RETHINKING TABULAR FOUNDATION MODELS ON DATA STREAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tabular foundation models (TFMs) outperform established machine learning models on tabular benchmarks through in-context learning. Building on this success, interest is growing in applying them to data streams, where data arrive continuously and evolve over time. On a stream, a TFM adapts by updating its context rather than its parameters, so its accuracy and cost depend on which examples it keeps and how often it rebuilds its context. We therefore present a systematic study of TFMs on data streams, covering memory management, computational cost, and stream-specific challenges such as concept drift and delayed labels. We find that TFMs achieve the highest predictive performance, and that simply retaining the most recent examples is as effective as existing memory management techniques. They also recover faster than streaming learners after drift and keep the highest accuracy under label delay. This accuracy, however, comes at a high serving cost, since a nearly unchanged context is re-encoded at every prediction. These results point to architectural efficiency as the way forward for in-context stream learning.

1 INTRODUCTION

Data streams arise across a wide range of tabular applications, from fraud and intrusion detection to sensor monitoring and clinical decision support (Dal Pozzolo et al., 2014; Shyaa et al., 2024; Arora et al., 2024). Unlike batch learning, stream learning must predict and adapt continuously under bounded memory and time, while the data-generating distribution changes (Gama et al., 2014; Bifet et al., 2018). Numerous stream learners have been proposed, from incrementally grown trees and drift-triggered ensembles (Domingos & Hulten, 2000; Gomes et al., 2017; 2019; Bifet et al., 2010b) to online AutoML systems that adapt the whole pipeline (Celik et al., 2023; Kulbach et al., 2022; Verma et al., 2024), each adapting by updating model parameters as data arrives. Tabular foundation models (TFMs) offer a different route, predicting from labelled examples held as context in a single forward pass without updating parameters (Müller et al., 2022; Hollmann et al., 2025), so adaptation to drift is handled entirely by updating the context memory. Recent approaches such as DualFIFO (Lourenço et al., 2026) and CURE (Lee et al., 2026) exploit this property and report large gains over streaming ensembles under a prequential protocol (Gama et al., 2013). However, their evaluations are narrow in the streams and costs they cover, leaving no controlled account of how backbone, memory, label delay, and drift jointly determine predictive performance and serving cost.

To close this gap, in this paper we systematically study TFMs on data streams. Our contributions are as follows:

- We compare TFM backbones against streaming learners and streaming AutoML systems under the same protocol on real and synthetic streams with different types of drift, and find that TFMs are the most accurate but also the most costly learners (Figure 1).
- We conduct a stream-specific analysis of memory composition, memory size, label delay, context refresh rate, and query batching, and find that backbone quality and memory size dominate predictive performance, while the memory management policy matters mainly under delayed labels.
- We identify repeated context encoding as the dominant serving cost of streaming TFMs, and show with compact causal models that reusing encoded context keeps cost nearly constant across memory sizes while staying close to re-encoding in accuracy.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

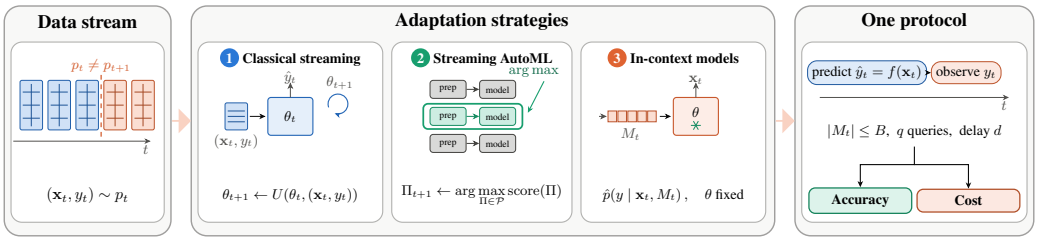

Figure 1: Approaches to adaptation in data streams, compared under a common prequential protocol measuring predictive performance and computational cost.

2 Background

Data streams. A data stream is an unbounded sequence of examples $(\mathbf{x}_t, y_t)$ arriving one at a time, where $\mathbf{x}_t \in \mathcal{X}$ is a feature vector and $y_t \in \mathcal{Y}$ its class label. Each example is drawn from a distribution p_t that may change over time, and such a change is a *concept drift*, which may be abrupt, gradual, incremental, or recurring (Gama et al., 2014; Lu et al., 2019). A stream classifier must use bounded memory and time per example, be ready to predict at any time, and adapt when p_t changes (Bifet et al., 2018). It is evaluated *prequentially* (test-then-train), in which each example first tests the current model and only then becomes available for learning (Gama et al., 2013), and in practice its label may arrive only after a delay of d examples (Grzenda et al., 2020). Most stream learners meet these requirements by updating the model as data arrives. Hoeffding trees grow incrementally from sufficient statistics (Domingos & Hulten, 2000; Bifet & Gavaldà, 2009), drift detectors such as ADWIN trigger a response when the distribution shifts (Bifet & Gavaldà, 2007), and ensembles such as Adaptive Random Forest, Streaming Random Patches, and Leveraging Bagging replace degraded members (Gomes et al., 2017; 2019; Bifet et al., 2010b). SAM-kNN instead retains past examples in short- and long-term memories (Losing et al., 2016), and is the classical learner closest to context-based adaptation. Online AutoML extends adaptation to the whole pipeline, re-running a search on detected drift (Celik et al., 2023), evolving a population of pipelines (Kulbach et al., 2022), or searching around the current best configuration (Verma et al., 2024).

Tabular foundation models. A tabular foundation model (TFM) is a transformer pretrained once on a large number of synthetic tables (Hollmann et al., 2023; Google Research, 2026). At prediction time, a context $D_{\text{ctx}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ of labelled rows and a query $\mathbf{x}^*$ pass through a single forward pass, $\hat{p}(y \mid \mathbf{x}^*, D_{\text{ctx}}) = f_\theta(\mathbf{x}^*, D_{\text{ctx}})$, and on small and medium tables TFMs now match or exceed tuned gradient-boosted trees (Erickson et al., 2025). TabPFN lets every context row attend to every other row directly (Hollmann et al., 2025), while TabICL first embeds each row from its columns and then lets rows attend to one another (Qu et al., 2026), and TabFM alternates attention over columns and rows before attending over compressed row vectors (Google Research, 2026), so in all cases encoding the context costs time quadratic in n. Drift-Resilient TabPFN (Helli et al., 2024) is the one backbone pretrained for shift, on temporally shifting priors with a known time covariate. Context construction has been studied for fixed tables, either by retrieving rows per query (Thomas et al., 2024) or by compressing and pruning the context (Feuer et al., 2024), but all assume a fixed pool of rows drawn from a single distribution.

Streaming TFMs. In a stream, the context becomes a memory M_t of at most B labelled rows that must be *maintained* as rows keep arriving, trading recent rows that allow fast adaptation against older rows that preserve earlier concepts (Lourenço et al., 2026). The closest prior works (Appendix E) are DualFIFO (Lourenço et al., 2026), which splits the memory into a short-term FIFO buffer and a class-balanced long-term store, and CURE (Lee et al., 2026), which evicts redundant rows to keep the memory informative. Their evaluations leave several gaps open. First, neither compares performance across drift types, since DualFIFO is evaluated only on real-world streams, where drift type is unknown, and CURE includes a single synthetic stream, with abrupt drift. Second, neither places serving cost beside that of classical learners. Because M_t and M_{t+1} differ by a single row, every step re-encodes almost the same set of rows, yet DualFIFO times ten instances per forward pass and CURE compares its per-step runtime only with DualFIFO. Third, both assume that each label arrives immediately, although labels in deployed streams often arrive late. These gaps motivate the design of our study.

Table 1: Evaluation streams.

Stream	Type	Instances	Features	Classes	Drift
`electricity` (Harries, 1999)	real	45,312	8	2	unknown
`noaa` (Elwell & Polikar, 2011)	real	18,159	8	2	unknown
`meter` (Souza et al., 2020a)	real	22,948	96	10	unknown
`rialto` (Losing et al., 2016)	real	82,250	27	10	unknown
`posture_no8` (Kaluza et al., 2010)	real	163,477	3	10	unknown
`agr_a` (Agrawal et al., 1993)	synth.	30,000	9	2	abrupt
`sea_g` (Street & Kim, 2001)	synth.	100,000	3	2	gradual
`rbf_m` (Bifet et al., 2010a)	synth.	100,000	10	5	incremental
`tree_r` (Domingos & Hulten, 2000)	synth.	60,000	10	2	recurring

3 EXPERIMENTAL SETUP

Streams. The evaluation covers real and synthetic classification streams (Table 1). Several real streams are distributed by the USP data stream repository (Souza et al., 2020b). Only datasets whose row order reflects the order in which observations arrived are included, since an arbitrary file order does not represent a true stream. The synthetic streams use data generators with abrupt, gradual, incremental, and recurring drift, the shapes drift takes over time (Lu et al., 2019). As each drift type is paired with a different generator, they serve only to describe the stream conditions and not to compare them directly.

Methods. The comparison covers dedicated streaming learners, streaming AutoML, and in-context TFMs. Classical streaming learners include Hoeffding Adaptive Tree (Bifet & Gavaldà, 2009), Adaptive Random Forest (Gomes et al., 2017), Streaming Random Patches (Gomes et al., 2019), Leveraging Bagging (Bifet et al., 2010b), and SAM-kNN (Losing et al., 2016). Streaming AutoML methods include ASML (Verma et al., 2024), EvoAutoML (Kulbach et al., 2022), and OAML (Celik et al., 2023). In-context TFMs include CURE (Lee et al., 2026), DualFIFO (Lourenço et al., 2026), and FIFO-based variants of TabICL v2 (Qu et al., 2026), TabPFN v1 (Hollmann et al., 2023), Drift-Resilient TabPFN (Helli et al., 2024), and TabFM (Google Research, 2026), where TabPFN v1 is also the backbone of DualFIFO and Drift-Resilient TabPFN. These TFMs span different backbone designs, pretraining strategies, and model scales, from models with tens of millions of parameters to a billion-scale backbone, allowing us to examine architecture, scale, and drift-oriented pretraining. No-Change and Majority-Class serve as simple reference baselines. Following established evaluation practice (Erickson et al., 2025; Lourenço et al., 2026), all methods use the settings and implementations described in their original work, with streaming learners and streaming AutoML methods running on CPU and TFMs on GPU.

Evaluation protocol and metrics. All methods follow the same prequential protocol, where each arriving instance is first predicted and becomes available for learning only after a label delay of d instances (Gama et al., 2013; Grzenda et al., 2020). Memory-based methods retain at most B labelled examples, and unless varied explicitly, all runs have no label delay d and share the same budget B across memory-based methods. We report accuracy together with balanced accuracy, which weights classes equally, and the chance-corrected statistics Cohen's κ, which adjusts for agreement expected by chance (Cohen, 1960), and temporal κ_t, which adjusts for the accuracy of the No-Change baseline (Bifet et al., 2015). Latency is reported in milliseconds per instance, measured as wall-clock serving time on each method's own host, and memory growth per instance.

To support reproducibility, we release the full experimental pipeline used in this study, including the code, run configurations for every experiment, and the raw result files behind all reported tables and figures.[1]

[1] `https://anonymous.4open.science/r/StreamTFM-Study`

Table 2: Predictive performance and cost of all methods, averaged over streams. Rank is the per-stream accuracy rank, time uses the geometric mean, and memory is growth per instance. Best **in bold**, second underlined.

