Neuralk

THE PREDICTIVE AI COMPANY

WHITEPAPER - SEPTEMBER 2026

Tabular Foundation Models: The basics

A guide to Predictive AI

Neuralk

THE PREDICTIVE AI COMPANY

TABLE OF CONTENT

EXECUTIVE SUMMARY

Enterprise operations rely primarily on tabular data: customer records, financial transactions, inventory, sensor logs, and the relational databases that drive business decisions. Yet, while foundation models have revolutionized AI for unstructured content like text, images, and code, predictive machine learning on tabular data has not yet experienced its foundational moment. It remains anchored to legacy methods, dominated by algorithms such as XGBoost or LightGBM that require intense manual feature engineering, custom training, and bespoke optimization for every new dataset.

This is now changing with the advent of Tabular Foundation Models (TFMs).

Tabular Foundation Models introduce a different approach. Instead of learning each prediction task from scratch, they are pre-trained on millions of synthetic datasets, enabling them to recognize statistical patterns and make accurate predictions on previously unseen tables through in-context learning. This significantly reduces the time and expertise required to build predictive models while delivering competitive performance, particularly on small and medium-sized datasets.

This whitepaper explains how Tabular Foundation Models work, why they represent an important evolution in predictive AI, where they outperform traditional machine learning approaches and where conventional methods remain the better choice.

It also explores the research challenges shaping the field and examines how TFMs, together with large language models, are beginning to redefine enterprise AI.

01 • THE FOUNDATION PROBLEM FOR ENTERPRISE DATA

Over the last several years, AI has undergone a massive shift in how it handles text, images, voice, video, and code. Each of these data types had its own foundation-model moment.

> **Whats is a foundational model?** A foundation model is a large AI model **pre-trained** on massive amounts of data, designed to be applied across many different tasks without needing to be retrained from scratch each time.

GPT-4 is a foundation model, specifically a large language model, or LLM. The internet gave these models an infinite pretraining corpus of trillions of words to learn language structure from. Because of this, they tackle translation, coding, and creative writing well without task-specific retraining. LLMs are designed to read continuous text word-by-word, not grids of rows and columns *(fig. 1)*.

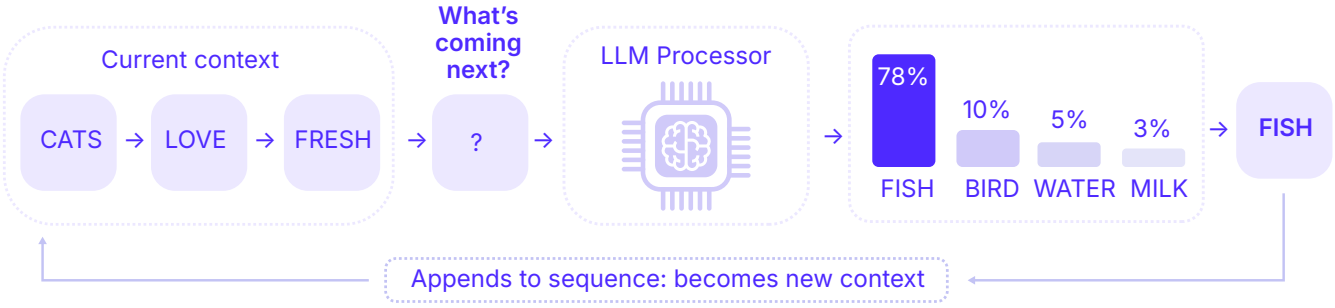


Figure 1

However, understanding language logic doesn't mean understanding structured business logic.

Ask the same model to predict which customers will churn next month from company CRM data, and the picture changes.

To process a table, an LLM has to squish the grid into a flat line of text, treating numbers like words. This flattens the table and breaks the hidden connections between your data. Tables organize data in grids, connecting information in all directions. AI language models only guess the next word in a single line, making them fundamentally unfit for grid-based data.

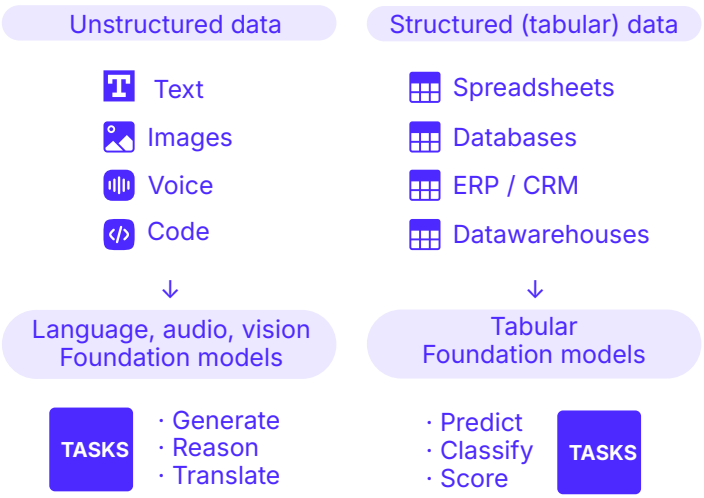


Figure 2

There's another branch of AI that's been working quietly to address this mismatch. It doesn't write poetry or generate images. It predicts what happens next, using the structured, tabular data that powers every critical decision in the enterprise (*fig 2*).

For years, the standard answer for tabular data has been gradient-boosted decision trees. Tools like XGBoost and LightGBM have powered business decisions. They perform very well, but they have zero memory. Because they never transfer knowledge between problems, every new dataset requires building the model over and over (*fig 3*). Tabular Foundation Models (TFMs) are the first serious attempt to change that.

What is tabular data? Tabular data is structured information organised in rows and columns, the records living in your ERP, CRM, data warehouse, databases and spreadsheets.

A TFM applies the foundation-model philosophy to structured data. Instead of learning from text or images, it learns from millions of tables with varying structures, column types, and relationships. The goal is to understand the deep patterns that exist across all tabular problems, from predicting house prices to classifying disease to forecasting demand.

The key insight: a customer-churn dataset looks nothing like a medical-diagnosis dataset but the underlying statistical patterns and relationships share surprising similarities. By training on vast, diverse tabular data, TFMs develop emergent **in-context learning capabilities**, the ability to recognize patterns and apply that knowledge to tables they've never seen before. That's something gradient-boosted trees, trained afresh on each dataset, simply can't do.

