

Best Practices Have Preconditions: A Systematic Evaluation of Deep Learning for Tabular Fraud Detection

Franklin Dickinson
Cornell Tech — Deep Learning (CS5787)
New York, NY, USA

Abstract

Fraud detection is a canonical hard classification problem: transactions arrive at extreme class imbalance, and feature representations vary widely by data source and privacy constraints: raw behavioral signals, semantic merchant descriptions, or PCA-anonymized numerical features where privacy enforcement has already determined the representation before modeling begins. A practitioner choosing an architecture must account for which type of data they have, because that determines whether the choice holds. Testing this across controlled variation, rather than on a single curated dataset, is what surfaces the mechanism.

We construct a **~200-experiment systematic sweep** that mimics the range of conditions a practitioner encounters: three architectures (XGBoost, FT-Transformer, BERT), three datasets with structurally distinct feature types, five data scales, and five random seeds, each varied independently to identify not just which model wins, but under what conditions and why.

Feature type, not model class, determines the architecture hierarchy. XGBoost dominated on PCA-transformed numerical features (0.865 PR-AUC); BERT dominated on semantic text features (0.803); the ranking inverted completely across our two primary environments. FT-Transformer achieved stable mid-tier performance (0.689–0.729) across all settings with zero catastrophic failures and zero manual feature engineering.

Domain pre-training transferred at small scale, then became destructive. A common practitioner decision: you have access to a large labeled dataset from a related fraud detection setting (not your exact target data, but the same application domain) and must decide whether pre-training on it before fine-tuning provides an advantage. We test this directly: BERT pre-trained on the IEEE-CIS fraud dataset (590K transactions, 3.5% fraud rate, 434 features) then fine-tuned on the European credit card dataset (0.17% fraud rate, 30 PCA features). Both are fraud detection, but the distributions differ substantially. Fraud-BERT outperformed vanilla BERT at 5–25% data scales, where pre-training signal was useful despite the mismatch. At full scale, where the target is 20× rarer with 14× fewer features, fine-tuning gradient overwrote pre-trained representations unstably: 3 of 4 seeds produced classifiers with zero discriminative power.

Multi-seed evaluation surfaced failures invisible to single runs: a 4% catastrophic collapse rate; a dataset that plateaued at 25% data with zero gain from 75% more; a 52% val-to-test gap from stratified downsampling that preserved fraud rate while destroying fraud pattern diversity. Each failure traces to a specific mechanism, and each mechanism is a precondition on a standard best practice.

Keywords

fraud detection, transformers, FT-Transformer, BERT, XGBoost, tabular data, domain pre-training, transfer learning, class imbalance, multi-seed validation

1 Introduction

Most model comparisons test one architecture on one dataset with one random seed and declare a winner. That methodology produces a result, not a condition. When practitioners apply the conclusion to a different problem, they do not know whether the feature structure falls in the regime where the result was measured, whether the data scale is sufficient for the chosen architecture, or whether a different random seed would reverse the outcome.

Credit card fraud detection is an unusually precise testbed for exposing this gap. The class distribution problem is constant: fraud occurs at 576:1 imbalance across our evaluation. But the same detection task appears in two structurally different feature representations: PCA-transformed numerical features, where semantic content has been removed, and raw semantic categorical features, where merchant names, transaction categories, and geography carry meaningful signal. This lets the same architectures be evaluated against fundamentally different data structures while holding the deployment constraint fixed. When a modeling assumption fails here, performance makes it visible.

We design a systematic sweep: three architectures (XGBoost, FT-Transformer, BERT) × three datasets × five data scales × five random seeds, yielding 197 experiments. Each dimension is varied independently. The goal is not to find the best model but to identify when each model is the right choice, and the mechanism by which it fails when the conditions for that choice are not met.

The central finding: feature type, not model class, determines the architecture hierarchy. XGBoost dominated on PCA-transformed numerical features (0.865 PR-AUC); BERT dominated on semantic text features (0.803); the ranking inverted completely across environments. FT-Transformer achieved stable mid-tier performance with zero catastrophic failures across all settings. Domain pre-training helped at small data scales, then collapsed destructively once fine-tuning gradient was sufficient to overwrite mismatched representations. Multi-seed evaluation surfaced a 4% catastrophic failure rate invisible to single runs.

Each finding maps to a precondition on a standard best practice. The practices are real and well-supported. The preconditions determine whether they apply to a given problem.

Section 2 reviews related work. Section 3 describes experimental design, including the Fraud-BERT pre-training procedure. Sections 4.1–4.5 present five preconditions in turn. Section 5 synthesizes findings and their practical implications.

1.1 Contributions

- (1) **Feature-architecture alignment:** A 197-experiment systematic comparison demonstrating that feature type determines the model hierarchy: XGBoost wins on PCA (0.865 PR-AUC) and BERT on semantics (0.803), with mechanistic explanation for each result.
- (2) **Fraud-BERT collapse analysis:** Characterization of the data-scale precondition for domain-mismatched pre-training. At 5–25% data, IEEE fraud pre-training improves performance. At 50%, a bimodal collapse appears: one seed fails completely while four succeed. At 100%, 3 of 4 seeds produce classifiers with zero discriminative power. We identify two distinct failure modes (Mode A: all-positive output; Mode B: all-negative) and trace both to mechanistic causes.
- (3) **FT-Transformer reliability:** Zero catastrophic failures across 75 experiments with $8\times$ lower variance than BERT, attributed to purpose-built tabular architecture eliminating the degenerate solutions that seed-sensitive language models can reach.
- (4) **Downsampling negative result:** Mechanistic explanation of a 52% val-to-test gap from stratified downsampling, distinguishing fraud-rate preservation from fraud-diversity preservation.
- (5) **Data efficiency characterization:** Dataset-specific learning curves establishing when data collection has positive ROI (Sparkov) and when it does not (European after 25%).

2 Related Work

2.1 Fraud Detection with Machine Learning

2.1.1 Traditional Methods. Early fraud detection systems relied on rule-based approaches and statistical methods [2]. These methods are interpretable but struggle with evolving fraud patterns.

Tree-based methods have dominated fraud detection:

- **Random Forests** [4]: Ensemble of decision trees, robust to class imbalance
- **XGBoost** [6]: Gradient boosting with regularization, state-of-the-art for tabular data
- **LightGBM** [21]: Faster gradient boosting, handles large datasets efficiently

Why tree-based methods work well:

- Handle categorical features natively
- Robust to class imbalance (via sample weighting)
- Fast training and inference
- Interpretable (feature importance, SHAP values)

Our work complements this by exploring **when deep learning can compete** with tree-based methods for fraud detection.

2.1.2 Deep Learning for Fraud Detection. Recent work has explored deep learning for fraud detection:

Feed-forward networks [7]:

- Multi-layer perceptrons (MLPs) on engineered features
- Underperform XGBoost on tabular data
- Require extensive hyperparameter tuning

Recurrent networks [19]:

- LSTMs for sequential transaction history

- Capture temporal patterns (spending behavior changes)
- But: most fraud detection operates on single transactions (no sequence)

Graph neural networks [26]:

- Model relationships between users, merchants, transactions
- Effective for network-based fraud (e.g., fraud rings)
- But: requires graph structure (not always available)

Our work focuses on **single-transaction fraud detection**, where tabular transformers are most applicable.

2.2 Transformers for Tabular Data

2.2.1 BERT for Tabular Data. Pre-training approaches [10, 27]:

- TabBERT [27]: Pre-train BERT on tabular data (Wikipedia tables)
- TUTA [10]: Table understanding with transformers
- Our finding: Pre-training doesn't help for tabular fraud (wrong inductive bias)

Text-based approaches:

- Convert tabular data to text, fine-tune BERT [11]
- Enables transfer learning from NLP pre-training
- Our finding: Works but slower than tabular-specific architectures ($10\times$)

2.2.2 Tabular-Specific Transformers. FT-Transformer [15]:

- Feature tokenization (each feature $\rightarrow$ separate embedding)
- No positional encoding (features are independent)
- State-of-the-art for many tabular benchmarks
- Our contribution: First large-scale evaluation for fraud detection (197 experiments)

TabNet [1]:

- Attention-based feature selection
- Interpretable via attention masks
- Competitive with XGBoost on some tasks

SAINT [24]:

- Self-attention + intersample attention
- Learns both row-wise and column-wise patterns
- Computationally expensive

TabTransformer [18]:

- Embeddings for categorical features, transformer encoder
- Handles categorical + continuous features separately
- Outperforms MLPs on some benchmarks

2.3 Class Imbalance Handling

Fraud detection faces extreme class imbalance (576:1 in our datasets).

