the performance of deep learning models on tabular data, narrowing or even closing the gap with gradient boosted decision trees on many benchmark datasets.

5 Sparsemax

5.1 From Softmax to Sparsemax

The softmax function has been the standard choice for attention mechanisms in transformers, converting raw attention scores into a probability distribution:

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{13}$$

However, softmax always assigns non-zero probability to all elements, which can lead to diffuse attention that is difficult to interpret. Even irrelevant features receive some attention weight, obscuring the model's true focus.

Martins and Astudillo (2016) introduced sparsemax as an alternative to softmax. Sparsemax maps input vectors to sparse probability distributions, frequently placing exact zeros on irrelevant elements:

$$\text{sparsemax}(z) = \arg\min_{p \in \Delta^{K-1}} ||p - z||^2 \tag{14}$$

where Δ^{K-1} is the (K-1)-dimensional simplex, and the solution can be computed efficiently through a projection operation.

This property of sparsemax makes it particularly attractive for attention mechanisms where we want to identify a small subset of truly important features. By producing sparse attention distributions, sparsemax potentially provides more interpretable feature importance scores.

5.2 Visual Comparison

To better understand the differences between softmax and sparsemax, we provide visual comparisons in different dimensions, as shown in Figure 6.

These visualizations highlight key differences between the two functions. Softmax never assigns exactly zero probability to any element, resulting in smooth, curved surfaces. In

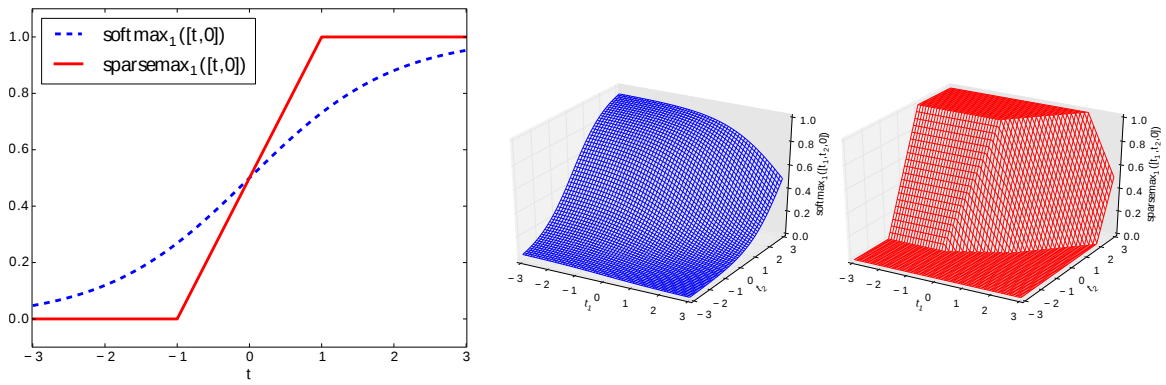

Figure 6: Visual comparison of softmax and sparsemax in 2D (**Left**) and 3D (**Right**) keeping one dimension constant. Figures reproduced from Martins and Astudillo (2016).

contrast, sparsemax produces sparse outputs with exact zeros for less relevant elements, creating distinct flat regions in its response surface. For attention mechanisms, this means sparsemax can completely ignore irrelevant features, potentially improving interpretability.

5.3 Practical Example with Feature Attention

To illustrate how these transformations apply to attention scores in practice, consider a simple 3×3 matrix of raw attention scores: Figure 7 demonstrates the key difference between these two transformations: softmax spreads attention across all elements, producing non-zero values throughout the matrix. In contrast, sparsemax concentrates attention only on the most important elements, setting the less relevant elements to exactly zero.

This sparsity property makes sparsemax particularly valuable for interpreting attention patterns in transformer models for tabular data. When using sparsemax, the attention weights directly highlight the most important features without the noise of small but non-zero weights on irrelevant features, providing clearer feature importance signals.

5.4 Sparsemax in Tabular Models

TabNet (Arik and Pfister, 2021) was one of the first tabular deep learning models to incorporate sparsemax for feature selection. It uses a sequential attention mechanism with sparsemax to select relevant features at each decision step, mimicking the feature selection process of decision trees. This design choice was motivated by the need for both interpretability and performance.

While TabNet showed promising results when introduced, more recent models like FT Transformer have demonstrated superior performance on many benchmarks. However, these newer models typically use the standard softmax attention, potentially sacrificing some interpretability. Our work explores whether incorporating sparsemax into these more powerful architectures can combine the strengths of both approaches.

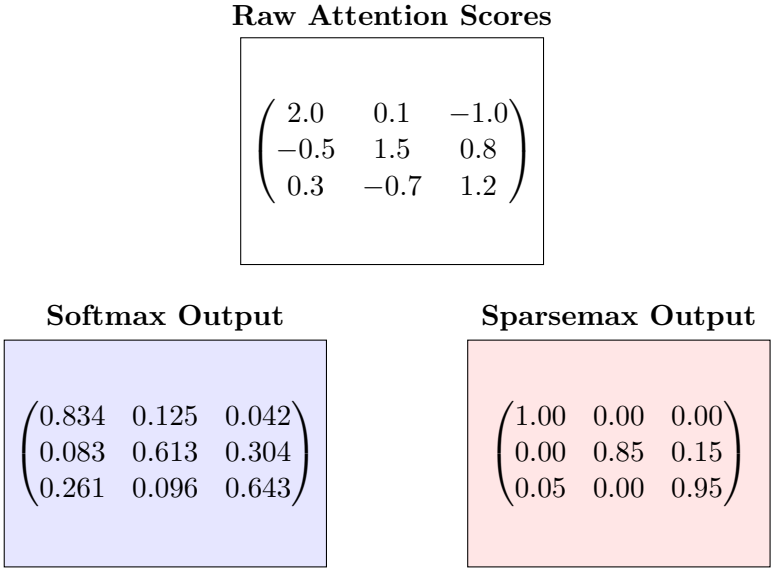

Figure 7: Comparison of softmax and sparsemax transformations applied instance- (row) wise to a matrix of attention scores. Due to rounding, entries might not sum to exactly 1 along rows.