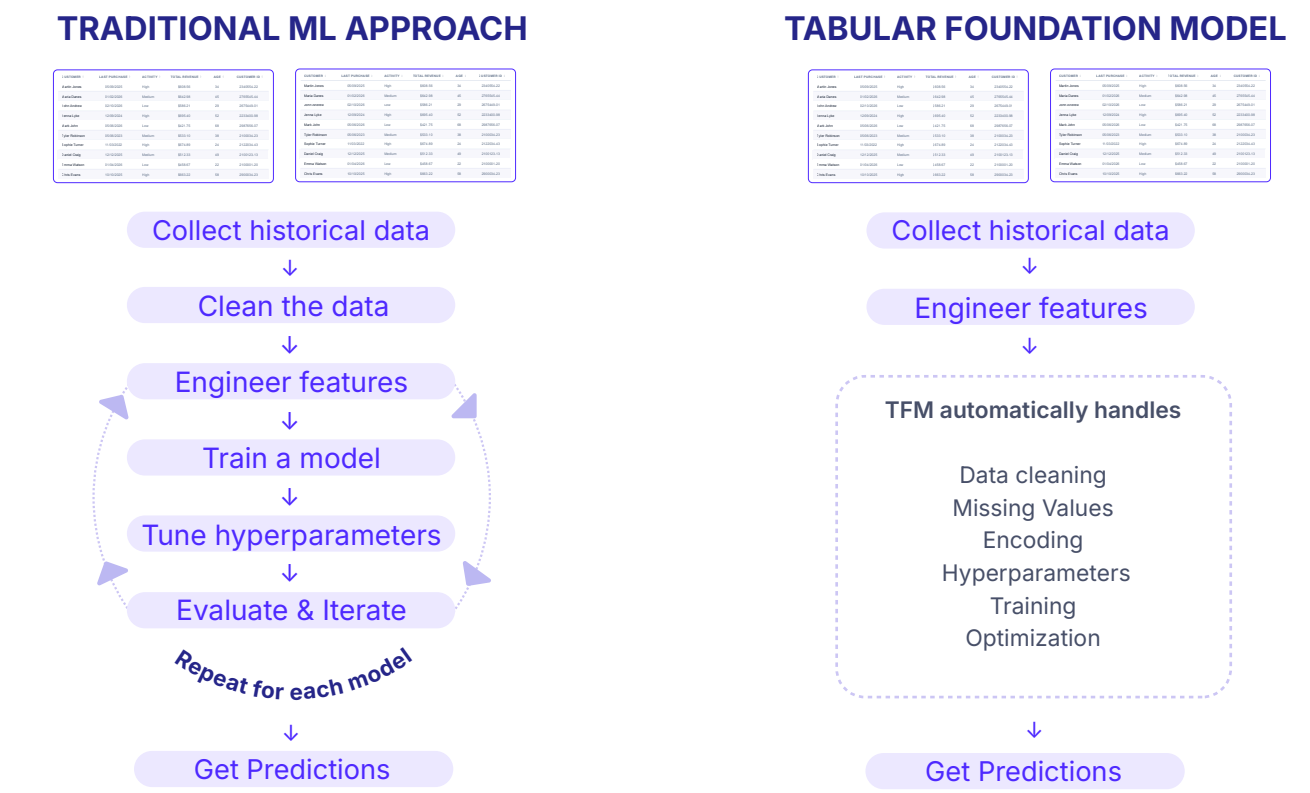


Figure 3

02 • HOW IT ACTUALLY WORKS

The architecture behind most TFMs is called a Prior-Data Fitted Network (PFN), a transformer, the same family of architecture behind LLMs, but trained to approximate Bayesian inference rather than language.

What is a transformer? A transformer is a neural network architecture that learns how different parts of information relate to each other by using context samples rather than processing step by step. It powers ChatGPT and other large language models.

> **Whats is in-context learning?** Making predictions by treating training examples as context at the time of prediction, rather than by adjusting model weights beforehand.

In practice, this means the model isn't trained on your data at all.

It's pretrained once on millions of synthetic tables. What it learns there is how columns relate to a target, how to separate signal from noise. None of that pretraining touches customer data.

At prediction time, it doesn't retrain or memorize. It treats your dataset as context and reasons over it in a single forward pass, the same way an LLM reasons over a prompt instead of updating its weights (*fig. 4*).

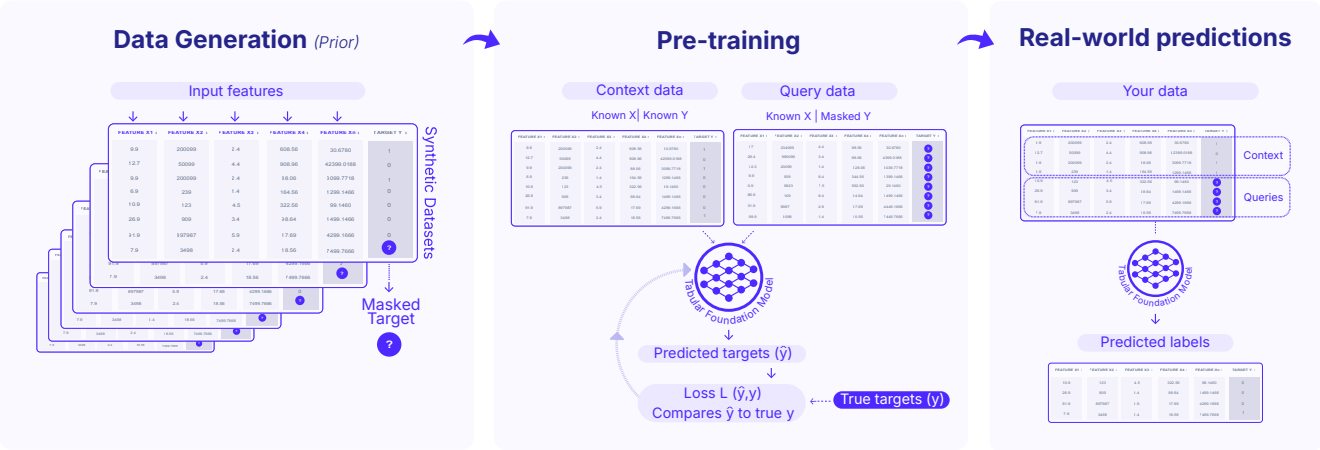


Figure 4

03 • WHY THIS MATTERS IN PRACTICE

- **Speed:** Traditional machine learning requires data cleaning, feature engineering, model selection, and hyperparameter tuning, typically several days of work. A TFM can return competitive predictions in seconds once the data is loaded.
- **Small or messy data:** Traditional methods need a reasonable volume of clean examples to find reliable patterns. A TFM performs better on small datasets (roughly under 10,000 rows) and tolerates missing values with less preprocessing.
- **Simplicity:** There's no equivalent of tuning dozens of hyperparameters. Data goes in, predictions come out. Less specialist expertise is needed to get a working model.
- **Many models at once**: The cost of traditional ML is not the first model, it's the fiftieth. Each one adds a pipeline to maintain and a retraining cycle to schedule. A TFM adds a dataset.

04 • HOW DID WE GET THERE?