	Rank		Accuracy			Balanced acc.		Cohen's κ		κ_t		Time (ms/inst.)		Memory (KB/inst.)	
Method	mean	med.	mean	med.	std.	mean	rank	mean	rank	mean	rank	mean	med.	mean	med.
TabFM + FIFO	**1.11**	**1**	**.869**	**.924**	.104	**.829**	**1.33**	**.772**	**1.22**	**.730**	**1.11**	167	129	4.80	1.50
TabICL + FIFO	2.56	2	.860	.906	.105	.820	2.89	.758	2.67	.712	2.56	58.4	71.4	3.73	1.81
CURE	2.89	3	.857	.907	.106	.819	2.89	.754	2.78	.704	2.89	87.3	97.9	5.94	1.96
DR-TabPFN + FIFO	5.33	5	.838	.868	.103	.794	5.67	.724	5.33	.659	5.33	599	542	5.38	1.78
TabPFN + FIFO	6.89	6	.805	.814	.122	.762	7.33	.677	7.44	.619	6.89	103	89.8	2.13	1.32
DualFIFO	6.89	6	.804	.814	.122	.770	5.67	.680	6.22	.619	6.89	127	116	5.63	2.21
EvoAutoML	7.67	7	.810	.878	.120	.761	8.00	.679	7.56	.627	7.67	36.7	22.9	7.58	2.67
ARF	8.00	9	.791	.798	.104	.742	8.22	.658	8.00	.592	8.00	6.30	6.50	6.93	6.87
SRP	8.11	8	.803	.802	**.100**	.749	9.22	.670	8.44	.605	8.11	8.47	8.50	10.7	7.15
Lev. Bagging	10.33	11	.752	.785	.122	.707	9.78	.601	10.22	.525	10.33	3.29	2.64	5.69	4.24
ASML	10.67	11	.763	.778	.126	.719	10.44	.616	10.67	.550	10.67	4.55	3.58	**0.97**	0.27
OAML	11.22	11	.763	.783	.127	.720	10.78	.613	11.11	.540	11.22	3.38	1.94	4.11	1.70
SAM-kNN	11.22	13	.755	.817	.134	.708	10.78	.582	11.22	.472	11.22	0.45	0.38	4.86	2.93
HAT	12.33	14	.694	.709	.166	.654	12.22	.530	12.33	.430	12.33	0.32	0.31	6.34	3.79
Majority-Class	15.22	15	.431	.523	.249	.323	15.67	-.011	15.67	-.139	15.22	0.19	0.16	4.48	**0.18**
No-Change	15.56	16	.413	.505	.306	.380	15.11	.112	15.11	.000	15.56	**0.17**	**0.12**	1.41	0.56

4 RESULTS

Predictive performance and cost. TabFM with FIFO ranks first in accuracy on all streams but one, ahead of both existing memory management systems and the streaming AutoML learners, with Drift-Resilient TabPFN narrowly ahead on the abrupt-drift stream (Table 2, with per-stream accuracies and a Wilcoxon–Holm comparison in Appendix C). The measured parameter counts separate scale from design. TabICL v2 clearly exceeds TabPFN v1 at almost the same parameter count, whereas TabFM adds only a small further gain for many times more parameters (Appendix B). Backbone quality therefore reflects architecture and pretraining at least as much as scale. Drift-Resilient TabPFN makes the pretraining effect visible on its own, since at the size and architecture of TabPFN v1 it gains several points on average, with large gains on meter, rialto and the recurring-drift stream and small changes elsewhere. Chance-corrected metrics mainly reorder the middle of the field and reveal where accuracy benefits from class imbalance or label autocorrelation. Electricity is the clearest case. Label autocorrelation lets No-Change reach a high accuracy there and some streaming learners fall below it, whereas TabFM+FIFO and TabICL+FIFO climb well above it and their temporal κ_t is roughly twice that of the best ensemble, which is why we also report κ_t.

Performance also varies across streams (Appendix C.3), with SRP showing the smallest spread among non-trivial methods, while SAM-kNN and EvoAutoML are more dependent on the stream. Figure 2 shows that the TFM advantage appears early and persists across several concept changes, although it is not uniform over time. On the abrupt-drift stream, the in-context methods start at higher predictive performance and recover to higher levels after the first drift. In a later, strongly autocorrelated segment, SRP and EvoAutoML overtake TabFM+FIFO. The size of the advantage depends on the drift type, since gradual and incremental streams separate almost nobody, while under recurring drift the swing within a single method exceeds the gap between paradigms. These differences show why aggregate performance should be read alongside temporal behaviour.

The predictive advantage comes at substantial computational cost (Figure 3). In-context methods occupy the most accurate and expensive end of the comparison, with the additional predictive gains requiring more than an order of magnitude higher latency than the cheapest ensemble, even though the TFMs run on GPUs. Because the asymmetry favours the TFMs, normalising for hardware would widen rather than close this gap, and the same ordering holds when times are converted to cloud cost (Appendix C.7). Both existing memory management systems are slightly less accurate and more expensive than FIFO on their own backbone, motivating the controlled memory studies below. Latency also varies severalfold across streams within a method and is highest on the widest stream, so mean cost alone does not capture the full deployment demands.

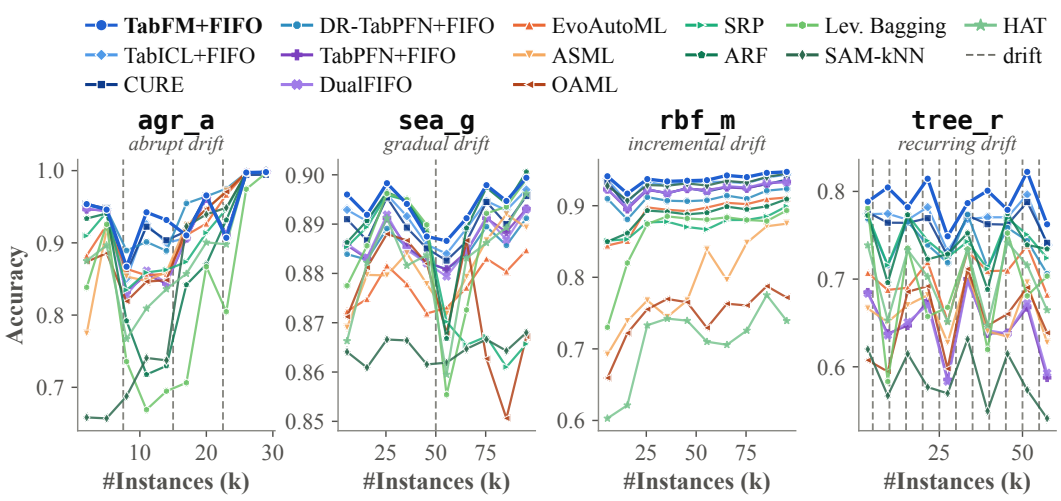

Figure 2: Accuracy over the synthetic streams, each with its own drift type, in stages of roughly a tenth of the stream. Dashes mark concept changes.

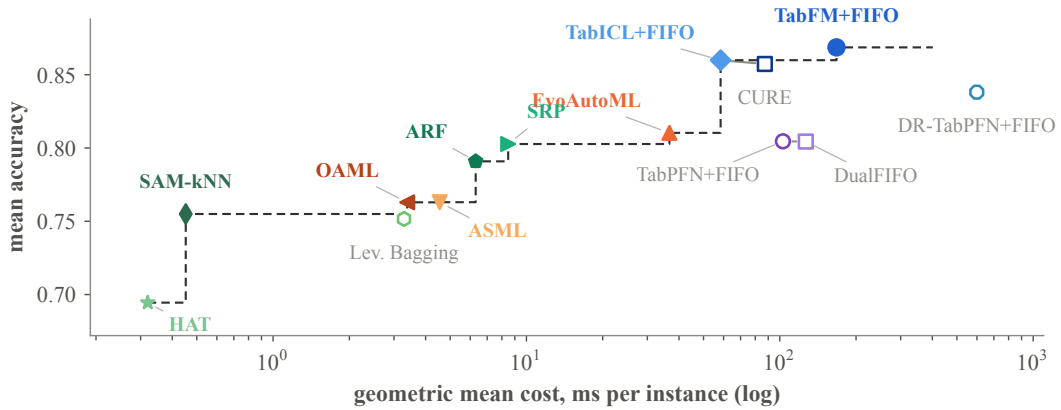

Figure 3: Mean accuracy against mean cost over all streams, coloured by method family. Filled marks (●) are on the Pareto front, hollow marks (○) are dominated, and grey ties (—) join each memory management system to FIFO on the same backbone.

Cold start and drift recovery. Figure 4 shows how quickly methods reach their initial plateau and fixed accuracy levels after the first abrupt drift. In-context methods begin near their plateau, while streaming ensembles and AutoML learners need longer, since a tree ensemble must see enough instances for its split tests, whereas an in-context model reads the concept directly from the rows in its memory. SAM-kNN, which also adapts through stored examples, is the exception among streaming learners. Figure 5 makes this visible on a synthetic stream with irrelevant features, where kNN and the TFMs share the same FIFO memory. The irrelevant features corrupt the distances of kNN, while the TFMs ignore them and follow the true boundary in the order of their overall accuracy (Appendix C.5). Until the next concept change, every in-context method recovers quickly, with TabFM+FIFO recovering furthest, while the AutoML learners recover more slowly and level off lower. SRP and HAT recover only late, and ARF, Leveraging Bagging and SAM-kNN never regain even the lowest level shown. Later drifts on this stream lead to easier concepts and are less informative.

The comparison above does not isolate the contributions of backbone, memory management and deployment conditions, which matter on fast streams where expensive methods may fall behind arrivals (Ghunaim et al., 2023; Alfarra et al., 2023). The following studies therefore vary memory composition, memory size, label delay, context refresh rate and query batching one at a time, holding the other settings fixed.

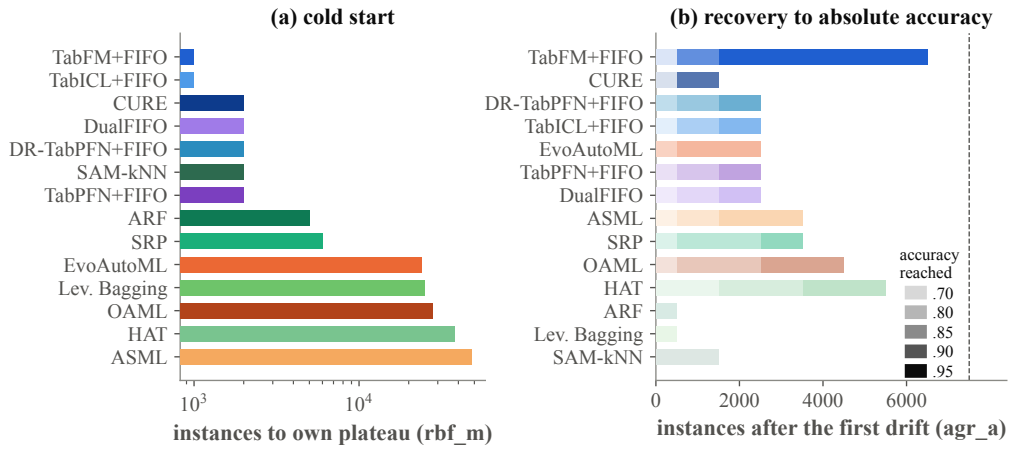

Figure 4: Instances to reach the initial plateau (left) and to regain accuracy levels after the first abrupt drift in `agr_a` (right). A missing bar marks a level not reached before the next drift (dashed line).

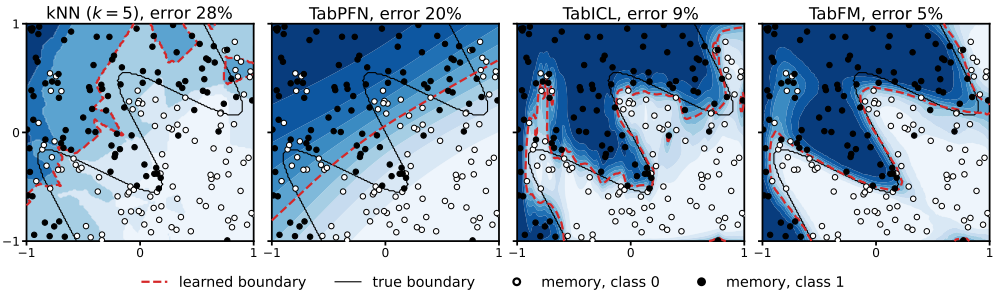

Figure 5: Decision functions of kNN and the TFMs on the same FIFO memory shortly after a gradual drift, with the misclassified share of the input area in each title.

Memory composition and size. Memory size has a substantially larger effect on predictive performance than the choice of memory management policy (Table 3). Replacing either existing memory management policy with FIFO on the same backbone produces little change in predictive performance at any budget while reducing serving cost (Table 2). Increasing the budget improves predictive performance more substantially, but also raises latency. TabICL gains most at smaller budgets before serving cost rises sharply, while TabPFN shows a more gradual increase in cost. Larger memories are not uniformly better, as the abrupt-drift stream reaches its best performance below the largest budget. These results favour tuning memory size before introducing more elaborate memory management.

Table 3: Accuracy and FIFO latency across memory budgets on a real and a synthetic abrupt-drift stream. Δ is FIFO minus the paired memory management policy. Best per stream in bold.