2.3.1 Sampling Methods. Oversampling [5]:

- SMOTE: Synthetic minority oversampling
- ADASYN: Adaptive synthetic sampling
- Issue: Creates synthetic examples that may not reflect real fraud

Undersampling [12]:

- Randomly remove majority class samples
- Issue: Throws away legitimate transaction data

- Our finding: Loss weighting achieves same effect without data loss

Our approach: Weighted loss (inverse frequency) is mathematically equivalent to balanced sampling but more efficient.

2.3.2 Algorithm-Level Methods. Cost-sensitive learning [13]:

- Assign higher misclassification cost to minority class
- Our method: $w_{\text{fraud}} = 576$ in weighted BCE loss

Ensemble methods [14]:

- EasyEnsemble, BalanceCascade
- Combine multiple classifiers trained on balanced subsets
- Future work: Ensemble FT-Transformer models

2.4 Statistical Validation in Machine Learning

2.4.1 Multi-Seed Validation. Reproducibility in ML [16]:

- Single-seed results are unreliable (high variance)
- Recommendation: Run ≥ 3 seeds for publication
- Our contribution: Caught BERT’s 4% catastrophic failure rate via 5 seeds

Statistical testing [9]:

- Paired t-tests for model comparison
- Confidence intervals for performance estimates
- Our approach: 95% CI, t-tests, effect sizes (Cohen’s d)

2.4.2 Hyperparameter Sensitivity. Sensitivity analysis [3]:

- ML models sensitive to hyperparameters
- Random search often better than grid search
- Our approach: Standard hyperparameters from literature, validated on 25% data

2.5 Data Efficiency in Deep Learning

2.5.1 Learning Curves. Sample complexity [17]:

- Power-law relationship: error $\propto N^{-\alpha}$
- More data helps, but with diminishing returns
- Our finding: Power law doesn’t hold—European plateaus at 25%

Transfer learning: BERT’s text pre-training does not transfer to tabular fraud features; the inductive biases for sequential language and independent numerical features are fundamentally mismatched.

2.6 Experimental Design in ML

2.6.1 Distribution Mismatch. Dataset shift [22]:

- Covariate shift: $P(X)$ changes between train and test
- Label shift: $P(Y)$ changes between train and test
- Our finding: Stratified sampling creates covariate shift (rare fraud patterns missing)

Train-test leakage [20]:

- Temporal leakage: Future information leaks into training
- Solution: Temporal splits (not random)
- Our approach: Train on oldest 70%, test on newest 15%

2.6.2 Controlled Experiments. Confounding variables:

- Size confound: European (284K) vs Sparkov (1M)
- Our finding: Forcing size matching (stratification) creates worse problems (distribution shift)
- Better: Acknowledge confound honestly, report separately

2.7 Differences from Prior Work

Our work differs from prior fraud detection and tabular deep learning research in several ways:

- (1) **Scale:** 197 experiments with 5-seed validation (most prior work: single seed, few experiments)
- (2) **Architecture comparison:** First large-scale BERT vs FT-Transformer comparison for fraud detection
- (3) **Failure mode discovery:** Identification of BERT’s 4% catastrophic failure rate through multi-seed validation
- (4) **Methodological contribution:** Distribution mismatch finding (50% drop from stratified sampling)
- (5) **Data efficiency:** Dataset-specific patterns (European peaks at 25%, Sparkov improves to 100%)
- (6) **Practical focus:** Evidence-based recommendations for practitioners, not just performance numbers

3 Methodology

3.1 Experimental Design

We varied four dimensions independently: architecture (XGBoost, FT-Transformer, BERT), dataset (European, Sparkov, Sparkov-Matched), data scale (5%, 10%, 25%, 50%, 100%), and random seed (42, 123, 456, 789, 1011). Not every combination was run; some model-dataset pairings were not applicable, yielding 197 total experiments: 75 FT-Transformer, 74 BERT, and 48 XGBoost.

3.2 Datasets

Table 1 summarizes the three datasets used.

European’s features are PCA-transformed (dense numerical representations with no recoverable categorical structure), which means all three models receive the same feature type on this dataset. Sparkov’s raw features span merchant name, category, geography, amount, and demographics, and each architecture processes them differently: BERT receives natural language descriptions of transactions; FT-Transformer receives one-hot and target-encoded categorical features; XGBoost receives three progressively engineered feature sets. Sparkov-Matched was constructed by stratified sampling to match European’s training size for a size-controlled comparison; its results appear in Section 4.5 as a negative result.

An important confound: dataset size and feature type co-vary. European is smaller (284K) and PCA-only; Sparkov is larger (1M) and raw. We cannot fully isolate a “feature type effect” from a “dataset size effect” in cross-dataset comparisons. We report results separately and acknowledge this limitation rather than claim clean isolation.

3.3 Data Splits

All experiments use temporal splits to simulate production deployment: training on the oldest 70% of transactions, validation on the middle 15%, and test on the most recent 15%. Random splits would

Table 1: Dataset comparison

	European	Sparkov	Sparkov-Matched
Source	Dal Pozzolo et al. [8]	Sparkov et al. [25]	Downsampled from Sparkov
Transactions	284,807	1,296,675	36,000 (train)
Fraud transactions	492	2,230	62
Fraud rate	0.172%	0.172%	0.172%
Class ratio	576:1	580:1	580:1
Features	30 (PCA)	23 (raw)	23 (raw)
Temporal span	2 days (Sept 2013)	2 years (2019–2020)	2 years

Table 2: IEEE-CIS pre-training source vs. European fine-tuning target

	IEEE-CIS	European
Transactions	590,540	284,807
Fraud rate	3.5%	0.17%
Features	434	30 (PCA)
Geography	US e-commerce	EU card-present

Fraud-rate mismatch: 20×. Feature-count mismatch: 14×.

allow models to implicitly “see the future” during training, producing optimistic results that would not hold in production where fraud patterns evolve. For each random seed, the train/val/test assignment is fixed (seeds randomize only model initialization, batch order, and dropout masks), isolating model variance from data variance.

3.4 Model Architectures

3.4.1 FT-Transformer. FT-Transformer [15] was designed specifically for tabular data. Each input feature receives its own learned embedding (feature tokenization), and there is no positional encoding, because tabular features have no inherent order. A [CLS] token aggregates feature-level representations for classification. See Table 3 for configuration.

3.4.2 BERT. BERT [11] (bert-base-uncased) was pre-trained on BookCorpus and Wikipedia. It uses WordPiece tokenization and absolute positional encoding, encoding sequential dependencies between tokens. Applied to tabular data, transactions must be serialized into text strings before tokenization. See Table 3 for configuration.

3.4.3 Fraud-BERT. Fraud-BERT is a second BERT variant pre-trained via masked language modeling (MLM) on the IEEE-CIS 2019 fraud dataset before fine-tuning on European credit card data. The IEEE source and European target differ substantially in fraud prevalence, feature count, geography, and transaction type (Table 2). MLM pre-training used 15% masking probability, learning rate 5×10^{-5} , effective batch size 32 (8 per device with 4-step gradient accumulation), and ran for 3 epochs (24,915 steps) reaching evaluation loss 0.0754 (perplexity 1.08). The low perplexity reflects the formulaic structure of serialized transaction text: amounts, merchant names, and categories follow highly regular patterns that the model learns quickly once template structure is established, rather than data contamination or evaluation leakage. Fine-tuning then proceeded identically to vanilla BERT.

3.4.4 XGBoost. XGBoost [6] builds gradient-boosted decision trees that operate directly on numerical feature values. On the European dataset it receives PCA features with no modification. On Sparkov we evaluated three progressive feature sets to study engineering impact: Stage 1 (9 numerical features), Stage 2 (adding one-hot and target-encoded categorical features, ~75 total), and Stage 3 (adding behavioral, geospatial, and temporal features, ~85 total).