Understanding tree-based methods makes it easier to see what a tabular foundation model changes and what it doesn't.

1. Decision Trees

Decision trees make predictions by following **a sequence of if-then rules.** Imagine predicting whether a loan applicant will default.

You will use the data you have on them and apply this **if / then** decision tree (*fig. 5*):

> Is their Annual revenue above $50k ? YES NO
>
> If YES, is their credit score above 700? If YES, predict Won't default
>
> If NO, do they have more than 3 existing loans? If YES, predict Will default

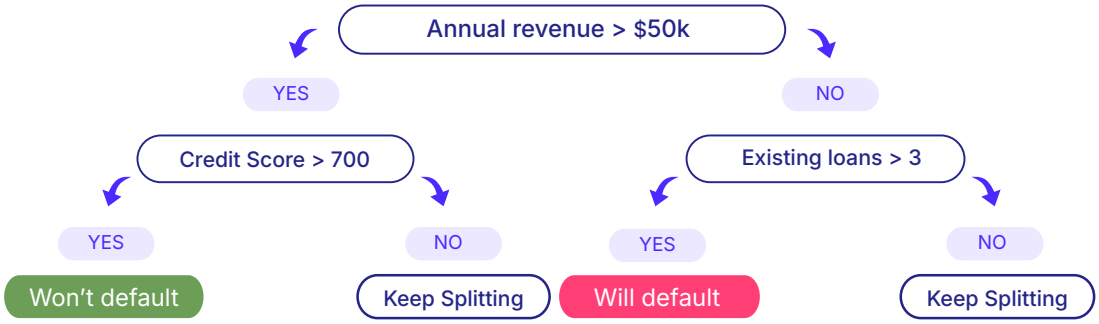


Figure 5

2. Random forests: the wisdom of crowds

Introduced by Leo Breiman in 2001, random forests address that by building hundreds of trees. Each tree trains on a different bootstrap sample of the data and at every split, it's only allowed to choose from a random subset of features rather than all of them.

Each tree trains on a different bootstrap sample, an approach called bagging (bootstrap aggregating). Combining many trees' predictions by majority vote is what turns a set of individually error-prone learners into something more stable than any single tree. It's the same logic behind a crowd's average guess beating most individual guesses.

The random feature subsetting at each split is one of the extra ingredients specific to random forests, with plain bagging. It's what stops the trees from all making the same mistakes in the same place (*fig. 6*).

Figure 6

3. Gradient Boosting

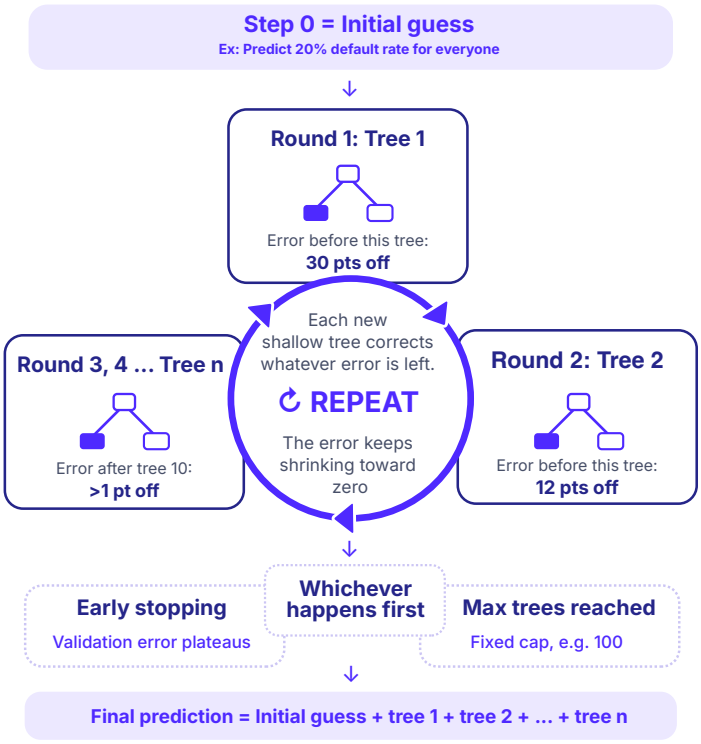


Figure 7

Gradient boosting builds trees sequentially instead of independently (*fig. 7*).

Each new tree is trained to correct whatever error is still left by the combined predictions of every tree before it, systematically closing the gap rather than relying on votes to cancel out mistakes.

XGBoost (2016), LightGBM (2016-17), and CatBoost (2017-18) are optimized implementations of this idea and despite the recent rise of tabular foundation models, they remain the benchmark that any new tabular method still gets measured against today.

4. Logistic Regression

Despite the name, logistic regression is a classification method, not a regression one.

It remains common wherever interpretability matters as much as raw accuracy: each feature's weight shows its effect on the prediction's log-odds (*fig. 8*), so you can point to exactly which factors pushed a decision up or down, and by how much, once features are on comparable scales. That transparency is why regulated industries like finance and healthcare often prefer it, even when a more complex model would score higher on pure accuracy.

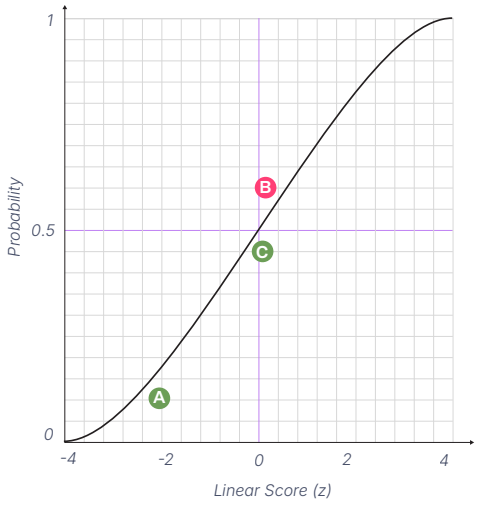
Step 1: compute z

Combine the features

$z = (W_1 * income) + (W_2 * creditscore) + (W_3 * age) + (\dots) + (Wn * feature\ n)$

z < 0 -> won't default
z > 0 -> will default

Step 2: decide with a threshold



Output: probability of default

Predicted to default if P > 0.5

- ***Customer A: z= -2.1 → P=0.11 →*** Won't default
- ***Customer B: z= 0.4 → P=0.6 →*** Will default
- ***Customer C: z= -0.2 → P=0.45 →*** Won't default

Figure 8

05 • THE TRADITIONAL MACHINE LEARNING WORKFLOW

So how do data scientists actually use these tools? Here's the typical workflow (*fig. 9*).

1. Data Collection & Exploration

Gather your data. Look at distributions, missing values, outliers. Understand what you're working with.

