

Why Large Language Models Fail at Tabular Prediction

Marta Garnelo

Wojciech M. Czarnecki



Abstract

Large language models (LLMs) have become the default tool for a remarkable range of tasks, yet they have had conspicuously little success at one of the most common machine-learning workloads: predictive analytics over tabular data. This gap is the founding premise of the fast-growing field of tabular foundation models, but the question of why generic LLMs fail has remained open. We study a frontier LLM in its purest inference regime - a single generation pass over a prompt containing the full training and test data, with no tools, no agentic scaffolding, and no fine-tuning - and systematically evaluate five hypotheses for the failure: (a) an inability to handle noisy or non-linearly-separable data; (b) the linearised CSV format obscuring column structure; (c) the tokenisation of numeric values; (d) the number of test points classified per query; and (e) the dimensionality of the input. Controlled experiments falsify (a)–(d). Dimensionality, in contrast, is decisive: sweeping random linear projections of thirty-one benchmark datasets, the LLM is the only method among nine whose accuracy decreases as dimensionality grows, while every classical baseline stays flat or improves. A behavioural comparison against 252 configured classical models finds that in two dimensions the LLM predicts like a local, distance-based method (up to 91.6% grid agreement), but in higher dimensions no classical model - even when augmented with tuned, dimension-dependent noise - reproduces its predictions. We do not claim to have identified the internal mechanism; our results show, more modestly, that the LLM's capability dissolves with dimension in a way no noise-corrupted classical learner mimics - which explains why LLMs, so capable elsewhere, keep losing to fifty-year-old baselines on tables, while leaving the mechanism of the prediction as an open question.

Introduction

Large language models (LLMs) have become, in the span of a few years, a default tool for an extraordinary range of tasks: drafting and editing text, writing and reviewing code, answering questions, and automating multi-step workflows [2, 3, 20]. Our collective understanding of these systems has not kept pace with their adoption. We mostly know LLMs through benchmark scores, and when the scores disappoint, the standard response is to build **around** the model - retrieval, tool use, agentic loops - rather than to ask what, precisely, the core model cannot do, and why.

One family of problems has conspicuously resisted the success story: **predictive analytics over tabular data**. Learning a classifier from a modest table of numeric features is the bread and butter of applied machine learning, and it is a task on which LLMs prompted with in-context examples routinely lose to baselines that predate the transformer by decades [4, 6, 12, 13] (this is exemplified in Figure 1 where the performance of an LLM is compared with that of traditional baselines on a set of 11 simple tasks). The gap is stark enough that a dedicated research field - large tabular models, or tabular foundation models - has formed around the premise that generic LLMs are simply not fit for this purpose and that purpose-built architectures are required [7, 14, 15, 24]. The premise is usually asserted, not explained. **Are LLMs bad at this task? And if so, why?**

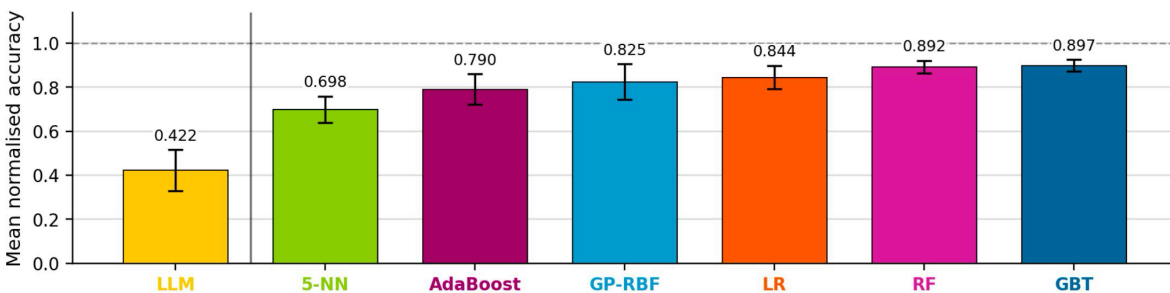


Figure 1: **LLMs significantly underperform on tabular prediction tasks compared to simple traditional methods.**

Mean normalised accuracy across the 11 selected tabular tasks described in Section C.1. Scores are normalised per dataset as $(\text{accuracy} - \text{majority}) / (\text{best baseline} - \text{majority})$ and then averaged across datasets; error bars show standard errors across datasets.

In this paper we do neither of the two usual things. We are not pursuing the best possible score, and we are not proposing a new architecture, prompting scheme, or agent. Instead, we take the core model in its purest inference regime - a single user turn containing the entire training set and test set, no system prompt, no tools, no fine-tuning - and treat it as the object of study. We put forward three questions:

- RQ1** Are LLMs capable of in-context learning over simple, classical classification problems?
- RQ2** What are the main factors determining their (in)capability to do so?
- RQ3** What is the most likely inner working - or, at least, the best behavioural model - of how they predict?

Our answers, in brief. **RQ1**: yes, but only while the table stays narrow. **RQ2**: of five candidate factors drawn from practitioner folklore - class overlap (H1), the linearised CSV format (H2), numeric tokenisation (H3), per-query test load (H4), and input dimensionality (H5) - controlled experiments falsify the first four.

Dimensionality alone is decisive: in a sweep of random linear projections over thirty one benchmark datasets, the LLM is the only method out of nine whose accuracy decreases as dimensionality grows.

RQ3: on two-dimensional tasks, where the model does work, its decision boundaries are reproduced almost exactly by simple distance-based methods - Gaussian processes with short length scales and low-k-nearest neighbours (91–92% grid agreement). In higher dimensions, however, we fail to identify any classical surrogate: across 252 configured baselines the best prediction agreement with the LLM is only 64.8%, and augmenting each baseline with a tuned, dimension-dependent label-noise model improves agreement by at most 0.64 percentage points. The LLM's high-dimensional predictions are thus not those of any standard learner in our library, nor of such a learner degraded into (majority-class) guessing with dimension; the mechanism behind the collapse remains open. Concretely, our contributions are:

- A controlled evaluation protocol for LLM in-context classification, including a memorisation probe that flags contaminated benchmark datasets before contamination can masquerade as capability (Section 5).
- Falsification, by targeted intervention, of four popular explanations of LLM failure on tabular prediction: separability, serialisation format, numeric precision, and test-batch size (Section 6).
- Evidence that dimensionality is the dominant limiting factor: the LLM is the only one of nine methods whose performance degrades as features are added, even when information content is held fixed by random projection (Section 7).
- A behavioural characterisation of the LLM in two dimensions as a local, distance-based predictor - Gaussian processes and low- k kNN reproduce its 2D decision boundaries at 91–92% agreement - together with a negative result in high dimension: no model among 252 configured classical baselines exceeds 64.8% prediction agreement with the LLM, and fitting a dimension-dependent noise model to every baseline adds at most 0.64 percentage points (Section 8).
- A qualitative analysis of reasoning traces, in which the model either admits it will guess or states decision rules that its own predictions do not follow (Section 9).

The paper is structured as follows. Section 2 pins down what we mean by "an LLM" and, as a result, the scope and generality of our claims. Section 3 states the five hypotheses and Section 4 the datasets and experimental setup. Section 5 reports the contamination check, Section 6 the experiments falsifying H1–H4, Section 7 the dimensionality result, and Section 8 the behavioural modelling. Section 9 examines the model's own verbal explanations of its predictions. We situate the findings in prior work in Section 10, discuss limitations and open questions in Section 11, before concluding in Section 12.

2

What do we mean by “LLM”?

Claims about LLMs are only as sharp as the definition behind them, so we fix ours. Throughout the paper, an **LLM** is an instruction-tuned, autoregressive transformer language model [25] accessed through a plain text interface. We probe it in what we call **pure inference mode**: a single user message containing the full training data and all test features either as attached CSV text or embedded in the prompt directly, answered in one generation pass under default sampling; no custom system prompt, no tools, no code execution, no retrieval, no multi-turn interaction, and no fine-tuning. Anything layered on top – agentic loops, scaffolds, feature-engineering harnesses – measures the harness. We want to measure the model.

Concretely, our experiments use `claude-opus-4-6` [2], a frontier model at the time of writing (Section C.3), alongside an initial generalizing experiment using Qwen to begin moving our claims from a single model to LLMs more broadly. Our claims are therefore claims about this regime and, strictly speaking, about the tested models; we phrase them about “the LLM” for readability. Note that pure inference mode is a stronger, not weaker, target than it may appear: it is exactly how in-context learning is advertised, and exactly the regime in which tabular foundation models such as Nexus [9] or TabPFN operate [14].

3

Five hypotheses

Why would a system that writes correct SQL and passes graduate exams fail to separate two point clouds? Practitioner folklore offers several explanations (Table 1).

H1 (Separability).

LLMs can only recover a decision rule when the classes are (nearly) separable; overlapping, noisy classes defeat them. **Prediction:** performance should track class separation, and fully separated classes should be easy.

H2 (Serialisation format).

A table flattened row by row into CSV text destroys the column structure a tabular learner relies on: the model “cannot read vertically”. **Prediction:** even trivially available signal - the answer sitting in a column - should be inaccessible.

H3 (Numeric tokenisation).

Real numbers become variable-length token strings; long mantissas inflate the context and blur magnitude comparisons. **Prediction:** reducing numeric precision should help.

H4 (Per-query test load).

All test rows are classified in a single pass, so the compute available per prediction shrinks with the size of the test set. **Prediction:** fewer test rows per query should help.

H5 (Dimensionality).

The model’s capability collapses as the number of feature columns grows, largely independently of the information those columns carry. **Prediction:** performance should degrade under dimensionality-increasing transformations that preserve information, while classical baselines stay flat or improve.

	Hypothesis	Test	Verdict
H1	Class overlap / non-separability	Controlled separation sweep (Figure 4)	rejected
H2	Linearised-CSV format	Needle in the haystack (Figure 5)	rejected
H3	Numeric tokenisation / precision	Decimal-places sweep (Figure 6)	rejected
H4	Test rows per query	Reduced size of test batches (Figure 7)	rejected
H5	Input dimensionality	Random-projection sweep (Figure 8)	supported

Table 1: The five hypotheses and their fate.

4

Datasets and experimental setup

Benchmarks We evaluate the LLM on two complementary testbeds:

- 1. Main benchmark** The first is a pool of 19 small classification datasets: 17 classical real-world benchmarks from scikit-learn and the UCI repository [17, 21] (e.g. iris, wine, breast cancer, sonar, heart disease), plus two synthetic generators built as controlled extremes: `linear`, which is linearly separable and noise-free at every dimensionality, and `sin`, a highly non-linear parity-of-votes task with feature interactions of order up to d . After the data-hygiene checks of Section 5, the main analyses retain 11 of the 19 datasets. When calling the LLM each query sends one train/test split to `claude-opus-4-6` as plain-text CSV attachments together with a fixed instruction, in a single user turn with no tools, no system prompt, and API-default sampling; the model must return a parseable vector of integer class predictions for all test rows. Splits are 5-fold stratified cross-validation repeated over 5 seeds, giving 25 splits per dataset and 475 queries in total.
- 2. 2D synthetic tasks** The second testbed is a suite of twenty two-dimensional tasks with qualitatively different decision boundaries (blobs, checkerboards, circles, moons, spirals, XOR, and others, in clean and noisy variants), rendered in Figure 16. Each task embeds 60 fixed labelled points directly in the prompt and the model labels a regular 20×20 grid over the training bounding box, queried in shuffled chunks.

Baselines and metric The LLM is compared against a fixed suite of eight classical classifier (kNN with $k \in \{1, 5, 10\}$, random forest, regularised logistic regression, AdaBoost, gradient boosting, and a Gaussian-process classifier), all seeing identical features and splits, plus 252 configured variants used in the behavioural analyses of Section 8. The base metric is per-split test accuracy; for cross-dataset aggregation we use a normalised score in which 0 corresponds to majority-class guessing and 1 to the best non-LLM baseline on the original, unmodified data, so values above 1 indicate exceeding that reference under a transformed condition. Full details, including dataset properties and generator code, the exact prompt, retry and parsing logic, data formatting, baseline configurations, and normalisation, are given in Section 4.

5

Data hygiene: which datasets does the model already know?

Before any accuracy can be read as in-context learning, one must rule out the boring explanation: the model has memorised these datasets. The classical benchmarks above have been reproduced in textbooks, tutorials, and code repositories for decades, and are all but certain to appear - labels included - in pre-training corpora of LLMs.

We therefore run a memorisation probe. For each dataset, repetition and fold, we hold out all rows of one target class, prompt/train on every row from the remaining classes, and score predictions on the held-out rows against their original labels. As a result, any predictor genuinely inferring the input-label mapping from the prompt must score ≈ 0 against the **original** labels; residual accuracy against the original labels can then only come from prior knowledge of the dataset. All eight classical baselines land at zero, by construction. The LLM does not: it recovers the original labels of breast cancer and iris almost perfectly, and those of bank and wine to a large extent (Figure 2).