6 Experimental Setup

6.1 Datasets

We evaluate our models on five diverse datasets covering different application domains and prediction tasks:

1. **California Housing (CA)**: Real estate data (Pace and Barry, 1997).
2. **Adult (AD)**: Income estimation dataset (Kohavi et al., 1996).
3. **Helena (HE)**: Anonymized dataset from the AutoML challenge (Guyon et al., 2019).
4. **Jannis (JA)**: Anonymized dataset from the AutoML challenge (Guyon et al., 2019).
5. **Higgs (HI)**: Simulated physical particles dataset (Baldi et al., 2014); we use the version with 98K samples available at the OpenML repository (Vanschoren et al., 2014).

The dataset properties are summarized in Table 1.

These datasets were chosen to represent a variety of challenges in tabular data modeling, including different sample sizes, feature types, and prediction tasks.

6.2 Feature Preprocessing

We applied minimal preprocessing to ensure a fair comparison: Numerical features are standardized (zero mean, unit variance), categorical features one-hot encoded. Missing values,

Table 1: Dataset Characteristics

	CA	HI	JA	HE	AD
#objects	20,640	98,050	83,733	65,196	48,842
#num. features	8	28	54	27	6
#cat. features	0	0	0	0	8
metric	RMSE	Acc.	Acc.	Acc.	Acc.
#classes	–	2	4	100	2

if present in the data, are imputed using a median imputation strategy. No feature engineering or selection was performed beyond this base preprocessing, allowing us to evaluate the models' ability to learn relevant patterns directly from the data.

7 Experiments

7.1 Models

We trained and evaluated four variants of the FT Transformer:

1. **FT-Linear**: FT Transformer with standard linear embeddings for numerical features and softmax attention
2. **FT-Sparse-Linear**: FT Transformer with linear embeddings and sparsemax attention
3. **FT-PLE**: FT Transformer with piecewise linear embeddings and softmax attention
4. **FT-Sparse-PLE**: FT Transformer with piecewise linear embeddings and sparsemax attention

All models share the same base architecture, differing only in their embedding approach (linear vs. piecewise linear) and attention mechanism (softmax vs. sparsemax).

7.2 Feature Importance from Attention Weights

A key advantage of attention-based models is the potential to derive feature importance directly from attention weights. We compute feature importance from attention as follows:

1. Extract attention weights from all heads and layers
2. For each feature, aggregate attention weights targeting that feature
3. Normalize the aggregated weights to sum to 1

For models with piecewise linear embeddings, the process is more complex as each numerical feature is represented by multiple bin embeddings. We aggregate attention weights across all bins belonging to the same feature, as shown in the pseudocode:

This approach allows us to obtain feature importance scores that can be compared with Permutation Feature Importance (PFI) to assess their reliability as interpretability measures.

Algorithm 2 Attention-based Feature Importance for PLE

Input: Attention weights A, feature-to-bin mapping F
Output: Feature importance scores
Initialize importance scores for each feature: $I = \text{zeros}(\text{num\_features})$
for each attention head h and layer l **do**
  for each feature f **do**
    Identify all bins b that belong to feature f
    Sum attention weights for these bins: $I[f] += \sum(A[h, l, :, b])$
  end for
end for
Normalize importance scores: $I = I/\text{sum}(I)$
return I

7.3 Evaluation Metrics

To evaluate our models, we use the following metrics:

1. **Performance metrics**:
   - For classification: Accuracy, F1 score (macro and weighted where appropriate)
   - For regression: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R^2 score, Mean Absolute Error (MAE)
2. **Interpretability metrics**:
   - Spearman rank correlation between attention-based feature importance and PFI
   - Comparison of top-k important features identified by each method

7.4 Training Protocol

All models were trained using the following protocol:

1. Initial hyperparameter tuning using Optuna (Akiba et al. (2019)) with 20 epochs per trial
2. Final training for up to 100 epochs with early stopping based on validation performance
3. Evaluation on a held-out test set

Hyperparameters were tuned separately for each model variant and dataset to ensure fair comparison. Please refer to Appendix C for the full tuning space.

8 Results

8.1 Correlation between Attention Weights and PFI

The primary goal of our study was to determine whether sparsemax attention provides more reliable feature importance scores compared to softmax. Table 2 shows the Spearman

rank correlation between attention-based feature importance and PFI for each model and dataset:

Table 2: Spearman Rank Correlation between Attention Map of the CLS Token and PFI

Model	California	Higgs	Jannis	Helena	Adult
FT-Linear	0.881	0.752	0.812	0.918	0.266
FT-Sparse-Linear	0.738	0.849	0.719	0.683	0.767
FT-PLE	0.857	0.699	0.807	0.829	0.758
FT-Sparse-PLE	0.429	0.793	0.698	0.464	0.763

These results reveal several interesting patterns:

1. For the Adult dataset, which contains both numerical and categorical features, sparsemax models show substantially higher correlation with PFI compared to the standard softmax model.

2. For datasets with complex numerical relationships (Higgs), sparsemax also improves correlation.

3. For some datasets (California, Helena), the sparsemax variants show lower correlation, suggesting that the relationship between attention weights and feature importance may be dataset-dependent.

4. In many cases, FT-PLE with standard softmax attention achieves high correlation, indicating that improved numerical representations through piecewise linear embeddings may enhance the interpretability of attention weights.