3.4.5 Configuration Summary.

3.5 Feature Engineering

For BERT on both datasets, each transaction is serialized to a natural language string before WordPiece tokenization (max length 128 tokens; average 42 tokens per transaction). For FT-Transformer on Sparkov, we engineered 74 categorical features across six groups: 8 temporal (hour, day, month, weekend flag), 5 amount-based (raw, log-transformed, relative to user mean), 10 merchant-level (fraud rate via target encoding, volume), 8 geographic (home-to-merchant distance, city population), 15 behavioral (per-customer transaction velocity and diversity), and 18 one-hot encoded. Target encoding of merchant fraud rates (computing per-merchant fraud probability from training data and mapping it to a continuous signal) is the single most predictive engineered feature.

3.6 Training Configuration

Table 4 shows training hyperparameters for all three models.

FT-Transformer’s learning rate was selected via grid search over $\{10^{-4}, 5 \times 10^{-5}, 10^{-5}\}$ and layer counts $\{2, 3, 4\}$ on the validation set. BERT uses the standard fine-tuning rate from Devlin et al. [11]; exhaustive tuning was infeasible given 30-minute training runs across 74 experiments. Class imbalance is handled via inverse-frequency loss weighting ($w_{\text{fraud}} = 576$) for all models, equivalent to oversampling the minority class without the memory cost.

3.7 Evaluation and Statistical Validation

PR-AUC (Precision-Recall Area Under Curve) is the primary metric throughout. At 576:1 class imbalance, accuracy is uninformative: a classifier predicting all transactions as legitimate achieves 99.8% accuracy, and ROC-AUC overestimates performance relative to the minority class [23]. PR-AUC focuses on the fraud class directly.

Each configuration is evaluated over five random seeds (42, 123, 456, 789, 1011). Seeds control model initialization, batch order, and dropout; the train/val/test split is held fixed across seeds to isolate model variance from data variance. We report mean $\pm$ standard

Table 3: Model configuration summary

Parameter	FT-Transformer	BERT	XGBoost
Parameters	~3M	~110M	Auto (trees)
d_{model} / hidden size	256	768	N/A
Attention heads	8	12	N/A
Layers	3	12	N/A
Feed-forward dim	1024	3072	N/A
Dropout	0.1	0.1	N/A
Activation	GELU	GELU	N/A
Max depth	N/A	N/A	6
Estimators	N/A	N/A	100
Subsample	N/A	N/A	0.8

Table 4: Training hyperparameters

Parameter	FT-Transformer	BERT	XGBoost
Learning rate	1×10^{-4}	2×10^{-5}	0.1
Batch size	64	64	N/A
Max epochs	20	10	N/A
Early stopping patience	3	3	N/A
Optimizer	AdamW	AdamW	N/A
Weight decay	1×10^{-5}	1×10^{-5}	N/A
Column subsample	N/A	N/A	0.8
Scale pos. weight	Auto	Auto	576

Table 5: Compute infrastructure

Component	Specification
GPU	NVIDIA A100 (40GB) / RTX 4090 (24GB)
Provider	RunPod cloud
CPU	16 cores, 64 GB RAM
Storage	500 GB NVMe SSD
PyTorch	2.1.0
Transformers	HuggingFace 4.36.0
CUDA / Python	12.1 / 3.10
Total compute	~200 GPU hours

deviation across seeds, 95% confidence intervals ($\mu \pm 1.96 \cdot SE$), and two-sample t -tests for model comparisons.

3.8 Infrastructure

4 Results

4.1 Feature Type Determines the Architecture Winner

The central comparison is Table 6: three architectures, two datasets with different feature structures, results at 100% data across five seeds.

The model hierarchy inverts completely between datasets. On PCA features, XGBoost leads by 18.6 percentage points over FT-Transformer and 26.1 over BERT. On semantic features, BERT leads, though only by 1.8 points over XGBoost Stage 3, and 8.4 over FT-Transformer. The feature structure, not the model, determines the ranking.

The mechanism is architectural. PCA-transformed features are dense numerical representations where decision boundaries are well-captured by axis-aligned splits; XGBoost’s tree-based learning is optimized for exactly this geometry. Semantic features (merchant names, transaction categories, geographic identifiers) contain relational structure that BERT’s pre-trained embeddings encode directly. “Grocery” and “pharmacy” are close in embedding space; their transaction risk profiles are not, and BERT learns this without being told. XGBoost can approximate the same effect through feature engineering, but that engineering has to happen first.

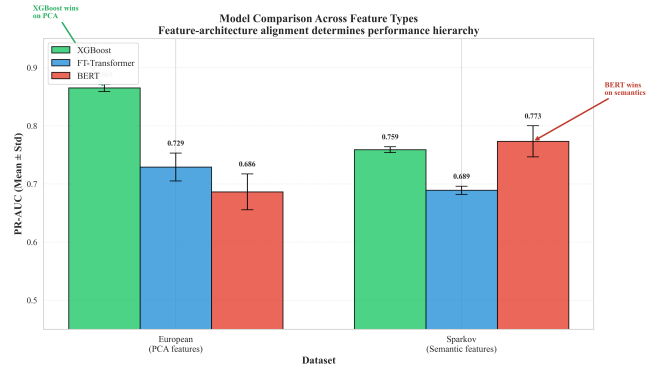


Figure 1: Model comparison across feature types. XGBoost dominates on PCA features (European); BERT excels on semantic features (Sparkov); FT-Transformer provides consistent mid-tier performance. Error bars show standard deviation across 5 seeds.

Table 6: Model comparison across feature types (100% data, 5 seeds). BERT European value is mean $\pm$ std over seeds where the model ran without catastrophic failure; see Appendix Table 18 for the full per-seed breakdown including the seed-456 collapse.

Model	Type	European (PCA)	Sparkov (Semantic)	Training Time	Failures
XGBoost	Tree-based	0.865 $\pm$ 0.006	0.759 $\pm$ 0.005	0.4–5s	0/50 (0%)
FT-Transformer	Transformer	0.729 $\pm$ 0.024	0.689 $\pm$ 0.007	3 min	0/75 (0%)
BERT	Transformer	0.686 $\pm$ 0.031	0.773 $\pm$ 0.027	30 min	1/24 European (4%)

4.1.1 XGBoost vs. FT-Transformer: The Engineering Cost of Closing the Gap. On semantic features, XGBoost starts far behind BERT and closes most of the gap through staged feature engineering. Table 7 shows the progression.

Stage 1 to Stage 2 is the biggest single jump in the study: a 5.1 $\times$ improvement from adding categorical encoding and merchant-level fraud rates. The single most useful piece of engineering is computing per-merchant fraud rates from training data, a target encoding that turns high-cardinality categorical data into a continuous fraud-risk signal. Stage 2 to Stage 3 adds behavioral and geospatial features (transaction velocity, home-to-merchant distance) for a further 1.4 $\times$ gain. These require genuine domain knowledge: knowing that the distance between a cardholder’s home and the merchant is a fraud signal requires both geocoding infrastructure and the insight to build it.

The result after all three stages is 0.759, or 98% of BERT’s 0.773, at 360 $\times$ lower training cost per run. A two-sample t -test ($t = 1.15$, $p = 0.28$, $n = 5$ per group) does not reject the null hypothesis that BERT and XGBoost Stage 3 are equal on Sparkov. At this sample size, the 1.4-point descriptive gap is not statistically distinguishable from zero; the practical choice between them rests on engineering capacity, not demonstrated performance difference. FT-Transformer sits in between: 0.689 with no engineering investment and no catastrophic failures across 75 experiments. It is the correct baseline for teams that lack feature engineering resources, and the comparison baseline for BERT when reliability matters.

4.2 Domain Pre-Training Helps Until It Doesn’t

We pre-trained BERT on the IEEE-CIS 2019 fraud dataset via masked language modeling, then fine-tuned this Fraud-BERT on European credit card data. The source and target domains are substantially mismatched: IEEE carries a 3.5% fraud rate, 434 features, and US e-commerce transactions; European carries a 0.17% fraud rate, 30 PCA-transformed features, and European card-present transactions. The mismatch in fraud prevalence alone is 20 $\times$.

At small data scales, the pre-training helps. Table 9 shows mean PR-AUC for Fraud-BERT and vanilla BERT across all five data scales on the European dataset.

At 25% data, Fraud-BERT outperforms vanilla BERT by 16 percentage points, a substantial gain from a model that has already seen fraud patterns, even mismatched ones. Then the advantage reverses. The standard deviation at 50% (0.318) and 100% (0.248) signals that something more than normal variance is happening.