2. Train/Test Split

This is crucial. You split your data into two parts:

- **Training set (~80%)**: The model learns from this data
- **Test set (~20%)**: Held back to evaluate how well the model generalizes to new, unseen data

Why split? If you evaluate on the same data you trained on, you're just testing the model's memory, not its understanding. A student who memorizes the textbook will ace questions from the book but fail when asked something slightly different. The test set is your final exam with questions the model has never seen.

In practice, data scientists design several splits onto which the model can be evaluated to make sure the trained model is really robust and that the performance is actually statistically significant. For the sake of keeping the explanation simple, we will keep a 1-fold evaluation process.

3. Feature Engineering and Preprocessing

This is where the magic (and suffering) happens. Raw data rarely feeds directly into models. You might:

- **Create or select new features:** ratio of debt to income, days since last purchase, etc.
- **Encode categorical variables**: convert "red" / "green" / "blue" to numbers
- **Handle missing values**: fill with median, use a flag or let the model handle it
- **Normalize/scale**: ensure features are on comparable scales

This step often determines success or failure. Domain expertise matters enormously.

4. Model Selection & Hyperparameter Tuning

Teams don't pick one algorithm up front. They train several (random forest, XGBoost, logistic regression) and keep whichever performs best. Each carries settings that aren't learned from data:

- How deep should trees grow?
- How many trees to build?
- What learning rate to use?

Finding the best settings requires trying many combinations, typically using **cross-validation**: repeatedly splitting the training data into sub-folds, training on some and validating on others, to estimate how well each setting will generalize. This step is time-consuming. Tuning XGBoost properly can take hours or days.

5. Model training

Train your model on the training set. The model adjusts its internal parameters to minimize prediction errors.

6. Evaluation

Finally, evaluate on the held-out test set. Common metrics include:

- **Accuracy**: What percentage of predictions are correct?
- **AUC-ROC**: How well does the model rank positive examples above negative ones?
- **Precision/Recall**: For imbalanced classes, how many predicted positives are true positives? How many positives did we find?
- **Check feature contributions**

This step is used to check whether there is any bias in the model decision. We can use feature importance scores or partial dependence plots to understand which features drive predictions and look for unexpected patterns. Validate that decisions align with domain knowledge and fairness requirements.

Figure 9

06 • WHY HAVE TREES DOMINATED?

Deep learning conquered images, text and speech. Why not tables? Several reasons:

1. Deep learning needs data that most tabular use cases don't have

Deep learning performs well when there is a lot of it. Most enterprise tabular problems run on tens of thousands of rows, not millions. Trees work at that scale.

2. Trees handle heterogeneity naturally

Tabular data mixes numeric columns (income: 52,000), categorical columns (city: "London"), and ordinal columns (education: "high school" < "bachelor's" < "master's"). Trees handle all of these without special preprocessing. Neural networks struggle more.

3. Trees are robust to feature scale

A neural network cares whether income is measured in dollars (50,000) or millions (0.05). A tree doesn't, it just finds the split point. This reduces preprocessing burden.

4. Trees can capture non-linear interactions

The split "income > $50k" has different effects depending on other features. Trees naturally capture these interactions without explicitly programming them.

5. Trees are relatively fast

Training and inference are efficient, even for large datasets.

07 •WHAT ARE THE LIMITATIONS OF TREE-BASED MODELS?

Of course, traditional methods aren't perfect:

• **Small data struggles:** With only a few hundred training examples, most classic Machine Learning models will overfit. There simply aren't enough patterns to learn from.

• **No transfer learning:** You start from scratch at every new dataset. The model learns nothing from the millions of previous tabular problems solved by others.

• **Extensive tuning required:** Getting the best performance requires significant hyperparameter tuning. TFMs have a clear advantage on all three of these aspects.

• **Calibration quantification:** Trees give you point predictions. Getting well-calibrated probability estimates ("I'm 73% sure this customer will churn") requires additional work.

08 • SETTING THE STAGE FOR TABULAR FOUNDATION MODELS

The knowledge gained from solving one tabular problem doesn't transfer to the next.

Tabular Foundation Models take a different approach: instead of starting from zero on every dataset, they arrive with prior knowledge already built in. By pre-training on millions of diverse tables, they arrive at your dataset with prior knowledge, patterns already learned, statistical intuitions already formed.

09 • HOW TABULAR FOUNDATION MODELS TRAINED?

Every foundation model needs a large amount of training data. Text and images are abundant on the open internet but tabular datasets are not.

Business data is proprietary, often regulated and, public datasets available on sites like Kaggle or OpenML number in the thousands, not the millions a foundation model needs. They are also too clean to reflect the statistical patterns of real business data.

The answer the field settled on: **generate the data.**

1. Why synthetic data works

At first, this seems absurd. How can a model learn to predict real-world phenomena from data that was never real?

The answer lies in what we're actually trying to teach the model.

A Tabular Foundation Model like Seldon doesn't need to learn that "income predicts credit default" or "tumor size correlates with malignancy." Those are specific relationships in specific domains.

What tabular foundation models need to learn is more fundamental:

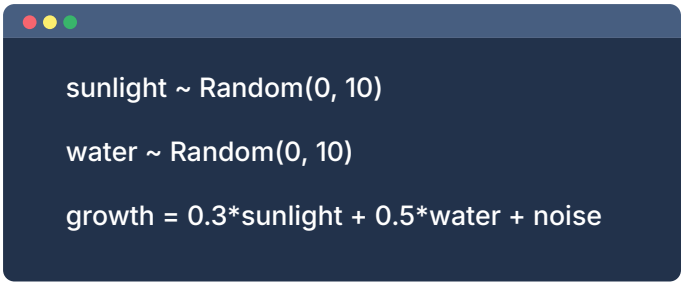
- How to detect patterns in columns of numbers
- How to handle different types of noise and outliers
- How to weight features of varying importance
- How to interpolate and extrapolate from training examples
- How to be appropriately uncertain when data is limited

These are meta-skills, skills for learning itself. And it turns out you can teach meta-skills using carefully constructed synthetic data.

2. The Structural Causal Model Approach

Tabular Foundation Models utilize **Structural Causal Models (SCMs)**, mathematical frameworks that explicitly define causal relationships and functional dependencies among variables, to generate diverse synthetic training data.

For example, in a baseline scenario modeling environmental impacts on plant growth, an SCM formalizes the data-generating process through explicit structural equations:



This framework produces a synthetic dataset where independent variables systematically dictate the target outcome, establishing a higher relative weight for water compared to sunlight. Through exposure to this structured data, the model learns to autonomously evaluate, isolate and weight varying degrees of feature significance.

Crucially, while an individual SCM establishes localized feature dynamics, the core advantage of Tabular Foundation Models is realized by scaling this mechanism across millions of distinct synthetic data structures.