This is direct evidence of dataset memorisation, and it dictates hygiene for everything downstream. First, datasets the model “knows by heart” are excluded wherever contamination could masquerade as capability. Second, datasets on which all baselines performed at chance are excluded from Figure 2 as there is nothing to learn from those tasks. More broadly, we suggest that a probe of this kind should be standard practice in any LLM-for-tabular evaluation.

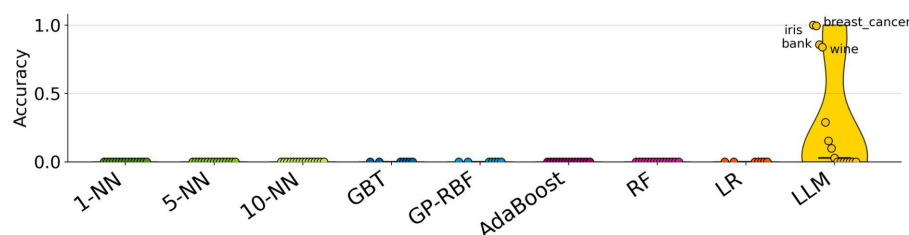


Figure 2: **Memorisation probe.** Accuracy against the **original** labels under a context manipulation that forces genuine in-context learners to zero. All classical baselines score zero; the LLM recovers breast cancer and iris almost perfectly, and bank and wine substantially. Datasets on which all baselines performed at chance are excluded from the plot.

6

Four hypotheses that fail

6.1 H1: it is separability

Design. For each dataset we construct a five-step sequence of variants by translating only class 1 in the original feature space. Let c_0 and c_1 be the two class centroids and $v = (c_1 - c_0) / \|c_1 - c_0\|$. For a shift t , each class-1 point x is replaced by $x + tv$, while class-0 points and labels are unchanged. The maximal-overlap endpoint uses $t = -\|c_1 - c_0\|$, so the two centroids coincide; the separated endpoint uses $t = \max\{\max_{x \in 0}(x - c_0)^T v - \min_{x \in 1}(x - c_0)^T v, 0\}$, the smallest shift that separates the projected class intervals along v . Intermediate variants are linearly spaced between these endpoints and indexed in the plots as step 1 (maximal overlap) through step 5 (separated). Figure 3 visualises the sequence in the first two principal components, with PCA fit on the `parkinsons` dataset along the five steps of the separation. The equivalent separation plots for the rest of the benchmark datasets are shown in Figure 14 in the Appendix.

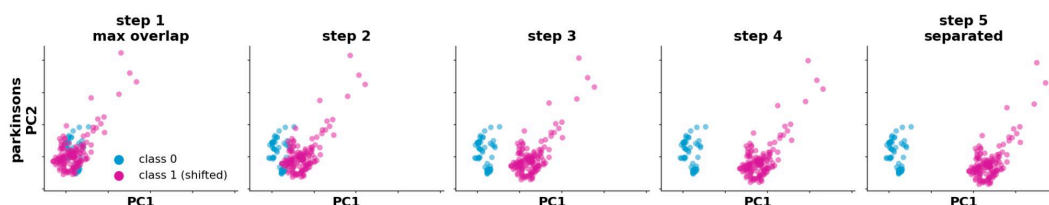


Figure 3: The **separability manipulation**, visualised in the first two principal components: class 1 is translated in five steps from maximal overlap (left) to full separation (right), for each retained dataset.

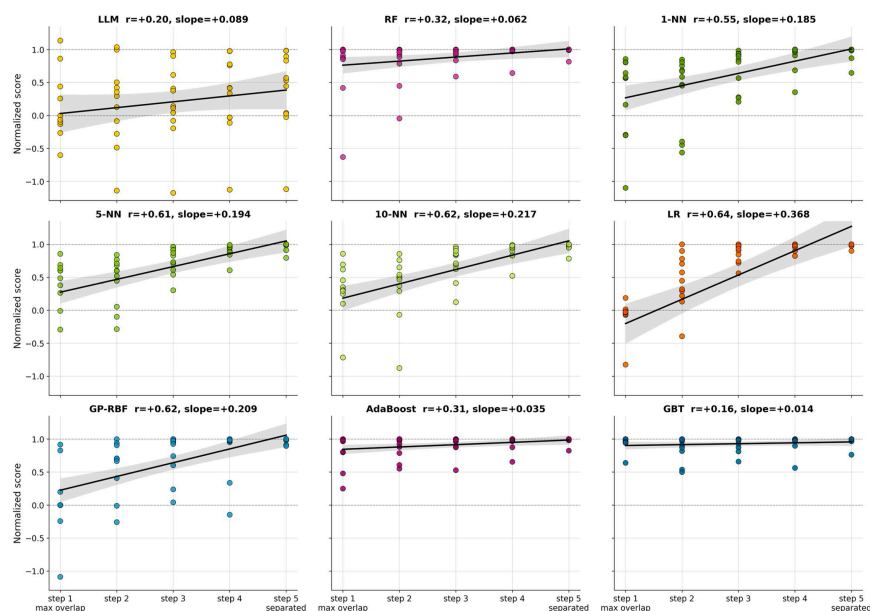


Figure 4: **H1 - separation experiments.** Normalised score against separation step. All methods improve with separation and succeed at the task with full separation while the LLM barely improves from a dummy baseline even when the classes are fully separated.

Result. If H1 were the story, separation should rescue the LLM. It does not (Figure 4): every method improves with separation – logistic regression most steeply (slope +0.368, $r = +0.64$) – but the LLM is the model that improves the **least** (slope +0.089, $r = +0.20$) and has the weakest correlation if we don't consider methods that already start close to the full score and saturate as the distance increases (RF, AdaBoost, GBT). Furthermore the LLM remains far below the baselines even at full separation (40% between the dummy baseline and full accuracy) while all other baselines succeed at full separation. The synthetic linear task sharpens the point from the other side: it is linearly separable by construction at every dimensionality, yet the LLM still fails on it as d grows (Section 7). Separability is not the limiting factor. **Verdict: rejected.**

6.2 H2: it is the format (needle in the haystack)

Design. If linearised CSV blinds the model to columns, it should fail even when the answer is sitting in one. We therefore make the answer trivially available in the serialised table: the target column itself is included among the features, for training and test rows alike, so that solving the task reduces to locating the correct column among up to 60 distractors and copying it. We sweep the native dimensionality of the host dataset.

Result. The LLM scores at or near ceiling at most dimensionalities, with a flat trend ($r = -0.17$, slope -0.002 ; Figure 5). This performance is similar to that of other baselines like distance based ones (nearest neighbours) and AdaBoost. Whatever prevents it from combining thirty columns, it is not an inability to read them – the model locates and copies a designated column at $d=60$ without loss. As a side effect, this also bounds vanilla long-context degradation over the tested range. **Verdict: rejected.**

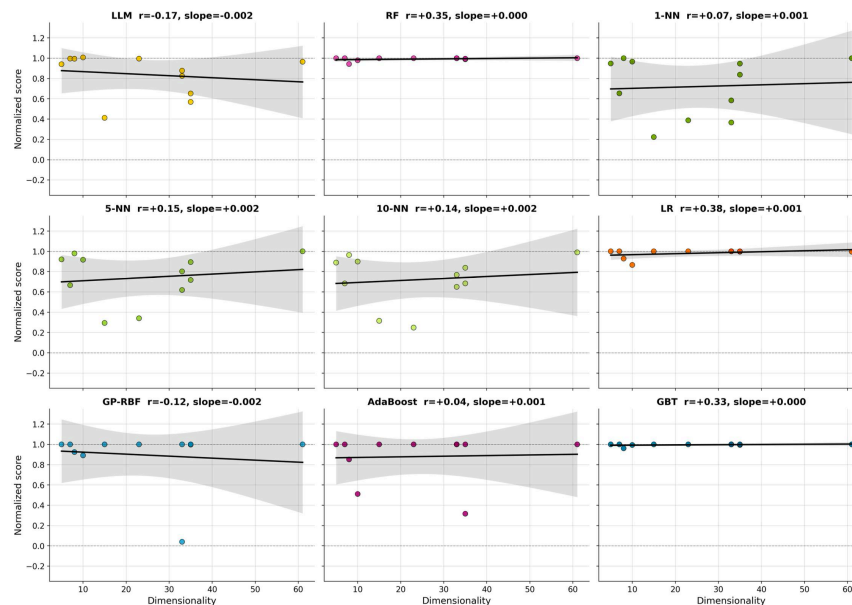


Figure 5: **H2 - Needle in the haystack experiments.** The target column is included in the context, i.e. among the features of every row. The LLM reads it at every dimensionality (flat trend, $r = -0.17$), as do all baselines.

6.3 H3: it is numeric precision

Design. We round all features to $p \in \{1, 4, 8\}$ decimal places and re-run the evaluation. Fewer decimals means shorter token strings and coarser values; if tokenisation of long numerals were the bottleneck, low p should help.

Result. Nothing moves (Figure 6): the LLM is flat ($r = -0.02$, slope -0.003), and so is every baseline.

Verdict: rejected.

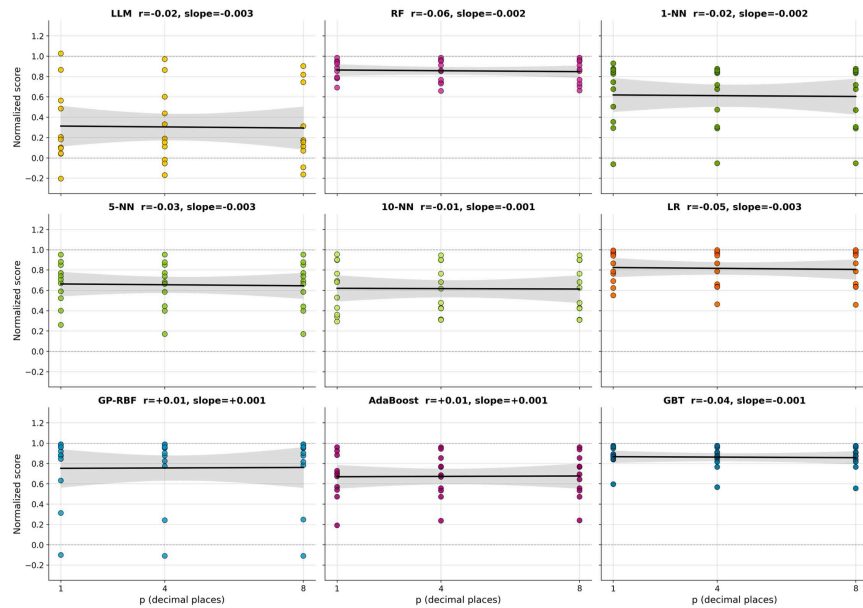


Figure 6: **H3 - Numerical precision experiments.** Features rounded to $p \in \{1, 4, 8\}$ decimal places. No effect on any method (LLM: $r = -0.02$).

6.4 H4: it is the number of target labels per query

Design. For each held-out fold, we start from the original condition in which the whole test fold is sent in one LLM call. We then create smaller target sets by splitting the same held-out rows into 2 or 4 chunks, sending each chunk in a separate LLM call with the same training context, and stitching the predictions back together before scoring the original fold. Thus the x-axis is the number of target labels requested per call: roughly $|D_{test}|$, $|D_{test}|/2$, or $|D_{test}|/4$, depending on the split factor. We run this only for the LLM: for conventional baselines, batching test rows is a scoring convenience rather than a change to the learned classifier, whereas for an LLM it changes how many labels must be produced in a single generation. If per-pass compute were being rationed across requested labels, accuracy should improve when the target set per query becomes smaller.

Result. It does not (Figure 7): over the 11 retained datasets and 33 dataset-size conditions, the relationship between target-set size and normalized LLM score is essentially flat ($r=-0.02$, slope -1.5×10^{-4} per target). The mean normalized score is also effectively unchanged across the original fold, half-fold, and quarter-fold target sets (0.422, 0.412, and 0.407; raw accuracy 0.643, 0.634, and 0.631). Asking the model for fewer labels at a time therefore does not rescue it. **Verdict: rejected.**

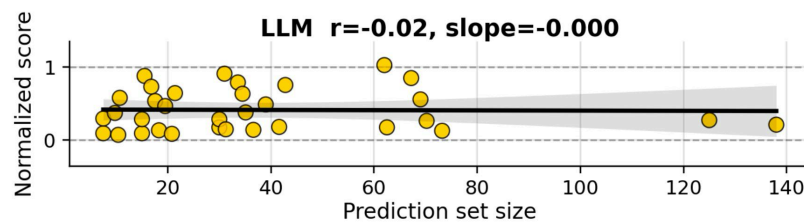


Figure 7: **H4 - number of targets per query experiments.** Normalized LLM score as a function of the nominal number of test labels requested in a single generation. The trend is flat ($r=-0.02$, slope -1.5×10^{-4} per target).