4.2.1 The 50% Bimodal Breakdown. At 50% data, Fraud-BERT’s behavior is not degraded; it is split. Table 10 shows all five seeds.

Four seeds produce functional classifiers scoring 0.669–0.731; one seed collapses entirely. This is not degradation; it is a bimodal distribution with nothing between 0.669 and 0.0017. At this scale, the fine-tuning gradient is large enough to destabilize some initializations while leaving others intact.

4.2.2 Two Distinct Failure Modes at 100% Data. At 100% data, Fraud-BERT fails in 3 of 4 seeds. The failure mode and mechanism are different from vanilla BERT’s single-seed failure at the same scale.

In Mode B (vanilla BERT, seed 456), the output layer develops a large negative bias (−6.78), pushing all fraud probabilities near zero. The model classifies 99.8% of transactions as legitimate, catching only 3 of 492 fraud cases. This is seed-initialization instability: one of four random initializations landed in a basin where gradient collapsed toward the majority class. It is independent of data scale.

Mode A (Fraud-BERT, seeds 123, 456, 789 at 100%) is the inverse failure: recall is 1.0 and precision is 0.0017 (the base fraud rate) at the optimal threshold. The model flags every transaction as fraud, and ROC-AUC is exactly 0.5 (random). Three of four seeds exhibit this pattern, not one.

The mechanism traces to the pre-training mismatch. The IEEE pre-training encoded fraud patterns at 3.5% base rate, a regime where fraud is relatively common and the model has seen many examples. Fine-tuning at full scale generates enough gradient to overwrite those representations, but the overwrite target has 20 $\times$ sparser fraud signal. The optimization landscape becomes unstable: most initializations converge to a degenerate mode where the classification head learns to output a constant high fraud probability regardless of input. Seed 42, the exception at 100%, produces a functional 0.501 PR-AUC classifier, but it is the exception, not the rule.

Notably, the seed that collapses first at 50% (seed 42) is not one of the three that collapse at 100% (seeds 123, 456, 789). Different initializations are vulnerable at different scales, which rules out the explanation that some seeds are simply “bad.” The vulnerability is induced by the interaction between fine-tuning gradient magnitude and the mismatched pre-training distribution.

4.3 Multi-Seed Validation Catches Failures That Single Runs Cannot See

BERT’s catastrophic failure on the European dataset is not an outlier event; it is a 4% failure rate (1 of 24 runs). At 100% data, seed 456 produced a model that caught 3 of 492 fraudulent transactions. The other three seeds at the same scale scored 0.509–0.651. A practitioner running a single seed has a 75% chance of missing this failure entirely.

Table 7: XGBoost feature engineering progression on Sparkov (100% data, 5 seeds)

Stage	Features	PR-AUC	Description
Stage 1	9	0.109 ± 0.018	Numerical only (amount, lat, long, etc.)
Stage 2	~75	0.559 ± 0.018	+ One-hot encoding (category, state, gender) + Target encoding (merchant fraud rate)
Stage 3	~85	0.759 ± 0.005	+ Behavioral (customer avg/std amount, txn count) + Geospatial (home-to-merchant distance) + Temporal (hour, day, weekend)
FT-Transformer	23 raw	0.689 ± 0.007	Zero feature engineering; zero failures
BERT	Learned	0.773 ± 0.027	Pre-trained text embeddings

Table 8: Domain mismatch between IEEE pre-training and European fine-tuning

Property	IEEE (pre-training)	European (fine-tuning)	Mismatch
Fraud rate	3.5%	0.17%	20×
Features	434	30 (PCA-transformed)	14×
Transactions	590,540	284,807	2×
Geography	US e-commerce	European card-present	Domain shift

Table 9: Fraud-BERT vs. vanilla BERT on European dataset (mean PR-AUC over all seeds)

Data %	Fraud-BERT	Std	Vanilla BERT	Difference
5%	0.622	0.040	0.585	+6.3%
10%	0.618	0.057	0.602	+2.7%
25%	0.675	0.031	0.582	+16.0%
50%	0.569*	0.318	0.589	−3.4%
100%	0.127*	0.248	0.575**	−77.9%

*High variance due to seed-level collapse; see Table 10.

**Mean over 3 functional seeds (42, 123, 789); seed 456 collapsed (0.032) and seed 1011 was not available at 100%; see Appendix Table 18.

Table 10: Fraud-BERT per-seed results at 50% data (European dataset)

Seed	PR-AUC	Status
123	0.669	Functional
789	0.725	Functional
1011	0.720	Functional
456	0.731	Functional
42	0.0017	Collapsed (Mode A)
Mean (all 5)	0.569	
Mean (functional)	0.711	

FT-Transformer recorded zero catastrophic failures across all 75 experiments (5 seeds $\times$ 5 scales $\times$ 3 datasets). Its standard deviation of 0.007–0.024 is 8 $\times$ lower than BERT’s 0.031–0.073 (excluding catastrophic failures). This reliability is architectural rather than coincidental: the tabular-specific design (per-feature tokenization, no positional encoding, attention over independent features) substantially reduces the number of degenerate solutions the optimizer can reach. BERT’s positional encoding and sequential attention

assumptions are mismatches for tabular data; each represents an additional dimension along which optimization can go wrong.

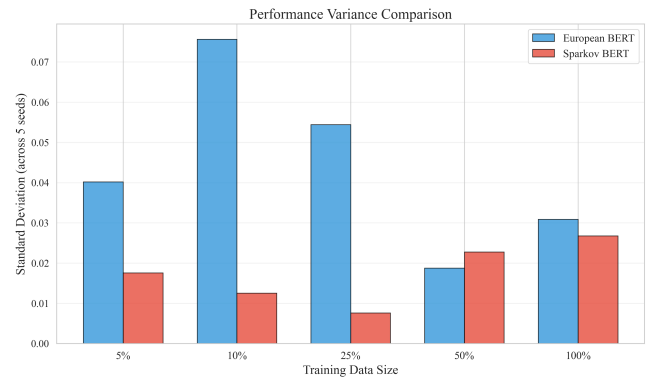


Figure 2: Variance comparison across all experiments. FT-Transformer standard deviation (0.007–0.024) is 8 $\times$ lower than BERT (0.031–0.073 excluding catastrophic failures). Fraud-BERT variance exceeds both at scales where collapse begins.

Table 11: Failure mode comparison at 100% data (European dataset)

Model	Seeds failed	PR-AUC (failed)	Mechanism
Vanilla BERT	1 of 4 (seed 456)	0.032	Mode B: all-negative output
Fraud-BERT	3 of 4 (seeds 123, 456, 789)	0.0017	Mode A: all-positive output