		TabICL v2				TabPFN v1			
Stream	B	FIFO	CURE	Δ	ms/inst.	FIFO	DualFIFO	Δ	ms/inst.
`noaa`	100	.788	.787	+.001	21.9	.784	.785	−.001	**28.3**
	250	.796	.798	−.002	**21.0**	.792	.785	+.006	30.8
	500	.814	.813	+.000	24.0	.808	.808	+.000	41.0
	1,000	**.818**	**.819**	−.001	71.8	**.814**	**.814**	+.001	99.1
`agr_a`	100	.921	.918	+.003	**17.3**	.901	.902	−.001	**24.9**
	250	.935	.935	+.000	18.4	.924	.922	+.001	29.7
	500	**.938**	**.938**	+.000	22.8	**.924**	**.925**	−.001	41.6
	1,000	.935	.937	−.002	71.4	.919	.920	+.000	113.0

Label delay. Delayed labels reduce predictive performance across the evaluated learners, while TabFM+FIFO retains the highest mean performance at every delay (Figure 6, with per-stream values in Table 10 of Appendix C.2). In relative terms the in-context methods lose about as much as the strongest ensembles and more than the weakest learners, whose lower starting accuracy leaves less to lose, so their advantage under delay comes from starting higher rather than from degrading less. On noaa the TFMs at the longest delay still match the best streaming ensemble with immediate labels, although this does not hold on the other streams. CURE becomes beneficial on the smart-meter stream, where it increasingly outperforms FIFO on the same backbone as delay grows and surpasses TabFM+FIFO at the longest delay. This suggests that selecting among older labelled examples can help when recent feedback is unavailable. The benefit is confined to this stream, which also accounts for much of the overall decline, so we do not claim that delayed feedback generally favours elaborate memory management.

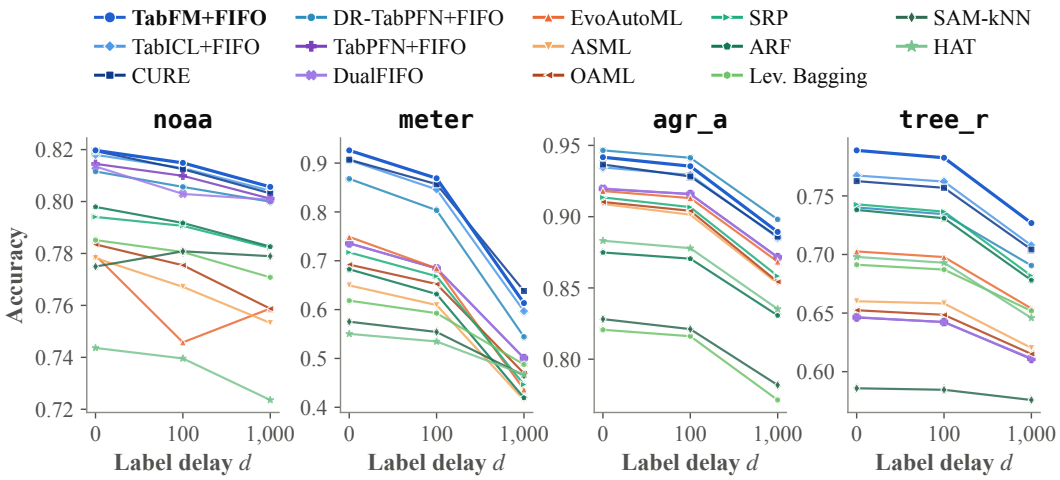

Figure 6: Accuracy of all methods as the label delay d grows, where each label arrives d instances after its prediction.

Context refresh rate and query batching. Deployment levers can spread the cost of encoding the memory over more than one prediction, one across time and one across queries. Refreshing the context every K instances while still predicting on every instance saves only a small share of latency at the first interval with little loss in predictive performance, and further reductions in update frequency buy almost no additional saving while degrading predictive performance, most sharply on the smart-meter stream (Table 4). Serving q queries against a shared context instead amortises the encoding across simultaneous predictions, and per-instance serving time falls substantially on every backbone while predictive performance remains similar on the recurring-drift stream and declines more noticeably on the smart-meter stream at larger batches (Table 5). Neither lever removes the underlying cost, since a refresh only stores the new memory while every prediction call re-encodes it.

Table 4: TabICL with FIFO when the context is refreshed every K instances but queried on every instance. Column headers give K, and the best per row and metric is in bold.

	Context updates				Accuracy				Time (ms/inst.)			
Stream	1	10	100	1,000	1	10	100	1,000	1	10	100	1,000
noaa	18,059	1,806	181	19	**.818**	.816	.814	.808	31.5	27.8	27.3	**27.1**
meter	22,848	2,285	229	23	**.906**	.903	.875	.710	70.3	62.0	61.3	**60.7**
agr_a	29,833	2,984	299	30	**.935**	.934	.932	.917	32.4	28.5	28.0	**27.9**
tree_r	59,900	5,990	599	60	.767	**.768**	.764	.737	35.0	31.0	30.7	**30.6**

Table 5: Query batching with q queries sharing one context, for each backbone with FIFO. Best per row and metric in bold.

		Time (ms/inst.)			Accuracy		
Backbone	Stream	q=1	q=10	q=100	q=1	q=10	q=100
TabICL v2	meter	88.17	13.08	**1.89**	**.906**	.903	.875
	tree_r	110.66	7.69	**0.92**	.767	**.768**	.764
TabPFN v1	meter	371.75	50.47	**5.58**	**.736**	.735	.709
	tree_r	246.53	11.75	**1.52**	**.646**	**.646**	.644
DR-TabPFN	meter	1367.64	570.62	**55.54**	**.868**	.866	.835
	tree_r	552.59	125.13	**12.46**	**.740**	.739	.735
TabFM	meter	658.69	142.97	**16.19**	**.926**	.924	.896
	tree_r	295.97	47.03	**5.34**	**.789**	**.789**	.784

Fixed and per-query cost. To characterise the batching gains, we approximate per-call serving time by $t(q) = A + q\gamma$, where A is the fixed overhead and γ the incremental cost per query, fitted to a separate sweep on noaa. The fixed term dominates single-instance serving time on every backbone (Table 6). It combines context encoding with preprocessing, data transfer and execution overhead, but separate measurements bound the non-encoding part. Dispatch does not depend on the memory budget, and preprocessing and transfer grow only linearly in it, yet the TabICL and TabPFN calls grow severalfold between the smallest and the default budget (Table 3), so most of the default-budget call is context encoding. Storing a new memory, which includes preprocessing, also takes only a small fraction of a call. The fixed term is therefore largely amortisable, although the fit does not measure the exact encoding share or the achievable cache speed-up. Together with the refresh-rate plateau, this motivates reusing context representations across predictions (Figure 7).

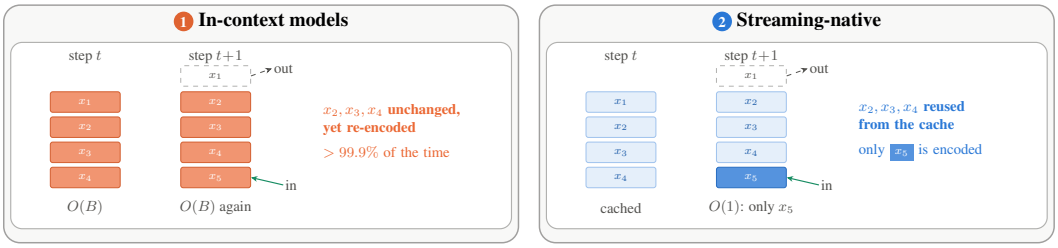

Figure 7: Repeated context encoding versus cached context reuse.

Table 6: Fixed cost A and per-query cost γ fitted to $t(q) = A + q\gamma$ on a separate noaa sweep. The lower block is derived from the fit.

	TabICL v2	TabPFN v1	TabFM
Fixed cost A (ms)	26.1	59.4	120
Per-query cost γ (μs)	13.2	22.3	117
Queries per unit of fixed cost A/γ	1,980	2,670	1,030
Fixed share of a single call	$> 99.9\%$	$> 99.9\%$	$> 99.9\%$

Cached serving. To test whether this coupling is inherent, we pretrain compact models on a synthetic prior with drift, never on an evaluation stream, with a causal datapoint axis so that each row is encoded once on arrival and then read from a cache (Appendix D). Serving one checkpoint both ways on several streams, the cached path stays flat in time and GPU memory as the memory grows, while re-encoding grows superlinearly in time and close to quadratically in memory, taking up to an order of magnitude longer at the largest budget (Figure 8). Where the checkpoint transfers, accuracy agrees at every budget except the smallest, where the gap stays small (Table 22), and on electricity both paths collapse to a constant prediction. Re-encoding remains cheaper at small memories, and the saving is far below the ceiling in Table 6, as the models are small and the cache unoptimised. This is a proof of concept that the bottleneck is removable, not a competitive model.

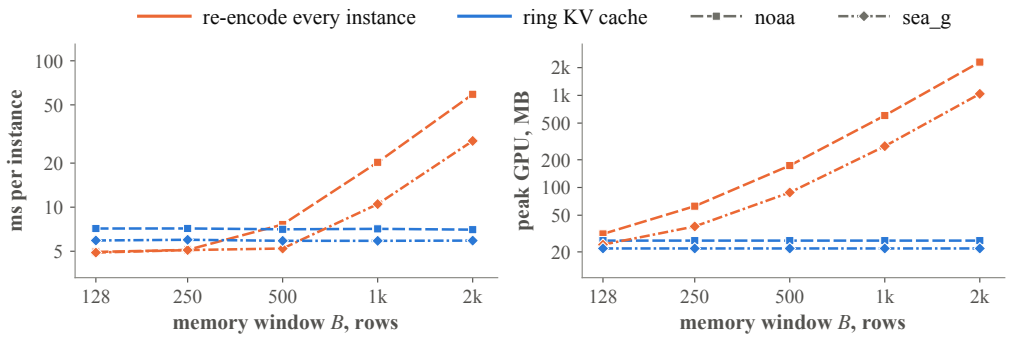

Figure 8: Per-instance serving time (left) and peak GPU memory (right) of one compact checkpoint served from a ring cache or by re-encoding every instance, as the memory window grows. The cached path stays flat, whereas re-encoding grows in both time and memory.

5 Conclusions and Future Work

We presented a systematic study of TFMs for data streams. TFMs achieve the strongest predictive performance in our evaluation, recover faster than streaming learners after drift, and remain the most accurate under label delay, but serve more than an order of magnitude slower than the cheapest streaming ensemble. Backbone choice and memory size contribute more than the memory management policy, which paid off only under delayed labels and only on one stream. In practice, FIFO with a strong backbone and a tuned memory size is a strong default, and query batching lowers cost where latency allows. The main computational limitation is repeated context encoding at every prediction. Our compact causal models show that reusing encoded context substantially reduces serving time and memory while staying close in accuracy wherever the checkpoint transfers. However, this result is limited to the compact models we pretrain, and whether existing TFMs support similar reuse remains open. Overall, efficient streaming TFMs will require architectural changes rather than more elaborate memory management.

Limitations. Our study covers stream classification only, so the findings may not carry over to regression or other tabular tasks. Our experiments are constrained by the context limits and serving costs of the evaluated TFMs, so substantially larger memories remain unexplored. In addition, context-based adaptation depends on the labelled examples available in memory, which limits adaptation under delayed feedback and changing concepts. The caching experiments are restricted to compact models and fixed attention windows, and therefore do not establish that the same efficiency gains will transfer directly to larger pretrained TFMs. Finally, unlabelled observations arriving during label delay are not used.

Future work. A key direction is to develop TFMs that encode arriving examples once and reuse their representations as the stream evolves. Approaches based on local attention (Beltagy et al., 2020), recurrent state reuse (Dai et al., 2019), and key-value caching (Xiao et al., 2024) provide useful foundations. Future models could also incorporate pretraining on drifting sequences to better balance recent information, recurring concepts, and delayed feedback. Scaling such architectures while retaining the predictive strength of current TFMs remains an important challenge.

AI Use Statement

We used generative AI assistants, including Claude and ChatGPT, to draft and refine manuscript text, improve readability, and develop and revise code for analysis, figures, and tables. The authors reviewed all AI-assisted output and verified every reported number using the released result files and scripts. Experimental design, scientific claims, and interpretation of results remain the authors' own. We take full responsibility for the final content of this work.

REFERENCES

Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Database mining: A performance perspective. In *IEEE Transactions on Knowledge and Data Engineering*, 1993.

Motasem Alfarra, Hani Itani, Alejandro Pardo, Shyma Alhuwaider, Merey Ramazanova, Juan C Pérez, Zhipeng Cai, Matthias Müller, and Bernard Ghanem. Evaluation of test-time adaptation under computational time constraints. *arXiv preprint arXiv:2304.04795*, 2023.

Shruti Arora, Rinkle Rani, and Nitin Saxena. A systematic review on detection and adaptation of concept drift in streaming data using machine learning techniques. *WIREs Data Mining and Knowledge Discovery*, 14(4):e1536, 2024.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1):152–161, 2016.

Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining (SDM)*, 2007.

Albert Bifet and Ricard Gavaldà. Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis (IDA)*, 2009.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive online analysis. *Journal of Machine Learning Research*, 11, 2010a.

Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2010b.

Albert Bifet, Gianmarco de Francisci Morales, Jesse Read, Geoff Holmes, and Bernhard Pfahringer. Efficient online evaluation of big data stream classifiers. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015.

Albert Bifet, Ricard Gavaldà, Geoffrey Holmes, and Bernhard Pfahringer. *Machine Learning for Data Streams with Practical Examples in MOA*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2018. ISBN 9780262037792.

