

METACASTER: Meta-Harness-Optimized Agent for End-to-End Few-Shot Learning of Lightweight Time Series Forecasters

ChengAo Shen¹, Wenchao Yu², Fangyu Wu³, Dongjin Song⁴, Hanghang Tong⁵,
Dongsheng Luo⁶, Wei Cheng², Haifeng Chen², Jingchao Ni¹

¹University of Houston, ²NEC Labs, ³University of Waterloo, ⁴University of Connecticut,

⁵University of Illinois at Urbana-Champaign, ⁶Singapore Management University

¹{cshen9, jni7}@uh.edu, ²{wyu, weicheng, haifeng}@nec-labs.com,

³fangyu.wu@uwaterloo.ca, ⁴dongjin.song@uconn.edu,

⁵htong@illinois.edu, ⁶dsluo@smu.edu.sg

Abstract

Time series forecasting (TSF) is evolving toward multimodal and agentic settings, yet using foundation models remains uneconomical in resource-constrained scenarios, where compact, specialized forecasters are more desirable. However, lightweight forecasters typically require substantial training data, limiting their use in domains with scarce, slowly accumulated, or privacy-sensitive time series. To address this dilemma, we investigate the challenging problem of few-shot learning for lightweight forecasters. We propose METACASTER, a meta-harness-optimized multi-agent framework that uses agentic data generation to automatically train specialized lightweight forecasters from only a few examples and textual contexts. Our work highlights a new TSF paradigm in which agents act not as forecasters but as intermediary engineers that prepare efficient, task-specific forecasters for deployment. Experiments on 18 datasets, 23 state-of-the-art lightweight forecasters, and 14 baselines demonstrate that METACASTER achieves both data efficiency and computational efficiency while maintaining high-quality TSF performance.¹

1 Introduction

Time series forecasting (TSF) underpins decision-making and intelligence across domains such as geoscience, healthcare, and energy (Koprinska et al., 2018; Morid et al., 2023; Ardid et al., 2025). Unlike traditional TSF, which focuses on numerical prediction at future timesteps, emergent TSF settings are multimodal, incorporating textual context describing domains, events, or external conditions (Jiang et al., 2025). This transition is driven by advances in large language models (LLMs) and AI agents. As Fig. 1 illustrates, recent LLM-based TSF methods have two main paradigms: (a) *LLM-as-Forecaster* – where pre-trained LLMs serve as

forecasting backbones with additional cross-modal adapters and TSF heads (e.g., TimeLLM (Jin et al., 2024), S2IPLLM (Pan et al., 2024), TimeVLM (Zhong et al., 2025), etc.); and (b) *Agent-as-Forecaster* – where LLM agents interpret time series in contextual prompts, reason about future trends, and generate forecasts conversationally or visually (e.g., LLMTime (Gruver et al., 2023), TimeOmni-1 (Guan et al., 2026a), Nexus (Das et al., 2026), etc.).

Despite the inspiring progress, using LLMs as the computational core for forecasting is largely obstructed by the modality gap between *discrete* language tokens and *continuous* time series values. Consequently, the effectiveness of the paradigms in Fig. 1(a) and (b), compared with specialized non-LLM forecasters, remains under debate (Tan et al., 2024; Merrill et al., 2024). Benchmark studies consistently show that dedicated numerical models, such as DLinear (Zeng et al., 2023) and PatchTST (Nie et al., 2023), can outperform LLM-based approaches (Shen et al., 2025a), motivating the rise of time series foundation models (TSFMs) such as Moirai (Woo et al., 2024), Chronos (Ansari et al., 2024), and Sundial (Liu et al., 2025c).

However, using large models – whether LLMs or TSFMs – as one-size-fits-all solutions raises concerns about sustainability and carbon footprint (Bolón-Canedo et al., 2024). The demand for rapid deployment in resource-constrained settings (e.g., edge devices and small organizations) has therefore sparked growing interest in lightweight forecasters (Ma et al., 2024), analogous to the rise of small language models (SLMs) (Belcak et al., 2025). Notably, recent studies show that lightweight forecasters can rival LLMs and TSFMs in individual TSF tasks (Shen et al., 2025b). Their main limitation is the need for sufficient downstream training data, as they lack large-scale pre-training.

In practice, collecting long-history time series for training can significantly delay deployment. In privacy-sensitive domains such as healthcare and

¹The code of METACASTER is available at <https://github.com/D2I-Group/metacaster>.

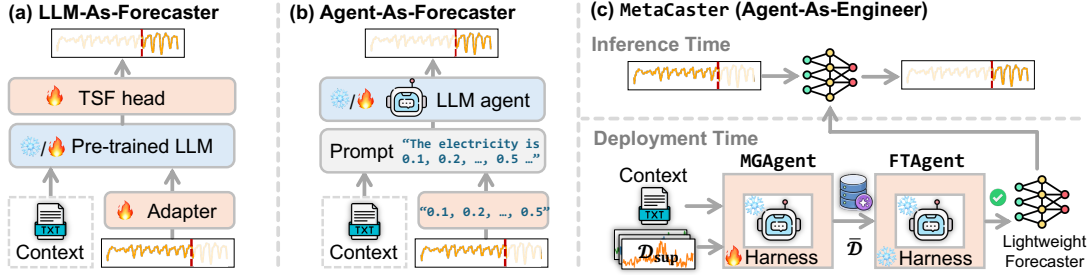


Figure 1: Comparison of different paradigms of using LLMs for TSF. METACASTER is the proposed model.

finance, acquiring large training datasets may even be infeasible. This raises a challenging question: *can lightweight forecasters be effectively trained with only a few samples?* Although such models are highly prone to overfitting without large-scale pre-training, we demonstrate that a solution is possible in the era of agentic AI.