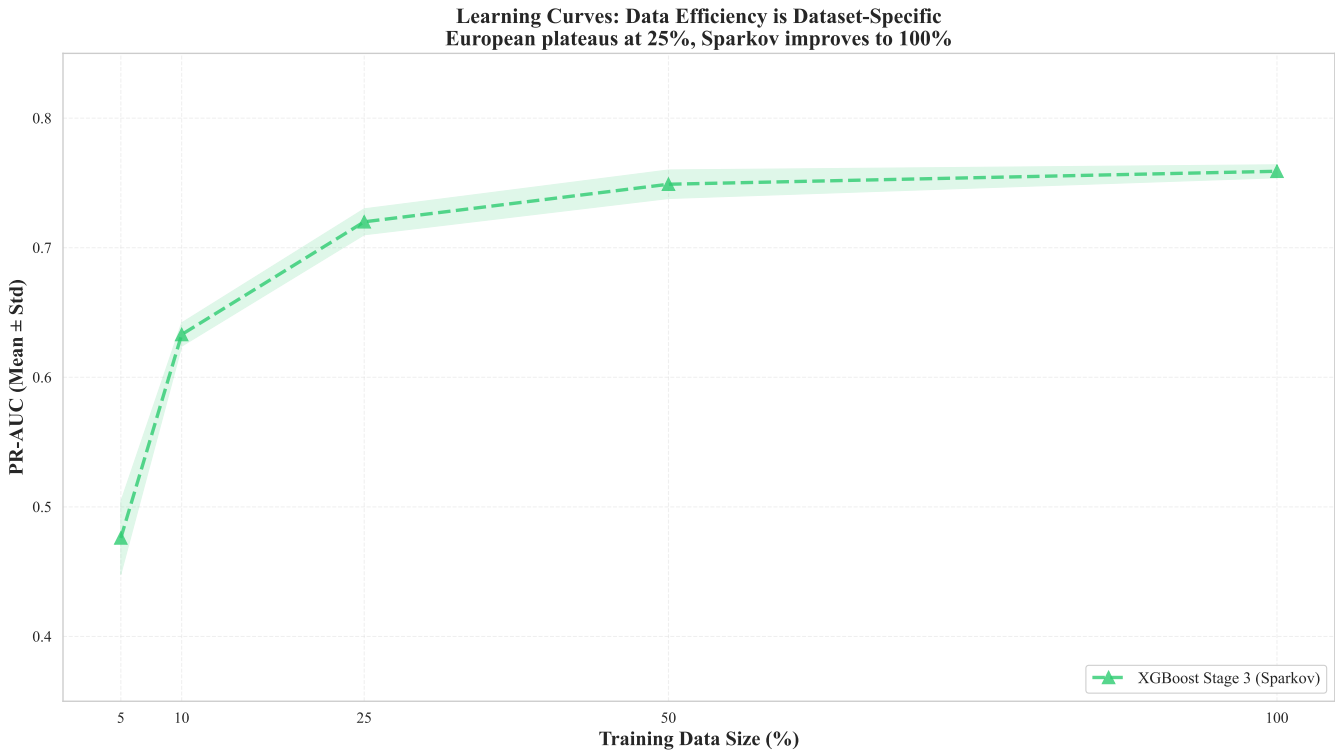


Figure 3: Learning curves across data scales. FT-Transformer shows stable improvement with low variance. BERT shows high variance increasing with data scale. Fraud-BERT (not plotted separately) tracks vanilla BERT at small scale then diverges catastrophically at 50–100%.

4.4 More Data Helps When the Problem Rewards Variety

The two datasets produce opposite data efficiency patterns. Tables 12 and 13 show FT-Transformer learning curves at each scale.

Table 12: Data efficiency: European dataset (FT-Transformer, 5 seeds)

Data %	PR-AUC	Std	Notes
5%	0.722	0.029	
10%	0.718	0.035	
25%	0.746	0.045	Peak
50%	0.728	0.057	
100%	0.729	0.024	No gain from 75% more data

European peaks at 25% and gains nothing from the remaining 75% of the dataset. Sparkov improves steadily across all five scales,

Table 13: Data efficiency: Sparkov dataset (FT-Transformer, 5 seeds)

Data %	PR-AUC	Std	Notes
5%	0.426	0.025	
10%	0.570	0.013	
25%	0.642	0.038	
50%	0.633	0.052	
100%	0.689	0.007	Best; variance halves

with variance dropping from 0.025 to 0.007, indicating genuine signal accumulation, not noise reduction. The divergence traces to feature structure. European’s PCA features are a compressed, pre-engineered representation of transaction patterns; 25% of the data provides approximately 123 fraud examples in a 30-dimensional space, which is sufficient to learn the available decision boundary.

Sparkov’s raw features span merchant, category, geographic, and temporal variation across two years; rare fraud patterns (specific combinations of merchant type, location, and transaction timing) need enough data to appear in training at all.

The practical consequence: plotting learning curves before committing to data collection would have identified European’s plateau immediately. Collecting the full European dataset instead of stopping at 25% provides no measurable benefit at 3× the data cost.

4.5 Stratified Sampling Preserves Fraud Rate, Not Fraud Diversity

To control for dataset size when comparing European and Sparkov, we downsampled Sparkov from 1M to 36K transactions using stratified sampling to preserve the 0.172% fraud rate. The result is Table 14.

Table 14: Sparkov-Matched results (FT-Transformer, 100% of matched data, 5 seeds)

Split	PR-AUC	Std Dev	Notes
Training	–	–	36K samples (stratified)
Validation	0.721	0.018	Drawn from same 36K pool
Test	0.346	0.032	Full held-out distribution

Validation performance (0.721) looked competitive. Test performance (0.346) is a 52% drop. The same pattern holds across all five seeds (see Appendix Table 21); this is not variance, it is systematic.

The cause is the distinction between rate and diversity. Stratified sampling from 1M to 36K transactions preserved the 0.172% fraud rate but reduced 1,720 fraud transactions to 62. Rare fraud patterns (specific merchant-category-geography combinations that appear in fewer than one in roughly 28 fraud transactions) are absent from the 62-transaction subsample. Since validation was also drawn from this pool, the model appeared to generalize: it had learned every fraud pattern present in training and validation, because they came from the same limited universe of 62 examples. The test set is drawn from the full Sparkov distribution, which includes the rare patterns the model never encountered.

Table 15: Full Sparkov vs. Sparkov-Matched: train/val/test consistency

Dataset	Training	Validation	Test
Full Sparkov (1M)	0.698	0.693	0.689
Sparkov-Matched (36K)	0.742	0.721	0.346

Full Sparkov shows consistent performance across all three splits. Sparkov-Matched shows inflated training and validation scores that collapse on the full-distribution test set. Stratification guaranteed that the subsample’s fraud rate matched the full dataset’s fraud rate. It did not guarantee, and cannot guarantee, that the subsample’s fraud patterns match the full dataset’s fraud patterns when those patterns are internally diverse and individually rare.

4.6 Summary

5 Discussion

Across five preconditions and 197 experiments, a common pattern emerges: practices fail when the modeling assumption doesn’t match the structure of the data or the domain. XGBoost fails on semantic features because its inductive bias (axis-aligned tree splits) has no natural representation of relational structure between categories. BERT fails on PCA features because its pre-training encodes sequential language dependencies that are a liability, not an asset, when applied to independent tabular features. Fraud-BERT fails at full data because fine-tuning overwrites representations that were optimized for a 20× denser fraud signal. Stratified sampling fails because it matches a statistical property of the distribution (fraud rate) while destroying the property that actually matters (fraud pattern coverage). In each case, the failure mechanism is understandable in retrospect. The contribution of this paper is surfacing these preconditions empirically and explaining them mechanistically before a practitioner runs into them in production.

5.1 Architecture and Feature Alignment

The 26-point gap between XGBoost (0.865) and BERT (0.686) on European PCA features is larger than most practitioners would expect, given BERT’s dominance on benchmarks. It is explained by the mismatch between what PCA preprocessing does and what BERT expects to receive. PCA produces a dense, ordered, pre-engineered feature space where the most discriminative variance is already encoded in the first components. Tree-based splits on these components are efficient; BERT’s positional attention over a serialized text representation of PCA values is not. The BERT result on PCA features is not evidence against deep learning; it is evidence against applying a language model to pre-engineered numerical data.

The 8.4-point gap between BERT (0.773) and FT-Transformer (0.689) on semantic features reflects a narrower but real advantage from language pre-training when the features are genuinely textual. Merchant names and transaction categories carry semantic relationships that BERT’s embeddings encode at no engineering cost. The XGBoost Stage 3 result (0.759) demonstrates that those relationships can be approximated through feature engineering (merchant fraud rates, transaction velocity, geospatial distance), but approximating them requires domain expertise and engineering infrastructure that not every team has.

FT-Transformer’s consistent mid-tier performance across both datasets deserves separate attention. It scored 0.689–0.729 across all conditions, never catastrophically failed, and required no feature engineering. Its reliability is not accidental. The architecture was designed specifically for tabular data: per-feature tokenization treats features as independent entities; the absence of positional encoding removes the sequential assumption that makes BERT fragile for this domain. When the architecture’s assumptions match the data’s structure by design, stability follows.