7

Dimensionality is the limiting factor

Design. To probe how performance depends on input dimensionality, we sweep over dimensionalities $d \in \{1, 2, 4, \dots, 2^{\lfloor \log_2 D \rfloor}\} \cup \{D\}$, i.e. powers of two up to the largest power not exceeding the native dimension, plus the native dimension itself. The projection matrix has entries drawn i.i.d. uniform, and a fresh projection is sampled at each target dimensionality. At every dimensionality, all methods are evaluated under the protocol of Section C.3 (5 repeats of 5-fold CV; identical projected, imputed features and identical splits for every method). The logic of the intervention: random projections approximately preserve the information available for classification [16] while varying only the number of columns the model must integrate – a method whose learning machinery scales with dimension has no reason to collapse.

Result. Figure 8 shows the outcome. Every classical baseline is flat or improves with dimensionality (slopes from +0.000 to +0.012; r from -0.01 to +0.32): more projected dimensions preserve more of the native signal, and the models use it. The LLM is the only method that moves the other way: $r = -0.21$, slope -0.009 per dimension, and every per-dataset trend is decreasing within its confidence interval. By $d \geq 16$, the LLM’s normalised score sits at or below majority-class guessing on most datasets.

This answers RQ1 and RQ2. LLMs **can** perform in-context classification – while the table stays narrow. Among the factors tested, only dimensionality produces the collapse, and only the LLM collapses.

One natural follow up question is whether the experimental data used has some hidden correlations between the dimensionality and some other characteristics inherent in the real data. To verify this claim we took simple synthetic 2D datasets described in Section C.2, and similarly to the previous experiment we used the random matrix to upscale them to dimension 4, 8, 16, 32, 64. This way data provably lives in the fixed 2D manifold, and all extra dimensions are just various linear combinations of it.

Figure 9 confirms, that in the synthetic regime the exact same property holds. Arguably the support for the hypothesis H5 is even stronger, as every baseline is flat or improving, while the LLM still rapidly deteriorates with the growing dimension.

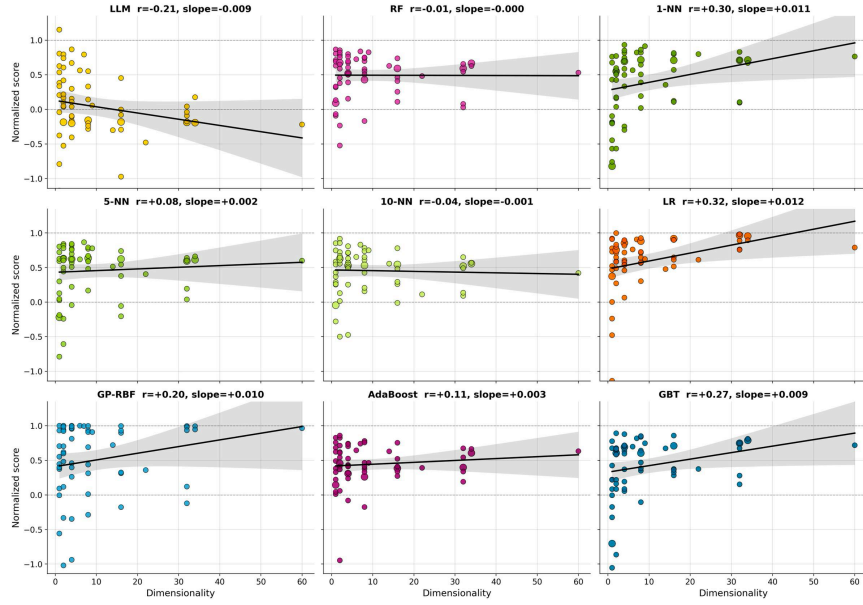


Figure 8: **H5 - dimensionality experiments.** Normalised performance of all methods on the 11 benchmark datasets projected to every power-of-two dimensionality up to, and including, the native dimension (Section 7). Normalisation is carried out as described in Section C.7.

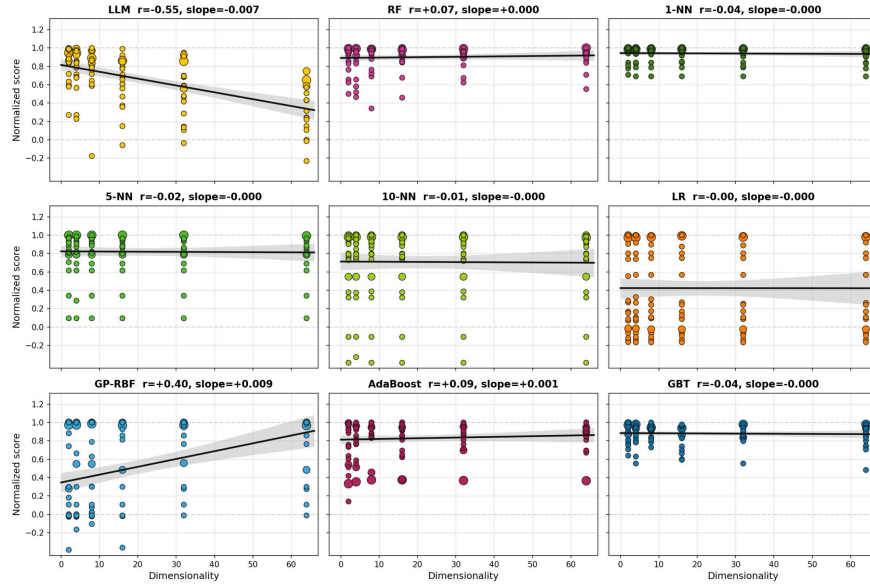


Figure 9: **H5 - dimensionality experiments.** Normalised performance of all methods on the 2D synthetic datasets projected to every power-of-two dimensionality up to 64. Normalisation is carried out as described in Section C.7.

What is the model actually doing?

Lets now focus on the question of **how** the LLM makes predictions when it is successful.

Baseline construction. We compare the LLM predictions against a broad library of non-LLM `scikit-learn` classifiers. The baseline pool consists of a systematic sweep over common classifier families and hyperparameters. Scale-sensitive models are wrapped in a `StandardScaler`, while `MultinomialNB` uses min-max scaling; tree-based, naive Bayes, discriminant-analysis, and dummy baselines are left unscaled. In total this gives 252 configured baselines; the hyperparameters chosen for these baselines are listed in Table 5 in the Appendix.

8.1 In 2D, the LLM behaves like a distance-based method

As we showed the performance of LLMs deteriorates in high dimensions, so we start where the model works: in two dimensions. We use the synthetic 2D datasets described in Section C.2 and query the LLM and the baselines following the protocol outlined in Section C.4. The decision boundaries predicted by the LLM are visualised in Figure 16 in the Appendix. Baselines are then ranked by their pointwise agreement with the LLM majority-vote grid prediction.

The story that crystallises from these 2D experiments is best described as **distance-based** (Figure 12). Across all 20 two-dimensional probes, the closest baseline is a standardized Gaussian process with a Matérn kernel (with length-scale 1, ν 1.5), which agrees with the LLM on 91.6% of grid cells; the top 5 matches are, in fact different variants of GPs. 1-NN is also close at 91.0%. The disagreement overlay in Figure 10 makes the qualitative picture clear: The GP and the LLM usually draw the same broad regions, and their disagreements concentrate near decision boundaries and task-specific ambiguous areas. Thus, in two dimensions, the LLM behaves much like a local distance-based classifier.

As a complementary view, we use MDS to visualize similarity between baselines' held-out accuracy profiles; details are in Section E. This plot should not be read as a pointwise prediction-similarity map. Rather, it asks whether models have similar patterns of accuracy across datasets, repetitions, and folds. In this accuracy-profile space, configured GP and nearest-neighbour baselines lie near the LLM, which is consistent with the direct grid-agreement evidence above, but weaker than it: two models can have similar accuracy profiles while making different predictions on individual points.

Comparing all state-of-the-art LLMs in our setup is out of scope for this paper. We did, however, run these 2D experiments on Qwen3-235B-A22B to see if these findings generalise. The specific baselines that best match Qwen differ from those matching Claude, as expected; what matters is that Qwen's top matches are all distance-based models as well (Figure 15), supporting our hypothesis about general LLM behaviour on two-dimensional tasks beyond a single model.

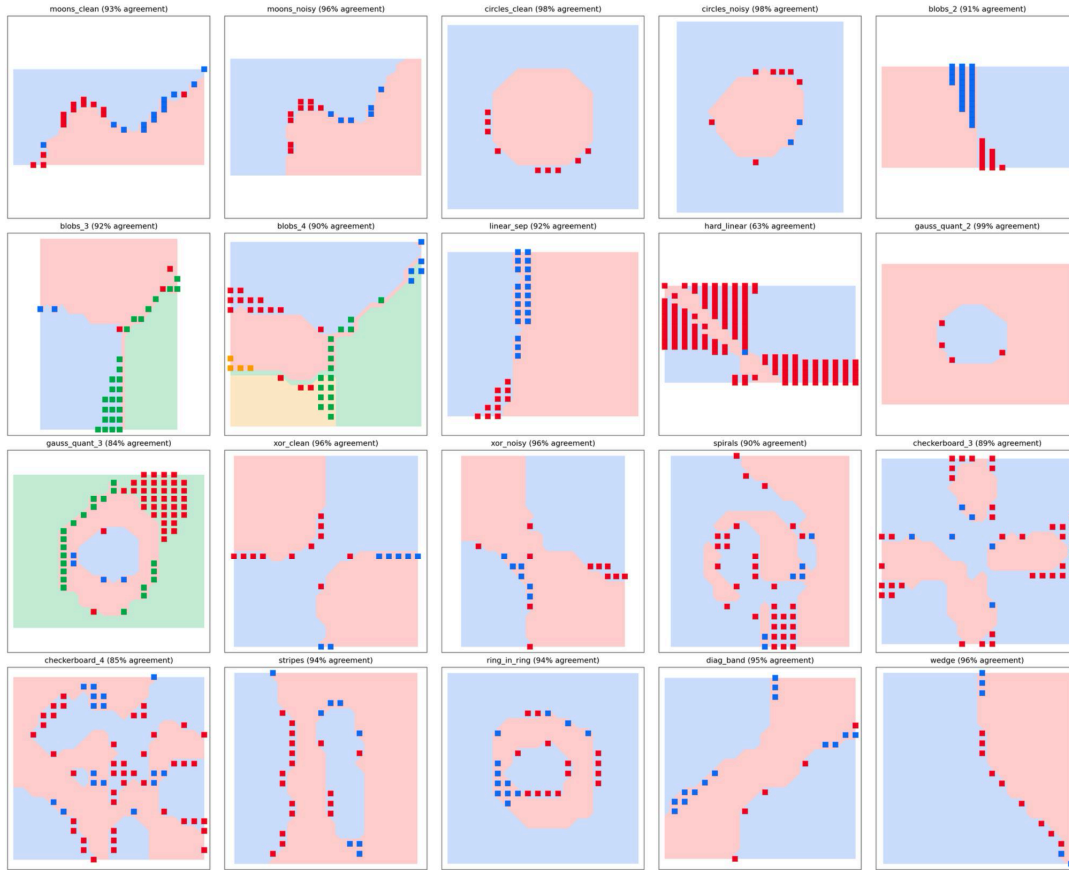


Figure 10: **Where the LLM and GP with most similar behaviour disagree.** Decision maps for the 20 two-dimensional synthetic probe tasks, with the LLM majority-vote prediction shown as the background. Markers indicate grid locations where the LLM prediction differs from a GP with Matérn kernel, length-scale 1, ν 1.5, the baseline with highest prediction agreement on the 2D probes. Disagreements concentrate near decision-boundary regions and task-specific ambiguous areas.

8.2 No classical model reproduces it in high dimension

We then repeat the identification attempt on the real tabular benchmark suite which goes beyond two dimensions, ranking the configured classical models by prediction agreement with the LLM. The search on the other hand does not provide a satisfactory winner. In the bottom panel of Figure 12, the most similar model to the LLM reaches only 64.8% mean dataset-level agreement, and the entire top 30 is compressed into 64.0%–64.8%. The closest entries include shallow random forests and large-length-scale Gaussian processes, but no family achieves enough similarity to be considered a good representation of an LLM-like predictor.

This is also confirmed in the Multidimensional Scaling (MDS) plot in Figure 11, where the LLM predictions are not close to any other model family. In fact, it is only the LLM's prediction and the dummy predictions that are far from all other models in this space. This is indicative in itself. In high dimensions, therefore, we cannot say that the LLM behaves like any standard learner in the library.

