Tabular Latent Fusion for Language Models A Token-Free Framework for Injecting TabFM Representations into Autoregressive Decoders

Chandan Kumar
IBM

***Abstract*—Large Language Models (LLMs) are increasingly used to reason over structured data, yet most practical systems still convert tables into textual prompts such as CSV, JSON, Markdown, or natural-language summaries. This text-serialization step increases token cost, weakens numerical fidelity, and obscures the two-dimensional row-column structure that makes tabular data meaningful. This paper presents a research-oriented framework for tabular latent fusion: a token-free mechanism that encodes tables with a Tabular Foundation Model (TabFM), projects the resulting continuous representation into an LLM-compatible hidden space, and fuses it with an autoregressive decoder through cross-modal attention, residual injection, or gated Bayesian correction. The accompanying notebook, research1.ipynb, compares direct table prompting, compact TabFM-vector prompting, local GPT-2 decoding, and TabFM-to-GPT-2 Bayesian/cross-attention injection. This work is positioned as a prototype-level architectural study rather than a full production benchmark. Prototype experiments are intended to test feasibility, token pressure, runtime behavior, and architectural trade-offs.**

***Index Terms*—tabular foundation models; large language models; TabFM; GPT; multimodal fusion; cross-attention; Bayesian correction; token efficiency; structured data reasoning.**

I. INTRODUCTION

Enterprise and scientific datasets are frequently stored as tables: financial order books, medical telemetry, industrial sensor streams, claims records, policy documents, and operational metrics all share an explicit row-column structure. However, LLMs operate on one-dimensional token sequences, following the Transformer language modeling paradigm that underlies modern GPT-style systems [1, 2]. A common engineering solution is to serialize a table into text and place it in the prompt, as explored in tabular prompting approaches such as TabLLM [4]. Although simple, this design introduces three recurring problems.

First, text serialization is token-expensive. A small table may fit into context, but long windows, high-frequency streams, or many feature columns rapidly consume input tokens. Second, serialization weakens numeric and relational fidelity: row order, column order, feature scale, and numerical precision may become artifacts of prompt formatting rather than learned structure. Third, agentic alternatives that ask an LLM to write and execute Python/Pandas code introduce latency, sandbox dependence, and code-generation failure modes.

Recent tabular foundation models suggest another path. Google Research presents TabFM as a zero-shot foundation model for tabular data, and its public model card describes tabular classification and regression support without task-specific fine-tuning in the standard use case [5, 6]. More broadly, tabular foundation model research, including TabPFN, shows that table-specific encoders can learn reusable priors over structured data [3]. This raises a natural question:

Can an LLM consume a table through a continuous tabular representation instead of through serialized text tokens?

This paper proposes a research framework to answer that question. We encode structured rows using TabFM-like latent representations, align those vectors to the LLM hidden dimension, and fuse them into a GPT-style decoder using a projection bridge,

cross-attention, residual conditioning, and optional Bayesian posterior correction. The remainder of this paper is organized as follows: we first describe the motivation for tabular latent fusion, then present the architectural overview, GPT decoder baseline, TabFM-style encoder, fusion mechanisms, Bayesian correction, implementation pathway, evaluation protocol, limitations, and future work.

1.1 Contributions

This paper makes the following contributions:

- A token-free tabular-to-LLM architecture. We define a structured pipeline that moves from table tensors to continuous tabular embeddings and then into an autoregressive decoder hidden space, avoiding full CSV/JSON-style table serialization.
- A TabFM-to-decoder latent fusion framework. We formalize a projection bridge that maps TabFM-style representations into the hidden dimension of a GPT-style decoder, enabling table information to be fused with language generation.
- Multiple fusion strategies. We describe projection-based conditioning, cross-attention fusion, gated residual injection, and Bayesian posterior correction as compatible mechanisms for incorporating tabular latent information into decoder inference.
- A notebook-aligned evaluation protocol. We define an evaluation pathway for comparing direct table prompting, compact TabFM-vector prompting, GPT-2 baseline decoding, and TabFM-to-GPT-2 injected decoding using token cost, latency, output quality, memory usage, and production-readiness metrics.
- A balanced discussion of benefits and limitations. We analyze where tabular latent fusion may reduce token pressure, preserve structural information, and reduce dependency on agentic code execution, while also identifying limitations such as projection training, closed-model access, and benchmark coverage.