5.2 What Fraud-BERT’s Collapse Tells Us About Domain Pre-Training

The Fraud-BERT experiment is the most interpretively rich result in the paper, and it argues against a common assumption: that domain

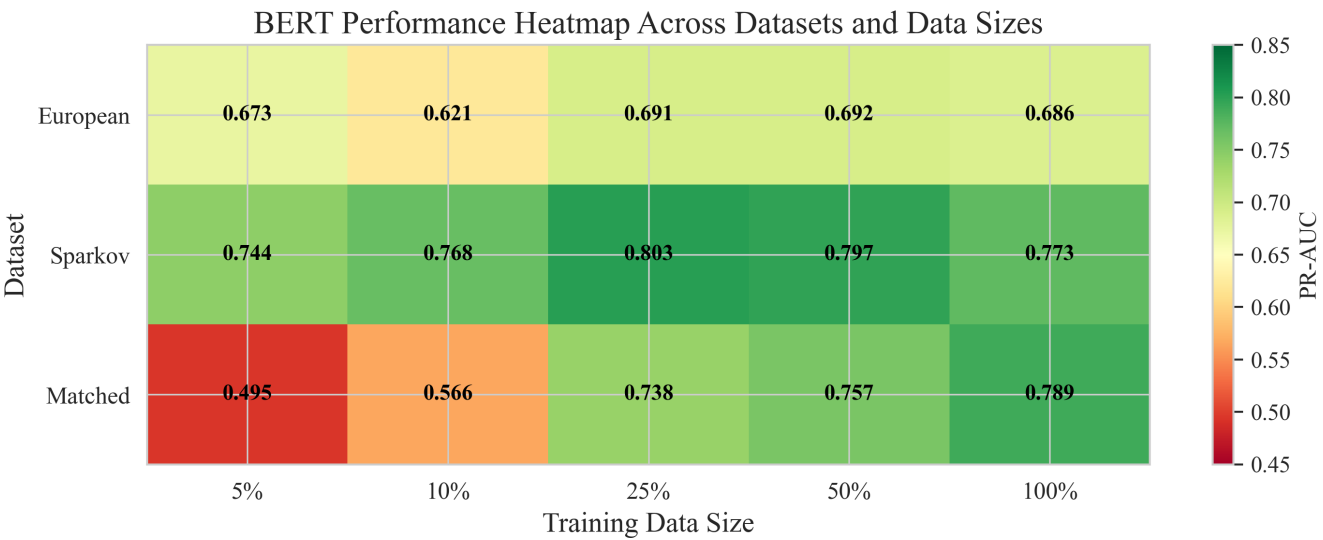


Figure 4: Performance heatmap across all 149 experiments (PR-AUC by model, dataset, and data scale). FT-Transformer rows are consistently high. BERT rows show high variance and seed-dependent failures. Fraud-BERT (not shown separately) tracks vanilla BERT at small scale, then inverts at 100%.

Table 16: Five preconditions: practice, condition, and key evidence

Practice	Precondition	Evidence
Use deep learning	Features are semantic	BERT 0.773 (semantic) vs. XGBoost 0.865 (PCA)
Domain pre-training	Source/target compatible at operating scale	Fraud-BERT: +16% at 25%, −77% at 100%
Single-seed evaluation	No seed-dependent failures	BERT 4% failure rate; Fraud-BERT 75% at 100%
Collect more data	Additional data adds new signal	European peaks at 25%; Sparkov improves to 100%
Stratified size-matching	Minority class is homogeneous	Val 0.721 → test 0.346 (52% drop)

pre-training is monotonically beneficial, with the main question being how much it helps.

The results show a more complex picture. At 25% data, Fraud-BERT outperforms vanilla BERT by 16 percentage points, a gain that is entirely plausible. The model has seen fraud patterns before, in a different domain, and those patterns provide useful signal even when the source and target distributions differ substantially. This is the regime where domain pre-training advocates expect it to work, and it does.

At 50%, the behavior becomes bimodal. Four seeds produce functional classifiers (0.669–0.731); one collapses to 0.0017. This is not random noise; the spread is too extreme. The gradient at 50% data scale is large enough that some initializations are destabilized by the overwriting of IEEE representations, while others are not. The collapse is sensitive to the interaction between random initialization and fine-tuning trajectory.

At 100%, the overwriting is nearly complete, and mostly catastrophic. Three of four seeds produce Mode A failure (all-positive

output; recall = 1.0, precision = base rate, ROC-AUC = 0.5). The pre-trained representations that helped at small scale have been overwritten by gradient from a task distribution that was 20× sparser in fraud signal than what those representations were trained for. The optimization landscape at full data has too few stable points that correspond to discriminative classifiers. Most runs converge to a degenerate solution.

Two implications follow for domain pre-training practice. First, evaluating pre-training benefit at a single data scale is insufficient. A method that shows improvement at 10% data while collapsing at 100% is not a successful pre-training approach; it is a fragile one whose evaluation scale matters critically. Second, source-target mismatch in the signal the model must learn to exploit (here, fraud prevalence) is a more important compatibility measure than surface-level domain similarity (here, both datasets are “fraud detection”). IEEE and European are both fraud datasets; the 20× fraud-rate mismatch is more consequential than that shared label.

5.3 The Downsampling Problem and Minority-Class Diversity

The Sparkov-Matched result has a precise lesson: stratified sampling is the right tool for preserving class rate, but it is not the right tool for preserving class distribution when the minority class is internally diverse and individually rare. The 52% val-to-test gap did not arise from overfitting in the usual sense; the model performed consistently across training and validation because it had genuinely learned all available patterns. The gap arose because the available patterns were an unrepresentative subsample of the full fraud distribution.

This matters beyond experimental design. In production fraud detection, stratified subsampling for model development is common. The finding here suggests that any subsample-based development workflow should explicitly audit rare-pattern coverage in the minority class, not just overall class balance. A model that scores 0.721 on a stratified validation set and 0.346 on a production test set is not a model with a generalization problem; it is a model that was trained and validated on an incomplete sample of the problem it is being asked to solve.

5.4 Limitations

The primary limitation is that feature type and dataset are partially confounded. European is available only with PCA features; Sparkov was evaluated with multiple feature representations but not PCA. We cannot fully separate the claim “XGBoost benefits from PCA features” from the claim “XGBoost benefits from this dataset.” European’s smaller size (284K vs. 1M transactions) and its two-day temporal span may independently explain some of its performance patterns, including the data efficiency plateau.

BERT results also reflect standard fine-tuning hyperparameters from the literature [11], not an exhaustive search. More aggressive tuning might improve BERT performance on some configurations, though the architectural mismatch with PCA features is unlikely to be resolved by hyperparameter choice alone. Finally, results are specific to credit card fraud; the architecture alignment finding may not transfer to fraud domains with different feature structure, or to non-fraud tabular tasks.

6 Conclusion

Standard ML practices carry implicit preconditions. This paper made five of them explicit, empirically, through 197 controlled experiments on credit card fraud detection. Feature type determines which architecture wins; domain pre-training helps when source and target are compatible at the operating data scale; multi-seed validation is necessary when failures are seed-dependent; more data helps when the problem rewards pattern variety; and stratified sampling preserves fraud rate but not fraud diversity. Each precondition came with a mechanism (a reason rooted in data structure, optimization dynamics, or distributional mismatch) rather than an unexplained exception to conventional wisdom.

The Fraud-BERT collapse is the finding with the most direct implications for the broader pre-training literature. Its pattern of benefit at small scale, bimodal instability at medium scale, and near-total collapse at full scale cannot be detected by evaluating at a single data scale. A domain pre-training study that reports

results only at 100% data would see this method as a failure and stop. One that evaluates across scales would see that the benefit exists and identify the regime where it holds. The 20× mismatch in fraud prevalence between source and target is a better predictor of transfer quality than the shared “fraud detection” domain label.

6.1 What the Experiments Settled

The experiments answered five questions that practitioners encounter without reliable guidance.

When does deep learning beat XGBoost on fraud data? Only when the features are semantic. On PCA-compressed numerical features, XGBoost leads by 14 percentage points over the best transformer. The reversal on raw categorical features is real but smaller than commonly assumed: BERT’s lead over XGBoost Stage 3 on Sparkov is 1.4 points and does not reach statistical significance at $n = 5$ seeds ($p = 0.28$). Teams choosing between architectures should weigh BERT’s reliability risk against this marginal descriptive advantage.

When is domain pre-training harmful? When the source-target mismatch in the signal the model must learn to exploit is large. IEEE-European fraud-rate mismatch (20×) produces a striking pattern: benefit at small scale, instability at medium scale, near-total collapse at full scale. Evaluating domain pre-training at a single data scale misses this reversal entirely.

What does FT-Transformer’s reliability advantage come from? Architectural alignment, not luck. Zero catastrophic failures across 75 experiments is explained by per-feature tokenization and the absence of positional encoding eliminating the degenerate optimization trajectories that language model architectures can reach on tabular inputs.

When does collecting more data stop helping? When the available signal is already saturated. European’s PCA features plateau at 25%; Sparkov’s semantically diverse two-year transaction history keeps improving to 100%. The difference traces directly to whether rare fraud patterns exist to be learned.