Bilge Celik, Prabhant Singh, and Joaquin Vanschoren. Online AutoML: an adaptive AutoML framework for online learning. *Machine Learning*, 2023.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Andrea Dal Pozzolo, Olivier Caelen, Yann-Aël Le Borgne, Serge Waterschoot, and Gianluca Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications*, 41(10):4915–4928, 2014.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2000.

Ryan Elwell and Robi Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Mutalik Desai, David Salinas, and Frank Hutter. TabArena: A living benchmark for machine learning on tabular data. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025.

Benjamin Feuer, Robin T Schirrmeister, Valeriia Cherepanova, Chinmay Hegde, Frank Hutter, Micah Goldblum, Niv Cohen, and Colin White. Tunetables: Context optimization for scalable prior-data fitted networks. *Advances in Neural Information Processing Systems*, 37:83430–83464, 2024.

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3), 2013.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 2014.

Yasir Ghunaim, Adel Bibi, Kumail Alhamoud, Motasem Alfarra, Hasan Abed Al Kader Hammoud, Ameya Prabhu, Philip HS Torr, and Bernard Ghanem. Real-time evaluation in online continual learning: A new hope. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11888–11897, 2023.

Heitor Murilo Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfahringer, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106, 2017.

Heitor Murilo Gomes, Jesse Read, and Albert Bifet. Streaming random patches for evolving data stream classification. In *IEEE International Conference on Data Mining (ICDM)*, 2019.

Heitor Murilo Gomes, Anton Lee, Nuwan Gunasekara, Yibin Sun, Guilherme Weigert Cassales, Justin Liu, Albert Bifet, and Bernhard Pfahringer. CapyMOA: Efficient machine learning for data streams in python. *arXiv preprint arXiv:2502.07432*, 2024.

Google Research. TabFM: A zero-shot foundation model for tabular data. `https://github.com/google-research/tabfm`, 2026. Software release, version 1.0.0, June 2026. Weights under the `tabfm-non-commercial-v1.0` licence. No technical report accompanies the model; benchmark details appear in an accompanying blog post.

Maciej Grzenda, Heitor Murilo Gomes, and Albert Bifet. Delayed labelling evaluation for data streams. *Data Mining and Knowledge Discovery*, 34(5):1237–1266, 2020. doi: 10.1007/s10618-019-00654-y.

Michael Harries. SPLICE-2 comparative evaluation: Electricity pricing. Technical report, University of New South Wales, 1999.

Kai Helli, David Schnurr, Noah Hollmann, Samuel Müller, and Frank Hutter. Drift-resilient TabPFN: In-context learning temporal distribution shifts on tabular data. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637, 2025.

Bostjan Kaluza, Violeta Mirchevska, Erik Dovgan, Mitja Lustrek, and Matjaz Gams. An agent-based approach to care in independent living. In *International Joint Conference on Ambient Intelligence (AmI)*, pp. 177–186, 2010.

Cedric Kulbach, Jacob Montiel, Maroua Bahri, Marco Heyden, and Albert Bifet. Evolution-based online automated machine learning. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2022.

Jinmo Lee, Doyun Choi, Moongi Choi, and Jaemin Yoo. Bounded context management for tabular foundation models on stream learning. *arXiv preprint arXiv:2606.18677*, 2026.

Viktor Losing, Barbara Hammer, and Heiko Wersing. KNN classifier with self adjusting memory for heterogeneous concept drift. In *IEEE International Conference on Data Mining (ICDM)*, 2016.

Afonso Lourenço, João Gama, Eric P. Xing, and Goreti Marreiros. In-context learning of evolving data streams with tabular foundational models. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '26, pp. 973–982, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722585. doi: 10.1145/3770854.3780305. URL https://doi.org/10.1145/3770854.3780305.

Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do Bayesian inference. In *International Conference on Learning Representations (ICLR)*, 2022.

Alexander Pfefferle, Johannes Hog, Lennart Purucker, and Frank Hutter. nanoTabPFN: A lightweight and educational reimplementation of TabPFN. *arXiv preprint arXiv:2511.03634*, 2025.

Ameya Prabhu, Hasan Abed Al Kader Hammoud, Puneet K Dokania, Philip HS Torr, Ser-Nam Lim, Bernard Ghanem, and Adel Bibi. Computationally budgeted continual learning: What does matter? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3698–3707, 2023.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICLv2: A better, faster, scalable, and open tabular foundation model. *arXiv preprint arXiv:2602.11139*, 2026.

Methaq A. Shyaa, Noor Farizah Ibrahim, Zurinahni Zainol, Rosni Abdullah, Mohammed Anbar, and Laith Alzubaidi. Evolving cybersecurity frontiers: A comprehensive survey on concept drift and feature dynamics aware machine and deep learning in intrusion detection systems. *Engineering Applications of Artificial Intelligence*, 137:109143, 2024.

Soda Team, Inria. nanoTabICL: A minimal reimplementation of the TabICLv2 architecture and prior. https://github.com/soda-inria/nanotabicl, 2026. Software release, BSD-3-Clause.

Vinicius Souza, Tiago Pinho, and Gustavo Batista. Evaluating stream classifiers with delayed labels information. In *2018 7th Brazilian conference on intelligent systems (BRACIS)*, pp. 408–413. IEEE, 2018.

Vinicius M. A. Souza, Trilce Estrada, Adnan Bashir, and Abdullah Mueen. LADPU smart meter data. Dryad, 2020a.

Vinicius M A Souza, Denis M Reis, André G Maletzke, and Gustavo E A P A Batista. Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery*, 34:1805–1858, 2020b. doi: 10.1007/s10618-020-00698-5.

W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2001.

Valentin Thomas, Junwei Ma, Rasa Hosseinzadeh, Keyvan Golestan, Guangwei Yu, Maksims Volkovs, and Anthony Caterini. Retrieval & fine-tuning for in-context tabular models. *Advances in Neural Information Processing Systems*, 37:108439–108467, 2024.

Nilesh Verma, Albert Bifet, Bernhard Pfahringer, and Maroua Bahri. ASML: A scalable and efficient AutoML solution for data streams. In *International Conference on Automated Machine Learning (AutoML)*, volume 256 of *Proceedings of Machine Learning Research*, 2024.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024.

A STREAMS

This appendix details the sources and tasks of the streams in Table 1 and the settings of the synthetic generators. The repository streams are available through the USPDS repository.[2] The smart-meter recordings are obtained from the Dryad data repository,[3] while the person-activity recordings are available from the UCI Machine Learning Repository.[4]

A.1 REAL STREAMS

- **`electricity`:** Half-hourly records from the New South Wales electricity market between 1996 and 1998, with eight features covering price, demand and interstate transfer. The label is whether the price moved up or down relative to a moving average of the preceding day. Consecutive labels are strongly autocorrelated, so repeating the last label already scores well.
- **`noaa`:** Fifty years of daily weather measurements from one NOAA station, with eight features such as temperature, pressure and wind speed, and a binary label for whether rain was observed. Seasonal and climatic variation change the inputs over time, although no drift points are annotated.
- **`meter`:** Electricity consumption recorded by smart meters in Los Alamos, New Mexico, sampled every fifteen minutes. Each row is one household-day, so the ninety-six features are the readings of that day in order, and the ten classes identify the household.
- **`rialto`:** Ten buildings beside the Rialto bridge in Venice, photographed by a fixed webcam over twenty consecutive days and encoded as a normalised twenty-seven-dimensional colour histogram. The label is which building the image shows. Changes in illumination, weather, and time of day alter the input distribution while the building identities remain fixed.
- **`posture_no8`:** Position readings from four tags worn by five people performing a scripted set of activities, giving three features, the tag's coordinates, and an activity label. The original release has eleven activities. Following CURE (Lee et al., 2026), we drop the rarest and keep ten, because TabPFN v1 accepts at most ten classes.

A.2 SYNTHETIC STREAMS

All four are built through CapyMOA (Gomes et al., 2024), which wraps the MOA implementations (Bifet et al., 2010a), and none of them appears in the pretraining prior of any model we train.

- **`agr_a`, abrupt:** The AGRAWAL generator describes loan applicants through nine attributes such as salary, age and loan amount, and labels them by one of ten fixed classification functions. We switch that function three times, from 1 to 4 to 7 to 10, at instances $7{,}500$, $15{,}000$ and $22{,}500$ of a $30{,}000$-instance stream, so the concept changes instantly at three known points while the inputs keep their distribution.
- **`sea_g`, gradual:** The SEA generator labels three numeric attributes by thresholding the sum of the first two. We move from function 1 to function 3 over a transition window of $10{,}000$ instances centred at the midpoint of a $100{,}000$-instance stream, so during the transition each instance is drawn from the old or the new concept with a probability that shifts smoothly, and there is no single point at which the concept changes.
- **`rbf_m`, incremental:** RandomRBF places fifty class centroids in a ten-dimensional space and draws instances around them, with five classes. Every centroid drifts continuously at a speed of 0.0001 per instance, producing continuous movement of the decision boundary over $100{,}000$ instances without discrete drift points.
- **`tree_r`, recurring:** RandomTree labels ten attributes, five numeric and five nominal, by a randomly generated decision tree. We build three such trees and cycle through them with abrupt transitions every $5{,}000$ instances, four visits to each concept, giving $60{,}000$ instances in total.

[2] https://sites.google.com/view/uspdsrepository
[3] https://doi.org/10.5061/dryad.m0cfxpp2c
[4] https://doi.org/10.24432/C57G8X

B REPRODUCIBILITY

All experiments use a single-pass prequential (test-then-train) evaluation in which predictions are made before labels are revealed. Default settings and study-specific variations are summarised in Table 7. Streaming learners and streaming AutoML methods run on the CPU host, while TFM backbones, compact models, and backbone efficiency measurements run on the GPU hosts listed in Table 8. Cross-method wall-clock times therefore reflect deployment cost on their respective platforms, whereas within-backbone efficiency comparisons use fixed hardware. The refresh, batching and budget sweeps are timed in separate runs whose absolute times sit below Table 13, so they should be compared only within each sweep. TFM backbones are isolated in separate virtual environments when required by incompatible dependencies, with TabPFN pinned to the evaluated v1 implementation. Running each paradigm on the hardware it is designed for follows the practice of published tabular benchmarks (Erickson et al., 2025; Hollmann et al., 2023; Lourenço et al., 2026). The asymmetry favours the TFMs, so any normalisation would increase their measured cost relative to the CPU methods. Drift-Resilient TabPFN uses its released checkpoints, which share the TabPFN v1 architecture, with four ensemble configurations. Its time covariate is the arrival index binned into equal-width domains, with each query placed one domain ahead of its memory. Its nine streams ran concurrently on the two A6000 hosts, five and four at a time, under the same harness as the other methods. TabFM is distributed without a technical report or a published parameter count, so we count parameters directly in the released checkpoints, summing tensor sizes from the safetensors header for TabFM and over the state dictionary for the other two. This gives 25.8M parameters for TabPFN v1, 27.6M for TabICL v2, and 1.64B for the TabFM classification checkpoint, all stored in single precision. The TabFM model card additionally documents an embedding width of 256, three column-attention blocks, three row-attention blocks, twenty-four in-context blocks, and a maximum of ten classes. Compact-model comparisons vary the datapoint-axis attention mask within each architecture family while keeping the remaining training configuration fixed. nanoTabPFN uses four layers, embedding width 160, eight heads, and feed-forward width 320, while nanoTabICL uses embedding width 96, four heads, two column blocks, two row blocks, and six in-context blocks over 64 class rows. Both are trained for $20{,}000$ steps with schedule-free AdamW on the same synthetic prior, which excludes the evaluation generators. Released result files contain the metrics, per-window histories, computational costs, memory measurements, host information, and configurations used to reproduce the reported results[5].

Table 7: Default evaluation settings and study-specific variations.

Setting	Value	Streams	Where
Default evaluation settings			
Warm-up	100 instances	9	all
Evaluation window	1,000, tumbling	9	all
Label delay	0 (immediate)	9	all
Retained memory	1,000 rows	9	all
Queries per prediction	1	9	all
Varied, one quantity per study			
Retained memory	100, 250, 500, 1,000	2	memory size
Label delay	0, 100, 1,000	4	label delay
Queries per batch	1, 10, 100	2	query batching
Context refresh	every 1, 10, 100, 1,000	4	context refresh
Fine-tuning data	7,500 or 10,000 instances, stream prefix	2	§D.1
Attention mask	bidirectional, causal, sliding	prior	§D.4
Attention window	32, 64, 128, 256, 512	prior	§D.4
Serving path	re-encode, ring cache	4	§D.6
Compact memory	128, 250, 500, 1,000, 2,000	4	§D.6
Recency decay, drift gate	on, off	9	§D.2

[5] https://anonymous.4open.science/r/StreamTFM-Study

Table 8: Hardware and software used for evaluation.