8.2 Model Performance

While our primary focus is interpretability, it's crucial to ensure that any modifications for interpretability don't significantly degrade model performance. Table 3 and summarizes the performance metrics for all model variants across the five datasets:

Table 3: Performance Metrics Across Datasets

Model	California (RMSE↓)	Higgs (Acc.↑)	Jannis (Acc.↑)	Helena (Acc.↑)	Adult (Acc.↑)
FT-Linear	0.489	0.724	0.715	0.375	0.871
FT-Sparse-Linear	0.491	0.723	0.718	0.357	0.867
FT-PLE	0.521	0.696	0.694	0.370	0.868
FT-Sparse-PLE	0.506	0.692	0.697	0.372	0.866

The performance results indicate that:

1. The sparsemax variants maintain competitive performance compared to the softmax counterparts, with minimal degradation in most cases.

2. For the Jannis dataset, the sparse linear model actually outperforms the standard linear model slightly.

3. Across most datasets, the linear embedding models outperform their piecewise linear counterparts in terms of raw performance. This is most likely due to the restricted tuning space of the number of bins used in PLE.

8.3 Correlation Analysis of Feature Importance Rankings

To provide a more comprehensive assessment of feature importance consistency across model variants, we computed the Spearman rank correlation between the feature importance rankings produced by different models on each dataset. Figure 8 presents these correlations as heatmaps, offering insights into how different embedding and attention mechanisms affect feature importance patterns.

These correlation heatmaps reveal several interesting patterns:

1. **High Internal Consistency:** For the California Housing dataset, we observe remarkably high correlations (0.95-1.00) between all model variants, suggesting that feature importance rankings remain highly consistent regardless of the embedding technique or attention mechanism used. This indicates that for this dataset, the model architecture choices have minimal impact on which features are identified as important.

2. **Moderate Variation in Mixed-Feature Datasets:** For the Adult dataset, which contains both numerical and categorical features, we see slightly lower but still substantial correlations (0.85-0.92). This suggests that different model variants maintain reasonable agreement on feature importance, though with some variations likely attributable to how they handle the mixed feature types.

3. **Greater Divergence in Complex Datasets:** The Helena dataset shows the most variation in feature importance correlations, with values ranging from 0.68 to 0.90. Notably, FT-Sparse-Linear shows the lowest correlation with other variants, indicating that for this complex dataset (100 classes), the choice of attention mechanism and embedding technique significantly influences which features are deemed important.

4. **Embedding Technique Impact:** Across multiple datasets (particularly Jannis and Helena), models sharing the same embedding technique (either both linear or both PLE) tend to have higher correlations with each other than with models using different embedding techniques. This suggests that the choice of embedding approach may have a stronger influence on feature importance patterns than the choice of attention mechanism.

5. **Attention Mechanism Effects:** When comparing models with the same embedding technique but different attention mechanisms (e.g., FT-Linear vs. FT-Sparse-Linear), we generally observe correlations above 0.80, indicating that while sparsemax does alter feature importance rankings, it maintains substantial agreement with softmax-based models.

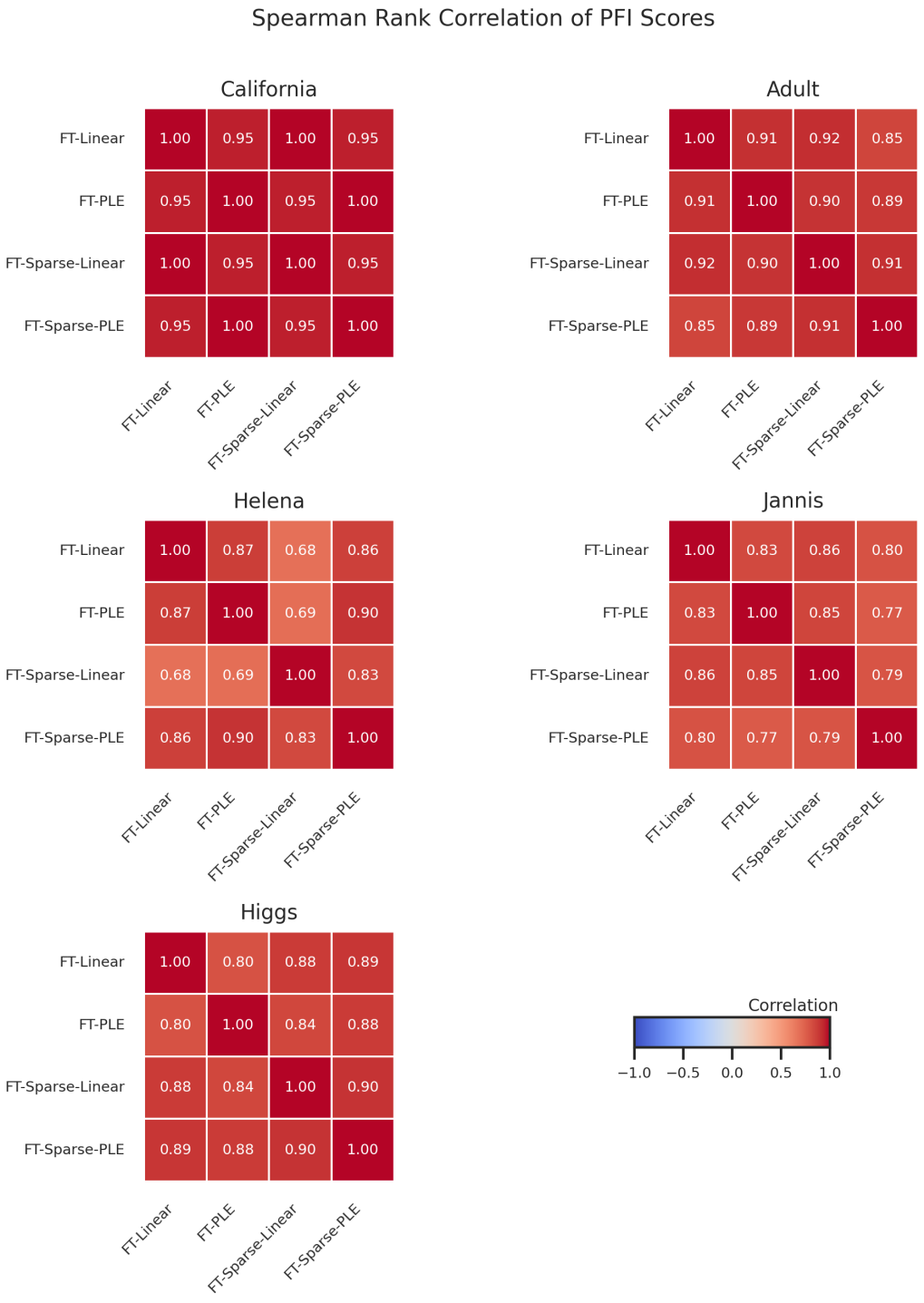

Figure 8: Spearman rank correlation of feature importance rankings between different model variants across the five datasets.