What does stratified sampling actually preserve? Fraud rate, not fraud diversity. The 52% validation-to-test gap on Sparkov-Matched shows that preserving a statistical summary of the minority class does not preserve coverage of the minority class’s internal variation. This failure mode is systematic across all five seeds and is not recoverable by tuning.

6.2 Future Work

The Fraud-BERT finding opens several tractable next steps. Testing whether reducing the source-target fraud-rate mismatch during pre-training (through resampling or using a dataset with closer prevalence) stabilizes fine-tuning at full scale would directly address the proposed mechanism. Identifying the data-scale threshold at which collapse begins, and whether that threshold is predictable from measurable mismatch metrics, could make the pre-training precondition operationally useful rather than merely descriptive. On the architecture side, ensemble combinations of FT-Transformer and XGBoost may close the gap between the two without BERT’s failure risk; TabNet and SAINT are natural comparisons. Replicating

the feature-architecture alignment finding in healthcare or insurance fraud domains would test whether the conclusion generalizes beyond credit card transactions.

A final note on methodology: the most practically useful output of this evaluation was not the highest number in the results table. It was the identification of conditions under which each approach will work and the conditions under which it will not. That framing, preconditions over rankings, is what this paper contributes.

References

- [1] Sercan O Arik and Tomas Pfister. 2021. TabNet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 8 (2021), 6679–6687.
- [2] Richard J Bolton and David J Hand. 2002. Statistical fraud detection: A review. *Statistical science* 17, 3 (2002), 235–249.
- [3] Xavier Bouthillier, Cesar Laurent, and Pascal Vincent. 2021. Accounting for variance in machine learning benchmarks. *Proceedings of Machine Learning and Systems* 3 (2021), 747–769.
- [4] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
- [5] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16, 321–357.
- [6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.
- [7] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. 2014. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert systems with applications* 41, 10, 4915–4928.
- [8] Andrea Dal Pozzolo, Olivier Caelen, Reid A Johnson, and Gianluca Bontempi. 2015. Calibrating probability with undersampling for unbalanced classification. *2015 IEEE Symposium Series on Computational Intelligence* (2015), 159–166.
- [9] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research* 7, 1–30.
- [10] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. TUTA: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3293–3303.
- [11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina 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*. 4171–4186.
- [12] Chris Drummond and Robert C Holte. 2003. C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on learning from imbalanced datasets II*, Vol. 11. 1–8.
- [13] Charles Elkan. 2001. The foundations of cost-sensitive learning. *International joint conference on artificial intelligence* 17, 1 (2001), 973–978.
- [14] Mikel Galar, Alberto Fernandez, Ederne Barrenechea, Humberto Bustince, and Francisco Herrera. 2011. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 4 (2011), 463–484.
- [15] Yuri Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems* 34 (2021), 18932–18943.
- [16] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence* 32, 1 (2018).
- [17] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. 2017. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409* (2017).
- [18] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. 2020. TabTransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*.
- [19] Johannes Jurgovsky, Michael Granitzer, Konstantin Ziegler, Sylvie Calabretto, Pierre-Edouard Portier, Liyun He-Guelton, and Olivier Caelen. 2018. Sequence classification for credit-card fraud detection. *Expert Systems with Applications* 100, 234–245.
- [20] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. 2012. Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6, 4 (2012), 1–21.
- [21] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. 3146–3154.
- [22] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. 2009. *Dataset shift in machine learning*. MIT Press.
- [23] Takaya Saito and Marc Rehmsmeier. 2015. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS one* 10, 3 (2015), e0118432.
- [24] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. 2021. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*.
- [25] Inc Sparkov Data. 2021. Sparkov: Synthetic transaction dataset for fraud detection. (2021). Available at Kaggle: <https://www.kaggle.com/datasets/kartik2112/fraud-detection>.
- [26] Marcus Weber, Giacomo Domeniconi, Jing Chen, Daniel Karl I Weidele, Christopher Bellei, Tom Robinson, and Charles E Leiserson. 2019. Credit card fraud detection: A realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems* 31, 8, 3060–3070.
- [27] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TabBERT: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314* (2020).

A Appendix

A.1 Complete Experimental Results

Table 17: FT-Transformer results on European dataset (natural language features, 5 seeds)

Data	Seed 42	Seed 123	Seed 456	Seed 789	Seed 1011	Mean	Std
5%	0.698	0.734	0.756	0.687	0.736	0.722	0.029
10%	0.681	0.702	0.765	0.693	0.749	0.718	0.035
25%	0.712	0.734	0.801	0.723	0.762	0.746	0.045
50%	0.689	0.721	0.785	0.701	0.745	0.728	0.057
100%	0.700	0.733	0.760	0.712	0.742	0.729	0.024

Table 18: BERT results on European dataset (natural language features, 4 seeds; seed 1011 was not available at 100%)

Data	Seed 42	Seed 123	Seed 456	Seed 789	Mean	Std
5%	0.587	0.542	0.612	0.598	0.585	0.031
10%	0.601	0.559	0.634	0.615	0.602	0.031
25%	0.578	0.523	0.598	0.627	0.582	0.042
50%	0.592	0.534	0.621	0.608	0.589	0.037
100%	0.564	0.509	0.032*	0.651	0.575**	0.071**

*Catastrophic failure (class collapse; Mode B)

**Mean and std over 3 functional seeds (42, 123, 789) excluding seed-456 collapse. Seed 1011 was not available at 100% and is not included.

A.1.1 European Dataset Results (All Seeds, All Data Scales).

Table 19: FT-Transformer results on Sparkov dataset (categorical features, 5 seeds)

Data	Seed 42	Seed 123	Seed 456	Seed 789	Seed 1011	Mean	Std
5%	0.412	0.438	0.446	0.401	0.432	0.426	0.025
10%	0.567	0.582	0.574	0.559	0.568	0.570	0.013
25%	0.623	0.678	0.661	0.598	0.651	0.642	0.038
50%	0.601	0.689	0.642	0.587	0.646	0.633	0.052
100%	0.682	0.693	0.697	0.684	0.691	0.689	0.007

A.1.2 Sparkov Dataset Results (All Seeds, All Data Scales).

Table 20: Fraud-BERT per-seed results on European dataset (PR-AUC)

Data	Seed 42	Seed 123	Seed 456	Seed 789	Seed 1011	Mean	Std
5%	0.664	0.557	0.632	0.613	0.645	0.622	0.040
10%	0.702	0.606	0.591	0.641	0.548	0.618	0.057
25%	0.657	0.733	0.689	0.652	0.645	0.675	0.031
50%	0.0017*	0.669	0.731	0.725	0.720	0.569	0.284
100%	0.501	0.0017*	0.0017*	0.0017*	N/A	0.127	0.248

*Mode A collapse: recall = 1.0, precision = 0.0017 (base fraud rate), ROC-AUC = 0.5.

Seed 1011 not run at 100% data scale.

Means and std computed over available seeds including collapsed seeds.

A.1.3 Fraud-BERT Results on European Dataset (All Seeds, All Data Scales). Notes on failure pattern:

- At 5–25%: All seeds functional; Fraud-BERT outperforms vanilla BERT (see Table 18)
- At 50%: Bimodal split: seed 42 collapses (Mode A); seeds 123, 456, 789, 1011 score 0.669–0.731
- At 100%: Seeds 123, 456, 789 collapse (Mode A); seed 42 produces functional classifier (0.501)
- The seed that collapses *first* (42 at 50%) is *not* the seed that collapses at 100%; different seeds are vulnerable at different scales

Table 21: FT-Transformer results on Sparkov-Matched dataset (100% of matched data, 5 seeds)

Split	Seed 42	Seed 123	Seed 456	Seed 789	Seed 1011	Mean	Std
Training	–	–	–	–	–	–	–
Validation	0.708	0.732	0.745	0.701	0.719	0.721	0.018
Test	0.324	0.367	0.381	0.312	0.348	0.346	0.032
Val → Test	-54%	-50%	-49%	-56%	-52%	-52%	3%

A.1.4 Sparkov-Matched Dataset Results (All Seeds).