To address this challenge, we propose METACASTER, a Meta-harness-optimized agent for end-to-end few-shot learning of lightweight forecasters. As Fig. 1(c) illustrates, rather than directly generating forecasts, METACASTER’s agents act as intermediaries that prepares a specialized forecaster for a target TSF task. At deployment time, given a few-shot support set $\mathcal{D}_{sup}$ and contextual description C , METACASTER employs two agents: (1) MGAGENT refers to $\mathcal{D}_{sup}$ and C , and generates a sufficient dataset $\bar{\mathcal{D}}$ that complies with domain constraints; and (2) FTAGENT trains and selects the best lightweight forecasters using $\bar{\mathcal{D}}$. To support FTAGENT, we compile 23 state-of-the-art (SOTA) lightweight forecasters (2022-2026) into LT-LIB library (§3.5) with a unified API. Similar to fine-tuning a foundation model, METACASTER performs task adaptation at deployment time, but more efficiently via the novel agents and LLM API calls. After deployment, only the selected lightweight forecaster is used for inference.

Unlike existing time series generation models that focus on data simulation (Huang et al., 2025b; Gu et al., 2025; Ge et al., 2025), METACASTER generates time series specifically to improve forecasting performance. To our knowledge, it is the first framework to align data generation with forecaster quality. Motivated by recent findings on the importance of agent Harnesses (Li et al., 2026a; Lee et al., 2026) – the infrastructure surrounding an LLM (e.g., system prompts, skills, tools) – we optimize MGAGENT’s Harness using a meta-harness HPAGENT (Fig. 2), which is more efficient than fine-tuning LLMs. The optimized Harness is also

transferable across different LLMs, enabling flexible API switching in downstream deployment, as demonstrated in §4.2. In summary, our contributions are as follows.

- We investigate the challenging problem of few-shot learning for lightweight forecasters.
- We propose METACASTER, a novel meta-harness-optimized multi-agent framework that achieves large-model-like performance with efficient inference for TSF tasks.
- We compile 20+ SOTA lightweight forecasters into LT-LIB, a unified library that will be released alongside METACASTER.
- We conduct comprehensive experiments on 18 datasets against 14 baselines, demonstrating the effectiveness of METACASTER.

2 Related Work

LLM-based TSF. As discussed in §1, many existing LLM-based TSF models adopt either *LLM-As-Forecaster* (Zhou et al., 2023; Jin et al., 2024; Pan et al., 2024; Liu et al., 2024, 2025a; Wang et al., 2025; Zhong et al., 2025) or *Agent-As-Forecaster* (Xue and Salim, 2023; Gruver et al., 2023; Das et al., 2026). Additionally, some multi-task time series QA agents can conduct TSF via LLM reasoning (Kong et al., 2025; Guan et al., 2026a; Yu et al., 2025; Wu et al., 2026; Guan et al., 2026b). In contrast, there are relatively fewer *Agent-As-Engineer* models that use Harness for TSF (Zhao et al., 2025; Garza and Rosillo Garcia, 2025; Zhang et al., 2025; Jatori et al., 2025). However, these models don’t generate times series (thus require large training data), and never automatically optimize their Harness for TSF, distinguishing them from METACASTER.

Time Series Generation. Our work is related to time series generation models, including data augmentation techniques (Luo et al., 2023; Yue et al., 2022; Iwana and Uchida, 2021; Wen et al., 2021), generative models that focus on data distributions

(Yoon et al., 2019; Desai et al., 2021; Jeon et al., 2022; Yuan and Qiao, 2024), and language-based models that can encode contexts (Naiman et al., 2024; Huang et al., 2025b; Gu et al., 2025; Ge et al., 2025). However, these standalone generators aim to simulate certain data properties, rather than directly optimize TSF performance, leaving a significant gap as we will demonstrate in §4.

Agent Harness Optimization. AI agents are undergoing a paradigm shift. Recent findings challenge the assumption that better models alone produce more reliable agents (Li et al., 2026a; Ning et al., 2026). Rather, improving the infrastructure layer around an agent, *i.e.*, the *agent harness*, can significantly enhance its performance (Li et al., 2026a), leading to growing attention to harness engineering (Lin et al., 2026), and emerging techniques for optimizing texts (Yuksekgonul et al., 2025) and harness (Lee et al., 2026) within agentic systems. Unlike time series agents that mostly rely on supervised fine-tuning or reinforcement learning (Guan et al., 2026a,b; Wu et al., 2026), the proposed METACASTER is, to our best knowledge, the first time series agent exposed to automatic harness optimization.

3 The Proposed Method

3.1 Problem Statement

Given a multivariate time series (MTS) $\mathbf{X} = [\mathbf{x}^1, \dots, \mathbf{x}^D]^\top \in \mathbb{R}^{D \times T}$ within a *look-back window* of length T , where $\mathbf{x}^d \in \mathbb{R}^T$ ($1 \leq d \leq D$) is a univariate time series (UTS) of the d -th variate, the goal of TSF is to estimate the most likely values of the MTS at future H time steps, *i.e.*, $\bar{\mathbf{Y}} \in \mathbb{R}^{D \times H}$, such that the difference between the estimation and the ground truth $\mathbf{Y} = \mathbf{X}_{T+1:T+H} \in \mathbb{R}^{D \times H}$ is minimized in terms of a metric, such as mean squared error (MSE).

In this work, we integrate multiple lightweight forecasters $\mathcal{F} = \{f_1, \dots, f_L\}$, where $f_l : \mathbb{R}^{D \times T} \rightarrow \mathbb{R}^{D \times H}$ ($1 \leq l \leq L$). We refer to $\mathcal{F}$ as LT-LIB (Lightweight Time series Library, §3.5). It is noteworthy that the forecasters in $\mathcal{F}$ are non-pre-trained due to their small capacities.

The Task. Given $\mathcal{F}$, a K -shot *support set* $\mathcal{D}_{\text{sup}} = \{(\mathbf{X}_i, \mathbf{Y}_i)\}_{i=1}^K$, and a textual context description $\mathbf{C}$ about the target domain (*e.g.*, climatology, healthcare), the task is to train the forecasters in $\mathcal{F}$ with optimal forecasting errors on a test set. The task is challenging when K is small, *i.e.*, in the few-shot setting, especially for the small forecasters in $\mathcal{F}$.