These findings suggest that while sparsemax attention and piecewise linear embeddings do influence which features are deemed important, there remains significant consistency in feature importance patterns across model variants. This consistency provides confidence that the feature importance rankings derived from these models are capturing genuine relationships in the data rather than artifacts of particular architectural choices.

The moderate to high correlations across model variants also validate our approach of using attention weights as indicators of feature importance, as they demonstrate that these weights capture stable patterns of feature relevance regardless of specific implementation details.

9 Limitations

While our study provides valuable insights into the use of sparsemax for interpretable tabular deep learning, several limitations should be acknowledged:

9.1 Tuning Regime

Due to computational constraints, we used different tuning regimes for the linear and piecewise linear embedding models. This makes direct performance comparisons between these groups less reliable. Specifically:

- For piecewise linear embeddings, a higher number of bins would likely improve performance but is computationally expensive.
- The quadratic complexity of the multi-head self-attention layer poses challenges when using piecewise linear embeddings, as the dataset with n instances, f features and b bins is transformed to shape n $\times$ (f $\times$ b) before parsing into the MHSA layer.

This limitation could be addressed with more computational resources or by exploring transformer variants with lower computational complexity.

9.2 Dataset Size and Diversity

Our experiments were carried out on five datasets that, although diverse, may not be sufficient to draw definitive conclusions about the generalizability of our findings. With proper resources, a more comprehensive evaluation on a larger benchmark or synthetic datasets with known ground truth feature importance would provide stronger evidence.

10 Outlook

Based on our findings and the limitations identified, several promising directions for future research emerge.

10.1 Computational Efficiency

Exploring integration of more efficient transformer architectures like Fastformer (Wu et al., 2021) or Performer (Choromanski et al., 2020) could address the computational challenges associated with piecewise linear embeddings. These architectures approximate attention

with lower computational complexity, potentially enabling the use of more bins for piecewise linear embeddings.

10.2 Advanced Embedding Techniques

Further investigation into how target-aware bins and periodic activation functions influence feature importance could yield additional insights. These techniques have shown performance benefits (Gorishniy et al., 2022), but their impact on interpretability remains unexplored.

10.3 Pre-trained Models

Examining whether the interpretability benefits observed in this study extend to more advanced solutions such as XTab (Zhu et al., 2023), where FT Transformer is pre-trained on a large corpus of tabular data, could be valuable. Pre-training might affect the attention patterns and, consequently, the reliability of attention-based feature importance.

10.4 Hybrid Approaches

Developing hybrid approaches that combine the strengths of sparsemax and piecewise linear embeddings while mitigating their limitations could lead to models that are both highly performant and interpretable. This might involve adaptive mechanisms that switch between softmax and sparsemax based on feature characteristics or task requirements.

11 Conclusion

In this paper, we investigated the impact of replacing softmax with sparsemax in the attention mechanism of Feature Tokenizer Transformers for tabular data. Our goal was to determine whether this change enhances interpretability, as measured by the correlation between attention-based feature importance and Permutation Feature Importance (PFI).

Our experiments across five diverse datasets revealed that sparsemax can indeed improve the alignment between attention weights and PFI, particularly for datasets with mixed feature types (numerical and categorical) or complex numerical relationships. This improvement comes with minimal performance degradation, suggesting that sparsemax is a viable alternative to softmax for applications where interpretability is crucial.

We also observed that piecewise linear embeddings for numerical features generally enhance the correlation between attention weights and PFI when using standard softmax attention. However, the combination of sparsemax and piecewise linear embeddings showed more varied results, highlighting the complexity of the relationship between embedding techniques, attention mechanisms, and interpretability.

These findings contribute to the ongoing effort to develop deep learning models for tabular data that are not only powerful but also interpretable. As tabular data continues to dominate in critical applications like healthcare and finance, such interpretable models will be essential for responsible deployment of machine learning systems.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):4308, 2014.

Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pages 4171–4186, 2019.

Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81, 2019.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in neural information processing systems*, 34: 18932–18943, 2021.

Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. *Advances in Neural Information Processing Systems*, 35:24991–25004, 2022.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, et al. Analysis of the automl challenge series. *Automated Machine Learning*, 177:177–219, 2019.

Sarthak Jain and Byron C Wallace. Attention is not explanation. *arXiv preprint arXiv:1902.10186*, 2019.

Ron Kohavi et al. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, volume 96, pages 202–207, 1996.

Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pages 1614–1623. PMLR, 2016.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Chuhan Wu, Fangzhao Wu, Tao Qi, Yongfeng Huang, and Xing Xie. Fastformer: Additive attention can be all you need. *arXiv preprint arXiv:2108.09084*, 2021.

Bingzhao Zhu, Xingjian Shi, Nick Erickson, Mu Li, George Karypis, and Mahsa Shoaran. XTab: Cross-table pretraining for tabular transformers. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 43181–43204. PMLR, 23–29 Jul 2023.