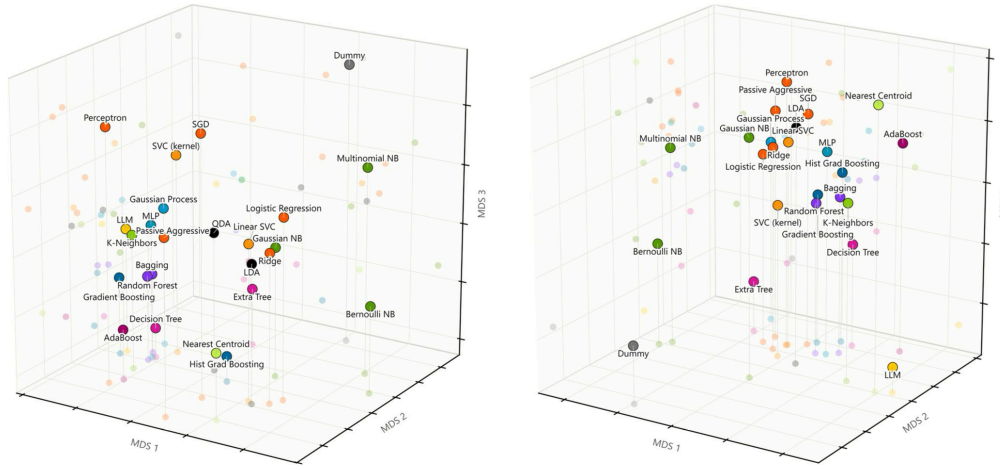


Figure 11: **3D MDS of model families' predictions.** Weighted metric MDS of configured baselines using mean-L1 distances between held-out accuracy vectors. Pairs involving the LLM are up-weighted in the MDS objective, so the layout emphasizes preserving LLM-to-baseline accuracy-profile distances. Colours denote model families. Proximity indicates similarity of accuracy profiles, not necessarily pointwise prediction agreement. **Left:** 2D benchmark predictions **Right:** main benchmark (real-world + `sin` and `linear`).

8.3 Dimension-dependent random noise is not enough

Given that performance of LLMs drops as the number of dimensions increase, one possible explanation could be that the LLM follows an ordinary classifier whose predictions deteriorate to random guesses with dimension. We tested this directly by adding dimension-dependent noise to each baseline. For each model m , we define

$$z_d = \sigma(a_m d + b_m),$$

and replace the model prediction with the training-fold majority-class prediction with probability z_d :

$$\tilde{f}_m(x) = \begin{cases} f_m(x), & \text{with probability } 1 - z_d, \\ f_{\text{maj}}(x), & \text{with probability } z_d. \end{cases}$$

The parameters a_m, b_m are tuned separately for every baseline to maximize prediction agreement with the LLM.

This helps only marginally. In Figure 12, the black hatched caps show the improvement from the optimized noise model. Across the top 30 baselines, raw agreement ranges from 64.0% to 64.8%, with mean 64.3%. After noise tuning, agreement ranges from 64.3% to 65.2%, with mean 64.7%. The average gain is only 0.38 percentage points, and the largest gain is 0.64 percentage points.

Thus, while dimensionality-dependent randomness can describe part of the LLM's degradation, it does not reproduce the LLM's prediction-level behaviour. The high-dimensional failure is not explained by taking a standard classifier and randomly replacing an increasing fraction of its predictions with the majority class.



Figure 12: **Nearest baseline prediction agreement.** Top-30 non-LLM baselines ranked by agreement with the LLM predictions on the main benchmark and 2D synthetic probes. For the main benchmark, hatched caps and parenthesized values show the additional agreement obtained after fitting a baseline-specific noise model. Agreement is computed over matched predictions on the same evaluation points.

9 What is the model saying it is doing?

Finally, when LLMs are prompted to carry out these predictive tasks they will usually also return a thinking trace that describes their decision process in words. As always with LLMs the explanations will sound convincing but can they be trusted?

We therefore ask whether the model's verbal explanations provide a coherent account of its prediction behaviour. On each of the 20 two-dimensional probe datasets, we ask the model to return both target predictions and an explanation of the rule it used to carry out the predictions. We then use this explanation to visualise the decision boundary the LLM claims to be following.

The results are mixed, but mostly cautionary. The LLM predictions fall into one of three categories: 1. both the LLM predictions and its explanation match the data, 2. the predictions match the data but the explanation does not or 3. neither the predictions nor the explanations match the data. We have visualised one example of each of these three cases in Figure 13 and the predictions and explanations for all of the datasets in the 2D probe datasets can be found in Figures 17-20. In 6 of 20 tasks, both the predictions and the explanation qualitatively match the data-generating structure (Appendix Fig. 17). In 4 further tasks, the

predictions largely match the data, but the explanation does not: the model gives a plausible-sounding rule whose implied boundary differs from the true one (Appendix Fig. 18). In the remaining 10 of 20 tasks, neither the predictions nor the explanation match the data (Appendix Figs. 19–20). Thus, explanations fail to match the data in 14 of 20 tasks overall, while only a smaller subset show agreement between accurate predictions and an appropriate stated rule.

These traces should therefore be read as diagnostic artefacts, not as direct evidence of the computation being performed. They show that the model can sometimes describe the structure it predicts, but they also show a substantial decoupling between predictions, explanations, and ground truth. This is consistent with known faithfulness gaps between stated reasoning and actual computation [23], and should discourage taking LLM self-explanations on tabular tasks at face value.

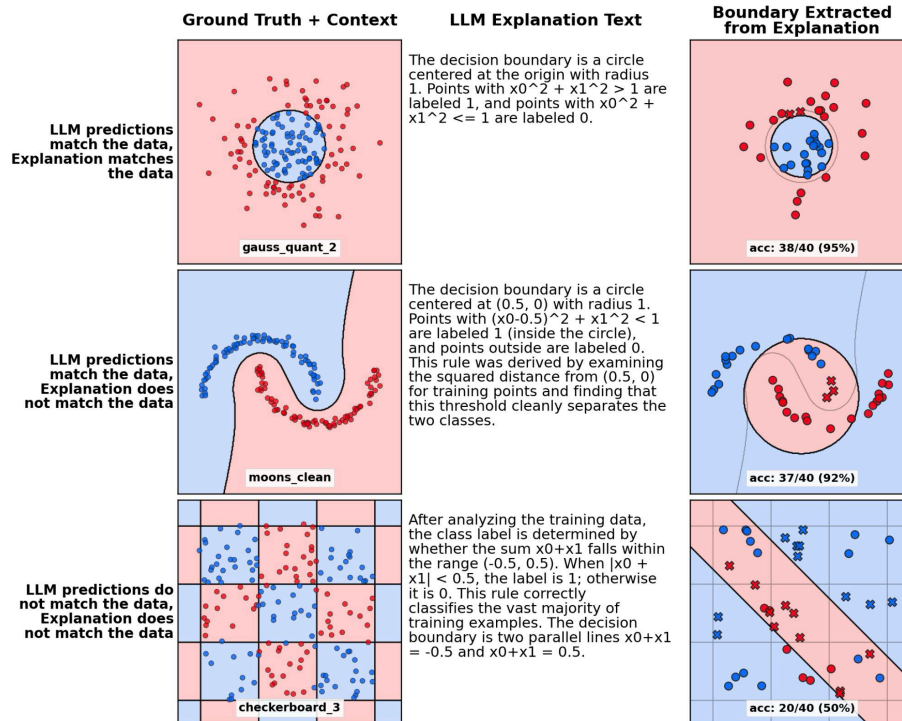


Figure 13: Examples from the 2D story experiments. Each row shows the ground-truth decision structure and context points, the rule described by the LLM, and the decision boundary obtained by evaluating that rule. We have selected one representative example from the three cases we observed: 1. the predictions and explanation both match the data 2. the predictions match the data but the LLM’s explanation does not or 3. neither the predictions nor the explanation matches the data.

Related work

LLMs for tabular prediction. Some work has attempted to use LLMs for tabular prediction. TabLLM [13] serialises tables into text and uses them for fine-tuning on few-shot classification; LIFT [4] fine-tunes language interfaces for non-language tasks; both of these methods, however, use fine tuning rather than in context learning, whereas our claims in this paper are with respect to the in context capabilities of LLMs. Gardner et al. [7] also use fine-tuning but the actual predictions are done via in context learning. However, this and the models before it work in the very low-shot, low-dimensional regime, the latter going only up to 32 context observations and none exceeding a few dozen features; real-world tables often have millions of rows, so these approaches are not relevant in that space. Fang et al. [6] survey the area.

Tabular foundation models. Given how unsuited LLMs are for the tabular prediction task, the field of Tabular Foundation Models has been growing over the past years. Models like Nexus [9], TabICL [22] and TabPFN [15] demonstrate that purpose-built in-context learners excel on exactly the types of tasks studied here, tree ensembles remain a stubborn baseline [12], and the case has been made that tabular foundation models deserve to be a research priority [24]. That literature is premised on generic LLMs being unsuited to tabular prediction; our results supply the missing causal account behind the premise.

Understanding in-context learning. Transformers trained for in-context regression can implement learning algorithms, including behaviour consistent with gradient descent or least squares [1, 8, 26]; other accounts cast ICL as implicit Bayesian inference [27] or interrogate the role of demonstrations [19]. The broader idea of amortising learning-from-a-context-set into a single forward pass predates transformers, as in Neural Processes [10, 11].

Limits of LLMs. Our dimensionality collapse joins a growing catalogue of systematic LLM limitations, including compositionality barriers [5], artefacts of the autoregressive training objective [18], and unfaithful self-explanations [23].

Limitations and open questions

Several limitations bound our claims. While our main experiments rely on a single frontier model (claude-opus-4-6), and we incorporate Qwen into our 2D suite as an initial step toward evaluating generalizability (Section H), the overall claims are empirical in nature and thus there is no guarantee they will hold for future models.

Furthermore, our account is behavioural, not mechanistic: we have no access to weights or activations, and our model matching experiments are a description of input–output behaviour, not of circuitry. The more theoretically grounded result connecting our observations to either a transformer architecture or the next token prediction training paradigm would be stronger.

A limitation of our evaluation is that it is restricted to toy-scale tabular datasets; however, this restriction is itself partly forced by the cost and context limits of the method being studied. For example, a 100K-row, 20-column numeric dataset contains 2×10^6 feature values; at roughly 8 input tokens per serialized CSV value, a single train/test prompt would require ~16 million input tokens before labels, instructions, retries, or output tokens. This already exceeds the current million-token context window of frontier Claude models, so the direct raw-table prompting protocol would be infeasible without chunking, sampling, or retrieval. Even ignoring that limit and pricing tokens at the same \$5/M input rate used in Table 2, the input alone would cost about \$80 per attempt, or roughly \$2,000 for a 5-repetition, 5-fold evaluation of one dataset, before retries and output costs.

Thus, the small scale of our experiments should not be read only as a limitation of this study, but also as evidence of a broader practical limitation of direct LLM-based tabular prediction. A single final run of the reported experiments costs roughly \$836 in input tokens alone (see the breakdown in Table 2). This does not include the many development, debugging, rerun, and failed-attempt calls made while designing the experiments. This cost profile makes direct LLM inference over raw tables hard to scale: real-world tabular prediction problems often involve far more rows and columns (often in the millions of rows and dozens if not hundreds of columns), repeated evaluation, tuning, and deployment-time use. Under that regime, naively placing the table in context is not merely expensive for research purposes, but likely infeasible as a practical prediction method.

Experiment	Eval Rows	LLM Calls	Attempts	Avg ctx.	Max ctx.	Input cost
Main benchmark	475	475	496	29.2k	80.3k	\$72
Memorisation probe	475	475	591	28.8k	84.2k	\$85
H1 experiment	1,375	1,375	1,512	32.9k	61.6k	\$248
H2 experiment	275	275	281	32.8k	63.3k	\$46
H3 experiment	825	825	836	17.6k	41.7k	\$74
H4 experiment	550	1,650	1,673	27.3k	55.7k	\$229
H5 experiment	1,500	1,500	1,531	23.1k	102.9k	\$177
2D suite experiment	200	800	800	2.4k	2.4k	\$9.6
Total	5,675	7,375	7,720	-	-	\$941

Table 2: Back-of-the-envelope LLM usage and input-token cost by experiment. Costs use Claude Opus 4.6 list pricing at \$5 per million input tokens. Average context is attempt-weighted, so retries are reflected in both the attempt count and the cost. The main benchmark and memorisation probe are all-19-dataset runs; H1–H5 use the retained f1-dataset analysis set, and the 2D suite is separate.

12

Conclusion

We set out to explain the failure of LLMs at in-context tabular prediction. A memorisation probe first separated capability from contamination. Controlled interventions then falsified four popular explanations – class overlap, the linearised CSV format, numeric tokenisation, and per-query test load – leaving one factor standing: dimensionality. In a random-projection sweep, the LLM was the only method of nine whose performance fell as columns were added. Where it works, in two dimensions, it predicts like a local, distance-based method: Gaussian processes and low-k nearest neighbours reproduce its decision boundaries on 91–92% of grid cells. In higher dimensions, however, no such characterisation survives: the best of 252 configured classical models agrees with the LLM on only 64.8% of predictions, and replacing an increasing, dimension-tuned fraction of any baseline’s predictions with the majority class adds at most 0.64 percentage points. The exact mechanics of the dissolution therefore remain unknown: dimension-dependent noise on top of a standard learner can describe part of the marginal performance drop, but not the actual predictions. It remains possible that the failure is a more structured form of guessing, particularly as LLMs have recently been shown to be poor generators of random numbers in the first place.