3. Millions of Variations

Tabular Foundation models are trained on millions of synthetic datasets, each generated with randomly varied parameters:

- **Relationship complexity**: Some datasets have simple linear relationships. Others have complex, non-linear interactions. Some have threshold effects ("growth only happens if water > 5"). This variety teaches the model to detect different pattern types.

- **Feature importance:** Sometimes all features matter equally. Sometimes only 2 out of 50 features are predictive. The model learns to identify which features are actually useful.

• **Noise levels:** Real data is noisy. By varying the amount of noise added to synthetic data, the model learns to distinguish signal from randomness.

• **Sample sizes:** Training datasets range from tiny (50 samples) to larger (thousands). This teaches the model how to behave with varying amounts of evidence.

• **Missing values**: Values are randomly removed to simulate real-world incompleteness. The model learns to handle gaps gracefully.

• **Class imbalance**: Some synthetic datasets have 50% positive cases, others have 5%. The model learns to calibrate probabilities appropriately.

• **Causal structures**: By varying the directed acyclic graphs (DAGs) that define causal relationships, the model sees simple independent causes, complex chains, confounders and mediators.

The result is a model that has "seen" virtually every type of statistical pattern that might appear in real data. It develops intuitions that generalize.

4. The Prior: Encoding Statistical Beliefs

Here's where things get philosophically interesting. The synthetic data generation process defines a prior distribution over possible tabular problems. In Bayesian statistics, a prior represents your beliefs before seeing data. The synthetic data generation process encodes beliefs like:

- Simple relationships are more common than complex ones (Occam's razor)
- Features tend to have varying importance
- Real relationships usually have some noise
- Missing values are a normal part of data

When the model then sees your actual dataset, it combines this prior knowledge with the evidence in your data to form predictions. If your training set is small, the prior matters more, so the model relies heavily on its general knowledge. If your training set is large, the data dominates, and the model adapts to the specifics of your problem. This is Bayesian inference, implemented through a neural network.

5. Introducing Prior-data Fitted Networks (PFNs)

The technical name for this approach is Prior-data Fitted Network (PFN).

A PFN is trained to approximate Bayesian inference. During pre-training, it sees millions of synthetic datasets (drawn from the prior) along with their optimal predictions. The network learns to internalize the entire inference process.

At inference time, when you give it a new dataset, it performs what would normally require expensive (Bayesian) computations but it does so in a single forward pass through the neural network. The prior is "baked in" to the network weights.

This is why Tabular Foundation Models can be so fast. Traditional Bayesian inference is computationally expensive. Neural networks running forward passes are fast. PFNs give you the best of both worlds.

6. The Advantages of Synthetic Data

- **No data privacy concerns**: You can't leak sensitive information from data that never contained real people.

- **Perfect labels**: In real data, labels are often noisy, human labelers make mistakes, measurements have errors. Synthetic data has ground truth.

- **Unlimited scale**: Need more training data? Generate it. There's no ceiling on dataset size.

- **Controlled diversity**: You can ensure the training data covers edge cases that might be rare in real datasets.

- **No benchmark contamination**: A major concern in ML is that models memorize benchmark datasets. If your foundation model was accidentally trained on the test set of a famous benchmark, its performance there is meaningless. Synthetic data sidesteps this entirely.

7. But What About Real Data?

Synthetic data isn't the only option, and recent research suggests combining approaches may be even better. Some Tabular Foundation Models like TabDPT take a different path: they pre-train on real tabular datasets from OpenML, using a technique called column-masking (hiding some columns and predicting them from others). This lets the model learn from actual empirical statistical patterns, not just simulated ones. Another TFM, Real-TabPFN uses a two-stage approach:

- **Stage 1**: Pre-train on synthetic data (like the original TabPFN).
- **Stage 2**: Continue pre-training on curated real-world datasets.

The intuition is that synthetic data provides broad coverage and prevents overfitting, while real data adds the subtle, domain-specific patterns that synthetic generators might miss.

It's like learning to play music: you can develop solid technique practicing scales and exercises (synthetic), but you also need to play real songs (actual data) to develop musicality.

At Neuralk, we trained our Tabular Foundation Model, Seldon, exclusively on synthetic data. However, our synthetic data generation is carefully guided by field data to accurately reproduce the behaviors and patterns found in enterprise datasets. This approach ensures that Seldon does not develop unintended biases or overfit to specific datasets, enabling robust generalization across diverse real-world applications.

8. The Craft of Data Generation

Creating the synthetic data generator is itself a research challenge. If your generator produces data that's too simple or too uniform, the model won't learn rich patterns. If it produces data that's too different from real-world distributions, the learned skills won't transfer.

Every Tabular Foundation Model team spends considerable effort engineering their generator, using:

- **Bayesian Neural Networks**: Neural networks with uncertainty over their weights, which naturally produce diverse input-output mappings.

- **Structural Causal Models**: Directed graphs defining causal relationships between variables.

- **Realistic noise models**: Not just Gaussian noise, but various distributions and patterns of missingness.

- **Varied dimensionality**: Different numbers of features and samples.

This is meta-engineering: building the generator that creates the data that trains the model.

9. When Synthetic Falls Short

Synthetic data isn't a silver bullet. The fundamental issue is that the generator itself has inherent biases and assumptions about what the data should look like. When those assumptions differ from the patterns or semantics of the real problem, they can negatively impact model performance.

There are two important ways this can happen:

- **Distribution mismatch:** The generator may fail to reproduce patterns that matter in the real data, such as specific temporal dynamics, correlations, or domain-specific structures.

- **Missing domain knowledge:** The generator may capture the statistical properties of variables without understanding their semantic meaning. For example, "age" and "temperature" are both numbers, but they have very different constraints and relationships in a real domain.

In other words, synthetic data is only as good as the assumptions built into the generator. This is why the field is actively exploring hybrid approaches, combining synthetic pre-training with real-data fine-tuning.

10 • THE ENTERPRISE REALITY CHECK

Let's move from theory to practice. You're a quantitative analyst at a hedge fund, or a data scientist at an insurance company. You've heard about Neuralk and its Tabular Foundation Models, Seldon. The question isn't "is this cool?" (it is). The question is: "Should I actually use this?" The answer, unsatisfyingly but honestly, is: it depends.

1. Where Tabular Foundation Models Shine: The Plethora of Models

Most enterprises don't build one model. They build dozens, sometimes hundreds: one for each region, each product line, each customer segment, each risk tier. Every one of those models needs its own training run, its own hyperparameters, its own monitoring and its own retraining schedule as data drifts. The maintenance burden scales with the number of models.

Traditional ML multiplies this cost. Each XGBoost or LightGBM model is its own artifact: versioned separately, retrained separately, and prone to silently degrading separately. A team that builds 5 churn models across 10 markets isn't just doing several times the initial work, it's carrying the ongoing operational load, indefinitely.