3.2 Framework Overview

To address the task, we propose METACASTER, a multi-agent system that generates a sufficient dataset $\bar{\mathcal{D}} = \{(\mathbf{X}_i, \mathbf{Y}_i)\}_{i=1}^{N'}$ with $N' \gg K$ based on the limited data in $\{\mathcal{D}_{\text{sup}}, \mathbf{C}\}$, and then splits $\bar{\mathcal{D}}$ into a training set $\bar{\mathcal{D}}_{\text{tr}}$ and a validation set $\bar{\mathcal{D}}_{\text{val}}$ for training the forecasters in $\mathcal{F}$.

Unlike existing generative models that focus on data realism (Huang et al., 2025b; Gu et al., 2025; Ge et al., 2025), METACASTER generates $\bar{\mathcal{D}}$ to ensure that forecasters trained on $\bar{\mathcal{D}}_{\text{tr}}$ perform comparably to those trained on real data of the same size (as $|\bar{\mathcal{D}}_{\text{tr}}|$) in the target domain. This objective difference is crucial: standard generators cannot guarantee faithful data reproduction, and the resulting mismatch may lead to biased forecasters. In contrast, METACASTER directly optimizes data generation for downstream forecasting performance, learning to produce training data that is better suited for building effective time series forecasters in the target domain.

To enable agent optimization, we adopt a meta-harness strategy (Lee et al., 2026) given recent findings on harness’s significant impact on agent performance (Li et al., 2026a). Fig. 2 illustrates the optimization framework of METACASTER, consisting of three key components: (1) a Meta-Generator (MGAGENT, §3.4); (2) a Forecaster Trainer (FTAGENT, §3.5); and (3) a Harness Proposer (HPAGENT, §3.6). Among them, HPAGENT optimizes the Harness of MGAGENT during the agent optimization process, which will be dropped at deployment and inference time (§3.7).

3.3 The Harness Optimization Problem

To enable METACASTER to generate domain-specific training data, we construct a corpus of time series datasets $\mathcal{C}_{\text{har}} = \{\mathcal{D}^m, \mathbf{C}_m\}_{m=1}^M$ for Harness optimization, where $\mathcal{D}^m = \{(\mathbf{X}_i, \mathbf{Y}_i)\}_{i=1}^{N_m}$ denotes the time series from the m -th domain and $\mathbf{C}_m$ its associated context. The corpus spans M diverse domains to improve METACASTER’s cross-domain generalizability.

As illustrated in Fig. 2, each dataset $\mathcal{D}^m$ is split into train/validation/test sets $\mathcal{D}_{\text{tr}}^m$, $\mathcal{D}_{\text{val}}^m$ and $\mathcal{D}_{\text{te}}^m$, representing the authentic data from the m -th domain. To simulate practical few-shot settings, K samples from $\mathcal{D}_{\text{tr}}^m$ are used to construct $\mathcal{D}_{\text{sup}}^m$. METACASTER then leverages $\{\mathcal{D}_{\text{sup}}^m, \mathbf{C}_m\}$ to generate $\{\bar{\mathcal{D}}_{\text{tr}}^m, \bar{\mathcal{D}}_{\text{val}}^m\}$. The objective is to minimize the performance gap between forecasters trained

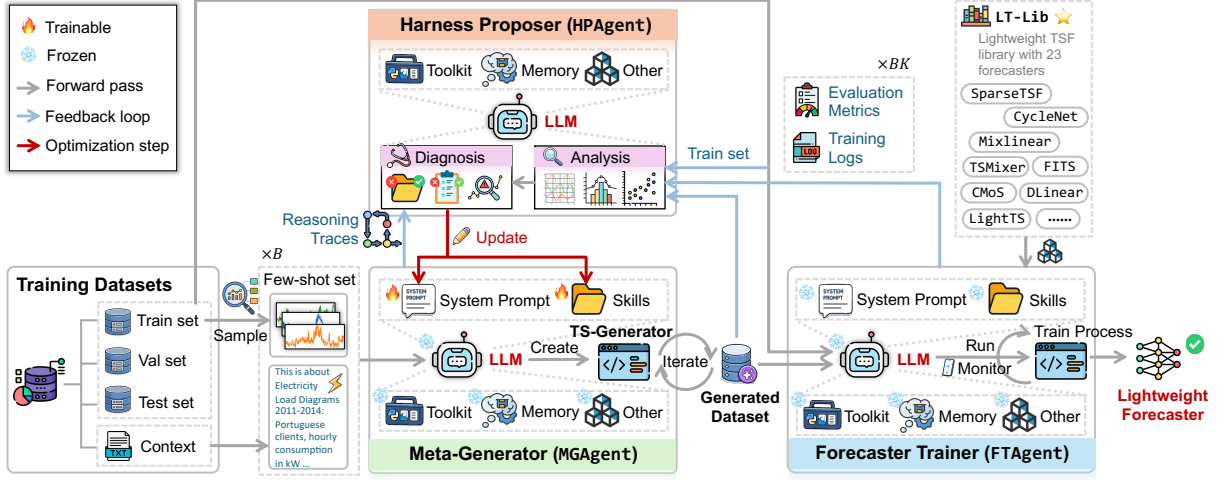


Figure 2: An illustration of the harness optimization framework of the proposed METACASTER system.

on $\{\mathcal{D}_{tr}^m, \mathcal{D}_{val}^m\}$ and those trained on $\{\bar{\mathcal{D}}_{tr}^m, \bar{\mathcal{D}}_{val}^m\}$, evaluated on the same test set $\mathcal{D}_{te}^m$.

Formally, let θ be the trainable Harness in METACASTER, f_l^m ($\bar{f}_l^m$) be the forecaster trained using $\{\mathcal{D}_{tr}^m, \mathcal{D}_{val}^m\}$ ($\{\bar{\mathcal{D}}_{tr}^m, \bar{\mathcal{D}}_{val}^m\}$), the Harness optimization problem is

$$\min_{\theta} \mathbb{E}_{1 \leq m \leq M, 1 \leq l \leq L} (\delta(\omega(f_l^m), \omega(\bar{f}_l^m))) \quad (1)$$

where $\omega(\cdot)$ is a metric that evaluates forecasting errors on $\mathcal{D}_{te}^m$, and $\delta(\cdot, \cdot)$ is a measure of difference between two metrics.