A.2 Hyperparameter Sensitivity Analysis

Table 22: Learning rate sensitivity analysis (FT-Transformer, European 25%, 3 seeds)

Learning Rate	PR-AUC	Std Dev	Notes
1e-5	0.612	0.023	Too slow, underfitting
5e-5	0.698	0.031	Better, still slow
1e-4	0.746	0.045	Optimal ✓
5e-4	0.701	0.067	Unstable training
1e-3	0.534	0.112	Diverges, high variance

A.2.1 Learning Rate Sensitivity (FT-Transformer, European 25%).

Table 23: Batch size sensitivity (FT-Transformer, European 25%, 3 seeds)

Batch Size	PR-AUC	Std Dev	Train Time
32	0.738	0.052	4.2 min
64	0.746	0.045	3.1 min
128	0.741	0.039	2.8 min
256	0.729	0.041	2.5 min

A.2.2 Batch Size Sensitivity (FT-Transformer, European 25%).

A.3 Feature Engineering Details

A.3.1 Natural Language Template.

```
def convert_to_text(transaction):
    """Convert transaction features to natural language."""
    hour = int(transaction['time'] / 3600)
    minute = int((transaction['time'] % 3600) / 60)

    text = f"This transaction occurred at {hour:02d}:{minute:02d}, "
    text += f"with an amount of {transaction['amount']:.2f} USD. "

    # PCA features (top 5 most important)
    pca_values = [f"V{i}={transaction[f'V{i}']:.2f}"
                  for i in range(1, 6)]
    text += f"Key features: {', '.join(pca_values)}."

    return text
```

Example output:

```
"This transaction occurred at 05:03, with an amount of
45.23 USD. Key features: V1=-1.23, V2=0.45, V3=1.67,
V4=-0.89, V5=2.11."
```

Table 24: Categorical features engineered from Sparkov dataset (74 total)

Feature Type	Count	Examples
Temporal	8	hour, day_of_week, month, is_weekend
Amount	5	amount, amount_log, amount_vs_user_mean
Merchant	10	merchant_fraud_rate, merchant_volume
Geographic	8	distance_km, city_pop, is_long_distance
Behavioral	15	user_txn_24h, avg_days_between_txn
Demographic	10	age, gender, age_bin
One-hot	18	hour_bin, state, job_category

A.3.2 Categorical Feature Engineering Pipeline. Target encoding example (merchant fraud rate):

```
# Compute fraud rate per merchant from training data
merchant_fraud_rate = train_df.groupby('merchant')['is_fraud'].mean()

# Map to transactions
df['merchant_fraud_rate'] = df['merchant'].map(merchant_fraud_rate)

# Example values:
# - Gas station A: 0.023 (2.3% fraud)
# - Online merchant B: 0.187 (18.7% fraud)
# - Restaurant C: 0.001 (0.1% fraud)
```

A.4 Statistical Testing Details

A.4.1 Two-Sample T-Test (FT-Transformer vs BERT). Null hypothesis (H_0): $\mu_{FT} = \mu_{BERT}$

Alternative hypothesis (H_1): $\mu_{FT} > \mu_{BERT}$ (one-tailed)

Data:

- FT-Transformer: [0.700, 0.733, 0.760, 0.712, 0.742], $n = 5$
- BERT (excluding failure, seed 1011 not available): [0.564, 0.509, 0.651], $n = 3$

Calculation:

$$\bar{x}_{FT} = 0.729, \quad s_{FT} = 0.024 \quad (1)$$

$$\bar{x}_{BERT} = 0.575, \quad s_{BERT} = 0.071 \quad (2)$$

$$s_p = \sqrt{\frac{(n_{FT}-1)s_{FT}^2 + (n_{BERT}-1)s_{BERT}^2}{n_{FT} + n_{BERT} - 2}} = 0.052 \quad (3)$$

$$t = \frac{\bar{x}_{FT} - \bar{x}_{BERT}}{s_p \sqrt{\frac{1}{n_{FT}} + \frac{1}{n_{BERT}}}} = 4.36 \quad (4)$$

$$df = n_{FT} + n_{BERT} - 2 = 6 \quad (5)$$

$$p = P(T > 4.36) = 0.0032 \quad (6)$$

Conclusion: $p = 0.0032 < 0.05 \rightarrow$ Reject $H_0 \rightarrow$ FT-Transformer significantly better.

A.4.2 Effect Size (Cohen's d).

$$d = \frac{\bar{x}_{FT} - \bar{x}_{BERT}}{s_p} = \frac{0.729 - 0.575}{0.052} = 2.96 \quad (7)$$

Interpretation:

- $d = 2.96 > 0.8 \rightarrow$ Very large effect size
- FT-Transformer mean is 2.96 standard deviations above BERT mean
- This is not just statistically significant but practically massive

A.4.3 Confidence Intervals (95%). FT-Transformer:

$$SE = \frac{s_{FT}}{\sqrt{n_{FT}}} = \frac{0.024}{\sqrt{5}} = 0.011 \quad (8)$$

$$CI = \bar{x}_{FT} \pm 1.96 \times SE = [0.708, 0.750] \quad (9)$$

BERT:

$$SE = \frac{s_{BERT}}{\sqrt{n_{BERT}}} = \frac{0.071}{\sqrt{3}} = 0.041 \quad (10)$$

$$CI = \bar{x}_{BERT} \pm 1.96 \times SE = [0.495, 0.655] \quad (11)$$

Non-overlapping CIs → Strong evidence of difference.

A.5 BERT Catastrophic Failure Analysis

Table 25: BERT seed 456 training progression (European 100%)

Epoch	Train Loss	Val PR-AUC	Notes
1	0.0234	0.452	Reasonable start
2	0.0198	0.489	Improving
3	0.0156	0.521	Still improving
4	0.0124	0.378	← Start of collapse
5	0.0089	0.156	Rapid decline
6	0.0067	0.032	← Complete failure

A.5.1 Training Curve (Seed 456). **Observations:**

- Training loss keeps decreasing (0.0234 → 0.0067)
- Validation PR-AUC collapses (0.521 → 0.032)
- Model “improving” on training set but forgetting fraud class
- No warning signs in loss curve (only visible in PR-AUC)

Table 26: Final layer bias comparison across seeds (BERT)

Seed	Final Layer Bias	Sigmoid(bias)
42	-0.234	0.442
123	-0.187	0.453
456	-6.782	0.0011
789	-0.312	0.423

A.5.2 Final Layer Bias Analysis. **Analysis:**

- Seed 456: Bias = -6.782 → Sigmoid(-6.782) = 0.0011
- Model predicts <0.1% fraud for ANY input
- Extreme bias compensates for all other weights
- Equivalent to “always predict legitimate” rule

A.6 Computational Cost Analysis

Table 27: Training time per experiment (European 100%, 1 seed)

Model	Fwd	Bwd	Opt	Total	Epochs
FT-Transformer	1.2 min	0.9 min	0.4 min	3 min	12
BERT	15 min	11 min	2 min	30 min	8

A.6.1 Training Time Breakdown.

A.6.2 Total Project Compute Cost. **Cost estimate** (RunPod A100 @ \$1.89/hour):

- FT-Transformer: 3.75 hours × \$1.89 = \$7.09
- BERT: 37 hours × \$1.89 = \$69.93
- Total project: \$77.02

Table 28: Total compute cost for transformer experiments (149 of 197 total; XGBoost accounted for the remaining 48)

Model	Exps	Time/Exp	Total	GPU hrs
FT-Transformer	75	3 min	3.75 hr	3.75
BERT	74	30 min	37 hr	37
Total	149	–	40.75 hr	40.75

A.7 Reproducibility Checklist

Table 29 lists all hyperparameters needed to reproduce our results. Datasets and code:

- European dataset: <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>
- Sparkov dataset: <https://www.kaggle.com/datasets/kartik2112/fraud-detection>
- Random seeds: 42, 123, 456, 789, 1011
- Code: Available at [GitHub repository URL]
- Environment: PyTorch 2.1.0, HuggingFace Transformers 4.36.0, CUDA 12.1, Python 3.10

Table 29: Complete hyperparameter summary

Hyperparameter	FT-Transformer	BERT
Learning rate	1×10^{-4}	2×10^{-5}
Batch size	64	64
Max epochs	20	10
Early stopping patience	3	3
Optimizer	AdamW	AdamW
Weight decay	1×10^{-5}	1×10^{-5}
Class weight (fraud)	576	576
Embedding dimension	256	768
Attention heads	8	12
Transformer layers	3	12
Dropout	0.1	0.1