Tabular Foundation Models sidestep this altogether. Because a single pre-trained model handles any new dataset through in-context learning, there's nothing to retrain, no separate weights to version, and no fleet of models to monitor for drift. What used to be a portfolio of models to manage becomes one model, called repeatedly.

This matters most for:

- Organizations running the same type of prediction across many segments, regions, product lines
- Teams without the headcount to own dozens of model lifecycles in parallel
- Any business where "keeping every model current" has quietly become a job in itself

2. Where Tabular Foundation Models Shine: Small Data Problems

Traditional ML methods need data to learn from. With only 500 training examples, XGBoost struggles to find reliable patterns. It might overfit, latching onto noise rather than signal. Cross-validation helps but there's only so much you can do with limited data.

Tabular foundation models arrive with prior knowledge. They've already "seen" millions of datasets and learned what statistical patterns typically look like. With your 500 examples, they don't need to learn everything from scratch. They just need to figure out which patterns from their experience apply here.

Benchmarks consistently show Tabular Foundation Models outperforming tuned XGBoost on datasets under 10,000 samples, often by a significant margin. The smaller the dataset, the larger the advantage.

Example real-world scenarios include:

- Rare disease prediction (few cases exist)
- New product demand forecasting (no historical data)
- Startup analytics (limited customer history)
- Research studies with small sample sizes

3. Where Tabular Foundation Models Shine: When ML Resources Don't Match the Need

Many organizations face more machine learning needs than their specialist teams can support. They may have data analysts comfortable with SQL and statistics, but not enough ML engineers to design, tune, and maintain a dedicated model for every use case. Tabular foundation models reduce the level of specialized expertise required. Users do not need to:

- Choose between random forests, gradient boosting, and neural networks
- Identify and tune the right hyperparameters
- Build sophisticated training and diagnose model-specific issues such as overfitting

Instead, they can load their data, call the model, and obtain strong predictions with minimal configuration effort. Expert oversight remains valuable, especially for evaluation and deployment, but the benefits of expert modeling become accessible to teams with more limited machine learning resources.

4. Where Tabular Foundation Models Shine: Well-Calibrated Calibration

Here's an underappreciated advantage: probability calibration. When a model says "70% chance of churn," you want that to actually mean 70%. If you take all the customers the model labeled 70%, roughly 70% should actually churn. This is called calibration.

Tree-based methods are notoriously poorly calibrated out of the box. They tend toward overconfidence. Getting good calibration requires additional post-processing (Platt scaling, isotonic regression, etc.).

We have seen in practice that Tabular Foundation Models produce naturally well-calibrated probabilities. The model's uncertainty reflects actual uncertainty.

This matters for:

- Risk assessment in finance and insurance
- Medical decision support (where confidence intervals matter)
- Any domain where the “how sure are you?” question is as important as the prediction itself

11 • WHERE TRADITIONAL METHODS STILL WIN (FOR NOW)

1. Production Latency Requirements

Tabular Foundation Models’ inference isn't slow, but it's not as fast as a single tree prediction. For real-time systems requiring sub-millisecond predictions (high-frequency trading, real-time ad bidding, fraud detection on payment transactions) every microsecond matters. Gradient boosted trees, once trained, are extremely fast. A single XGBoost prediction might take 10 microseconds. Neuralk offers the fastest Inference with Seldon.

2. Interpretability Requirements

Regulated industries often require model explainability. Why was this loan denied? Why was this claim flagged? Simple linear models and shallow decision trees can be intrinsically interpretable: their decision mechanism can be inspected directly through coefficients, splits, and human-readable rules. More complex tree ensembles are not intrinsically transparent, but they benefit from mature post-hoc interpretability methods:

- Feature importance scores
- SHAP values showing per-prediction explanations
- Partial dependence plots showing feature effects
- Surrogate trees or rule-extraction methods approximating model behaviour

Tabular foundation models are not intrinsically interpretable. Post-hoc methods such as SHAP, integrated gradients, ablations, and sensitivity analyses can provide empirical explanations, but they do not directly reveal the model’s reasoning process and are less mature for TFMs than for tree-based models. For applications where regulatory compliance demands clear explanations (medical diagnostics subject to review for example) the traditional interpretability advantage matters.

This is changing quickly, though. Adoption of foundation models in these use cases is growing rapidly, while interpretability remains one of the most active areas of research in the field. As these tools mature, foundation models should become increasingly viable for regulated and high-stakes applications over the next few years.

3. Domain-Specific Feature Engineering

Sometimes, domain expertise encoded in features is the main driver of model performance. Consider fraud detection. Raw transaction data might include: amount, timestamp, merchant ID, card type. But domain experts know to engineer features like:

- Velocity (transactions in last hour)
- Distance from home address
- Time since last transaction
- Ratio of current amount to average

These engineered features capture domain knowledge that dramatically improves predictions. Traditional methods with carefully engineered features often outperform foundation models on raw data.

Tabular foundation models use engineered features too but if you're investing in sophisticated feature engineering anyway, the "zero-effort" advantage diminishes.

Neuralk is developing industry or use-case specific finetuning approaches that will bundle industry knowledge directly in the model's feature handling capabilities. If you're interested, reach out.

12 • THE DECISION FRAMEWORK

Here's a practical decision tree:

Start with Tabular foundation models if:

- You lack ML engineering expertise
- You want results in minutes, not days
- Calibrated probabilities are important
- You have to train custom models per category/geography/segment/...

Start with traditional methods if:

- You have ML engineers who have the time and resources to properly tune and maintain the system
- You need sub-millisecond inference latency
- Regulatory compliance requires transparent explanations

13 • ENTERPRISE CONSIDERATIONS BEYOND ACCURACY

Tabular foundation models do more than replace one modelling technique with another: they change how data-science teams allocate their time, how models are deployed, and where the associated costs arise.

1. A Different Data-Science Workflow

Traditional machine learning requires teams to repeat much of the modelling process for every new use case: feature engineering, model selection, hyperparameter tuning, validation, and subsequent maintenance. TFMs reduce this repetitive work by providing a strong pretrained model that can be applied to many different tasks with little or no task-specific training.

This changes how data scientists spend their time:

- **More time for problem framing:** What are we actually trying to predict, and what decision will that prediction inform?
- **More focus on data quality:** Identifying missing information, leakage, biases, and unreliable labels remains essential.
- **More attention to evaluation:** Faster modelling makes it possible to test more ideas, but also makes rigorous validation more important.

While feature engineering and hyperparameter-tuning skills remain valuable, they become less central to the day-to-day workflow, allowing time previously spent tuning individual models to be redirected toward improving the data, evaluation protocol, and overall system.

2. A Different Production Model

Putting trainable models into production requires a complete lifecycle of training, validation, versioning, deployment, monitoring, and retraining, often repeated for every task-specific model. TFMs simplify this process by allowing a single pretrained model to support many use cases.