3.4 Meta-Generator (MGAGENT)

Given the few-shot set $\{\mathcal{D}_{sup}^m, C_m\}$, MGAGENT seeks to generate $\bar{\mathcal{D}}^m$, that is

$$\bar{\mathcal{D}}^m = \text{MGAGENT}(\mathcal{D}_{sup}^m, C_m) \quad (2)$$

which will be split into $\{\bar{\mathcal{D}}_{tr}^m, \bar{\mathcal{D}}_{val}^m\}$ by FTAGENT (§3.5). Here, C_m provides semantic cues that guide MGAGENT to generate domain-compliant signals using methods pertinent to the m -th domain.

As shown in Fig. 2, MGAGENT is centered on a (replaceable) LLM. Rather than generating MTS directly, the LLM uses its Harness to create a TS-Generator program that integrates domain-specific knowledge, rules, and models pertinent to the m -th domain, circumventing LLM’s limited capability in direct time series inference (Merrih et al., 2024). Instead, MGAGENT exploits the LLM’s strengths in planning, reasoning, and coding to orchestrate more suitable tools for time series inference, hence the “Meta-” prefix.

Upon receiving $\{\mathcal{D}_{sup}^m, C_m\}$, MGAGENT (1) analyzes the MTS in $\mathcal{D}_{sup}^m$, (2) creates TS-Generator to generate $\mathcal{D}^m$, and (3) performs quality checks.

It accepts $\mathcal{D}^m$ if all checks pass. Otherwise, it goes to step (2) to revise TS-Generator.

In this process, the Harness exposes several components to the LLM, including a *system prompt*, *skills*, a *toolkit*, *long-term memory*, and other execution *middleware*. The system prompt defines MGAGENT’s behavior, goals, constraints, and tool-usage rules. Skills (Li et al., 2026b) are reusable modules containing instructions, metadata, and tool bindings for the time series generation task in Eq. (2). The toolkit provides APIs for tools such as Python and web search, while the long-term memory stores information about generated data to support iterative review and refinement in steps (2) and (3).

In the Harness, the system prompt and skills define key behavior of MGAGENT. Thus, we freeze the LLM and use them as the trainable parameters θ . Instead of Harness engineering (Li et al., 2026a), θ is edited by HPAGENT (§3.6) in an outer loop to automatically optimize Eq. (1).

3.5 Forecaster Trainer (FTAGENT)

FTAGENT splits $\bar{\mathcal{D}}^m$ into $\{\bar{\mathcal{D}}_{tr}^m, \bar{\mathcal{D}}_{val}^m\}$ according to the partition sizes of $\{\mathcal{D}_{tr}^m, \mathcal{D}_{val}^m\}$, orchestrates computational resources to train the forecasters in $\mathcal{F}$, and evaluates them on $\mathcal{D}_{te}^m$, producing trained forecasters, evaluation metrics, and training reports:

$$\begin{aligned} & \{ \{ f_l^m, \bar{f}_l^m, \omega(f_l^m), \omega(\bar{f}_l^m) \}_{l=1}^L, R \} \\ & = \text{FTAGENT}(\{\mathcal{D}_{tr}^m, \mathcal{D}_{val}^m, \mathcal{D}_{te}^m\}, \{\bar{\mathcal{D}}_{tr}^m, \bar{\mathcal{D}}_{val}^m\}) \end{aligned} \quad (3)$$

where $\omega(\cdot)$ is the evaluation metric such as MSE and MAE, and R denotes the training reports.

Lightweight TSF Library (LT-LIB). To construct $\mathcal{F}$, we collect 23 SOTA lightweight forecasters proposed from 2022 to 2026, including

Linear-Layer models (e.g., MixLinear (Ma et al., 2024)), MLP-based forecasters (e.g., TSMixer (Chen et al., 2023)), and Frequency-domain models (e.g., FITS (Xu et al., 2024)). Their sizes, as listed in Appendix C.1, are much smaller than SOTA Transformer-based forecasters (e.g., PatchTST (Nie et al., 2023): $\sim 3.2\text{M}$) and TSFMs (e.g., Chronos (Ansari et al., 2024): $\sim 700\text{M}$). We compile them in LT-LIB with a unified interface to facilitate training calls in FTAGENT.

As shown in Fig. 2, FTAGENT composes programs to train each forecaster with a grid search of hyperparameters. It organizes training jobs – i.e., (forecaster, hyperparameter, dataset) triplets – into a queue and assigns available GPUs for maximally parallel execution. During training, it monitors processes, resolves errors when encountered, and recovers interrupted jobs without human intervention. This procedure ends when the queue is emptied.

FTAGENT’s Harness shares the same components as MGAGENT’s, with additional access to LT-LIB. Unlike MGAGENT, however, its Harness is not optimized, since the training process is relatively standard and has less impact on the resulting forecasters than the generated dataset $\bar{\mathcal{D}}^m$.

3.6 Harness Proposer (HPAGENT)

HPAGENT is the meta-harness (Lee et al., 2026) that configures the Harness of MGAGENT to optimize Eq. (1). Upon receiving $\{\omega(f_l^m), \omega(\bar{f}_l^m)\}_{l=1}^L$ from FTAGENT, HPAGENT evaluates Eq. (1) using a hinge-loss based measure:

$$\delta(\omega(f_l^m), \omega(\bar{f}_l^m)) = \max \left\{ \frac{\omega(\bar{f}_l^m) - \omega(f_l^m)}{\omega(f_l^m)}, 0 \right\} \quad (4)$$

which induces a penalty when the forecasting error $\omega(\bar{f}_l^m)$ of the forecaster trained on the generated $\bar{\mathcal{D}}_{\text{tr}}$ is larger than $\omega(f_l^m)$ of the forecaster trained on the authentic dataset $\mathcal{D}_{\text{tr}}$, suggesting $\bar{\mathcal{D}}_{\text{tr}}$ needs improvements. Otherwise, Eq. (4) won’t induce any penalty when $\bar{f}_l^m$ performs better than f_l^m .