Component	CPU host	GPU hosts
Processor	2× AMD EPYC 7702, 128 cores	AMD EPYC 7543, 32 cores
Memory	1 TiB DDR4	251 GiB DDR4
Accelerator	none	NVIDIA RTX A6000, 48 GB
OS	Ubuntu 24.04.4 LTS	Rocky Linux 9.5
Kernel	Linux 6.8.0	Linux 5.14.0
Python	3.14	3.12 (3.11 for the v1 environment)
Runtime	CapyMOA 0.13, OpenJDK 21	CapyMOA 0.13, OpenJDK 17, PyTorch 2.13 (cu126)

C Additional results

C.1 Accuracy

Table 9 gives the per-stream accuracy behind Table 2. TabFM+FIFO is best on every stream except `agr_a`, where Drift-Resilient TabPFN leads, and the in-context methods separate most from the classical learners on `meter` and `rialto`.

Table 9: Per-stream accuracy with mean and mean rank across methods, lower rank is better and best values are bold.

Method	elec	noaa	meter	rialto	posture	agr_a	sea_g	rbf_m	tree_r	Mean	Rank
No-Change	.853	.680	.011	.000	.203	.647	.561	.260	.505	.413	15.56
Majority-Class	.575	.686	.011	.100	.333	.632	.673	.343	.523	.431	15.22
HAT	.839	.744	.550	.412	.532	.883	.882	.709	.698	.694	12.33
ARF	.900	.798	.683	.728	.619	.875	.890	.888	.738	.791	8.00
SRP	.894	.794	.717	.802	.608	.914	.878	.874	.743	.803	8.11
Lev. Bagging	.890	.785	.618	.612	.600	.821	.886	.861	.691	.752	10.33
SAM-kNN	.825	.775	.575	.817	.594	.828	.865	.930	.586	.755	11.22
ASML	.900	.778	.650	.754	.541	.909	.881	.794	.660	.763	10.67
EvoAutoML	.928	.779	.749	.880	.568	.918	.878	.890	.703	.810	7.67
OAML	.888	.783	.692	.799	.519	.910	.874	.748	.652	.763	11.22
CURE	.936	.819	.907	.920	.621	.937	.890	.923	.763	.857	2.89
DualFIFO	.917	.814	.736	.802	.596	.920	.886	.922	.646	.804	6.89
TabICL + FIFO	.944	.818	.906	.917	.629	.935	.891	.932	.767	.860	2.56
TabPFN + FIFO	.917	.814	.736	.802	.598	.919	.887	.922	.646	.805	6.89
DR-TabPFN + FIFO	.903	.812	.868	.865	.615	**.947**	.885	.909	.740	.838	5.33
TabFM + FIFO	**.950**	**.820**	**.926**	**.924**	**.635**	.942	**.894**	**.938**	**.789**	**.869**	**1.11**

C.2 Label delay

Table 10 gives the per-stream accuracy behind Figure 6. TabFM+FIFO has the best mean at every delay. Most of the decline comes from `meter`, the only stream where CURE overtakes TabFM+FIFO at the longest delay.

C.3 Variation across streams

Figure 9 complements the aggregate results with the distribution of temporal κ_t across streams. Similar medians can conceal different spreads, while differences between means and medians indicate sensitivity to individual datasets. The variation across methods, including TabFM, shows that a strong overall ranking does not imply uniform performance across stream conditions.

Table 10: Accuracy under label delay d on four streams, all sixteen methods. Rank is the mean per-stream rank. Best per column in bold.

	noaa			meter			agr_a			tree_r			Mean			Rank		
Method	0	100	1000	0	100	1000	0	100	1000	0	100	1000	0	100	1000	0	100	1000
No-Change	.680	.562	.557	.011	.102	.101	.647	.642	.633	.505	.502	.498	.461	.452	.447	15.8	15.5	15.8
Majority-Class	.686	.686	.686	.011	.100	.101	.632	.632	.632	.523	.523	.523	.463	.485	.486	15.2	15.5	15.2
HAT	.744	.740	.724	.550	.535	.467	.883	.878	.835	.698	.693	.646	.719	.711	.668	11.8	11.8	10.8
ARF	.798	.792	.783	.683	.632	.419	.875	.871	.831	.738	.731	.678	.773	.756	.678	8.8	8.8	9.5
SRP	.794	.791	.782	.717	.668	.446	.914	.907	.858	.743	.737	.682	.792	.776	.692	7.0	7.0	8.0
Lev. Bagging	.785	.781	.771	.618	.593	.488	.821	.816	.771	.691	.687	.652	.729	.719	.670	11.0	11.2	9.8
SAM-kNN	.775	.781	.779	.575	.554	.465	.828	.821	.782	.586	.585	.576	.691	.685	.650	13.2	12.2	11.5
ASML	.778	.767	.753	.650	.609	.415	.909	.901	.853	.660	.658	.620	.749	.734	.660	10.8	10.8	11.8
EvoAutoML	.779	.746	.759	.749	.685	.436	.918	.913	.868	.703	.698	.654	.787	.760	.679	7.5	8.0	9.2
OAML	.783	.775	.759	.692	.652	.469	.910	.904	.854	.652	.648	.615	.760	.745	.674	9.8	10.0	10.0
CURE	.819	.812	.803	.907	.857	**.638**	.937	.928	.886	.763	.757	.704	.856	.839	.758	2.5	3.0	2.5
DualFIFO	.814	.803	.801	.736	.684	.500	.920	.916	.871	.646	.642	.611	.779	.761	.696	7.0	7.8	7.2
TabICL + FIFO	.818	.813	.804	.906	.846	.597	.935	.929	.885	.767	.762	.708	.857	.838	.748	3.0	2.5	2.8
TabPFN + FIFO	.814	.810	.801	.736	.684	.501	.919	.916	.872	.646	.642	.611	.779	.763	.696	7.5	7.0	6.8
DR-TabPFN + FIFO	.812	.806	.800	.868	.803	.544	**.947**	**.941**	**.898**	.740	.734	.691	.842	.821	.733	4.0	3.8	3.8
TabFM + FIFO	**.820**	**.815**	**.806**	**.926**	**.869**	.613	.942	.935	.889	**.789**	**.783**	**.727**	**.869**	**.850**	**.759**	**1.2**	**1.2**	**1.5**

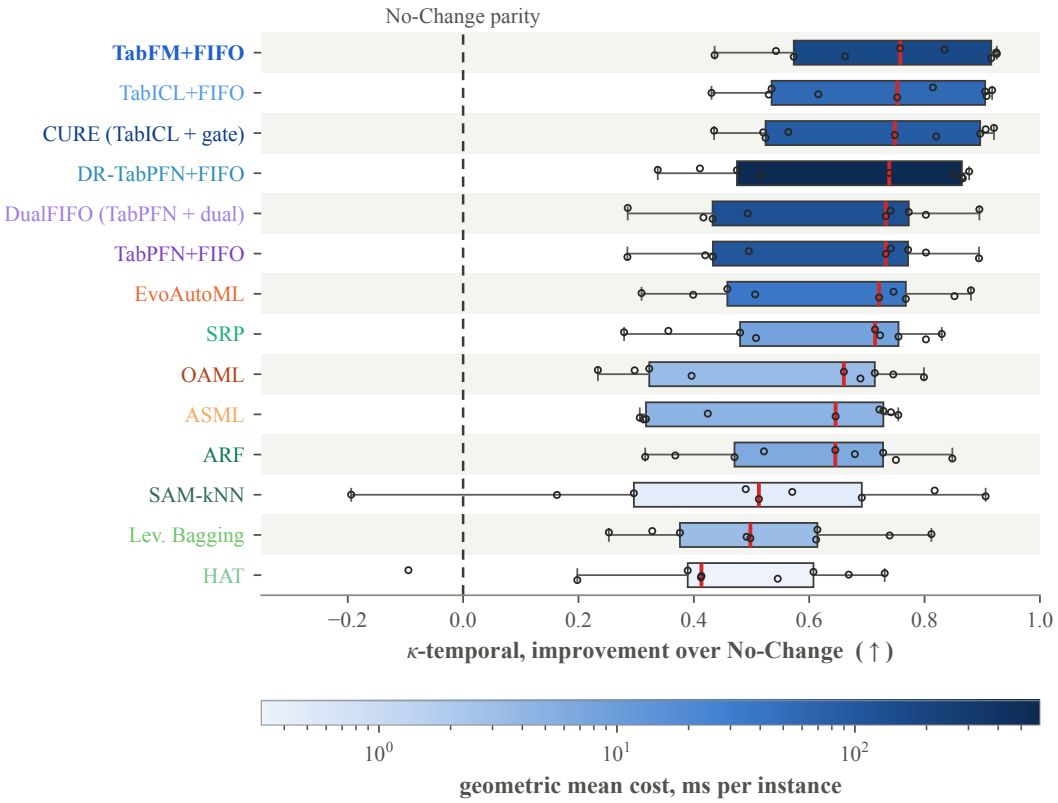

Figure 9: Temporal κ_t across evaluation streams, with methods ordered by median. Fill colour indicates per-instance serving cost on a logarithmic scale.

C.4 Statistical comparison

Figure 10 summarises accuracy ranks of the non-trivial methods, excluding No-Change and Majority-Class, so its ranks differ from those in Table 2. We use two-sided exact Wilcoxon signed-rank tests on paired per-stream accuracies, with Holm correction for comparisons against the top-ranked method (Benavoli et al., 2016). TabFM+FIFO outperforms every comparator on every stream except Drift-Resilient TabPFN on the abrupt-drift stream. With few streams, however, even this out-

come gives a two-sided p-value just above the first Holm threshold, so the diagram does not separate the leading in-context methods. We therefore emphasise the descriptive result that TabFM+FIFO ranks first on all streams but one.

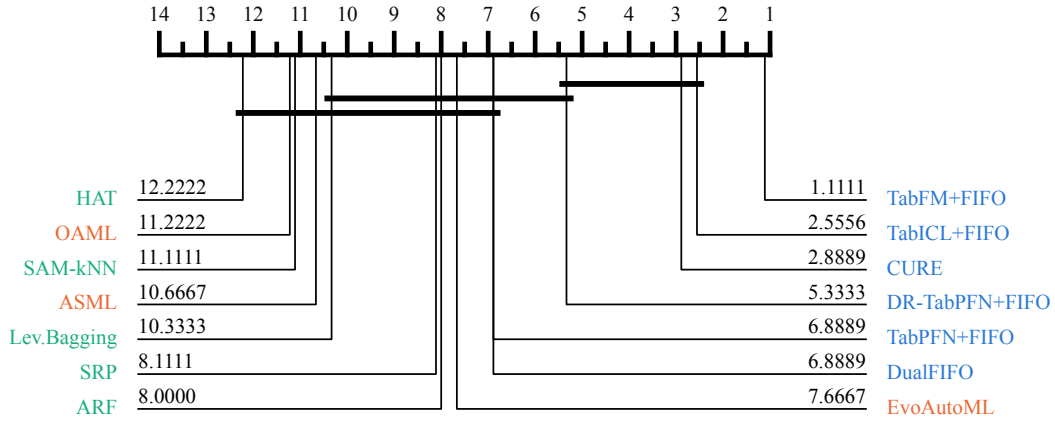

Figure 10: Mean accuracy ranks with Holm-corrected two-sided Wilcoxon comparisons against the top-ranked method.

C.5 Decision boundaries on a synthetic stream

To make the difference between in-context and instance-based learning visible, we generate a two-dimensional stream with irrelevant uniform features and label noise, whose sine-shaped concept switches abruptly to a rotation of itself, recurs, and then drifts gradually to another rotation. A kNN classifier, TabPFN v1, TabICL v2 and TabFM are fitted on the identical FIFO memory and evaluated on a grid over the two informative features (Figure 11). Because all four see the same examples, every difference comes from how they read them. The irrelevant features corrupt the distances of kNN, while the TFMs ignore them and follow the true boundary in the order of their overall accuracy. Right after each drift all four fail alike, since the memory still holds the previous concept, so under FIFO the speed of adaptation is set by memory turnover and the quality of the recovered boundary by the backbone. Memory size exposes the resulting trade-off. A larger memory sharpens the steady-state boundary of TabICL and TabFM but still holds the old concept long after a drift (Figure 12), and recovery time grows nearly in proportion to memory size (Table 11). Choosing the size per window with hindsight is worth several points, and for TabICL and TabFM a selector over all sizes captures part of this headroom, whereas switching between only the smallest and largest memory does not (Table 12). These results come from one stream and one seed, so they indicate headroom for memory management rather than a method.

Table 11: Instances after the first drift until windowed accuracy regains 90 percent of its pre-drift level, by FIFO memory size B, with the slope on logarithmic axes. The smallest values sit at the measurement floor.