II. BACKGROUND AND MOTIVATION

2.1 Why Serialized Tables are Insufficient

Let a table be represented as $X_{tab} \in \mathbb{R}^{B \times R \times C}$, where B is batch size, R is the number of rows, and C is the number of columns. In text-prompting systems, X_{tab} is converted into a string s:

$$s = \text{Serialize}(X_{tab}; \text{CSV, JSON, Markdown, or text}). \quad (1)$$

The LLM then receives token IDs $t = \text{Tokenizer}(s)$ and reasons over token embeddings E_{text}. This design is easy to implement but suffers from an information bottleneck: the table is forced through a language tokenizer not designed for numeric scale, feature covariance, or row-column relational structure.

2.2 Tabular Latent Fusion

Instead of converting the table into words, tabular latent fusion keeps the table in tensor form:

$$H_{tab} = f_{TabFM}(X_{tab}), \quad (2)$$

where f_{TabFM} is a table-specific encoder. The latent table representation is then projected to the decoder hidden dimension:

$$Z_{tab} = \text{LayerNorm}(H_{tab} W_p + b_p). \quad (3)$$

The projected representation Z_{tab} can be used as a continuous conditioning signal for an LLM decoder. This shifts the design from:

Table → Text → Tokens → LLM

to:

Table → Tabular Latents → LLM Hidden Space

III. ARCHITECTURAL OVERVIEW

Figure 1 presents the complete proposed pipeline. The framework has four major stages: table preparation, TabFM encoding, cross-modal fusion, and autoregressive generation. The design is conceptually related to multimodal systems that project non-text representations into language-model-compatible spaces [7, 8].

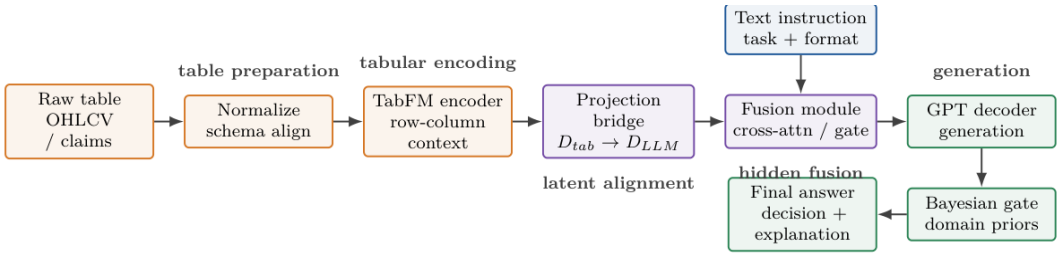

Figure 1: Compact end-to-end architecture. The prompt provides the task instruction, while the table enters the model through continuous TabFM-derived latent representations rather than long CSV/JSON serialization.

IV. STANDARD GPT DECODER BASELINE

A GPT-style decoder receives tokenized text, converts token IDs into embeddings, applies positional information, and passes the hidden states through stacked causal decoder blocks. Each block contains masked self-attention, feed-forward transformations, residual connections, and layer normalization. Figure 2 gives a compact standard decoder schematic.

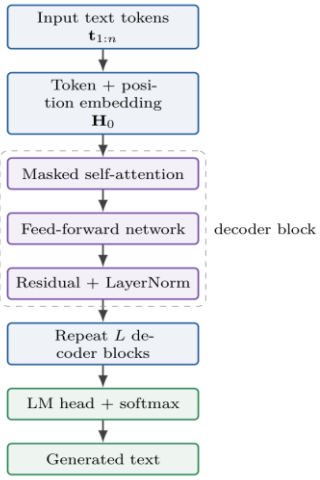

Figure 2: Standard GPT-style autoregressive decoder. In a pure text baseline, all external information must enter through tokenized text.

In this baseline, a table can influence the model only if it is serialized into text. The proposed fusion pathway leaves the standard decoder intact but adds a continuous external condition derived from TabFM.

V. TABFM-STYLE TABULAR ENCODER

TabFM-style encoders are designed for two-dimensional data where both rows and columns matter. The encoder receives a table context, applies transformations that capture feature interactions and row-level dependencies, and returns continuous table representations. Figure 3 illustrates the conceptual encoder used in this paper.