As in Fig. 2, HPAGENT oversees the entire pipeline through three stages: (1) *Self-Planned Analysis*, which collects evidence on MGAGENT and FTAGENT performance, including the loss in Eq. (1), statistical differences between $\mathcal{D}_{\text{tr}}$ and $\bar{\mathcal{D}}_{\text{tr}}$ (e.g., MMD), and training issues from logs R; (2) *Diagnosis*, which identifies potential causes of performance degradation from MGAGENT’s reasoning traces, system prompt, and skills, then determines how to update the Harness θ of MGAGENT; and

(3) *Update*, which edits θ . Effectively, HPAGENT serves as the optimizer of Eq. (1), as defined by its system prompt.

Accordingly, HPAGENT’s lifecycle is the entire optimization process with multiple epochs of updating θ , whereas MGAGENT and FTAGENT each functions within a single epoch. HPAGENT therefore relies more heavily on long-term memory, which stores dataset snapshots, evaluation metrics, training logs, analysis/diagnosis results, θ update logs, etc., across epochs. This memory enables rollback of harmful updates and allows HPAGENT to output the best θ at the end of optimization.

3.7 Training and Inference

Harness Training. We summarize the optimization process in Appendix A. In every epoch, we draw $K \sim \text{Uniform}([10, 50])$. Notably, instead of feeding the K -shot support set $\{\mathcal{D}_{\text{sup}}^m, \mathcal{C}_m\}$ to METACASTER individually, we sample a batch of B datasets from the corpus $\mathcal{C}_{\text{har}}$ to extract B support sets, and let METACASTER process them simultaneously. This extension allows HPAGENT to evaluate generalizability across datasets and mitigate overfitting to any specific dataset.

Inference. After optimization, HPAGENT outputs the best θ^* , and is discarded. The Harnesses of MGAGENT (including θ^*) and FTAGENT (including LT-LIB) are retained, while their LLMs are removed. For a new target domain g , these Harnesses are attached to user-selected LLM APIs to instantiate MGAGENT and FTAGENT. Given K -shot $\mathcal{D}_{\text{sup}}^g$ and context $\mathcal{C}_g$, the agents execute the forward process in Fig. 2 to produce the trained forecaster $\bar{f}_*^g$. At TSF stage, only $\bar{f}_*^g$ needs to be maintained, while MGAGENT and FTAGENT are dropped, as shown in Fig. 1(c).

Therefore, the entire downstream deployment and inference process can be accomplished using low-cost devices capable of running the Harnesses and the LT-LIB framework, making it more efficient than relying on GPU-intensive TSFMs.

4 Experiments

4.1 Experimental Setup

Datasets. To evaluate METACASTER, we employ the GIFT-Eval benchmark (Aksu et al., 2024), previously adopted for training TSFMs (Woo et al., 2024). We use 18 datasets across 9 domains (e.g., energy, weather, traffic), split into a train corpus of 8 datasets (i.e., $\mathcal{C}_{\text{har}}$ in §3.3), an *in-domain* (IND)

	Dataset	Ours	Generation Models					Augmentation Methods				References	
		METACASTER	TimeDP	VerbalTS	T2S	DiffTS	TimeVAE	Repeat	Bootstrap	Jitter	MagWarp	$\mathcal{D}_{\text{sup}}^m$	$\mathcal{D}_{\text{tr}}^m$
IND ($K = 10$)	ETTM1	0.376	1.049	1.004	0.922	1.154	0.749	0.781	0.748	0.746	1.919	0.687	0.316
	Electricity	0.300	0.288	1.061	1.150	0.365	0.328	0.380	0.379	0.375	7.057	0.721	0.121
	Seattle	1.522	1.420	1.800	1.792	1.643	1.759	1.955	1.849	1.925	43.24	2.341	1.024
	SZTaxi	0.104	0.124	0.116	0.118	0.111	0.114	0.134	0.118	0.123	1.122	0.187	0.096
	Sales	2.616	2.587	2.419	2.758	4.323	2.795	4.175	2.949	2.946	4.321	3.004	2.927
	Bitbrains	0.068	0.079	0.098	0.094	0.121	0.109	0.197	0.149	0.159	0.826	0.152	0.150
	Solar	0.158	0.208	0.513	0.552	0.238	0.237	0.278	0.272	0.272	0.446	0.440	0.157
OOD	Saugeen	1.960	1.492	1.474	1.578	2.281	2.705	3.119	2.887	2.908	3.893	1.859	1.183
	USbirths	0.702	2.690	1.501	1.540	1.775	0.607	0.752	0.715	0.714	12.08	1.204	0.250
	M4 [†]	2.093	2.178	2.331	2.444	2.688	2.993	3.250	3.101	3.057	8.654	2.628	2.330
IND ($K = 30$)	ETTM1	0.345	1.049	1.027	0.910	1.211	0.684	0.714	0.664	0.662	0.943	0.602	0.316
	Electricity	0.226	0.287	1.055	1.140	0.363	0.268	0.305	0.287	0.290	1.644	0.657	0.121
	Seattle	1.177	1.414	1.806	1.777	1.673	2.158	1.738	1.656	1.622	21.15	1.785	1.024
	SZTaxi	0.114	0.121	0.116	0.115	0.112	0.156	0.162	0.141	0.135	1.626	0.150	0.096
	Sales	2.362	2.479	2.455	2.942	4.439	2.598	3.603	3.131	3.158	3.422	2.611	2.927
	Bitbrains	0.125	0.084	0.097	0.095	0.130	0.114	0.463	0.469	0.489	0.777	0.137	0.150
	Solar	0.152	0.220	0.515	0.576	0.240	0.234	0.254	0.252	0.248	0.356	0.379	0.157
OOD	Saugeen	1.464	1.495	1.480	1.593	2.286	2.019	2.419	2.266	2.277	2.620	1.547	1.183
	USbirths	0.533	1.980	1.509	1.536	1.732	0.474	0.546	0.524	0.525	5.107	0.803	0.250
	M4 [†]	2.112	2.248	2.394	2.522	2.663	3.402	3.126	3.109	3.169	10.321	2.950	2.330
IND ($K = 50$)	ETTM1	0.267	1.063	1.035	0.931	1.284	0.548	0.582	0.556	0.546	0.592	0.456	0.316
	Electricity	0.191	0.279	1.053	1.197	0.352	0.227	0.244	0.234	0.233	1.287	0.472	0.121
	Seattle	1.125	1.423	1.819	1.779	1.732	2.810	1.537	1.462	1.497	12.78	1.533	1.024
	SZTaxi	0.110	0.120	0.114	0.113	0.113	0.318	0.230	0.222	0.216	0.352	0.143	0.096
	Sales	2.835	2.605	2.474	2.852	4.459	2.683	2.778	2.590	2.560	3.039	2.501	2.927
	Bitbrains	0.119	0.097	0.099	0.095	0.124	0.155	0.929	0.976	0.987	1.214	0.119	0.150
	Solar	0.149	0.234	0.520	0.553	0.250	0.547	0.234	0.231	0.229	0.333	0.296	0.157
OOD	Saugeen	1.415	1.464	1.482	1.591	2.291	1.767	2.462	2.380	2.371	2.528	1.515	1.183
	USbirths	0.424	2.366	1.509	1.531	1.687	0.367	0.396	0.380	0.376	5.787	0.574	0.250
	M4 [†]	1.916	2.157	2.455	2.444	2.896	3.279	3.330	3.300	3.282	12.210	2.560	2.330
Wins (of 30)		19	3	3	1	1	3	0	0	0	0	–	–