Because these models require substantial compute and are primarily improved by their providers, accessing them through an API is often the most natural option, enabling users to benefit from model updates without operating the underlying infrastructure.

Models such as Neuralk's Seldon are available through both a Python package and an API, with on-premise deployment offered when sensitive data must remain within the organization. This provides greater control over data and model versions but transfers the infrastructure burden to the user. In all deployment modes, updates still require careful validation. While TFM ecosystems are still evolving, they are rapidly catching up with traditional ML ecosystems, with increasingly mature tools for monitoring drift and performance degradation.

3. A Different Cost Structure

TFMs may involve direct costs, including API usage, licensing, inference infrastructure, or on-premise compute. Some open-source models are available without licensing fees, but may provide less support or have limitations around dataset size, inference speed, and production deployment.

These costs should be considered alongside the resources that TFMs can save:

- Engineering time spent developing task-specific pipelines
- Compute used for training and hyperparameter tuning
- Maintenance and MLOps work across multiple models
- Maintenance and MLOps work across multiple models

The value of a TFM therefore does not necessarily come from having the lowest cost per model inference. It comes from rebalancing expenditure: higher model or infrastructure costs can be offset by substantial time savings, while enabling teams to explore more use cases and devote more effort to improving the systems built around the model.

14 • WHERE THE VALUE LIES: APPLYING TABULAR FOUNDATION MODELS IN THE ENTERPRISE

The table below covers some use cases generally deployed across business functions.

Finance & Risk
- Revenue forecasting
- Cash flow forecasting
- Credit risk scoring
- Fraud detection
- Customer lifetime value (CLV)
- Payment default prediction
- Collections prioritization

Sales & Marketing
- Customer churn prediction
- Next Best Offer
- Lead scoring
- Customer lifetime value
- Campaign response prediction
- Cross-sell / Upsell
- Collections prioritization

Operations & Supply Chain
- Demand forecasting
- Inventory optimization
- Stockout predictions
- Supply chain risk
- Delivery delay prediction
- Capacity planning

Customer Success & Support
- Escalation risk prediction
- Ticket prioritization
- Customer satisfaction prediction
- Renewal likelihood
- Next Best Action

Product & Digital
- Product recommendation
- User engagement prediction
- Feature adoption prediction
- Conversion optimization
- Customer journey prediction
- Personalization
- User lifetime value

Manufacturing & Industrial Operations
- Predictive maintenance
- Quality defect detection
- Yield optimization
- Equipment failure prediction
- Production planning
- Asset utilisation optimization

HR & People Ops
- Employee attrition prediction
- Hiring success prediction
- Workforce planning
- Internal mobility prediction
- Training effectiveness prediction

IT, Security & Fraud
- Fraud detection
- Cybersecurity threat detection
- Incident prediction
- Network outage prediction
- Capacity forecasting
- Infrastructure failure prediction
- Anomaly detection

15 • THE OPEN QUESTIONS

1. Why Do They Actually Work?

The theory says: train on synthetic data drawn from a prior and the model learns to approximate Bayesian inference. In practice, the models often exceed what theory predicts. They generalise in ways that surprise even their creators. Some hypotheses:

- The transformer architecture's inductive bias makes it well-suited to capturing feature interactions in tabular data that other architectures might miss.
- Meta-learning across millions of datasets creates emergent capabilities
- Synthetic priors don't need to look like real data. They need to behave like real data. By modeling the joint distribution between features and labels, the model learns the rules of classification during in-context learning, even if it has never seen the specific features before.

2. What's the Right Prior?

Every tabular foundation model interprets problems through the lens of a prior: a formalised set of beliefs about what tabular datasets typically look like. Different choices of priors naturally lead to different models.

But what's the "right" prior for real-world tabular data? Most current generators rely on structural causal models or Bayesian neural networks, producing diverse datasets that cover many possible patterns. Yet, they may still miss structures that are common in specific domains.

The research community's focus on public datasets like OpenML's, and the publication pressures that come with them can exacerbate this problem. Just as computer vision has been overfitting to ImageNet errors, overreliance on standardised datasets risks overfitting models to quirks of benchmark data rather than real-world problems.

At Neuralk, we take a different approach: we analyse patterns in industrial tabular data. Some of the recurring aspects we observe include:

- Presence of categorical features with high cardinality
- Values missing not at random
- High noise levels in features and labels
- Tables with many irrelevant columns

We use these observation to inform our data generation strategies, always striving to achieve the best combined performance on both research benchmarks and real-world industrial datasets.

TFMs have been developed with some assumptions, the most important of which is that rows in tables are identically and independently distributed (iid). This assumption, which is shared by traditional ML models, is often violated in the real world, and much work goes into engineering features so as to adhere to it as much as possible.

We believe that TFM architectures already have the capacity to tackle datasets where this assumption is violated, and that the main obstacle resides in crafting priors that correctly reflect realistic dependencies (like the temporal dynamics and regime shifts encountered in financial data, or the periodic patterns and sensor drifts in industrial sensor data).

The remaining key question is: can a single prior/model realistically cover the kind of dependencies observed in all domains, or is some form of specialisation preferable?

3. When Do They Fail?

Every model fails somewhere. For certain tabular foundation models, failure modes can include:

- **Distribution shift:** If your real data looks fundamentally different from anything the synthetic generator could produce, the model struggles. Edge cases outside the prior distribution can produce poor predictions with unwarranted confidence. For example, when positive cases in a classification dataset are 0.1% of your data, even well-calibrated models face challenges. The prior may not adequately capture such extreme imbalance.

- **Adversarial vulnerability:** Recent research found that tabular foundation models can be vulnerable to small, carefully designed perturbations. Changing a few feature values in specific ways can flip predictions. For security-critical applications, this matters.

- **Complex temporal dependencies:** Tabular foundation models treat each row as independent. If your problem requires understanding sequences (how a customer's behaviour evolved over time, not just their current state) the standard approach struggles.

- **Handling long context:** how to keep state of the art performance when facing very long context?

Knowing these failure modes help us design appropriate safeguards and mitigate impact. For example, Seldon has shown to be extremely performant, even with extreme class imbalances.

16 • WHAT's ACTUALLY NEXT?

Based on current research trajectories, here's what to expect in the near term:

1. Larger Scale

TabPFN started with 1,000 samples. Version 2.5 handles 50,000. Neuralk's Seldon can handle tens of millions. Foundation models are now viable for the large-scale enterprise datasets where traditional methods currently dominate.

2. Hybrid Systems

Pure synthetic pre-training vs. pure real-data training is a false dichotomy. The future is hybrid:

- Pre-train on synthetic data for broad coverage
- Continue pre-training on real-world data for domain adaptation

3. Better Integration with Causal Inference

Much of data science isn't about prediction, it's about understanding causation. Will this marketing campaign cause higher sales, or just correlate with factors that would increase sales anyway?