Tables are the humble two-dimensional data structure of applied machine learning, and our results say the failure on them is real but specific: the model reads tables fine – it retrieves a designated column at $d = 60$ without loss – but it cannot think with tables once the second (column) dimension grows. For practitioners, the implication is that no amount of prompt polishing aimed at format, precision, or batching should be expected to fix high-dimensional in-context prediction, and purpose-built tabular models remain the right tool [9]. For researchers, the bottleneck we isolate is quantitative and behavioural – a concrete target for mechanistic work.

13

Acknowledgements

We would like to thank Irene, Matt, Teresa, Manuel and Irasema for their existence, Sorin for his absence, all the Kapybaras and Boo for being cute and the Research team for being lit. Wojtek would like to give a special shout-out to Marta and conversely Marta would like to take this opportunity to express her gratefulness to Wojtek.

14

References

- [1] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? Investigations with linear models. In **International Conference on Learning Representations (ICLR)**, 2023.
- [2] Anthropic. Claude Opus 4.6 system card. <https://www.anthropic.com/claude-opus-4-6-system-card>, February 2026. System card. Accessed: 2026-07-28.
- [3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. **Advances in Neural Information Processing Systems**, 33:1877–1901, 2020.
- [4] Tuan Dinh, Yuchen Zeng, Ruisu Zhang, Ziqian Lin, Michael Gira, Shashank Rajput, Jy-yong Sohn, Dimitris Papailiopoulos, and Kangwook Lee. LIFT: Language-interfaced fine-tuning for non-language machine learning tasks. In **Advances in Neural Information Processing Systems**, 2022.
- [5] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, et al. Faith and fate: Limits of transformers on compositionality. In **Advances in Neural Information Processing Systems**, 2023.
- [6] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, et al. Large language models (LLMs) on tabular data: Prediction, generation, and understanding – a survey. **Transactions on Machine Learning Research**, 2024.

- [7] Josh Gardner, Juan C. Perdomo, and Ludwig Schmidt. Large scale transfer learning for tabular data via language modeling. In **Advances in Neural Information Processing Systems**, 2024.
- [8] Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? A case study of simple function classes. In **Advances in Neural Information Processing Systems**, 2022.
- [9] Marta Garnelo and Wojciech Marian Czarnecki. Developing foundation models for real-world tabular data. 2025. URL <https://fundamental.tech/whitepaper.pdf>.
- [10] Marta Garnelo, Dan Rosenbaum, Chris J. Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J. Rezende, and S. M. Ali Eslami. Conditional neural processes. In **International Conference on Machine Learning (ICML)**, 2018.
- [11] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural processes. In **ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models**, 2018.
- [12] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In **Advances in Neural Information Processing Systems, Datasets and Benchmarks Track**, 2022.
- [13] Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. TabLLM: Few-shot classification of tabular data with large language models. In **International Conference on Artificial Intelligence and Statistics (AISTATS)**, 2023.
- [14] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In **International Conference on Learning Representations (ICLR)**, 2023.
- [15] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmester, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. **Nature**, 637:319–326, 2025.
- [16] William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. **Contemporary Mathematics**, 26:189–206, 1984.
- [17] Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. UCI machine learning repository, 2023. <https://archive.ics.uci.edu>.
- [18] R. Thomas McCoy, Shunyu Yao, Dan Friedman, Mathew D. Hardy, and Thomas L. Griffiths. Embers of autoregression show how large language models are shaped by the problem they are trained to solve. **Proceedings of the National Academy of Sciences**, 121(41), 2024.

[19] Sewon Min, Xinxu Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In **Empirical Methods in Natural Language Processing (EMNLP)**, 2022.

[20] OpenAI. GPT-4 technical report, 2023. arXiv:2303.08774.

[21] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, 12:2825–2830, 2011.

[22] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. Tabicl: A tabular foundation model for in-context learning on large data. In **ICML 2025-Forty-Second International Conference on Machine Learning**, 2025.

[23] Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don’t always say what they think: Unfaithful explanations in chain-of-thought prompting. In **Advances in Neural Information Processing Systems**, 2023.

[24] Boris van Breugel and Mihaela van der Schaar. Position: Why tabular foundation models should be a research priority. In **International Conference on Machine Learning (ICML)**, 2024.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In **Advances in Neural Information Processing Systems**, 2017.

[26] Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In **International Conference on Machine Learning (ICML)**, 2023.

[27] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit Bayesian inference. In **International Conference on Learning Representations (ICLR)**, 2022.

A: Multi-dimensional data

Name	URL	Columns	Rows	Classes	Retained?	Discard reason
Breast Cancer Wisconsin (Diagnostic)	UCI 17	30	569	2	×	memorised
Iris	UCI 53	4	150	3	×	memorised
Wine	UCI 109	13	178	3	×	memorised
Ionosphere	UCI 52	34	351	2	✓	
Australian Credit Approval	UCI 143	14	690	2	✓	
Banknote Authentication	UCI 267	4	1372	2	×	memorised
Sonar	UCI 151	60	208	2	✓	
Heart Disease	UCI 45	13	303	5	×	nothing to learn
Parkinsons	UCI 174	22	195	2	✓	
Vertebral Column	UCI 212	6	310	3	✓	
Glass Identification	UCI 42	9	214	6	✓	
Ecoli	UCI 39	7	336	8	✓	
Haberman Survival	UCI 43	3	306	2	×	nothing to learn
Blood Transfusion Service Center	UCI 176	4	748	2	×	nothing to learn
Tic-Tac-Toe Endgame	UCI 101	9	958	2	×	nothing to learn
Dermatology	UCI 33	34	366	6	✓	
Balance Scale	UCI 12	4	625	3	✓	
Linear synthetic	Generated (code below)	32	150	2	✓	
Sine synthetic	Generated (code below)	32	150	2	✓	

Table 3: Datasets considered in the experiments. URL entries are clickable UCI record links. Retained datasets are those in the curated analysis set. The linear and sine rows are generated synthetically in the repository rather than fetched from UCI.

The table summarises the filtering step used to construct the multi-dimensional benchmark. We began from a small set of standard UCI tabular classification datasets and retained only those that gave a non-trivial learning problem under our protocol. Datasets marked as memorised were excluded when the task appeared too likely to be recoverable from prior exposure rather than from the provided examples; datasets marked as having nothing to learn were excluded when none of the baselines were able to outperform the Dummy baseline. The retained set is therefore not meant to be exhaustive, but to provide a compact collection of tabular tasks with varied dimensionality, sample size, and number of classes. The two synthetic datasets are included as controlled complements, where the target rule is known exactly and can be varied independently of any dataset-specific artefacts. They are generated as follows:

```

def generate_linear_data(num_samples=150, num_features=32, random_state
=42):
    np.random.seed(random_state)
    X = np.random.rand(num_samples, num_features) * 2 - 1
    true_coefficients = np.random.normal(size=num_features)
    y = X @ true_coefficients > 0
    return X, y

def generate_sin_data(num_samples=150, num_features=32, random_state=42):
    np.random.seed(random_state)
    X = np.random.rand(num_samples, num_features) * 2 - 1
    y = np.sin(X[:, :-1] * np.pi)
    larger = X[:, -1:] > y / 2
    result = np.sum(larger, axis=-1) % 2
    return X, result

```

B: 2D synthetic datasets

The 2D synthetic datasets provide a controlled probe of rule-learning behaviour in a setting where the full input space can be visualised. Each example consists of a point in the plane, with labels assigned by a geometric rule implemented in the generator code below. These tasks are not intended to mimic real tabular datasets; instead, they serve as simple diagnostic cases where the target structure is known and the induced decision boundary can be inspected directly.

This makes the 2D setting useful as a sanity check alongside the higher-dimensional experiments: when a model succeeds or fails, the result can often be related back to a concrete geometric pattern rather than to hidden dataset-specific artefacts.

Listing 1: Generation of the 2D probe datasets - helper functions.

```

from sklearn.datasets import (
    make_blobs, make_circles, make_classification,
    make_gaussian_quantiles, make_moons,
)

def _xor(n, noise, rng):
    X = rng.uniform(-1, 1, size=(n, 2))
    y = ((X[:, 0] > 0) ^ (X[:, 1] > 0)).astype(int)
    return X + rng.normal(scale=noise, size=X.shape), y

def _spirals(n, noise, rng):
    n2 = n // 2
    theta = np.sqrt(rng.uniform(0, 1, n2)) * 3.5 * np.pi
    r = theta + np.pi
    x1 = np.stack([np.cos(theta) * r, np.sin(theta) * r], axis=1)
    x2 = np.stack([-np.cos(theta) * r, -np.sin(theta) * r], axis=1)
    X = np.vstack([x1, x2]) / 10.0
    y = np.concatenate([np.zeros(n2, int), np.ones(n2, int)]
    return X + rng.normal(scale=noise, size=X.shape), y

def _checkerboard(n, noise, rng, k=3):
    X = rng.uniform(-1, 1, size=(n, 2))
    y = (
        np.floor((X[:, 0] + 1) * k / 2).astype(int)
        + np.floor((X[:, 1] + 1) * k / 2).astype(int)
    ) % 2
    return X + rng.normal(scale=noise, size=X.shape), y

def _stripes(n, noise, rng, k=4):
    X = rng.uniform(-1, 1, size=(n, 2))
    y = np.floor((X[:, 0] + 1) * k / 2).astype(int) % 2
    return X + rng.normal(scale=noise, size=X.shape), y

def _ring_in_ring(n, noise, rng):
    n3 = n // 3
    radii = np.concatenate([
        rng.uniform(0.0, 0.25, n3),
        rng.uniform(0.45, 0.65, n3),
        rng.uniform(0.85, 1.0, n - 2 * n3),
    ])
    theta = rng.uniform(0, 2 * np.pi, n)
    X = np.stack([radii * np.cos(theta), radii * np.sin(theta)], axis=1)
    y = np.concatenate([
        np.zeros(n3, int), np.ones(n3, int),
        np.zeros(n - 2 * n3, int),
    ])
    return X + rng.normal(scale=noise, size=X.shape), y

def _diag_band(n, noise, rng):
    X = rng.uniform(-1, 1, size=(n, 2))
    y = (np.abs(X[:, 0] - X[:, 1]) < 0.5).astype(int)
    return X + rng.normal(scale=noise, size=X.shape), y

def _wedge(n, noise, rng):
    X = rng.uniform(-1, 1, size=(n, 2))
    theta = np.arctan2(X[:, 1], X[:, 0])
    y = ((theta > -np.pi / 4) & (theta < np.pi / 2)).astype(int)
    return X + rng.normal(scale=noise, size=X.shape), y

```


Listing 2: Generation of the 2D probe datasets - assembly code.

```
def make_all_datasets(n=200, seed=17):
    rng = np.random.default_rng(seed)
    rs = lambda: int(rng.integers(1_000_000_000))

    datasets = [
        ("moons_clean", *make_moons(n, noise=0.05, random_state=rs())),
        ("moons_noisy", *make_moons(n, noise=0.25, random_state=rs())),
        ("circles_clean", *make_circles(
            n, noise=0.04, factor=0.55, random_state=rs())),
        ("circles_noisy", *make_circles(
            n, noise=0.18, factor=0.50, random_state=rs())),
        ("blobs_2", *make_blobs(
            n, centers=2, cluster_std=1.2, random_state=rs())),
        ("blobs_3", *make_blobs(
            n, centers=3, cluster_std=1.0, random_state=rs())),
        ("blobs_4", *make_blobs(
            n, centers=4, cluster_std=0.9, random_state=rs())),
        ("linear_sep", *make_classification(
            n_samples=n, n_features=2, n_informative=2,
            n_redundant=0, n_clusters_per_class=1,
            class_sep=2.0, random_state=rs())),
        ("hard_linear", *make_classification(
            n_samples=n, n_features=2, n_informative=2,
            n_redundant=0, n_clusters_per_class=1,
            class_sep=0.7, random_state=rs())),
        ("gauss_quant_2", *make_gaussian_quantiles(
            n_samples=n, n_classes=2, random_state=rs())),
        ("gauss_quant_3", *make_gaussian_quantiles(
            n_samples=n, n_classes=3, random_state=rs())),

        ("xor_clean", *_xor(n, noise=0.05, rng=rng)),
        ("xor_noisy", *_xor(n, noise=0.18, rng=rng)),
        ("spirals", *_spirals(n, noise=0.04, rng=rng)),
        ("checkerboard_3", *_checkerboard(n, noise=0.03, rng=rng, k=3)),
        ("checkerboard_4", *_checkerboard(n, noise=0.03, rng=rng, k=4)),
        ("stripes", *_stripes(n, noise=0.03, rng=rng, k=4)),
        ("ring_in_ring", *_ring_in_ring(n, noise=0.03, rng=rng)),
        ("diag_band", *_diag_band(n, noise=0.04, rng=rng)),
        ("wedge", *_wedge(n, noise=0.04, rng=rng)),
    ]

    return datasets
```

C: Full experimental details

C.1 Benchmark datasets

We assemble a pool of 19 small classification datasets: classical real-world benchmarks available through the scikit-learn library and the UCI repository [17, 21] - iris, wine, breast cancer, ionosphere, sonar, glass, ecoli, dermatology, vertebral column, balance scale, australian credit, parkinsons, tic-tac-toe, bank, heart disease, haberman survival, blood transfusion together with two synthetic generators designed as controlled extremes:

linear. $x \sim \mathcal{U}([-1, 1]^d)$, a random direction $w \sim \mathcal{N}(0, I_d)$, and labels

$$y = \begin{cases} 1 & \text{if } w^\top x > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Linearly separable by construction at every dimensionality, and noise-free.

sin. $x \sim \mathcal{U}([-1, 1]^d)$; each of the first $d-1$ coordinates casts a vote

$$v_j = \begin{cases} 1 & \text{if } x_d > \sin(\pi x_j)/2, \\ 0 & \text{otherwise,} \end{cases}$$

and the label is the parity of the votes, $y = (\sum_{j=1}^{d-1} v_j) \bmod 2$. Highly non-linear, with feature interactions of order up to d .