Table 1: TSF performance comparison on IND and OOD corpora for $K \in \{10, 30, 50\}$ in terms of MSE. Lower MSE is better. **Red** (**blue**) values indicate the best (second-best) MSE per row. [†]M4 uses instance-normalized MSE to address large distribution shifts. MAE results are available in Appendix E.

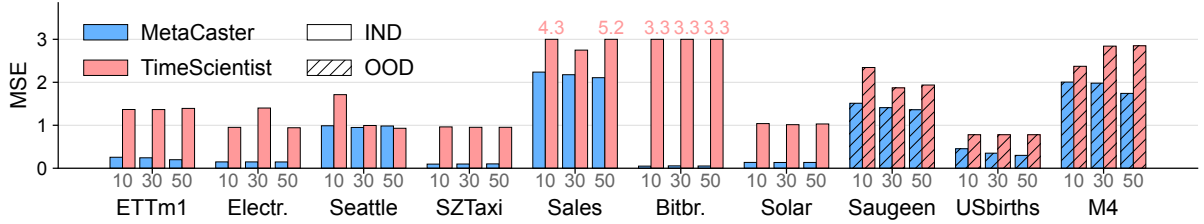


Figure 3: Comparing the selected (trained) forecasters of agent pipelines METACASTER and TimeScientist.

test corpus (7 datasets), and an *out-of-domain* (OOD) test corpus (3 datasets). The IND corpus shares domains with $\mathcal{C}_{\text{har}}$, while the OOD corpus covers unseen domains to evaluate generalization. All test datasets are disjoint from $\mathcal{C}_{\text{har}}$. Each dataset $\mathcal{D}^m$ has a textual context $\mathcal{C}_m$. Time series are split chronologically into 80%/10%/10% train/validation/test sets. Following standard protocol (Nie et al., 2023), the look-back window T is set as 336, the prediction horizon H is 192. Full dataset details and approaches to avoid data leakage are deferred to Appendix B.1.

Baselines. We compare METACASTER with the most relevant SOTA methods, including *time series generation models*: (1) TimeVAE (Desai et al., 2021), (2) DiffTS (Yuan and Qiao, 2024), (3) T2S (Ge et al., 2025), (4) TimeDP (Huang et al., 2025b),

(5) VerbalTS (Gu et al., 2025); *time series augmentation techniques* (Iwana and Uchida, 2021; Luo et al., 2023); (6) Repeat, (7) Bootstrap, (8) Jitter, (9) MagWarp; *pre-trained TSFMs*: (10) Chronos (Ansari et al., 2024), (11) Moirai (Woo et al., 2024), (12) VisionTS (Chen et al., 2024), (13) Time-LLM (Jin et al., 2024); and a SOTA *agent pipeline*: (14) TimeScientist (Zhao et al., 2025). Details about these methods are in Appendix B.2.

Settings. We draw $K \in \{10, 30, 50\}$ samples from each test dataset to constitute $\mathcal{D}_{\text{sup}}^m$, which is fed with $\mathcal{C}_m$ to the compared methods. Generation models and augmentation methods use $\{\mathcal{D}_{\text{sup}}^m, \mathcal{C}_m\}$ to produce $\bar{\mathcal{D}}^m$ of size N'_m , which is used to train the forecasters in our LT-LIB. TSFMs use $\mathcal{D}_{\text{sup}}^m$ for fine-tuning. TimeScientist uses $\mathcal{D}_{\text{sup}}^m$ to directly train forecasters as it cannot generate time series.

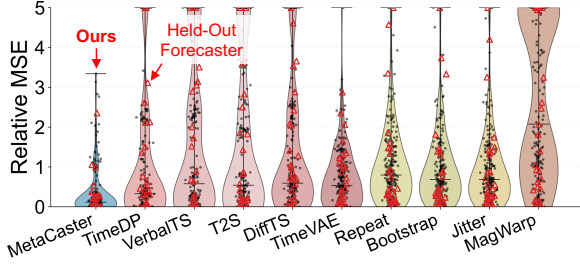


Figure 4: Performance distribution. Each dot represents the performance of a forecaster $\bar{f}_l^m$ on a test set $\mathcal{D}_{te}^m$, covering all forecasters in LT-LIB and all datasets in both of IND and OOD corpora.