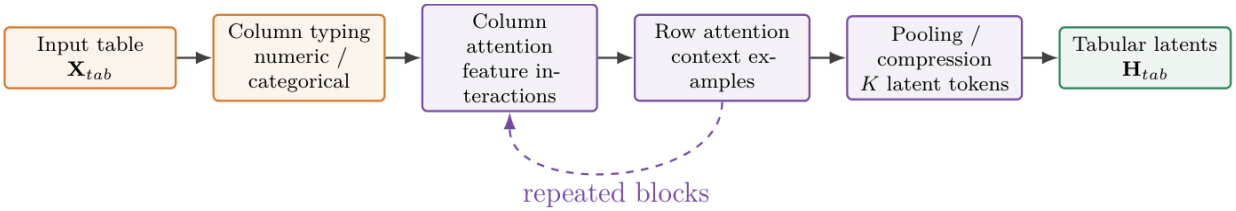

Figure 3: Compact TabFM-style tabular encoder. The encoder preserves row-column structure before producing continuous latent table tokens.

For a table window X_tab, the encoder produces:

H_tab = f_TabFM(X_tab) ∈ ℝ^(B×K×D_tab), (4)

where K is the number of compressed latent table tokens. In simple implementations, K = 1 after mean pooling. In richer implementations, K may preserve multiple row-level, segment-level, or feature-group-level embeddings.

VI. FUSION DESIGN

The main design question is how to connect H_tab to the GPT decoder. This paper defines three fusion variants: projection-only prompting, residual hidden-state injection, and cross-attention fusion.

6.1 Projection Bridge

The projection bridge aligns TabFM hidden size D_tab with LLM hidden size D_LLM:

$$Z_{tab} = \text{LayerNorm}(H_{tab} W_p + b_p), \quad W_p \in \mathbb{R}^{(D_{tab} \times D_{LLM})}. \tag{5}$$

This bridge can be trained using supervised next-token loss, contrastive alignment, instruction-output pairs, or task-specific decision labels.

6.2 Residual Injection

For decoder layer l, residual injection adds the tabular signal to the language hidden state:

$$\tilde{H}_l = \text{DecoderBlock}_l(H_{(l-1)}) + \alpha_l \cdot g_l(Z_{tab}), \tag{6}$$

where α_l is a learnable or manually tuned scalar, and $g_l(\cdot)$ maps tabular latent tokens into a layer-compatible representation. A small α_l prevents the tabular signal from overpowering language fluency.

6.3 Cross-Attention Fusion

A stronger fusion method allows decoder hidden states to attend to tabular latent tokens:

$$Q_l = H_{(l-1)} W_{(Q,l)}, \tag{7}$$

$$K_{tab} = Z_{tab} W_{(K,l)}, \tag{8}$$

$$V_{tab} = Z_{tab} W_{(V,l)}, \tag{9}$$

$$\text{CrossAttn}_l = \text{softmax}((Q_l K_{tab}^{\top})/\sqrt{d_k}) V_{tab}. \tag{10}$$

The decoder update becomes:

$$\tilde{H}_l = \text{DecoderBlock}_l(H_{(l-1)}) + \beta_l \cdot \text{CrossAttn}_l. \tag{11}$$

This is more expressive than simple residual addition because language tokens can selectively retrieve relevant table latent information.

6.4 Fusion Diagram

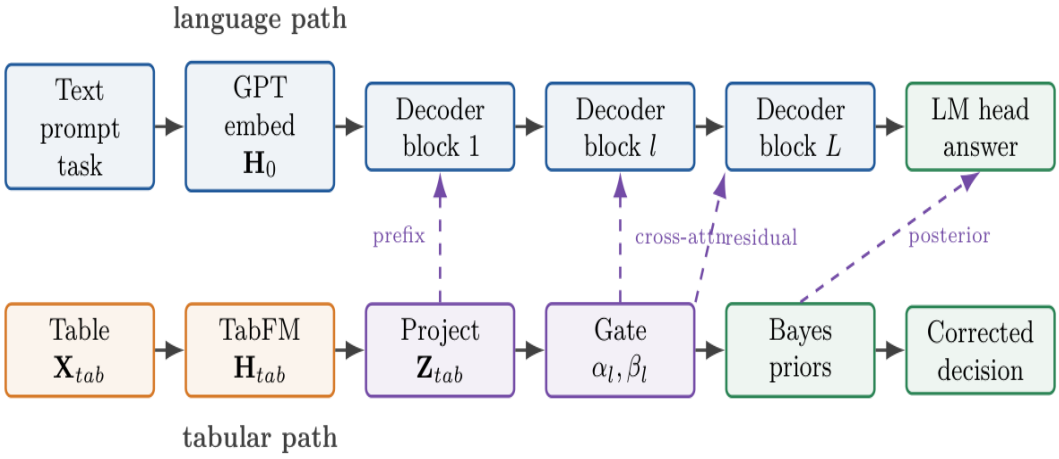

Figure 4: Compact fusion strategy. The upper lane is the GPT language path and the lower lane is the TabFM tabular path. Projection and gates inject table information into selected decoder layers, while posterior correction is applied only for decision-oriented outputs.

