[ICML 2026 Spotlight] TabSwift: An Efficient Tabular Foundation Model with Row-Wise Attention

---

Presenter Kyungseon Lee

2026-06

Seoul National University

Background — Tabular Foundation Model

Tabular prediction.
Train dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, test dataset $\boldsymbol{x}_q \in \mathbb{R}^d$, y_i is a class label or a real value.

Tabular foundation model, TFM.
A Transformer-based model f_θ is pretrained once on synthetic tasks and then reused for every dataset. Prediction is done by **in-context learning**. Let $\boldsymbol{x}_q$ be the *test data (query)* and $\hat{y}_q$ the *estimated label*.

$$\hat{y}_q = f_\theta(\mathcal{D}, \boldsymbol{x}_q)\,.$$

No gradient update at test time. The most well-known model is **TabPFN**.

Problem.
Recent TFMs use heavier and heavier architectures to gain accuracy.
Goal: improve the old lightweight TFM and reach high performance.

Background — TabPFN v1: Row-wise Attention

One row = one token.
N train rows and T test rows, so $n = N + T$ tokens in total.

1. **Unify the input dimension.** Each dataset has a different number of features F, so we map all of them to a fixed size F_{max}, using zero-padding or PCA. $\tilde{\boldsymbol{x}} \in \mathbb{R}^n \times \mathbb{R}^{F_{\text{max}}}$.
2. **Embedding.** Train and test rows are each embedded into the model dimension d_{model}. $\tilde{\boldsymbol{x}} \in \mathbb{R}^n \times \mathbb{R}^{d_{\text{model}}}$.
3. **Inference (regression case)** $\hat{y} = f_{\text{TabPFNv1}}(\tilde{\boldsymbol{x}}) \in \mathbb{R}^n$

Computational cost. $\mathcal{O}(n^2 d_{\text{model}})$ per attention layer.

Background — TabPFN v2: Row & Column Attention

The difference from TabPFN version 1 is that **one cell = one token**. That is, the input is treated as an $n \times d$ grid of tokens.

❶ **Turn each scalar cell into a vector token.** With learnable weights $\boldsymbol{w}_{\text{val}}, \boldsymbol{b}_{\text{val}}, \boldsymbol{c}_j \in \mathbb{R}^{d_{\text{model}}}$,

$$\boldsymbol{h}_{ij}^{(0)} = x_{ij}\,\boldsymbol{w}_{\text{val}} + \boldsymbol{b}_{\text{val}} + \boldsymbol{c}_j \in \mathbb{R}^{d_{\text{model}}}, \qquad x_{ij} \in \mathbb{R}$$

❷ **Attention in two directions.** Arranging $\boldsymbol{h}_{ij}^{(0)}$ into an $n \times d$ grid gives $\boldsymbol{H} \in \mathbb{R}^{n \times d \times d_{\text{model}}}$. Let $\boldsymbol{H}_{:,j} \in \mathbb{R}^{n \times d_{\text{model}}}$ be the column tokens of feature j, and $\boldsymbol{H}_{i,:} \in \mathbb{R}^{d \times d_{\text{model}}}$ the row tokens of instance i. Let Attn be self-attention over a token sequence, and assume a single attention layer.

row attention: $\boldsymbol{H}_{:,j} \leftarrow \mathrm{Attn}(\boldsymbol{H}_{:,j}) \;\; \forall j,$ column: $\boldsymbol{H}_{i,:} \leftarrow \mathrm{Attn}(\boldsymbol{H}_{i,:}) \;\; \forall i$

The cost is $\mathcal{O}(dn^2 d_{\text{model}}) + \mathcal{O}(nd^2 d_{\text{model}})$, more than d times that of v1.

TabSwift — Key Question

Can the lightweight row-wise backbone of v1 reach v2-level performance if we add modern techniques?

The authors add three things to the original TabPFN v1.

- Add missing model capacity - Learnable register tokens.
  - A technique proven in ViT. K learnable global tokens are appended to the input and used as memory slots.
- Stabilize the attention update - Gated attention.
  - A technique proven in LLMs. Multiply the attention output by an element-wise learnable gate.
- Make it more efficient - Adaptive early exit.
  - Easy test data finish their prediction at a shallow layer and stop computing. The early-exit decision is made token by token, not for the whole test set at once.

TabSwift — Register Tokens

Input. After preprocessing steps 1 and 2 from v1, we start from $\tilde{\boldsymbol{x}} \in \mathbb{R}^{n \times d_{\text{model}}}$.

Register tokens. K learnable tokens $\boldsymbol{R}^{(0)} \in \mathbb{R}^{K \times d_{\text{model}}}$ are placed in front of the row tokens:

$$\boldsymbol{X} = \left[\boldsymbol{R}^{(0)};\ \boldsymbol{H}^{(0)}_{\text{rows}}\right] \in \mathbb{R}^{S \times d_{\text{model}}}, \qquad S = K + n.$$

- Learned independently of the data and shared across all tasks. They pass through every layer and get updated together.
- Role: a global memory that gathers information about the whole dataset through attention.
- Since $K \ll n$, the extra cost is negligible.

TabSwift — Element-wise Gated Attention

Layer input $\boldsymbol{X} \in \mathbb{R}^{S \times d_{\text{model}}}$.
Assume the number of heads in multi-head attention is fixed to one.