For our METACASTER, we set batch size $B = 8$, and use GPT-5.4 (Lee et al., 2026) as the default LLM. We also test different LLMs in our ablation study (Table 2). For all methods that can generate $\bar{\mathcal{D}}^m$, the size N'_m is set to $|\mathcal{D}_{tr}^m| + |\mathcal{D}_{val}^m|$. Following (Nie et al., 2023; Tan et al., 2024), we use MSE and MAE to evaluate the TSF performance.

4.2 Experimental Results

Table 1 reports MSE results comparing generation models, augmentation methods, and METACASTER. We randomly hold out 3 forecasters from LT-LIB and evaluate the average performance over the remaining 20; the held-out forecasters are used to assess METACASTER’s generalization to unseen forecasters (Fig. 4). MAE results are provided in Appendix E, and Appendix F reports the best-performing forecaster among the 20 as trained by the compared methods.

Table 1 also reports results of training forecasters with K -shot $\mathcal{D}_{sup}^m$ and full set $\mathcal{D}_{tr}^m$, serving as lower and upper performance references, respectively. From Table 1, we highlight several observations: (1) METACASTER outperforms the generation/augmentation baselines in most cases, demonstrating the benefit of optimizing data generation for forecasting; (2) METACASTER’s performance improves with larger K , showing effective few-shot utilization; (3) When $K \geq 30$, METACASTER approaches or even surpasses $\mathcal{D}_{tr}^m$, suggesting that raw data may be noisy and optimized data can improve training; (4) In the extreme case when $K = 10$, METACASTER remains competitive; and (5) Despite increased difficulty, METACASTER generalizes well to OOD datasets, where it generally outperforms baselines.

Fig. 4 shows the MSE distribution of individual forecasters trained by the models in Table 1, normalized by the upper reference ($\mathcal{D}_{tr}^m$)

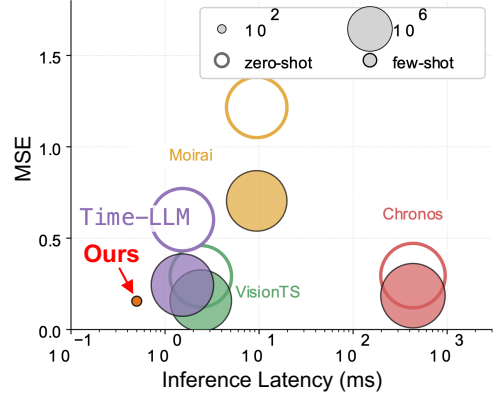


Figure 5: Comparing METACASTER with TSFMs on Solar dataset. Full results are in Appendix E.

(i.e., Eq. (4)). METACASTER produces more high-quality forecasters with lower variance. Moreover, it generalizes better to held-out forecasters than the baselines. Fig. 3 compares METACASTER with TimeScientist under $K \in 10, 30, 50$. Without data generation capability, TimeScientist struggles to train generalizable forecasters and its performance does not scale with K , indicating that a pure training pipeline is insufficient. Appendix E reports results when TimeScientist uses LT-LIB, yielding consistent conclusions.

Comparing with TSFMs. Fig. 5 compares MetaCaster with TSFMs on the Solar dataset under $K = 30$ in terms of performance vs. cost. After deployment, METACASTER trains and selects a lightweight forecaster for inference and incurs no further agent overhead – here, MixLinear (243 parameters) is selected at runtime. In contrast, TSFMs remain expensive from fine-tuning through inference. At comparable performance, METACASTER achieves up to $10^3 \times$ lower latency and $10^5 \times$ fewer parameters than TSFMs.

Ablation Study. Table 2 presents an ablation study using averaged MSE with $K = 30$, with METACASTER as our original model. In (a), we study the effect of forecasting-oriented optimization in Eq. (1) by replacing Eq. (1) with MMD and Wasserstein distances to directly align the generated set $\bar{\mathcal{D}}_{tr}^m$ with the authentic set $\mathcal{D}_{tr}^m$. In (b), we remove contextual cues C_m to evaluate their contribution. In (c), we examine the impact of different LLMs.

From Table 2(a), minimizing data distribution discrepancy generally degrades performance, as it is not directly aligned with the forecasting objective. Table 2(b) shows that contextual cues C_m are crucial, as they guide MGAGENT in selecting domain-relevant knowledge for generating time se-

Dataset ($\rightarrow$) Method ($\downarrow$)	ETTM1	Electricity	IND Corpus				Solar	OOD Corpus			Overall
			Seattle	SZTaxi	Sales	Bitbrains		Saugeen	USBirths	M4	
METACASTER	0.345	0.226	<u>1.177</u>	<u>0.114</u>	<u>2.362</u>	0.125	0.152	1.464	<u>0.533</u>	2.112	0.267
(a) Loss $\rightarrow$ MMD	0.708	0.157	1.204	0.150	3.581	0.575	0.254	2.339	0.403	2.556	0.764
(a) Loss $\rightarrow$ Wasserstein	0.702	0.284	1.734	0.152	3.553	0.483	0.254	2.406	0.535	3.105	0.940
(b) Remove context C_m	<u>0.386</u>	0.293	1.425	<u>0.144</u>	2.595	0.158	0.281	2.004	0.534	2.312	0.521
(c) LLM $\rightarrow$ Gemini-3.1-Pro	0.430	0.226	<u>1.177</u>	<u>0.114</u>	<u>2.362</u>	0.120	0.151	1.396	<u>0.533</u>	<u>2.129</u>	<u>0.288</u>
(c) LLM $\rightarrow$ Claude-Opus-4.7	0.494	0.226	1.189	<u>0.114</u>	<u>2.362</u>	0.151	<u>0.150</u>	1.491	0.541	<u>2.129</u>	0.321
(c) LLM $\rightarrow$ Qwen3.5-122B-A10B	0.494	0.223	<u>1.177</u>	<u>0.114</u>	<u>2.362</u>	0.199	0.152	1.729	<u>0.533</u>	<u>2.129</u>	0.366
(c) LLM $\rightarrow$ GPT-5.3-Codex	0.557	<u>0.209</u>	1.115	0.087	2.229	0.098	0.148	<u>1.465</u>	1.488	2.269	0.677

Table 2: Ablation analysis in terms of MSE. “Overall” assesses the normalized MSE (Eq. (4)) across datasets.