VII. BAYESIAN POSTERIOR CORRECTION

For decision-oriented outputs, such as BUY/SELL, fraud/non-fraud, or approve/reject, the decoder probabilities may be corrected with a lightweight Bayesian process gate. Let the decoder produce a likelihood-like score $P(I \mid y)$ for candidate label y. Given a domain prior $P(y)$, the posterior follows the standard Bayesian correction principle [9]:

$$P(y \mid I) = [P(I \mid y)P(y)] / [\Sigma_{(y' \in Y)} P(I \mid y')P(y')]. \quad (12)$$

This step does not replace model reasoning; it regularizes the final decision when domain priors are known. For example, in a market analysis setting, priors may encode trend direction, volatility regime, or risk constraints. In an insurance setting, priors may encode policy eligibility or claim class frequencies.

VIII. IMPLEMENTATION PATHWAY

The notebook implementation is organized into the following execution stages:

1. Load tabular stream. Load OHLCV, sensor, or business table windows.
2. Normalize features. Apply scaling and schema alignment.
3. Encode table. Produce TabFM-style dense vectors.
4. Create bridge vector. Map dense vectors into the LLM hidden dimension.
5. Run baselines. Compare raw table prompting and compact vector prompting.
6. Run local decoder. Evaluate GPT-2 baseline without tabular conditioning.
7. Run fusion decoder. Inject TabFM features through residual, cross-attention, or Bayesian gates.
8. Measure outputs. Record token count, runtime, estimated API cost, quality signals, limitations, and best-use cases.

Algorithm 1 TabFM-to-GPT Fusion Inference

```
procedure FusionInference(X_tab, t_prompt)
    X_norm ← NormalizeAndAlign(X_tab)
    H_tab ← f_TabFM(X_norm)
    Z_tab ← LayerNorm(H_tab W_p + b_p)
    H_0 ← TokenEmbedding(t_prompt) + PositionEmbedding
    for l ← 1 to L do
        U_l ← DecoderBlock_l(H_(l−1))
        if fusion is residual then
            H_l ← U_l + α_l g_l(Z_tab)
        else if fusion is cross-attention then
            C_l ← CrossAttention(U_l, Z_tab)
            H_l ← U_l + β_l C_l
        else
            H_l ← U_l
        end if
    end for
    o ← LMHead(H_L)
    p ← Softmax(o)
    p* ← BayesianPosteriorGate(p, priors)
    return Decode(p*)
end procedure
```

IX. EVALUATION PROTOCOL

The notebook compares four broad paths, summarized in Table 1. This table defines the evaluation pathways used by the accompanying implementation. Final reported results should include measured token counts, runtime, memory usage, quality scores, and estimated cost.

Table 1: Notebook-aligned evaluation paths for comparing text prompting, compact latent prompting, local decoding, and TabFM-GPT fusion.

Path	Input to LLM	Primary metric	Purpose
Direct OHLCV → GPT-4o mini	Raw table/statistics in prompt	Token cost, quality	Measures the practical text-prompting baseline.
TabFM vector → GPT-4o mini	Compact encoded vector plus statistics	Token reduction, quality	Tests whether a compact latent summary can reduce prompt size while preserving useful signal.
GPT-2 baseline	Text prompt only	Local runtime, fluency	Establishes local decoder behavior without tabular conditioning.
TabFM → GPT-2 fusion	Projected TabFM latents injected into decoder	Alignment, accuracy, latency	Tests whether latent fusion improves local generation and decision alignment.

9.1 Suggested Metrics

Table 2: Recommended evaluation metrics for the notebook and paper.

Metric	Definition	Why it matters
Prompt tokens	Number of input tokens sent to API/local decoder	Measures cost and context pressure.
Completion tokens	Number of generated output tokens	Measures output budget and response length.
Runtime latency	End-to-end execution time in milliseconds	Critical for production feasibility.
Peak memory	Maximum memory consumed during inference	Important for local GPU/CPU deployment.
Quality score	Rule-based or human-evaluated output quality	Measures grounding, relevance, and repetition.
Token accuracy	Accuracy of expected target-token generation	Useful for decision labels such as BUY/SELL.
Domain coverage	Whether output mentions key domain facts	Detects hallucination or topic drift.
Estimated cost	API cost estimated from usage	Compares direct prompting vs compact latent prompting.