Generator code is given in Section A. We generate $n = 150$ points with $d = 32$. After the hygiene checks of Section 5, the main analyses retain 11 of the 19 datasets. A full list of the considered datasets and their properties can be found in Table 3 of the appendix.

C.2 The 2D suite

Complementing the benchmarks, we construct a suite of twenty two-dimensional tasks with qualitatively different decision boundaries: Gaussian blobs with two to four classes, checkerboards, concentric circles and rings, diagonal bands, quantised Gaussians, hard and easy linear separations, moons, spirals, stripes, wedges, and XOR, in clean and noisy variants. A visual rendering of the datasets can be found in Figure 16 and the code to generate these is provided in Section B in the Appendix.

C.3 Protocol: benchmark experiments

Model configuration. The model queried is `claude-opus-4-6` with a maximum of 16,000 output tokens and a 60 s per-request timeout. Sampling is left at the API defaults (temperature 1.0; no top- p or system prompt is set); each query is a single user turn with no tools and no multi-turn interaction. Each split is attempted up to 10 times. After a failed request on zero-indexed retry $k \in \{0, \dots, 8\}$, the code sleeps for $\min(2 \cdot 2^k, 60) + u_k$ seconds, where $u_k \sim \text{Unif}(0, 1)$, before retrying; if all 10 attempts fail, the split is logged as NaN. A reply that fails to parse, or that parses to an array of the wrong length, is retried; a result is accepted only when the returned array length equals the number of test rows.

Data presentation. Each query is one train/test split sent as three plain-text CSV documents plus one text instruction, all in a single user message: `x_train_csv.csv` (training features), `y_train_csv.csv` (training labels), and `x_test_csv.csv` (test features), each attached as a document content block with media type `text/plain`. CSVs are produced with `pandas.DataFrame(...).to_csv(index=False)`: a header row of integer column indices $0..d-1$, no row index, and full floating-point precision in the main run (`rounding=None`). Features are mean-imputed with the imputer fit on the training fold only; on clean datasets this is a no-op. No scaling, standardisation, or dimensionality reduction is applied to the LLM input – scaling exists only inside the sklearn baseline pipelines. The label files themselves contain integer-encoded targets in $0, \dots, K-1$. In the runs reported here, the instruction text also included a legacy binary-label clarification ("True means class 1 and False means class 0"); this phrase was redundant for binary integer labels and inapplicable to multiclass datasets, where the CSV labels remained integer class IDs.

The exact prompt. With n_{test} the number of test rows:

```
You are given training data (x_train_csv.csv with features, y_train_csv.csv with labels) and test features (x_test_csv.csv). In y_train_csv.csv, True means class 1 and False means class 0.

Analyse the relationship between features and labels in the training data, then predict the correct class label for each of the n_test rows in x_test_csv.csv. Labels are non-negative integers starting from 0.

End your response with a line in exactly this format: PREDICTIONS: [p1,p2,...,pn_test] where every value is an integer label (no True/False), and the array has exactly n_test elements.
```

Output parsing and metric. Predictions are extracted with a regular expression, required to match n_{test} in length, and scored with `sklearn.metrics.accuracy_score`. The raw predicted label vector is stored alongside the score for the behavioural analyses of Section 8.

Context vs. target sizes (the in-context split). The split is 5-fold stratified cross-validation (`StratifiedKFold(n_splits=5, shuffle=True, random_state=rep)`), repeated over 5 seeds (`rep = 0.4`), giving 25 train/test splits per dataset and $19 \times 25 = 475$ LLM queries in the main run. Per split, the context is the 80% train fold and the target is the remaining 20% test fold, classified in a single query.

C.4 Protocol: 2D experiments

The LLM querying protocol is the same in spirit as above – same model, 16,000 maximum output tokens, API-default sampling, no system prompt, no tools, and up to 10 retries with exponential backoff – but the 2D probe is not run as stratified cross-validation over train/test folds. Each task first generates 200 labelled source points. A fixed random subsample of 60 of these points is used as the labelled in-context training set. Separately, we construct a regular 20×20 grid of 400 query points over the training-set bounding box and ask the LLM to label those grid points. The 400 grid points are queried in shuffled chunks of 100, so one dataset/seed requires four LLM calls rather than one call for an entire held-out fold. The data are embedded directly in the prompt as rounded coordinate-label text, not attached as CSV documents, and the required output format is one ‘INDEX LABEL’ line per test point rather than a final ‘PREDICTIONS: [...]’ array. The 10 probe seeds change only the shuffle order of grid points; the 60 training points are fixed.

C.5 Protocol: explanation experiments

The explanation experiments use the same synthetic 2D tasks described above, but they are a separate diagnostic protocol from the 400-point grid probe. For each task, we generate 200 labelled source points and take one deterministic stratified split: a 5-fold `StratifiedKFold` with `shuffle=True` and `random_state=0`, using fold 0 as the held-out target set. Thus each explanation run gives the LLM 160 labelled context points and scores predictions on 40 held-out target points. We make one LLM call per dataset, using the same request settings as the other LLM experiments: 16,000 maximum output tokens, API-default sampling, no system prompt, no tools, and up to 10 retries with exponential backoff. The prompt supplies `x_train_csv.csv`, `y_train_csv.csv`, and `x_target_csv.csv` as text CSV documents, and asks the model to return exactly one JSON object containing a length-40 `predictions` list, a natural-language explanation, and a compact vectorized NumPy `rule_expression` over `x0`, `x1`, and `np`. We parse and validate the JSON response, require exactly 40 valid labels, and report point accuracy as $c/40$ in the figure panels. The `rule_expression` is evaluated only to render the extracted decision boundary; the displayed accuracy is computed solely on the 40 held-out target rows.

C.6 Baselines

The fixed comparison suite consists of k -nearest neighbours with $k \in \{1, 5, 10\}$, a depth-limited random forest, ℓ_2 -regularised logistic regression on standardised features, AdaBoost (100 estimators), gradient boosting, and a Gaussian-process classifier with an RBF kernel (length scale 100) on standardised features. All baselines see the same projected, imputed features and the same cross-validation splits as the LLM. For the behavioural search of Section 8 we additionally instantiate **252 configured variants** spanning the kNN family (choices of k , metric, and weighting), decision trees, random and extra forests, bagging, boosting, SVMs, Gaussian processes across length scales, MLPs, naive Bayes, discriminant analysis, nearest centroid, and dummy predictors. All configuration details for our baselines can be found in Table 4 in the Appendix.

C.7 Metrics and normalisation

The base metric is test-set accuracy per split. For aggregation across datasets we use the normalised score

$$\frac{\text{acc} - \text{acc}_{\text{maj}}}{\text{acc}_{\text{best}} - \text{acc}_{\text{maj}}},$$

where acc_{maj} is the accuracy of always predicting the most common class. For the property-sweep figures, acc_{best} is held fixed as the best non-LLM baseline mean accuracy on the original, unmodified dataset, rather than recomputed separately within each transformed condition. Thus 0 corresponds to majority guessing, 1 corresponds to the original-data best-baseline reference, and values above 1 indicate that a model under a transformed condition exceeds that original-data reference.

D: Full Separation experiments visualisation

This section collects the full visual summary of the separation experiments. The figure shows the PCA projections used to inspect how the datasets evolve across the experimental pipeline, providing a compact qualitative check alongside the quantitative results reported in the main text.

E: Calculating MDS

For the MDS visualization, each configured baseline is represented by its vector of held-out accuracies across evaluation slots, where an evaluation slot is a dataset, repetition, and fold. We define the distance between two configured baselines p and q as the mean L1 distance between their accuracy vectors,

$$D(p, q) = (1/T) \sum_{t=1}^T |s_{p,t} - s_{q,t}|$$

We then apply weighted metric MDS to this distance matrix, up-weighting pairs involving the LLM so that distances from the LLM are preserved more faithfully. Model families are used only to color points in the plot.

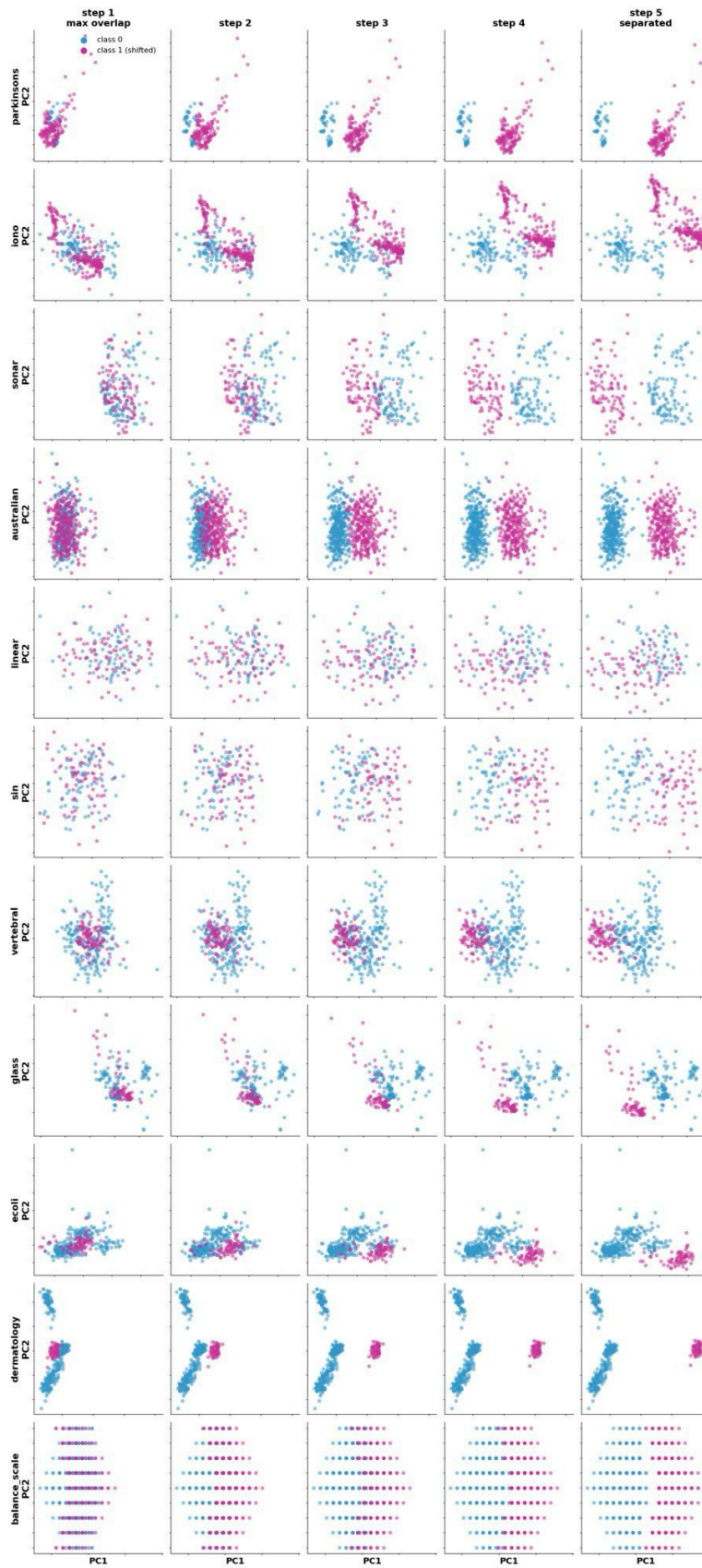


Figure 14: Visualisation of the separability experiments.