Appendix A. Data Splits

All datasets were split into training (70%), validation (15%), and test (15%) sets, stratified by target for classification tasks. Table 4 provides details on the exact split sizes:

Table 4: Dataset Split Sizes

Dataset	Training	Validation	Test	Total
California	14,448	3,096	3,096	20,640
Higgs	68,635	14,707	14,708	98,050
Jannis	58,613	12,560	12,560	83,733
Helena	45,637	9,779	9,780	65,196
Adult	31,258	7,815	9,769	48,842

Appendix B. Hardware and Implementation

All experiments were performed on a Nvidia T4 GPU with 16GB of vRAM on Google Colab. The implementation was based on the PyTorch framework, with the FT Transformer and PLE code adapted from the original implementation by Gorishniy et al. (2021) and Gorishniy et al. (2022), respectively.

Appendix C. Hyperparameter Tuning

Hyperparameter tuning was performed using Optuna, with 20 trials per model variant and dataset, but was limited to a time budget of 30 min. Each trial involved training for 20 epochs, and the best hyperparameter configuration was used for the final training (100 epochs with early stopping).

Table 5: Hyperparameter Search Space

Hyperparameter	**Search Space**
Token dimension (d_token)	[32, 128]
Number of heads (num_heads)	[2, 8]
Number of layers (num_layers)	[1, 3]
Feed-forward dimension (d_ffn)	[64, 256]
Learning rate (lr)	[1e-4, 1e-2] (log scale)
Dropout rate ($dropout$)	[0.0, 0.5]
Number of bins in PLE (n_bins)	Dataset specific

Appendix D. Batch Sizes and Number of Bins

Depending on the size of the dataset, different batch sizes were used in training and inference.

Table 6: Dataset-Specific Parameters

Dataset	**Bin Range**	**Batch Size Training**	**Batch Size Val / Test**
Higgs (HI)	[2, 10]	512	1024
Helena (HE)	[2, 50]	512	8192
Jannis (JA)	[2, 10]	512	8192
Adult (AD)	[10, 100]	64	128
California Housing (CA)	[10, 100]	64	128

The number of bins was chosen so that the model could still be tuned in a meaningful way given the available computational resources. For better performance, Gorishniy et al. (2022) recommends tuning the number of bins up to a size of 256.

Appendix E. Additional performance results

Below we present detailed performance metrics for each dataset and model configuration. Table 7 shows regression metrics for the California Housing dataset, while Tables 8 and 9 present classification metrics for binary and multi-class tasks respectively.

Table 7: Regression Metrics for California Housing Dataset

Model	MSE↓	RMSE↓	R^2↑	MAE↓
FT-Linear	0.239	0.489	0.820	0.345
FT-Sparse-Linear	0.241	0.491	0.818	0.339
FT-PLE	0.271	0.521	0.795	0.358
FT-Sparse-PLE	0.256	0.506	0.806	0.354

As seen in the results, models with linear embeddings generally achieved lower error rates on regression tasks compared to their piecewise linear counterparts. This is very likely to be the result of the restrictive tuning space for the number of bins.

Table 8: Binary Classification Metrics

Model	Adult			Higgs		
	Precision↑	Recall↑	AUC↑	Precision↑	Recall↑	AUC↑
FT-Linear	0.757	0.659	0.922	0.748	0.720	0.801
FT-Sparse-Linear	0.751	0.645	0.920	0.736	0.742	0.803
FT-PLE	0.732	0.691	0.923	0.720	0.693	0.767
FT-Sparse-PLE	0.772	0.606	0.917	0.705	0.716	0.763

For binary classification tasks, we observe that sparsemax variants maintain competitive performance across precision, recall, and AUC metrics.

Table 9: Multi-class Classification Metrics

Model	Jannis		Helena	
	F1 Macro↑	F1 Weighted↑	F1 Macro↑	F1 Weighted↑
FT-Linear	0.551	0.343	0.221	0.343
FT-Sparse-Linear	0.569	0.345	0.220	0.335
FT-PLE	0.546	0.340	0.219	0.340
FT-Sparse-PLE	0.530	0.349	0.230	0.349