Attention output. $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{\times d_{\text{model}}}$:

$$\boldsymbol{O} = \text{softmax}\left(\frac{\boldsymbol{Q}(\boldsymbol{K})^\top}{\sqrt{d_{\text{model}}}}\right)\boldsymbol{V} \in \mathbb{R}^{S \times d_{\textbf{model}}}, \qquad \boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q$$

Gating. Build a gate from the input and multiply it into the output element-wise. $\boldsymbol{W}_g \in \mathbb{R}^{d_{\text{model}} \times d_{\textbf{model}}}$, $\boldsymbol{b}_g \in \mathbb{R}^{d_{\textbf{model}}}$, σ is the sigmoid:

$$\boldsymbol{G} = \sigma(\boldsymbol{X}\boldsymbol{W}_g + \boldsymbol{b}_g) \in (0,1)^{S \times d_{\textbf{model}}}, \qquad \widetilde{\boldsymbol{O}} = \boldsymbol{G} \odot \boldsymbol{O}$$

The extra cost of the gate is linear, $\mathcal{O}(S\, d_{\text{model}})$. Negligible compared to the $\mathcal{O}(S^2 d_{\text{model}})$ of attention.

TabSwift — Early Exit (1): Architecture and Training

At every layer, compute the prediction we would get if we stopped now, and how much we can trust it.

Notation.
Let $\boldsymbol{H}^{(e)} \in \mathbb{R}^{S \times d_{\text{model}}}$ be the hidden state of layer e. Let $\boldsymbol{z}^{(e)} \in \mathbb{R}^{T \times d_{\text{model}}}$ be the rows of $\boldsymbol{H}^{(e)}$ at the test-data positions, and $\boldsymbol{r}^{(e)} \in \mathbb{R}^{K \times d_{\text{model}}}$ the rows at the register-token positions. There are T test rows and K register-token rows.

Prediction head $h^{(e)}_{\text{pred}}$.
The intermediate prediction using only the layers up to e

$$\hat{y}^{(e)} = h^{(e)}_{\text{pred}}\left(\boldsymbol{z}^{(e)}\right)$$

Exit head $h^{(e)}_{\text{exit}}$.
Concatenate with the register mean $\boldsymbol{r}^{(e)} = \frac{1}{K}\sum_k \boldsymbol{R}^{(e)}_k \in \mathbb{R}^{d_{\text{model}}}$ and output a logit

$$s^{(e)} = h^{(e)}_{\text{exit}}\left(\left[\boldsymbol{z}^{(e)};\, \boldsymbol{r}^{(e)}\right]\right) \in \mathbb{R}$$

TabSwift — Early Exit (2): Stopping Rule and Choosing τ

Train. The exit head learns a binary label: "is the intermediate prediction at layer e already correct?"

$$t^{(e)} = \mathbb{I}\Big\{\hat{y}^{(e)} = y\Big\} \in \{0, 1\}$$

The loss is binary cross-entropy:

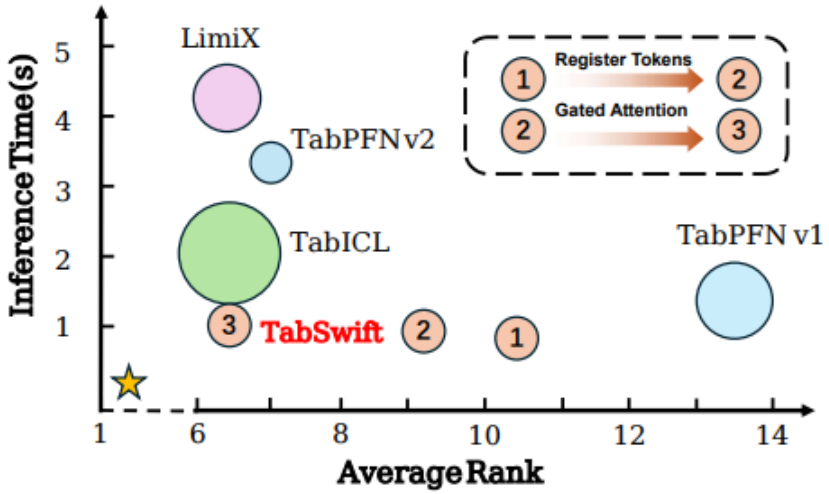
$$\mathcal{L}_{\text{exit}}^{(e)} = \text{BCE}(\sigma(s^{(e)}), t^{(e)})$$

Inference. Compute layers in order starting from 1, check $\sigma(s^{(e)})$ at every layer, and stop at the first layer where it passes the threshold τ:

$$e^\star = \min\Big\{\, e : \sigma(s^{(e)}) \geq \tau \,\Big\}, \qquad \text{output is } \hat{y}^{(e^\star)}.$$

If no layer passes, use the prediction from the full depth L.

Results



Backbone	# tokens	attention cost
Row-wise only, v1	n	$\mathcal{O}(n^2 d_{\text{model}})$
TabSwift	$S = K + n$	$\mathcal{O}(S^2 d_{\text{model}}) \approx \mathcal{O}(n^2 d_{\text{model}})$
Row & column, v2	$n \times d$	$\mathcal{O}(dn^2 d_{\text{model}} + nd^2 d_{\text{model}})$