	B=50	100	200	500	1,000	slope
kNN (k=5)	50	50	55	130	271	0.99
TabPFN v1	50	82	109	244	271	0.58
TabICL v2	60	97	203	334	601	0.67
TabFM	60	67	165	335	601	0.80

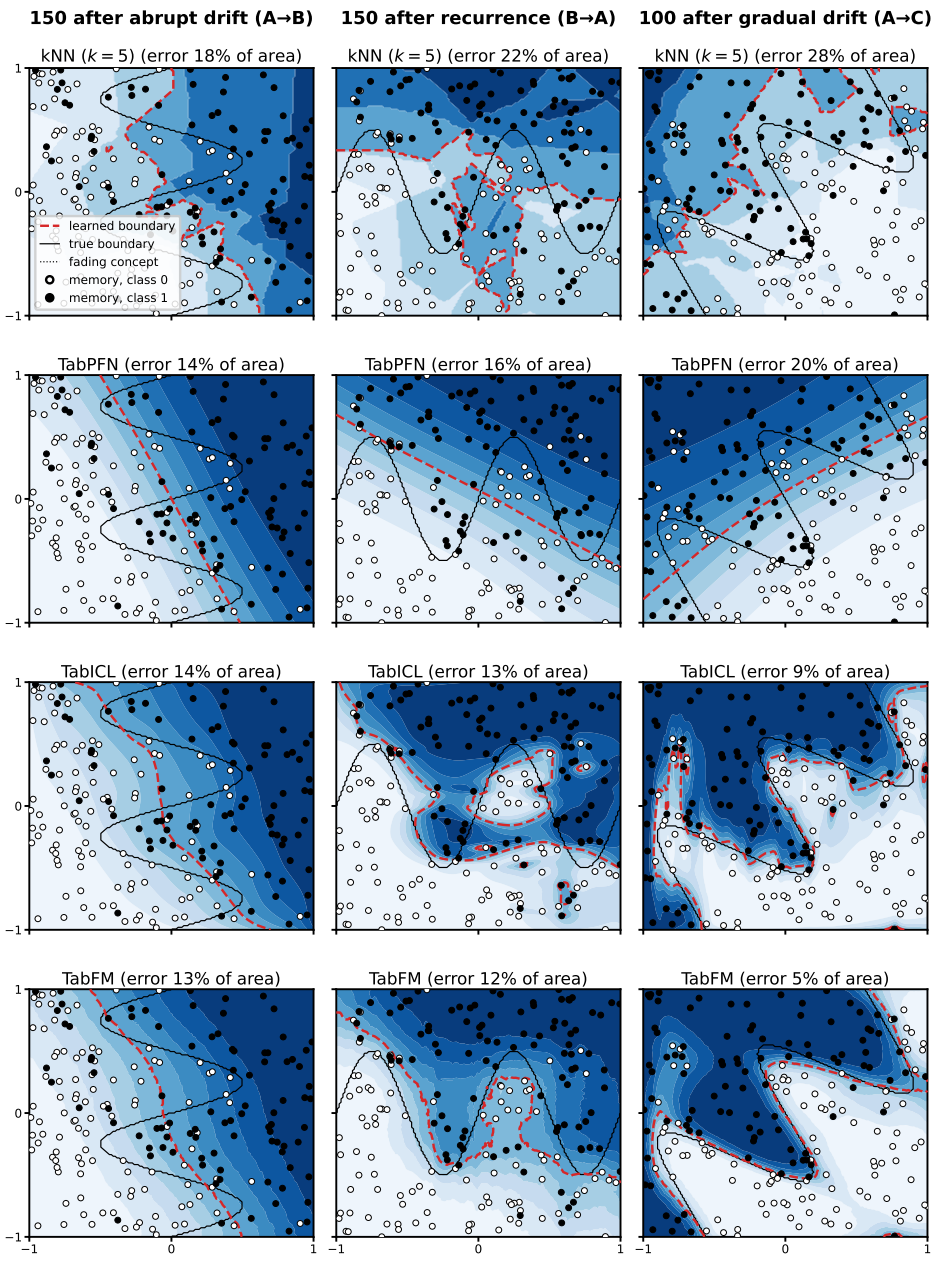

Figure 11: Decision functions of kNN and the TFMs on the same FIFO memory after an abrupt drift, a recurrence and a gradual drift, with learned (red dashed) and true (black) boundaries and memory contents as dots.

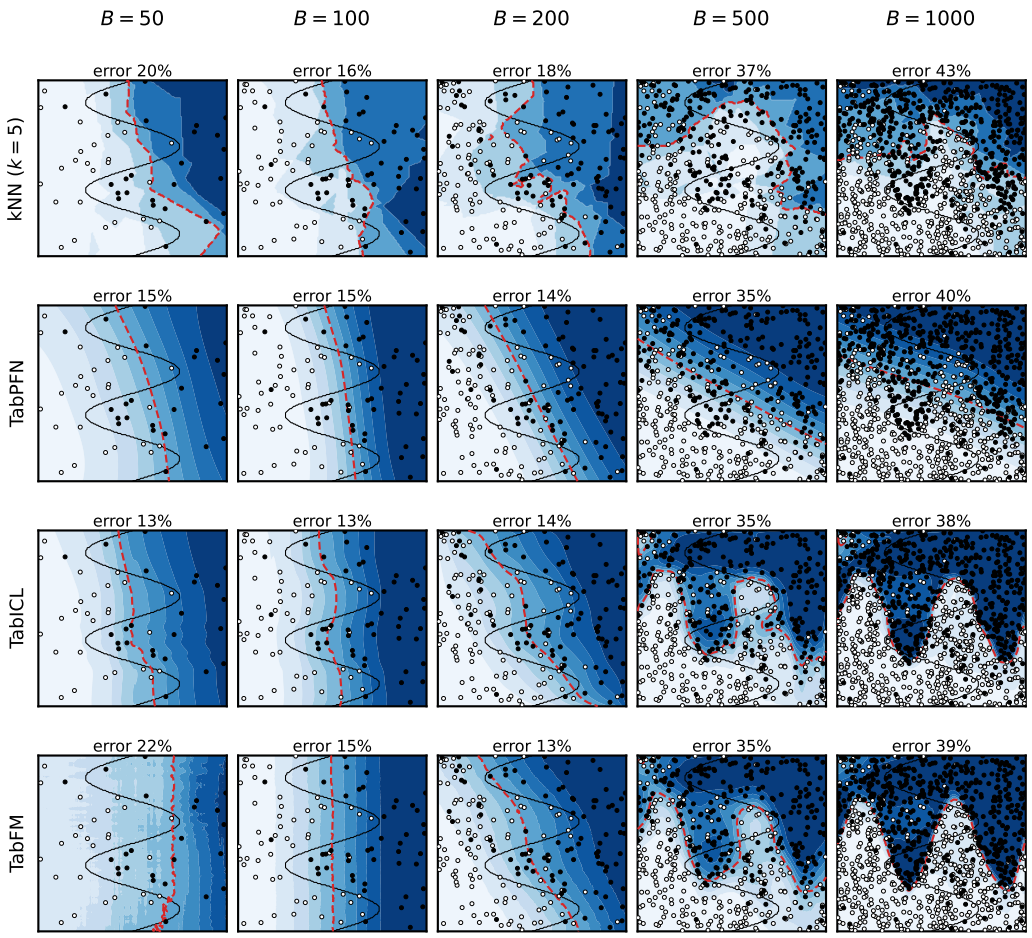

Figure 12: Decision functions after the first abrupt drift for increasing FIFO memory sizes. Large memories still hold the previous concept.

Table 12: Mean accuracy around the first drift (all) and just after it (+300) for fixed memory sizes, oracles and selectors.

	kNN		TabPFN v1		TabICL v2		TabFM	
	all	+300	all	+300	all	+300	all	+300
Best fixed size	.766	.760	.822	.780	.878	.787	.886	.780
$B = 50$	.740	.760	.808	.780	.829	.783	.811	.767
$B = 1{,}000$	.718	.593	.748	.613	.811	.610	.814	.610
Window oracle	.808	.777	.848	.807	.923	.823	.928	.830
One-switch oracle	.774	.760	.821	.780	.885	.783	.878	.767
Selector, all sizes	.764	.740	.823	.790	.893	.787	.907	.807
Selector, 50/1,000	.758	.753	.816	.780	.868	.763	.873	.770
Majority vote	.779	.713	.817	.757	.888	.773	.888	.757

C.6 Computational Cost

Table 13 reports per-instance serving time across streams, which captures the variation in deployment cost across datasets. Table 14 reports resident memory growth per instance between the first and last evaluation windows. This measure reduces the effect of initial process allocation and model loading, although it still includes allocator and evaluation-process overhead. The reported values are process-level memory behaviour rather than isolated model state; the raw resident and GPU traces are in the released result files.

Table 13: Per-stream serving time in ms per instance and mean latency rank, with lower values indicating faster serving.

Method	elec	noaa	meter	rialto	posture	agr_a	sea_g	rbf_m	tree_r	Mean	Rank
No-Change	0.20	0.31	0.86	0.19	0.12	0.08	0.10	0.10	0.11	0.17	1.44
Majority-Class	0.30	0.38	0.67	0.16	0.22	0.11	0.09	0.10	0.12	0.19	1.78
HAT	0.31	0.82	1.02	0.35	0.20	0.31	0.17	0.21	0.18	0.32	3.11
ARF	6.50	6.91	41.4	9.46	2.07	6.25	2.49	4.07	6.85	6.30	7.33
SRP	8.87	7.10	53.8	13.3	3.59	8.50	2.89	5.33	10.6	8.47	8.56
Lev. Bagging	2.44	2.96	35.2	6.39	1.64	2.95	1.00	2.64	2.14	3.29	5.78
SAM-kNN	0.31	0.36	1.96	0.51	0.38	0.57	0.27	0.48	0.26	0.45	3.67
ASML	2.66	3.76	59.2	15.2	3.58	2.36	0.70	5.24	2.98	4.55	7.00
EvoAutoML	15.6	21.1	457	70.2	22.9	20.7	13.2	50.9	35.8	36.7	10.44
OAML	2.88	3.92	144	22.2	1.06	1.41	0.54	1.03	1.94	3.38	6.44
CURE	31.0	97.9	150	116	65.0	98.8	93.2	92.0	101	87.3	12.22
DualFIFO	113	116	523	167	44.5	114	99.6	124	117	127	13.67
TabICL + FIFO	30.7	71.8	88.2	83.2	62.5	71.4	28.1	35.2	111	58.4	10.89
TabPFN + FIFO	54.4	99.1	372	89.8	87.5	113	44.7	65.8	247	103	12.78
DR-TabPFN + FIFO	542	509	1368	628	471	540	486	614	553	599	16.00
TabFM + FIFO	122	124	659	157	114	129	114	133	296	167	14.89

Table 14: Per-stream resident memory growth in KB per instance and mean rank, with lower values indicating less growth. Dashes mark runs without a usable memory trace; means and ranks use the available streams.

Method	elec	noaa	meter	rialto	posture	agr_a	sea_g	rbf_m	tree_r	Mean	Rank
No-Change	0.56	2.65	1.22	6.30	1.83	0.07	**0.01**	**0.01**	0.07	1.41	4.44
Majority-Class	0.74	2.77	30.3	5.97	**0.18**	0.18	0.02	0.05	**0.05**	4.48	5.11
HAT	4.88	3.79	33.6	2.16	0.80	4.27	1.02	1.28	5.27	6.34	9.89
ARF	0.63	6.87	4.83	0.56	4.41	16.6	7.11	10.3	11.1	6.93	11.22
SRP	11.9	4.88	16.1	6.35	5.57	23.4	7.15	6.46	14.8	10.7	14.00
Lev. Bagging	1.09	22.8	2.49	**0.24**	4.24	6.07	7.08	5.06	2.10	5.69	10.11
SAM-kNN	7.67	13.8	—	2.82	1.46	9.72	2.83	2.93	9.10	4.86	10.44
ASML	**0.28**	2.89	1.12	3.54	0.26	0.27	0.17	0.17	—	**0.97**	**4.22**
EvoAutoML	1.65	6.73	43.3	3.99	0.48	2.67	0.64	1.88	6.90	7.58	10.00
OAML	1.70	3.43	25.7	3.87	1.77	0.05	0.13	0.14	0.20	4.11	7.22
CURE	1.69	3.60	30.0	9.36	1.68	—	1.94	1.96	3.26	5.94	9.89
DualFIFO	2.92	3.15	32.7	7.34	2.21	—	0.70	0.72	0.92	5.63	9.78
TabICL + FIFO	1.81	6.72	**0.92**	2.29	2.19	19.5	0.02	0.10	0.10	3.73	7.50
TabPFN + FIFO	1.32	3.39	4.43	6.71	1.96	—	0.62	0.60	0.11	2.13	7.11
DR-TabPFN + FIFO	1.64	1.78	30.9	4.30	3.71	3.00	1.15	1.21	0.80	5.38	9.00
TabFM + FIFO	1.65	**1.70**	30.9	7.24	1.50	**0.02**	0.02	0.05	0.09	4.80	6.06

C.7 Cost at cloud prices

Table 15 converts the geometric-mean serving times of Table 13 into cloud cost per million instances. The CPU methods are charged for a full 8-vCPU on-demand instance at $0.384 per hour, although most use a single core, and the TFMs are charged the median on-demand price of an RTX A6000 across thirteen providers, $0.56 per hour, both as listed on 9 September 2026. Charging the CPU methods for a whole instance is conservative for our claim, since it raises their cost. The ordering of Figure 3 is unchanged: the least expensive TFM costs about ten times the most expensive classical ensemble and over two hundred and fifty times the Hoeffding tree.