Appendix F. Feature Importance plots

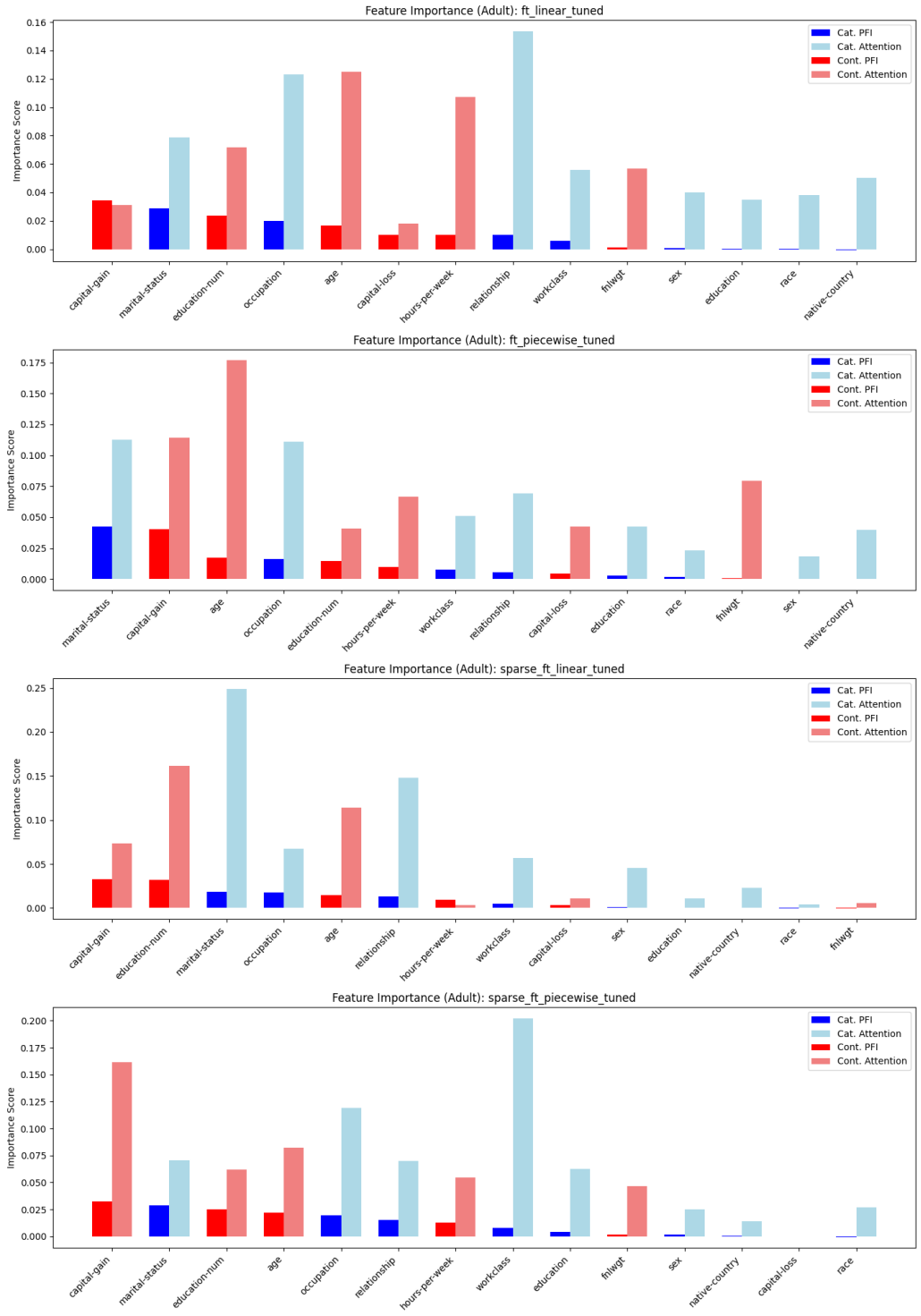

Figure 9: Feature Importance for the Adult dataset for all models.

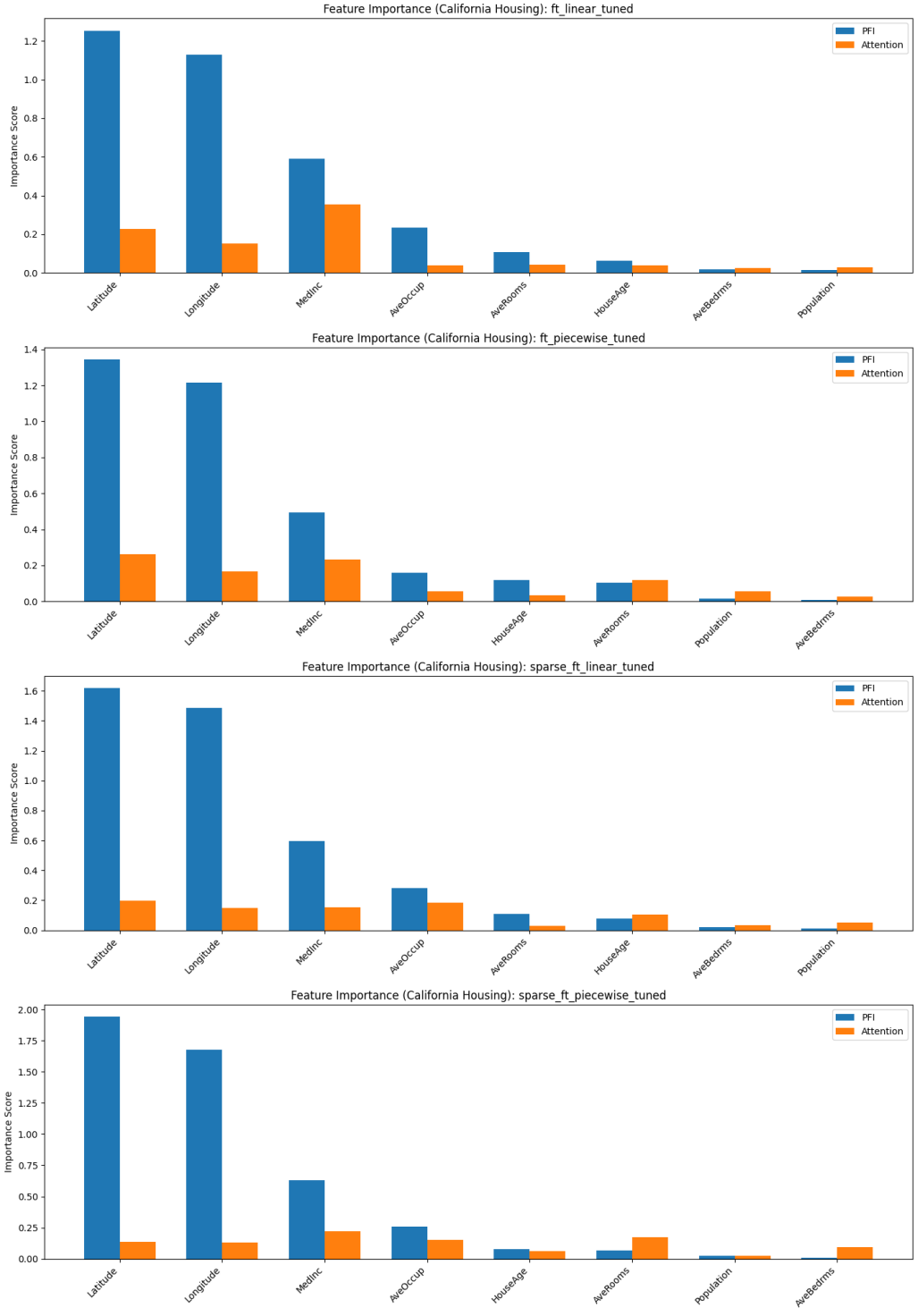

Figure 10: Feature Importance for the California Housing dataset for all models.