F: Baseline configuration - main baselines

For the experiments that test the five different hypothesis we consider eight baselines that we compare to the LLM. The following table describes the hyperparameter settings for said baselines:

Label	Estimator	Model-specific preprocessing	Configuration
LR	Logistic regression	StandardScaler	LogisticRegression with max_iter=10000 and random_state=0; sklearn defaults otherwise: L2 penalty, C=1.0, solver=lbfgs.
GBT	Gradient boosting trees	None	GradientBoostingClassifier with n_estimators=100 and random_state=0; sklearn defaults otherwise: loss=log_loss, learning_rate=0.1, depth=3 regression trees.
1-NN	k-nearest neighbours	None	KNeighborsClassifier with n_neighbors=1; sklearn defaults otherwise: uniform weights, Minkowski distance with p=2.
5-NN	k-nearest neighbours	None	KNeighborsClassifier with n_neighbors=5; sklearn defaults otherwise: uniform weights, Minkowski distance with p=2.
10-NN	k-nearest neighbours	None	KNeighborsClassifier with n_neighbors=10; sklearn defaults otherwise: uniform weights, Minkowski distance with p=2.
GP-RBF	Gaussian process classifier	StandardScaler	GaussianProcessClassifier with kernel 1.0 * RBF(length_scale=100.0) and random_state=0; sklearn defaults otherwise: one-vs-rest multiclass handling and max_iter_predict=100.
Ada	AdaBoost	None	AdaBoostClassifier with n_estimators=100 and random_state=0; sklearn defaults otherwise: decision-stump base estimator, learning_rate=1.0, algorithm=SAMME.R.
RF	Random forest	None	RandomForestClassifier with max_depth=4 and random_state=0; sklearn defaults otherwise: n_estimators=100, Gini splits, bootstrap samples, max_features=sqrt.

Table 4: Main baseline configurations used in the compact/haystack panels; All baselines share the experiment-level preprocessing: categorical feature encoding where needed, 5 repetitions of 5-fold StratifiedKFold with shuffle=True and random_state=rep, and mean imputation fit on the training fold only. Defaults refer to sklearn 1.5.1, the version used in this workspace.

G: Baseline configuration - baseline sweep

For the behaviour matching experiments we consider a large sweep of baselines with the following hyperparameters.

Family	Count	Preproc.	Sweep / variants
<i>Linear and margin-based models</i>			
Logistic regression	11	Standard	$C \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$; L1 variants; elastic-net; unregularized; default reference.
Ridge classifier	10	Standard	$\alpha \in \{10^{-3}, 10^{-2}, 10^{-1}, 0.5, 1, 2, 5, 10, 50, 500\}$.
SGD classifier	10	Standard	Loss in {hinge, log_loss, modified_huber, perceptron, squared_hinge}; selected α and L1 variants.
Perceptron	10	Standard	Penalties {none, l1, l2, elasticnet}; selected $\alpha \in \{10^{-4}, 10^{-3}, 10^{-2}\}$.
Passive-aggressive	10	Standard	$C \in \{0.01, 0.1, 1, 10, 100\}$ crossed with hinge and squared_hinge.
Linear SVC	10	Standard	$C \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$; hinge-loss and L1-penalized variants.
SVC	10	Standard	RBF kernels over selected C and γ ; linear, polynomial degree 2/3, and sigmoid kernels.
<i>Distance, centroid, and Gaussian-process models</i>			
k -NN	13	–	$k \in \{1, 2, 3, 5, 7, 10, 15, 25\}$ with uniform/distance weighting for selected values.
Nearest centroid	10	–	Euclidean/manhattan metrics with shrink thresholds in {0.1, 0.5, 1, 2} plus no-shrink variants.
Gaussian process	21	Standard	RBF length scales from 0.01 to 100; Matérn $\nu \in \{0.5, 1.5, 2.5\}$; constant-RBF and optimizer/multiclass variants.
<i>Tree and ensemble models</i>			
Decision tree	10	–	Depths {1, 2, 3, 5, 10, None} with Gini; selected entropy variants.
Extra tree	10	–	Depths {1, 3, 5, 10, None} with Gini; selected entropy/log-loss variants.
Random forest	11	–	Selected combinations of $n_{\text{trees}} \in \{10, 50, 100, 200, 500\}$ and depth {2, 3, 4, 8, 16, None}.
AdaBoost	11	–	Selected combinations of $n_{\text{estimators}} \in \{10, 25, 50, 100, 200, 500\}$ and learning rate {0.1, 0.5, 1, 2}.
Gradient boosting	11	–	Selected combinations of estimators, learning rate, and depth; includes the default reference model.
Histogram gradient boosting	10	–	Max iterations {50, 100, 200, 500}, learning rate {0.01, 0.1, 0.5, 1}, and selected leaf counts.
Bagging	10	–	Selected combinations of $n_{\text{estimators}} \in \{10, 50, 100, 200\}$ and max-sample fractions {0.3, 0.5, 0.7, 1}.
<i>Neural, naive-Bayes, discriminant, and dummy models</i>			
MLP	10	Standard	Hidden layers from one- and two-layer networks; selected α and learning-rate values.
Gaussian NB	10	–	Variance smoothing in $\{10^{-12}, 10^{-10}, 10^{-9}, 10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}, 1, 10, 100\}$.
Bernoulli NB	10	–	Selected combinations of $\alpha \in \{0.01, 0.1, 0.5, 1, 10\}$ and binarization threshold {0, 0.5, 1, 2}.
Multinomial NB	10	Min-max	$\alpha \in \{10^{-3}, 10^{-2}, 10^{-1}, 0.5, 1, 2, 5, 10, 50, 100\}$.
Linear discriminant analysis	10	–	Solvers svd, lsqr, and eigen; selected shrinkage values.
Quadratic discriminant analysis	10	–	Regularization in {0, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9}.
Dummy classifier	4	–	Strategies most_frequent, prior, uniform, and stratified.

Table 5: Baseline classifier library used for prediction-agreement analyses. The matched comparison pool contains 252 baselines.

H: Results using Qwen as the LLM

The results reported in this paper are obtained with Claude Opus 4.6. We also ran the 2D behaviour matching experiments with Qwen3-235B-A22B. The highest agreement between Qwen and the sweep of baselines is shown in the following plot:

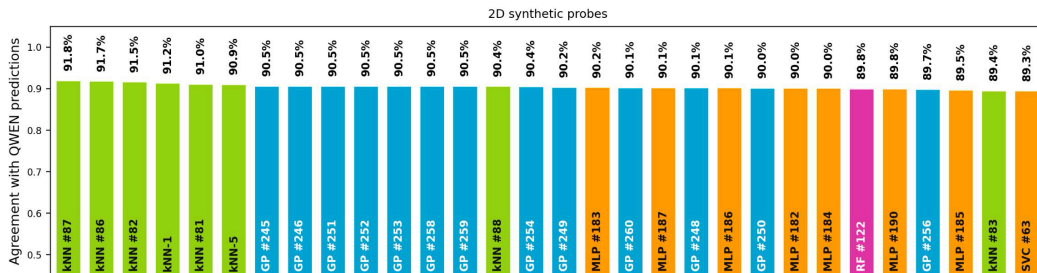


Figure 15: **Nearest baseline prediction agreement between Qwen and the baselines.** Top-30 non-LLM baselines ranked by agreement with the Qwen predictions on the 2D synthetic datasets. Agreement is computed over matched predictions on the same evaluation points.

I: 2D predictions - LLM decision boundaries

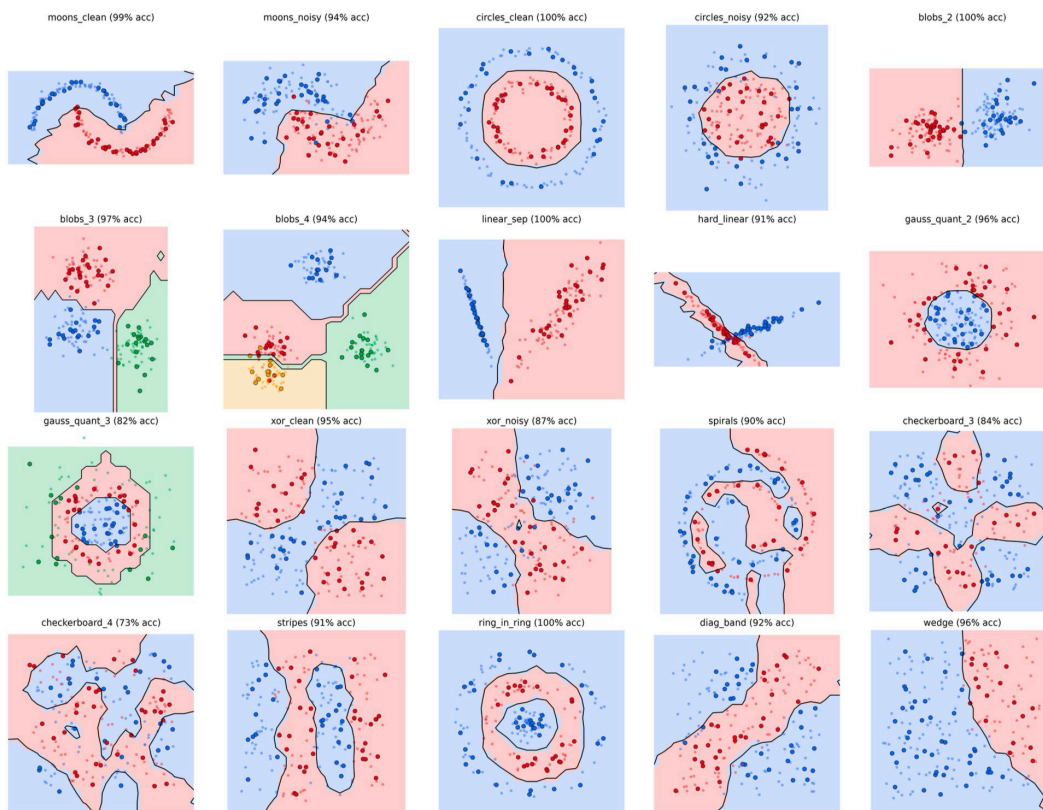


Figure 16: **LLM decision boundaries on synthetic 2D tasks.** For each probe task, the LLM is given 60 labeled examples and queried on a 20×20 grid. Colors show the LLM majority-vote class prediction across ten shuffled query orders, illustrating the decision boundary induced by the in-context examples.

J: LLM predictions vs its explanations

LLM predictions match the data, Explanation matches the data

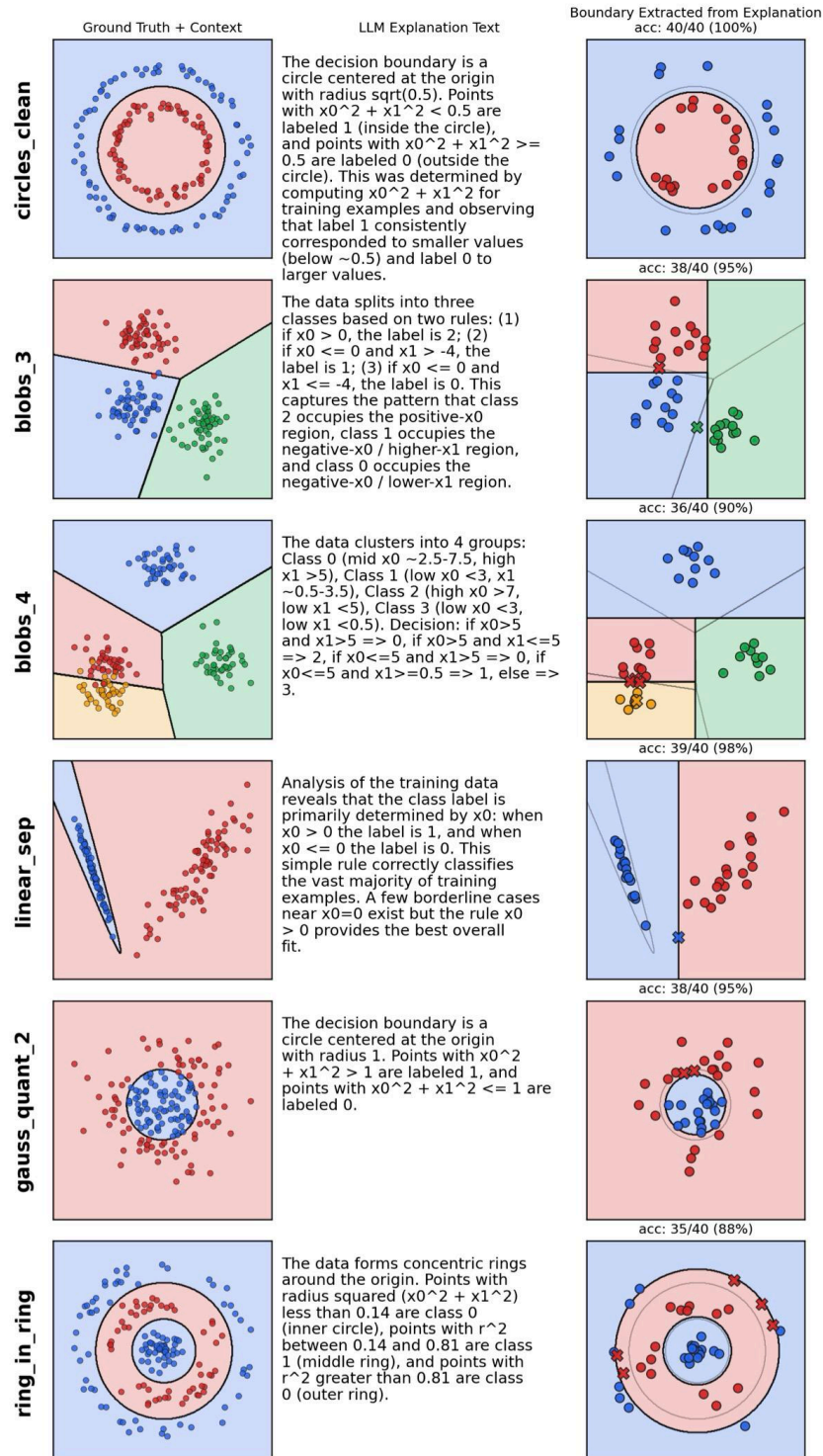


Figure 17: **LLM predictions and explanations both match the data.** Each row shows one 2D probe task. The left panel shows the ground-truth decision regions with the labelled context points, the centre panel shows the LLM explanation, and the right panel overlays the LLM target predictions on the ground-truth background together with the decision boundary extracted from the explanation. Marker colour denotes the predicted class; circles are correct predictions and crosses are incorrect predictions.