Table 15: Serving cost at on-demand cloud prices, derived from Table 13. Prices are the AWS m6i.2xlarge rate and the median RTX A6000 rate on 9 September 2026 and are indicative only.

Method	ms/inst.	Host	USD per 10^6 inst.
HAT	0.32	CPU	0.03
SAM-kNN	0.45	CPU	0.05
Lev. Bagging	3.29	CPU	0.35
OAML	3.38	CPU	0.36
ASML	4.55	CPU	0.49
ARF	6.30	CPU	0.67
SRP	8.47	CPU	0.90
EvoAutoML	36.7	CPU	3.91
TabICL + FIFO	58.4	GPU	9.08
CURE	87.3	GPU	13.6
TabPFN + FIFO	103	GPU	16.0
DualFIFO	127	GPU	19.8
TabFM + FIFO	167	GPU	26.0
DR-TabPFN + FIFO	599	GPU	93.2

D Backbone studies

D.1 Fine-tuning

We test whether prefix-based fine-tuning improves TabICL beyond context adaptation. Table 16 compares the pretrained and selected fine-tuned checkpoints using FIFO memory, with tuning and scored windows kept disjoint. The tuning prefix includes a held-out validation segment, and the selected checkpoint is evaluated only on the remaining stream. Fine-tuning produces negligible changes and no clear improvement in either case, although this result is specific to the evaluated prefixes, training budget, and selection procedure.

Table 16: TabICL performance before and after prefix-based fine-tuning with FIFO memory.

		Windows		Accuracy			κ_t		
Stream	prefix	tuning	scored	pretrained	updated	Δ	pretrained	updated	Δ
electricity	10,000	10	36	**.9394**	.9388	$-.0006$	$+.\mathbf{5639}$	$+.5600$	$-.0039$
agr_a	7,500	8	22	.9404	**.9406**	$+.0002$	$+.\mathbf{6570}$	$+.6566$	$-.0004$

D.2 Similarity instead of attention

To test whether the backbone advantage can be reproduced without learned attention, we replace it with a similarity-based predictor that stores labelled rows and applies cosine-similarity-weighted voting with recency decay and a drift-dependent gate. Table 17 compares this predictor using raw features and frozen TabICL row representations. Raw features perform better on most streams, while both variants remain below TabFM+FIFO. The frozen representations improve performance only on the smart-meter stream and require substantially greater serving time, suggesting that representations learned for use within the backbone do not necessarily transfer well to cosine-similarity voting.

Table 18 further examines the contribution of recency decay and the drift gate. Recency weighting improves overall performance, particularly on several real streams, while the drift gate is more beneficial on the synthetic drifting streams and shows mixed effects on real data. These results indicate that similarity-based adaptation captures some streaming behaviour but does not reproduce the predictive advantage of the full backbone.

Table 17: Accuracy and serving time for similarity-based prediction using raw features, frozen TabICL representations, and TabFM+FIFO.

Stream	raw features	TabICL row repr.	TabFM + FIFO
rbf_m	.924	.720	**.938**
sea_g	.862	.783	**.894**
agr_a	.838	.805	**.942**
electricity	.802	.774	**.950**
noaa	.745	.705	**.820**
rialto	.618	.539	**.924**
tree_r	.610	.596	**.789**
posture_no8	.532	.509	**.635**
meter	.527	.662	**.926**
mean	.717	.677	**.869**
ms/inst.	**0.24**	15.3	167.4

Table 18: Ablation of recency decay and the drift gate in the raw-feature similarity predictor. Δ is off minus on, so negative values favour the component.

	recency decay			drift gate	
Stream	off	on	Δ	off	Δ
Real streams					
rialto	.540	.618	$-.078$	.659	$+.042$
meter	.503	.527	$-.024$	.534	$+.006$
electricity	.789	.802	$-.013$	.815	$+.013$
noaa	.743	.745	$-.002$	.744	$-.001$
posture_no8	.533	.532	$+.001$	.503	$-.029$
Synthetic drifting streams					
agr_a	.837	.838	$-.001$	.810	$-.028$
sea_g	.863	.862	$+.001$	.852	$-.010$
rbf_m	.924	.924	$+.001$	.910	$-.014$
tree_r	.611	.610	$+.000$	.584	$-.027$
mean	.705	.717	$-.013$	.712	$-.005$

D.3 Pre-training small models

We pre-train compact *nanoTabPFN* (Pfefferle et al., 2025)[6] and *nanoTabICL* (Soda Team, Inria, 2026)[7] models to test whether restricted context interactions enable more efficient streaming inference. Bidirectional attention allows changes in memory to affect all retained representations, whereas causal attention restricts each row to earlier observations and can support cached reuse when the remaining architecture is compatible. Sliding attention further limits each row to a window of W observations, giving $O(W)$ attention cost per step at fixed model size. Cached serving and re-encoding are not equivalent because cached rows may retain information from observations that have since been evicted, while re-encoding uses only the current memory. Within each architecture family, the attention mask is varied while the remaining training configuration is fixed, and all models are pretrained from scratch on synthetic drifting sequences that exclude the evaluation streams.

[6] https://github.com/automl/nanoTabPFN

[7] https://github.com/soda-inria/nanotabicl

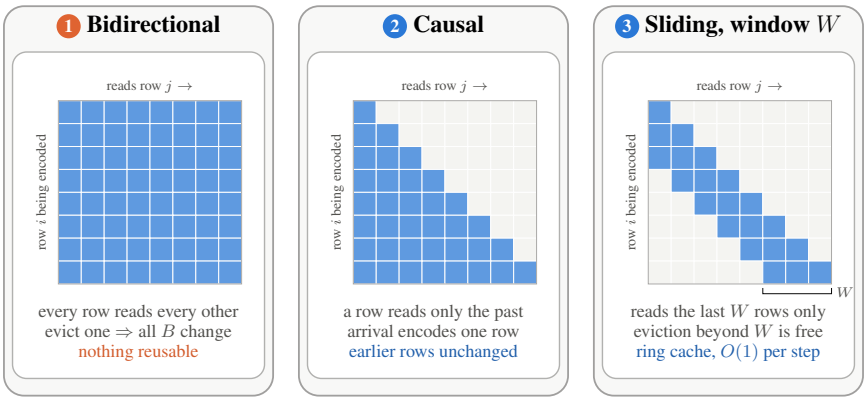

Figure 13: Datapoint-axis attention masks for bidirectional, causal, and sliding attention.

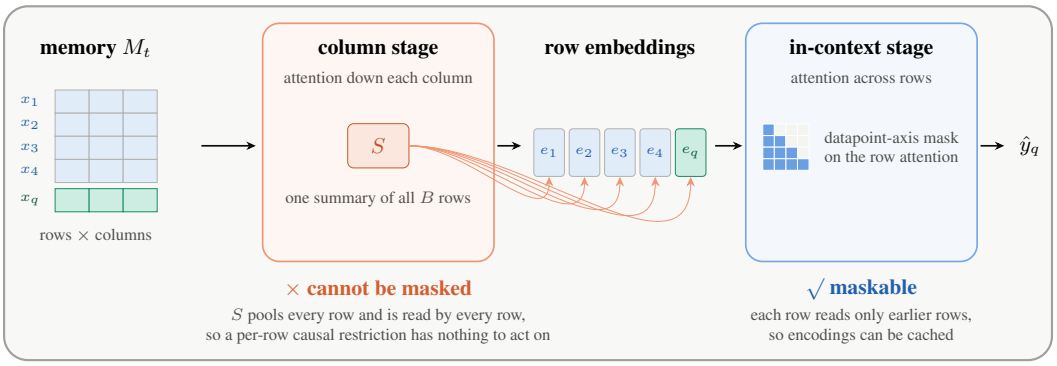

Figure 14: Cache dependencies in the evaluated TabICL-style architecture.

D.4 Datapoint-axis masks

Table 19 and Figure 16 report the predictive effects of the attention masks illustrated in Figure 13. Causal and sliding attention improve accuracy under abrupt, gradual, and incremental drift in both architecture families, with larger gains for nanoTabPFN. Recurring drift differs across architectures, while Figure 14 shows the shared column-stage dependencies in nanoTabICL that limit full cache compatibility. Table 20 and Figure 15 examine sensitivity to the sliding-attention window. Wider windows mainly benefit recurring and stationary sequences, while the other drift types change less beyond moderate window sizes. The mask comparison and window sweep are separate training experiments, so values should be compared within each experiment.

Table 19: Accuracy across datapoint-axis attention masks and drift conditions.

	nanoTabPFN, 356k			nanoTabICL, 8.1M		
Drift type	bidirectional	causal	sliding	bidirectional	causal	sliding
abrupt	.715	.880	**.891**	.749	**.793**	.784
gradual	.687	.876	**.889**	.743	**.758**	.755
incremental	.648	.819	**.856**	.761	**.781**	.780
recurring	**.679**	.620	.623	.737	**.747**	.747
stationary	.883	.875	**.900**	**.875**	.874	.870
mean	.722	.814	**.832**	.773	**.791**	.787

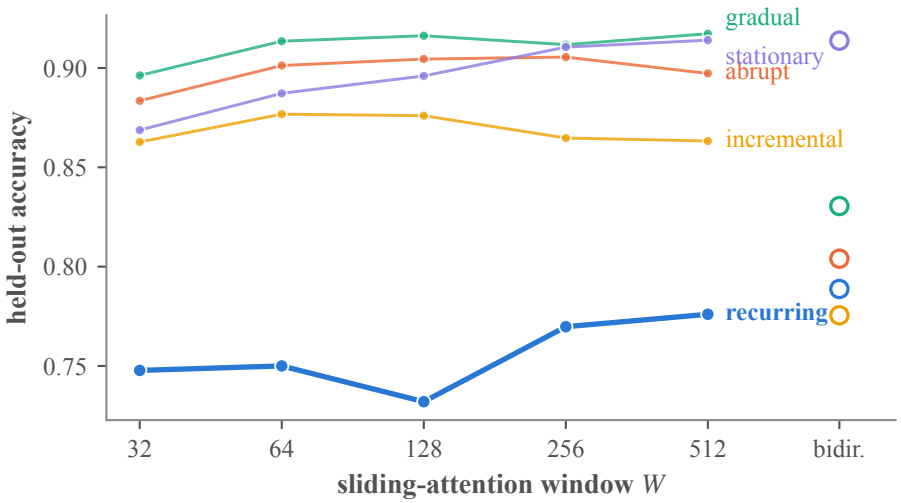

Figure 15: Accuracy sensitivity to the sliding-attention window size.

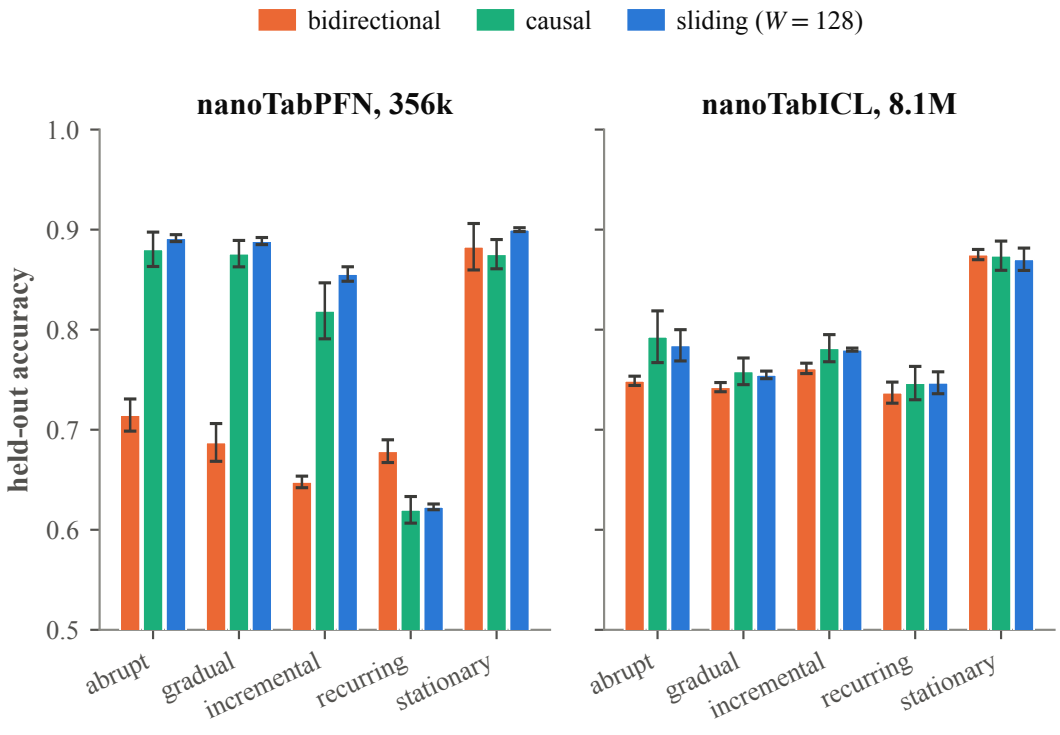

Figure 16: Accuracy across bidirectional, causal, and sliding attention masks.