9.2 Expected Qualitative Findings

The notebook design is expected to reveal a trade-off rather than a single universal winner:

- Direct prompting usually gives the most interpretable API response because the model can see the raw rows, but token cost grows quickly.
- Compact TabFM prompting may reduce tokens, but the vector representation needs careful explanation or alignment for closed API models.
- GPT-2 baseline is useful for local control but lacks structured tabular grounding.
- TabFM-to-GPT-2 fusion is the most research-relevant path because it tests whether hidden-space conditioning can improve local generation without serializing the table.

X. BENEFITS OF THE PROPOSED FUSION

Table 3: Practical benefits of tabular latent fusion compared with direct table serialization.

Benefit	Why it happens	Practical impact
Lower token pressure	The full table does not need to be serialized into the text prompt.	Larger table windows can be represented compactly.
Better structural fidelity	TabFM-style encoders preserve row-column context before fusion.	Numeric and relational patterns can be retained as learned latent features.
Reduced code-execution risk	The LLM does not need to write Pandas scripts for every query.	Fewer runtime failures and sandbox dependencies.
Local deployment path	GPT-2/open-source decoders can be modified at hidden-state level.	Useful for experimentation, privacy, and cost control.
Flexible fusion levels	Injection can occur at input, mid-layer, or late-layer decoder blocks.	Enables ablation studies on where tabular information is most useful.
Decision regularization	Bayesian priors can correct unstable final probabilities.	Useful for noisy domains such as markets, fraud, or claims.

XI. LIMITATIONS AND THREATS TO VALIDITY

This work is a prototype-level architectural study and should not be interpreted as a fully validated production benchmark. Several limitations remain and should be considered when interpreting the proposed framework.

- **C**losed-model access. Direct hidden-state injection is generally not available in closed API-only LLMs. The proposed fusion mechanism is therefore most practical for open-source decoders or controlled model-serving environments.
- Projection training requirement. The projection bridge between TabFM embeddings and LLM hidden states must be trained or calibrated. A randomly initialized bridge may not produce meaningful language alignment.
- Limited benchmark coverage. The current notebook-based evaluation is intended to test feasibility and architectural behavior. Broader validation is required across public classification, regression, and tabular reasoning benchmarks.
- Interpretability trade-off. Continuous tabular embeddings are less human-readable than raw table prompts. Additional explanation methods are needed to make latent fusion outputs interpretable.
- Prior calibration risk. Bayesian posterior correction is useful only when priors are reliable and domain-valid. Poorly calibrated priors may bias the final decision.
- Compute trade-off. TabFM encoding introduces additional compute before decoding. The benefit should therefore be measured against token-cost savings, latency, memory usage, and output-quality improvements.

XII. FUTURE WORK

Future experiments should focus on the following directions:

1. Train the projection bridge with supervised instruction-output pairs.
2. Compare residual injection, cross-attention, prefix tuning, adapters, and LoRA-based fusion.
3. Evaluate on public tabular benchmarks in addition to OHLCV streams.
4. Add human evaluation for factuality, usefulness, and explanation quality.
5. Report full ablation results for injection layer depth and gate strength.
6. Measure scaling behavior as rows, columns, and context windows increase.
7. Add a safety layer for high-stakes domains where generated advice must be constrained.

XIII. CONCLUSION

This paper presents a structured formulation of tabular latent fusion for LLMs. The central idea is simple: instead of forcing a table through text serialization, encode it with a tabular foundation model, project the representation into the LLM hidden space, and fuse it directly with the decoder. This approach offers a meaningful research path toward lower token cost, better structural fidelity, and more controllable local generation. The accompanying implementation provides an initial experimental scaffold; the next step is systematic evaluation across datasets, fusion mechanisms, and decoder sizes.

Code and notebook artifacts are maintained at: https://github.com/chandanuexp-star/table2Text

REFERENCES

[1] 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, 2017.

[2] Tom B. Brown et al. Language models are few-shot learners. Advances in Neural Information Processing Systems, 2020.

[3] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. International Conference on Learning Representations, 2023.

[4] Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. TabLLM: Few-shot classification of tabular data with large language models. AISTATS, 2023.

[5] Weihao Kong and Abhimanyu Das. Introducing TabFM: A zero-shot foundation model for tabular data. Google Research Blog, 2026. Google Research Blog

[6] Google Research. TabFM 1.0.0 PyTorch model card. Hugging Face, 2026. Hugging Face model card

[7] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. arXiv preprint arXiv:2304.08485, 2023.

[8] Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. Google Technical Report, 2024.

[9] Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.