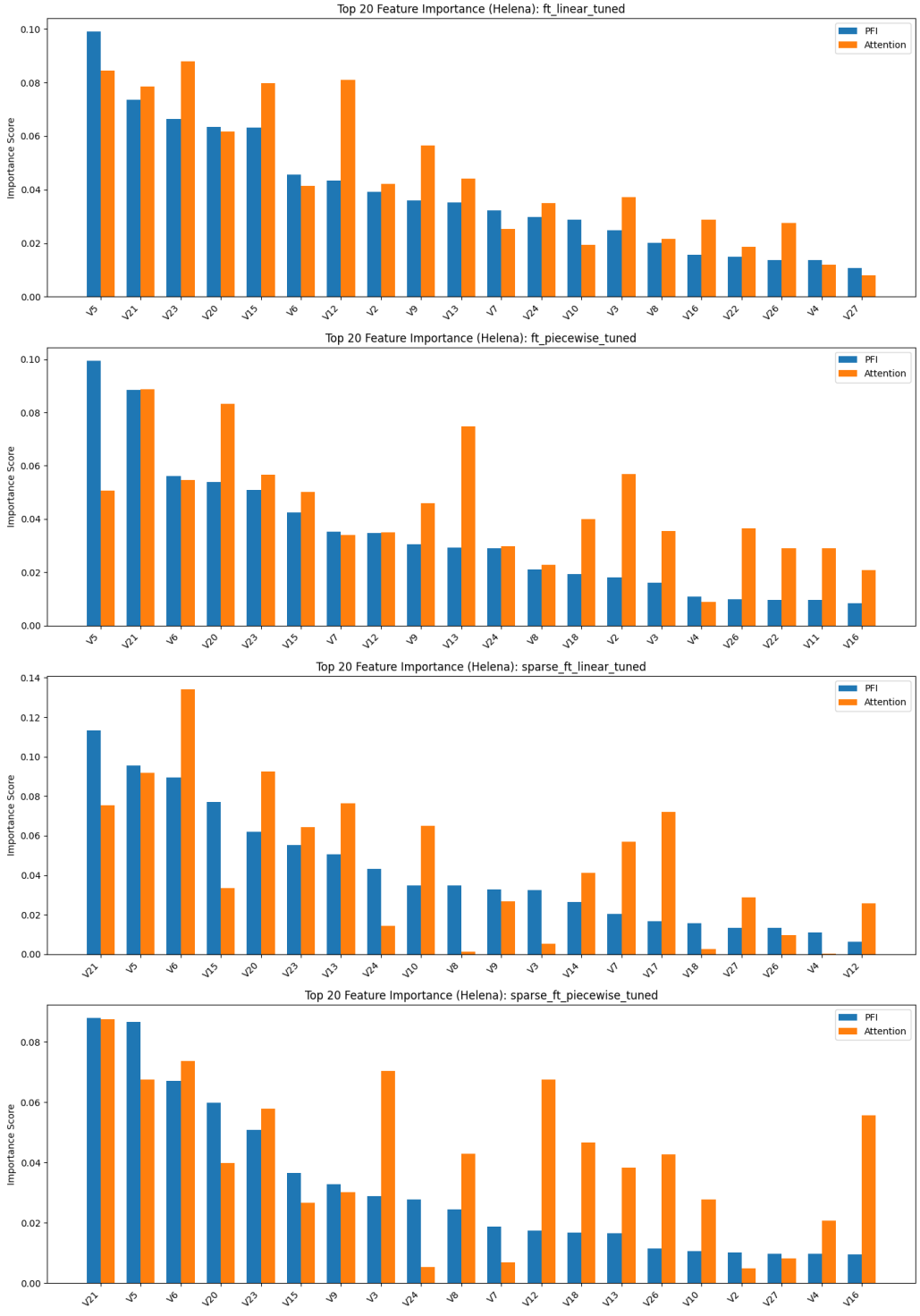

Figure 11: Feature Importance for the Helena dataset for all models.