D.5 Transfer to evaluation streams

Table 21 evaluates the compact pretrained models on evaluation streams outside the pretraining prior. Both variants outperform No-Change on several streams but perform poorly on electricity, where short-lag label autocorrelation is strong, indicating limited use of recent temporal information. A learned recency bias improves NOAA but reduces performance elsewhere, and its slopes remain at their initial values, providing no evidence that useful recency weighting was learned. Both compact models also remain below the released TabICL backbone, leaving the effects of greater capacity and broader pretraining unresolved.

Table 20: Accuracy across sliding-attention window sizes for nanoTabPFN, trained separately from Table 19.

W	abrupt	gradual	incremental	recurring	stationary	mean
32	.883	.896	.863	.748	.869	.852
64	.901	.913	**.877**	.750	.887	.866
128	.904	.916	.876	.732	.896	.865
256	**.906**	.912	.865	.770	.910	.872
512	.897	**.917**	.863	.776	**.914**	**.874**
bidirectional	.804	.831	.776	**.789**	**.914**	.823

Table 21: Transfer accuracy and temporal κ_t on evaluation streams. Bold marks the better compact model.

	electricity		noaa		sea_g		tree_r	
Model	acc.	κ_t	acc.	κ_t	acc.	κ_t	acc.	κ_t
No-Change	.853	.000	.680	.000	.561	.000	.505	.000
TabICL + FIFO	.944	+.616	.818	+.431	.891	+.753	.767	+.530
sliding	.602	−1.71	.732	+.160	**.814**	**+.575**	**.559**	**+.108**
sliding, learned recency bias	.428	−2.90	**.742**	**+.194**	.732	+.389	.551	+.091

D.6 CACHED SERVING

Figure 8 compares cached serving with re-encoding using the same checkpoint and stream prefix. At a fixed attention window, cached serving remains nearly constant across memory budgets in both time and peak GPU memory, while re-encoding becomes increasingly expensive in both, with memory growing faster than time because the attention footprint scales with the square of the retained memory.

Table 22: The same sliding-window checkpoint served from cache and by re-encoding on a stream prefix, per memory budget. Δ is cached minus re-encoded accuracy, time is per instance, and peak GPU memory is the allocator high-water mark, which depends only on the feature count and B. Electricity was timed on a second host of the same type.

		Accuracy			κ_t		Time (ms/inst.)		Peak GPU (MB)	
Stream	B	Cached	Re-enc.	Δ	Cached	Re-enc.	Cached	Re-enc.	Cached	Re-enc.
electricity	128	.609	.395	+.214	−1.469	−2.824	8.2	5.6	26.5	31.3
	250	.609	.609	.000	−1.469	−1.469	7.7	5.7	26.5	62.7
	500	.609	.404	+.205	−1.469	−2.768	8.1	10.4	26.5	173
	1,000	.609	.405	+.205	−1.469	−2.764	8.2	28.0	26.5	604
	2,000	.609	.430	+.179	−1.469	−2.605	8.2	83.1	26.5	2,297
noaa	128	.750	.766	−.016	.221	.271	7.1	4.9	26.5	31.3
	250	.750	.754	−.004	.221	.233	7.2	5.1	26.5	62.7
	500	.750	.752	−.002	.221	.227	7.1	7.6	26.5	173
	1,000	.750	.750	.000	.221	.222	7.1	20.3	26.5	604
	2,000	.750	.746	+.004	.221	.208	7.0	59.0	26.5	2,297
agr_a	128	.653	.658	−.006	.223	.236	7.2	4.8	27.4	32.8
	250	.653	.649	+.003	.223	.215	7.2	5.1	27.4	67.5
	500	.653	.652	+.001	.223	.220	7.3	8.0	27.4	190
	1,000	.653	.653	.000	.223	.222	7.4	22.2	27.4	671
	2,000	.653	.655	−.002	.223	.227	7.2	65.4	27.4	2,549
sea_g	128	.824	.829	−.004	.609	.619	5.9	4.9	21.8	24.0
	250	.824	.829	−.004	.609	.619	6.0	5.1	21.8	37.8
	500	.824	.831	−.006	.609	.623	5.9	5.2	21.8	88.2
	1,000	.824	.831	−.006	.609	.623	5.9	10.5	21.8	281
	2,000	.824	.829	−.004	.609	.619	5.9	28.4	21.8	1,039

E PRIOR WORK

This appendix describes the methods evaluated in this study, with the limitation of each that is most relevant here.

Memory management for TFMs. *DualFIFO* (Lourenço et al., 2026) keeps a short-term FIFO buffer of recent rows and a long-term store, and when the store is full it drops the oldest row of the most represented class, so that the memory stays class-balanced. It is evaluated only on real-world streams, where the drift type is unknown, and its cost is reported with ten instances predicted per forward pass, so neither the effect of drift type nor single-instance serving cost is measured. *CURE* (Lee et al., 2026) splits the budget into a short bank of recent rows and a long bank. Rows leaving the short bank enter the long bank only if their prediction entropy was high, and the long bank evicts a row that has a close same-class neighbour. Its evaluation includes one stream with a known drift type, and its runtime is compared only with DualFIFO, so its behaviour across drift types and its cost relative to classical learners remain open.

TFM backbones. *TabPFN v1* (Hollmann et al., 2023) is pretrained on synthetic tables and lets every context row attend to every other row. It is limited to small contexts and at most ten classes, and it is the backbone shared by DualFIFO and Drift-Resilient TabPFN. *Drift-Resilient TabPFN* (Helli et al., 2024) pretrains the same architecture on a prior whose causal model changes over time and receives a time index as input. It is designed and evaluated for batch data split into time domains rather than for streams, so applying it to a stream requires choosing how arrival order maps to domains. *TabICL v2* (Qu et al., 2026) first embeds each row from its columns and then lets rows attend to one another, which scales to larger contexts than TabPFN v1. Its column stage attends over all rows, so the context cannot be encoded incrementally. *TabFM* (Google Research, 2026) alternates attention over columns and rows, compresses each row, and then attends over the compressed rows. It is released without a technical report and accepts at most ten classes.

Stream learners. *Hoeffding Adaptive Tree* (Bifet & Gavaldà, 2009) grows a decision tree from running statistics and replaces subtrees when an ADWIN detector signals change. As a single tree, it adapts slowly after a drift. *Adaptive Random Forest* (Gomes et al., 2017) and *Streaming Random Patches* (Gomes et al., 2019) train ensembles of Hoeffding trees on resampled instances and random feature subsets, and swap in background trees when a detector fires. Both need many instances before their trees split, which gives them a slow cold start. *Leveraging Bagging* (Bifet et al., 2010b) increases the resampling weight and resets its worst member on detected change, so each reset rebuilds a member from scratch. *SAM-kNN* (Losing et al., 2016) keeps a short-term memory sized by recent error and a long-term memory of consistent older examples, and predicts by nearest neighbours. It adapts through stored examples like a TFM, but its distances degrade with irrelevant features.

Online AutoML. *OAML* (Celik et al., 2023) re-runs a pipeline search on a buffer when drift is detected, so it depends on the detector and adds search cost at every trigger. *EvoAutoML* (Kulbach et al., 2022) evolves a population of pipelines and replaces the weakest by mutation, with a cost that grows with the population. *ASML* (Verma et al., 2024) searches around the current best pipeline, balancing exploration and exploitation. Like the other AutoML systems, its accuracy depends on search running alongside prediction.

Evaluation under delay and compute limits. Prequential evaluation assumes that each label arrives before the next prediction (Gama et al., 2013). Delayed-label protocols relax this assumption (Grzenda et al., 2020; Souza et al., 2018), and work on online continual learning and test-time adaptation shows that a method slower than the arrival rate must skip or delay updates (Ghunaim et al., 2023; Alfarra et al., 2023; Prabhu et al., 2023). Existing TFM stream methods assume immediate labels and report cost separately from accuracy, which our label-delay and cost studies address.

Reusing encoded context. Language models avoid re-encoding their past through segment-level recurrence (Dai et al., 2019), local attention (Beltagy et al., 2020) and key-value caching (Xiao et al., 2024). These rely on causal attention over the sequence, which current TFM backbones lack, and our cached-serving proof of concept adds it to a compact TFM.

F ACCURACY AND COST OF STREAMING TFMS

This appendix gives a simple account of why streaming TFMs are accurate and why they are costly. Let L denote the number of layers, h the embedding width, W the attention window, B the memory size and d the label delay.

F.1 WHY TFMS ARE ACCURATE ON STREAMS

Let Π be the pretraining prior and $p_\Pi(y \mid \mathbf{x}, M)$ its posterior predictive distribution. For any predictor f, the prior-averaged log-loss satisfies

$$\mathcal{L}_\Pi(f) - \mathcal{L}_\Pi(p_\Pi) = \mathbb{E}_\Pi\Big[\mathrm{KL}\big(p_\Pi(\cdot \mid \mathbf{x}, M) \,\big\|\, f(\cdot \mid \mathbf{x}, M)\big)\Big] \geq 0, \tag{1}$$

so pretraining drives f_θ towards p_Π (Müller et al., 2022). This does not guarantee optimal accuracy on a given stream. It explains why a TFM can predict well from a small memory when the stream is well represented by the prior, and why it fails when it is not (Table 21).

With a full FIFO memory and label delay d, the memory at step t holds rows $t-d-B, \ldots, t-d-1$. After a single abrupt drift whose first new-concept instance is τ, the share of old-concept rows is

$$\rho_t = 1 - \frac{\min\{B, \max\{0,\, t-\tau-d\}\}}{B}, \tag{2}$$

and the mean age of the rows in memory is $\bar{a}_t = d + (B+1)/2$. The memory is therefore fully replaced $B + d$ instances after the drift, without any parameter update. This is a replacement time rather than a recovery bound, since accuracy may recover earlier or remain incomplete after replacement. A larger memory or a longer delay increases the age of the context, which matches the slower recovery and the delay losses in Tables 10 and 11.

A Hoeffding tree splits a leaf once the observed gap $\widehat{\Delta}_n$ between the two best candidates exceeds

$$\epsilon_H(n, \delta) = R\sqrt{\frac{\ln(1/\delta)}{2n}}, \tag{3}$$

where n counts instances at that leaf, R is the range of the split criterion and δ the failure probability (Domingos & Hulten, 2000). Resolving a gap of η therefore takes on the order of $R^2 \ln(1/\delta)/(2\eta^2)$ instances per leaf, and this is paid again after every reset. This offers one explanation for the slower cold start and drift recovery of the tree ensembles in Figure 4.

F.2 WHY TFMS ARE COSTLY AND HOW TO REDUCE THE COST

For a row transformer of width h, with the feature count held fixed, re-encoding the memory at every prediction costs

$$C_{\text{re-encode}} = O\big(L(B^2 h + B h^2)\big) \tag{4}$$

per instance, although only one row changes per step. The fitted serving time of Table 6 gives a per-instance cost of

$$\bar{t}(B, q) \approx \frac{A(B)}{q} + \gamma(B) \tag{5}$$

when q queries share one call. Within the tested range, batching approaches the fitted floor $\gamma(B)$ but does not remove $A(B)$ from each call. With causal attention over a fixed window of W rows, each row is encoded once and read from a cache, so

$$C_{\text{cache}} = O\big(L(Wh + h^2)\big), \qquad S_{\text{cache}} = O(LWh), \tag{6}$$

which no longer depends on B. Causality alone is not enough, since attention over the full cached history still grows with B. Cached and re-encoded predictions also need not agree exactly, because cached rows may reflect rows that have since been evicted (Table 22). This matches the flat cached curves in Figure 8. Current TFMs attend over rows in both directions, so they cannot reuse encoded rows in this way without architectural change.