LLM predictions match the data, Explanation does not match the data

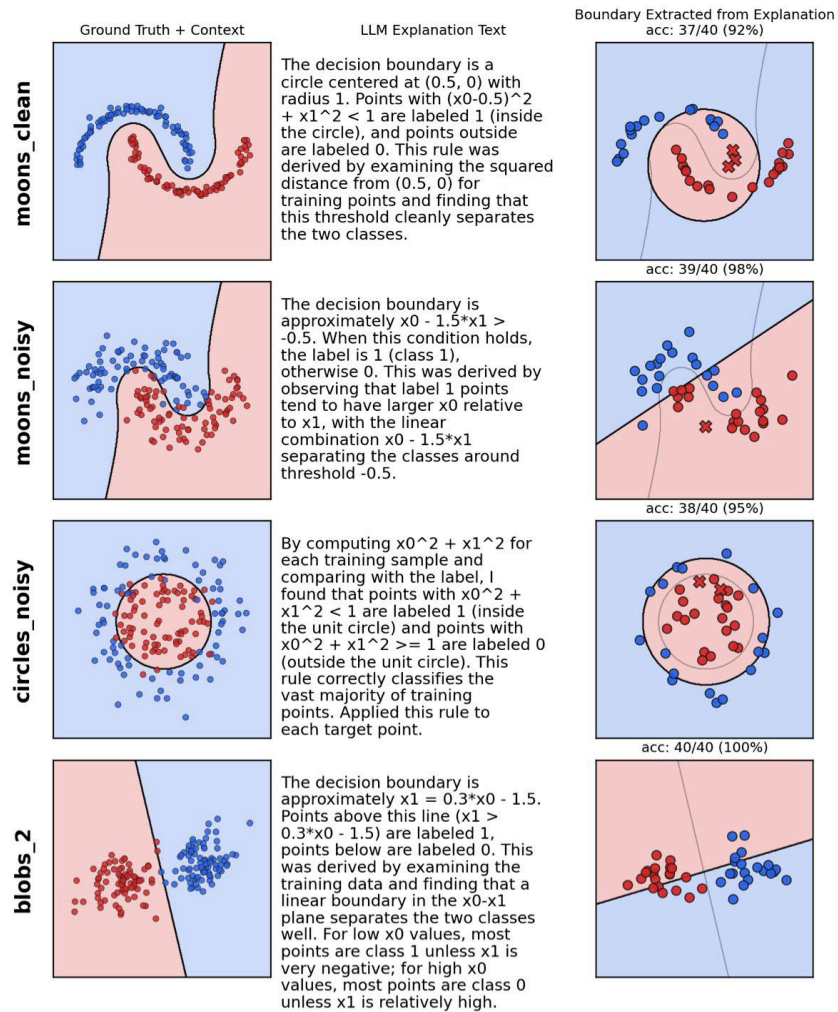


Figure 18: **LLM predictions match the data but the explanations do not.** Format as in Fig. 17. Here the target predictions largely agree with the ground truth, but the stated rule and the boundary extracted from it are qualitatively mismatched to the data-generating structure.

LLM predictions do not match the data Explanation does not match the data (part 1)

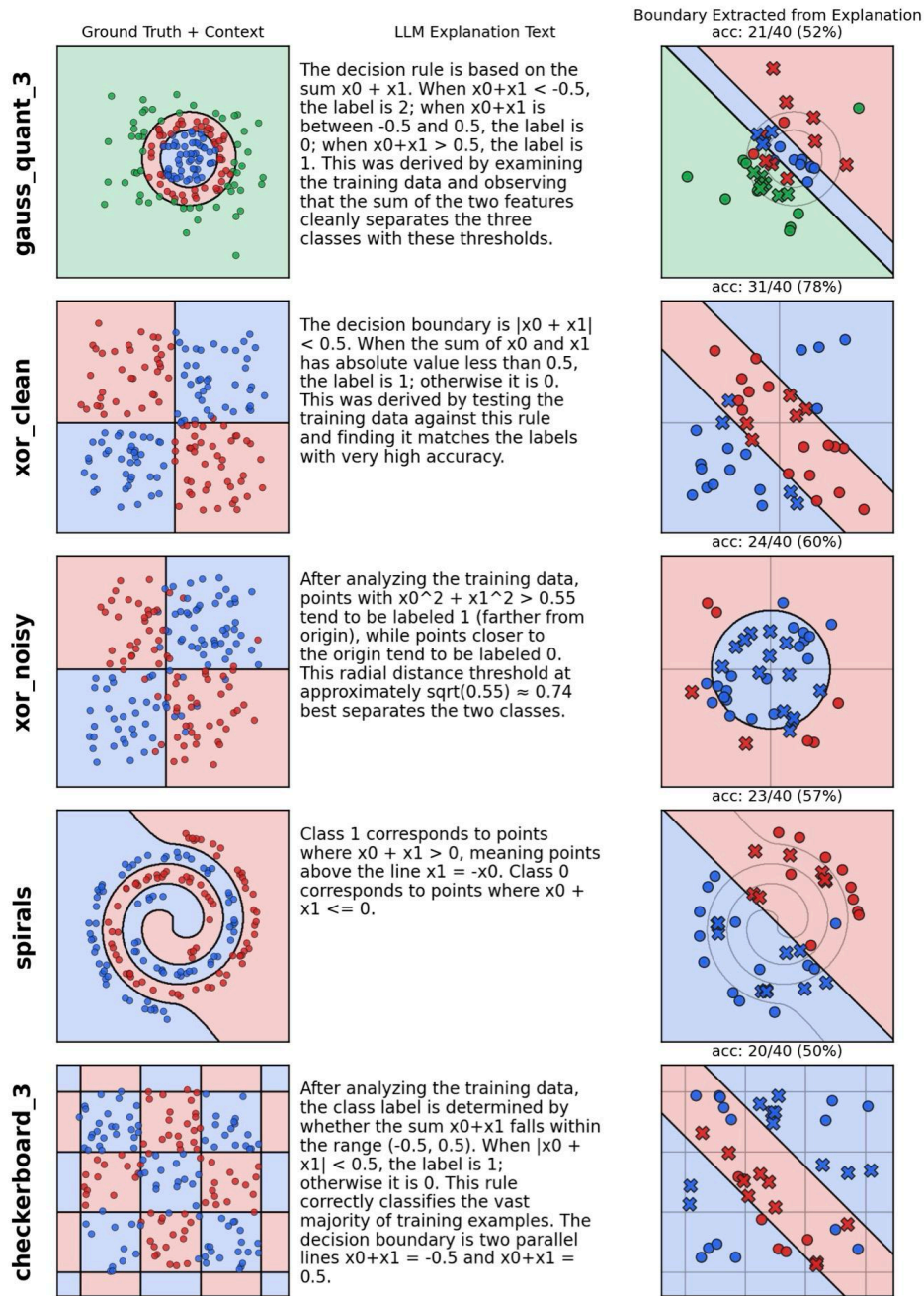


Figure 19: **Neither LLM predictions nor explanations match the data.** Format as in Fig. 17. For these tasks, the model fails to recover the target labels and its explanation describes a boundary that is qualitatively different from the ground truth. This panel shows the first half of this category.

LLM predictions do not match the data Explanation does not match the data (part 2)

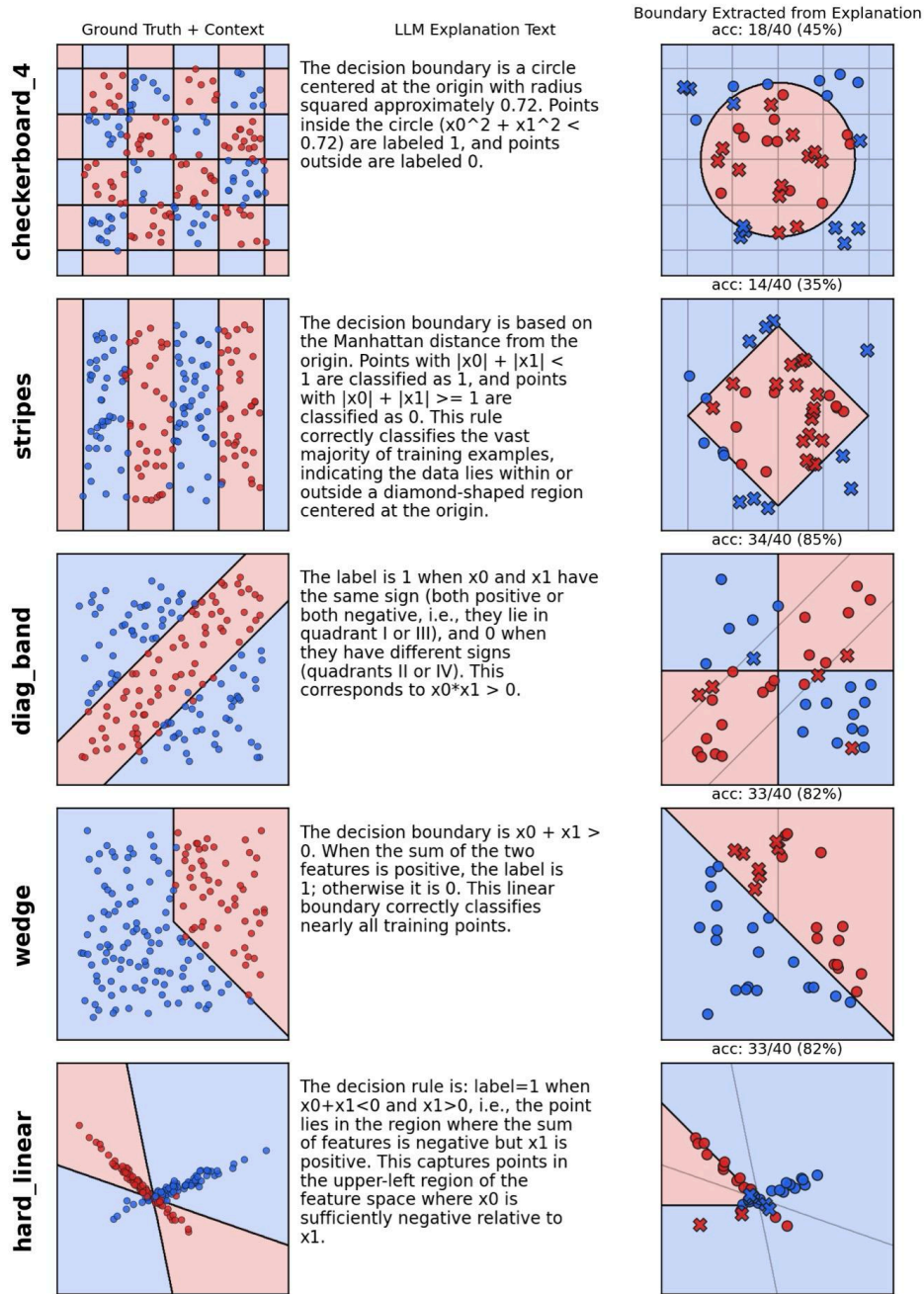


Figure 20: **Neither LLM predictions nor explanations match the data.** Format as in Fig. 17. This panel shows the second half of the cases where both the predictions and the extracted explanatory boundary fail to match the data.