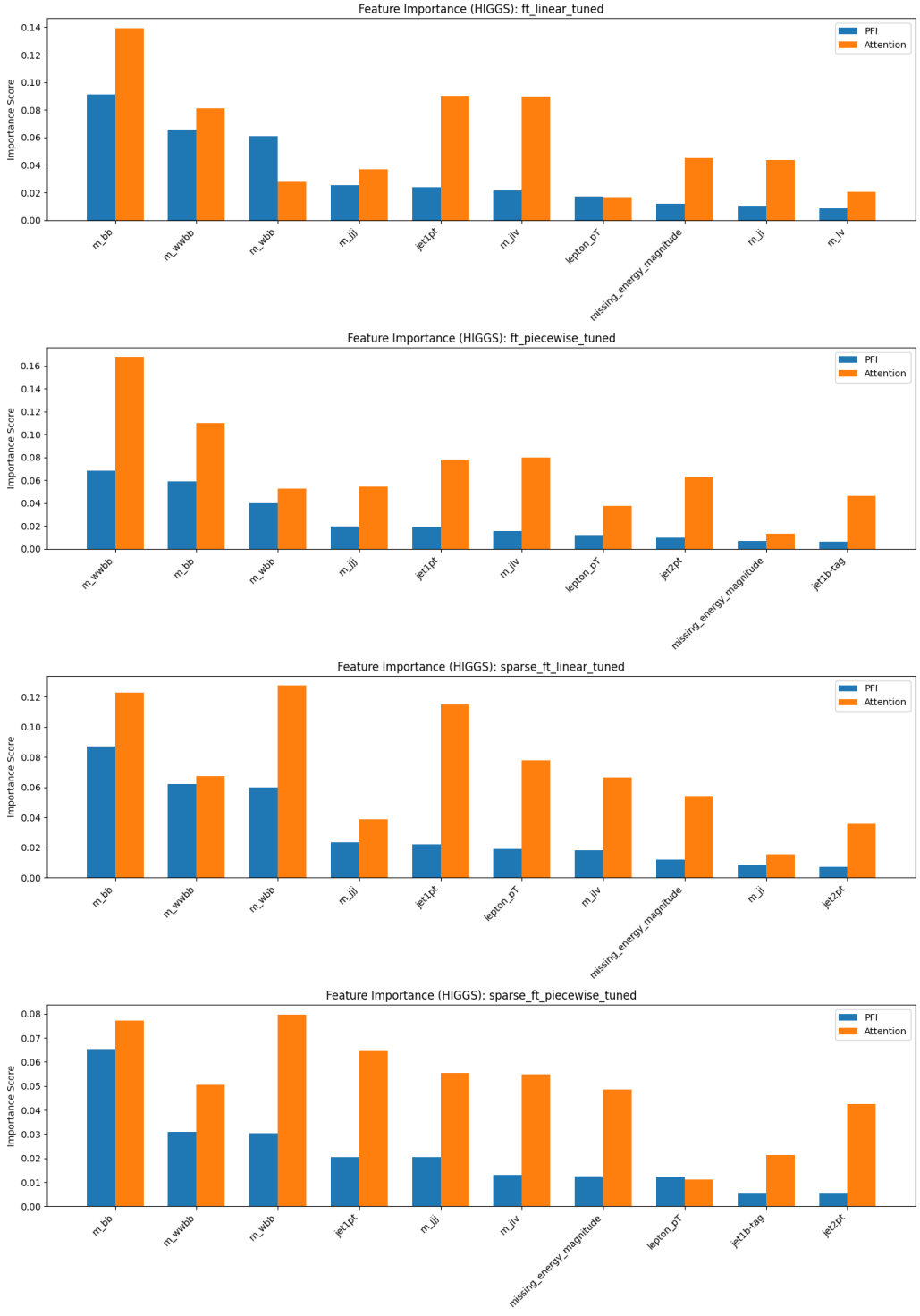

Figure 12: Feature Importance for the Higgs dataset for all models.

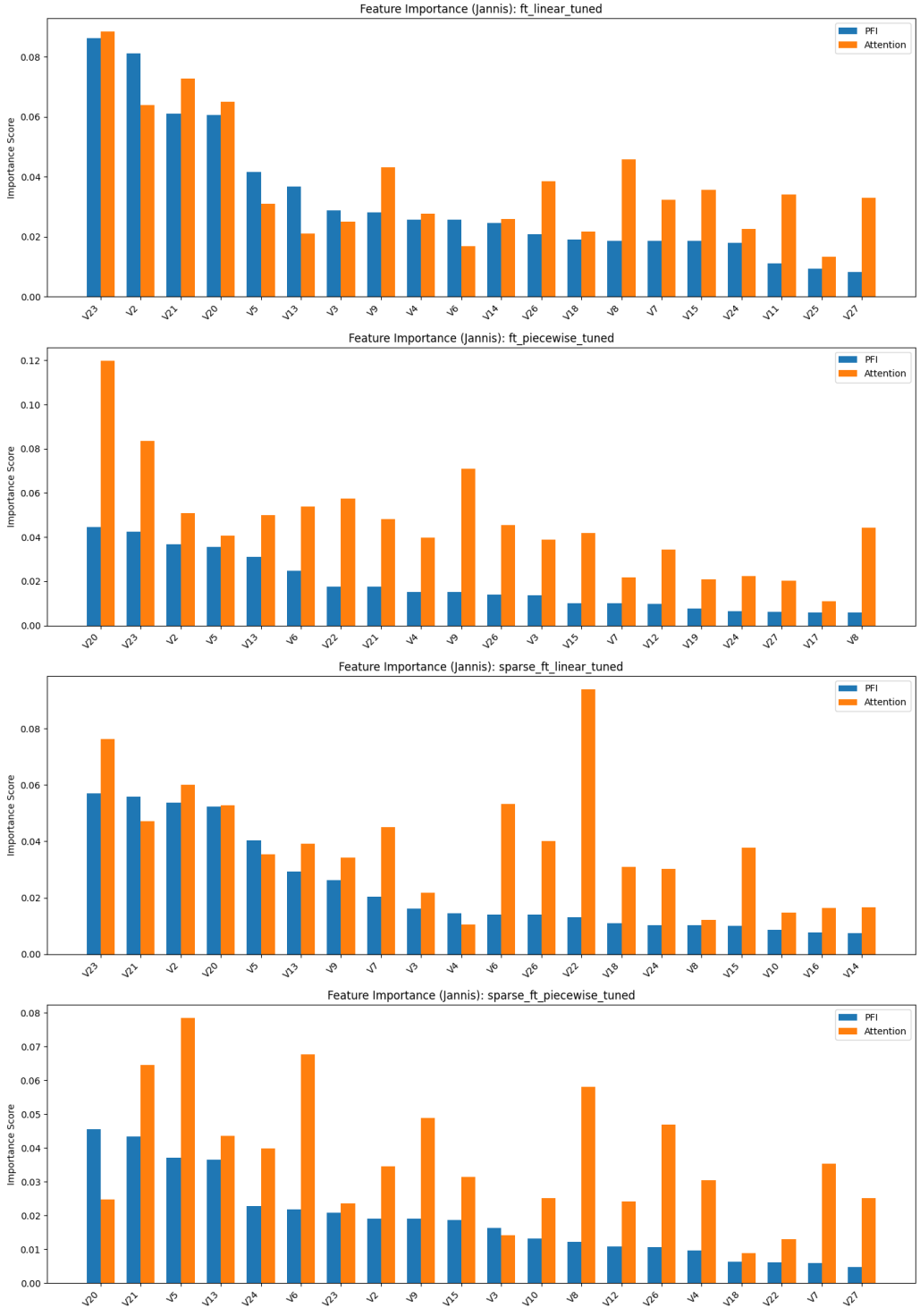

Figure 13: Feature Importance for the Jannis dataset for all models.