Tabular foundation models, trained on structural causal models, are positioned to move beyond prediction into causal inference. Early work like CausalFM extends the PFN framework to estimate causal effects. Expect more development here.

4. Automated Data Science

Combine LLMs with tabular foundation models that connect directly to the raw data, remove all Data Science expertise (from framing to data prep to model training/hyperoptimization) and MLOps and put it in the hands of any decision maker.

17 • 2026, THE TIPPING POINT FOR TABULAR FOUNDATION MODELS

For more than a decade, gradient-boosted tree ensembles have been the gold standard for tabular prediction. We are witnessing a paradigm shift.

A recent large-scale benchmarking shows a shift in performance: Tabular Foundation Models win 82.5% of head-to-head matchups against hyper-tuned tree ensembles, confirming that pre-trained in-context models now outperform traditional approaches on open data.

Among these, Seldon currently achieves the strongest overall performance on real industrial data, demonstrating that foundation models are no longer just competitive with traditional machine learning but can surpass it across a broad range of real-world tasks.

If you'd like to explore the results in detail, visit our TabBench V2 benchmark, which compares the latest generation of Tabular Foundation Models against traditional machine learning methods and read our deep dive into Seldon, our industrial-scale Tabular Foundation Model.

18 • MEET SELDON

Good news: Our API is open for you to try it for free.

If you want to get high-performance predictions without the heavy lifting of conventional model engineering, you can now access Seldon through our API. You can be ready to go in less than 3 minutes.

To explore how Tabular Foundation Models fit into your AI strategy, get in touch with our team.

CONCLUSION

Tabular Foundation Models represent the most significant advance in predictive machine learning for structured data since the emergence of gradient-boosted decision trees. By transferring statistical knowledge across millions of learning problems, they fundamentally change how predictive models are built: instead of beginning every project from scratch, models arrive with prior knowledge that can be applied immediately to new datasets.

This does not mean traditional machine learning is obsolete. Tree-based methods remain an excellent choice for latency-critical systems, highly specialized workflows, and applications where interpretability is paramount.

But the balance is shifting. As the latest generation of Tabular Foundation Models matures, independent benchmarks increasingly show that pre-trained in-context models can match or outperform carefully tuned classical approaches across a broad range of prediction tasks, while dramatically reducing engineering effort. Recent evaluations such as TabBench V2, spanning 189 real-world classification datasets, suggest that the question is no longer whether Tabular Foundation Models are viable, but where their remaining limitations lie.

For enterprises, this changes the economics of predictive AI. Building accurate models is becoming less about selecting algorithms and tuning hyperparameters, and more about identifying valuable prediction problems, assembling high-quality data, and integrating predictions into business decisions. The competitive advantage increasingly shifts from model engineering toward problem formulation, data quality, and operational execution.

The field remains young. Interpretability, multimodal reasoning, causal inference, domain adaptation, and ever-larger context windows are all active areas of research. Many of the techniques that will define the next generation of predictive AI have yet to be invented.

Large language models transformed how we work with unstructured information. Tabular Foundation Models are beginning to do the same for structured enterprise data. Whether that transition takes five years or ten, the direction is becoming increasingly clear: predictive AI is moving from task-specific models toward general-purpose foundation models, and enterprise machine learning is likely to evolve with it.

GLOSSARY

• **Bagging:** Bootstrap Aggregating is a specific ensemble method where you train multiple instances of the same model on different random subsets of the training data (sampled with replacement), then average or vote on their outputs, with Random Forest being the textbook example (page 7).

• **Bayesian inference:** A statistical approach that updates beliefs based on evidence, considering all possible explanations (page 20).

• **Benchmark contamination**: When a model is accidentally trained on test data, inflating its apparent performance (page 14).

• **Calibration**: How well predicted probabilities match actual frequencies (page 16).

• **Causal inference:** Statistical methods for determining cause-and-effect relationships, not just correlations (page 21).

• **Column-masking**: A training technique where some columns are hidden and the model predicts them from others (page 14).

• **Cross-validation**: Repeatedly splitting training data into folds to estimate model performance (page 16).

• **Data drift:** When the statistical properties of input data change over time (page 15).

• **Decision tree**: An algorithm that makes predictions by following a series of if-then rules (page 4, 6, 7, 17, 18).

• **Directed Acyclic Graph (DAG)**: A graph with directed edges and no cycles, used to represent causal relationships (page 13).

• **Distribution shift:** When the data a model encounters differs from the data it was trained on (page 21).

• **Feature engineering:** Creating new input variables from raw data to help a model make better predictions and improve overall performance (page 6, 9, 17, 18).

• **Forward Pass:** The process of feeding input data through a neural network layer by layer, applying weights and activation functions at each step, to produce an output prediction (page 6, 13).

• **Foundation model:** A large AI model pre-trained on massive data, designed to be applied across many tasks.

• **Gradient Boosting:** Building trees sequentially, each correcting the errors of previous trees (page 8, 16).

• **Hyperparameters:** Model configuration settings not learned directly from the data (e.g., tree depth, learning rate) that control how a machine learning model trains (page 5, 6, 15).

• **In-context learning:** Making predictions by using training examples as context, without adjusting model weights (Page 5, 6, 15, 20, 22).

• **Inductive bias:** The assumptions a learning algorithm makes to generalize beyond training data (page 20).

• **Latency:** Time delay between a request and response in a system (page 17, 18).

• **Overfitting:** When a model memorizes training data rather than learning generalizable patterns (page 14, 16, 20).

• **Prior:** The implicit knowledge embedded in the model from pre-training (page 13, 21)

• **Statistical patterns**: How numerical features typically relate to outcomes (page 5, 14).

• **Feature importance patterns**: Which types of columns tend to be predictive (page 10, 12, 17).

• **Prior distribution**: In Bayesian statistics, your beliefs about possible outcomes before seeing data (page 6, 11, 13, 20).

• **Prior-data Fitted Network (PFN)**: A neural network trained to approximate Bayesian inference on new datasets (page 5, 13).

• **Random Forest:** An ensemble of many decision trees, each trained on random data subsets (page 7, 9, 16).

• **SHAP values**: A method for explaining individual predictions by attributing contribution to each feature (page 17).

• **Structural Causal Model (SCM)**: A mathematical framework describing how variables cause and influence each other (page 12).

• **Tabular data**: Data organized in rows and columns, such as spreadsheets, database tables, and CSV files (page 4, 5, 6, 10, 11, 14, 20).

• **Train/Test split**: Dividing data into portions for learning and for evaluating generalization (page 9).


Neuralk
THE PREDICTIVE AI COMPANY
www.neuralk.ai