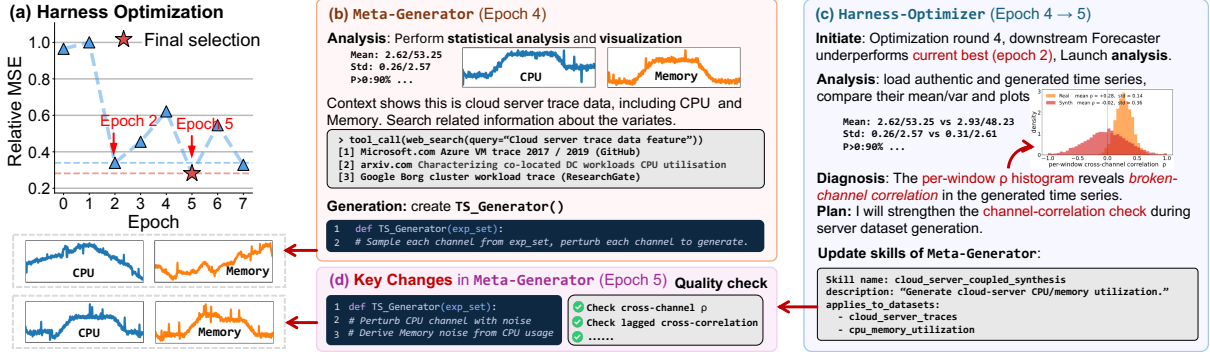


Figure 6: An illustration of Harness optimization by METACASTER: (a) hinge loss (Eq. (4)) during optimization iterations. (b)-(d) key traces of MGAGENT and HPAGENT from Epoch 4 to 5 on alibaba_cluster_2018 dataset.

ries. Interestingly, Table 2(c) shows that different LLMs yield comparable results in many cases, suggesting the Harness – rather than the LLM backbone – is the key factor in METACASTER, consistent with prior findings (Li et al., 2026a). Although GPT-5.3-Codex wins in some cases, it is unstable on datasets such as ETTm1 and USbirths, leading to worse overall results; thus, we adopt GPT-5.4 as the default LLM for its consistent performance.

Further Analysis. We have evaluated computational efficiency, token usage, robustness of performance (using standard deviation), performance change *w.r.t.* K , performance of top-ranked forecasters, visualization of generated time series and data distribution, which are in Appendix C.3, F.

4.3 Case Study

Fig. 6(a) shows the evolution of hinge loss (Eq. (4)) over 8 Harness optimization epochs, where METACASTER converges quickly and selects the final Harness from epoch 5. Fig 6(b)-(d) illustrates MGAGENT and HPAGENT traces from epoch 4 to 5 on the alibaba_cluster_2018 dataset in C_{har} . MGAGENT (Fig 6(b)) analyzes the few-shot examples D_{sup}^m , inspects statistics, retrieves domain knowledge about the two variates CPU and memory, and constructs a TS_Generator. HPAGENT (Fig. 6(c)) detects performance degradation at epoch 4 via

memory, then performs Analysis and Diagnosis. The Analysis uses analytical tools pertinent to the issue identified in the reasoning traces of MGAGENT and the training logs of FTAGENT. The Diagnosis identifies broken inter-variate correlation as the root cause. It then updates MGAGENT’s skills, leading to an improved TS_Generator that strengthens the CPU-memory correlation and an addition of consistency checks. Consequently, epoch 5 produces correlated time series that are more compliant with the few-shot examples than epoch 4. This example illustrates a simple Harness update. In fact, METACASTER’s optimization is more complex and it deals with $B = 8$ datasets in C_{har} concurrently. The final output is a trained lightweight forecaster rather than the generated time series.

5 Conclusion

In this work, we study few-shot learning for lightweight forecasters via a novel meta-harness-optimized multi-agent, METACASTER, which automates an end-to-end pipeline for time series generation, forecaster training, parameter tuning, and model selection. Experiments not only validate the effectiveness of METACASTER, but also set a groundwork for delving into the Agent-as-Engineer paradigm in agentic time series forecasting.

Limitations

In this work, we study the challenging yet practical problem of improving lightweight time series forecasters in a few-shot setting for rapid deployment, avoiding delays from costly data acquisition and privacy constraints. This is enabled by leveraging the knowledge and reasoning capabilities of AI agent systems.

However, our current work does not address the extreme zero-shot setting, where no reference examples are available. Without reference time series, the agent lacks basic statistical grounding of the target domain, leading to unreliable data generation. This is a common limitation of time series generation-based models (Jeon et al., 2022; Yuan and Qiao, 2024; Naiman et al., 2024; Huang et al., 2025b; Gu et al., 2025; Ge et al., 2025) and highlights the advantage of TSFMs, which can transfer pre-trained knowledge in zero-shot settings. In contrast, by leveraging only a small number of examples – which is often feasible in practice – METACASTER achieves competitive performance with TSFMs while maintaining significantly lower computational cost.

Additionally, our experiments cover 18 time series datasets spanning a limited set of domains. We plan to extend this to the full pre-training corpora of modern TSFMs to further improve Harness optimization. Meanwhile, LT-LIB currently includes a set of SOTA lightweight forecasters we have collected so far. It may not be exhaustive as new lightweight forecasters continue to emerge. In future work, we will continuously update LT-LIB to strengthen the METACASTER system.

Acknowledgments

This work was partially supported by a research gift from NEC Laboratories America and by the NVIDIA Academic Grant Program.

References

- Taha Aksu, Gerald Woo, Juncheng Liu, Xu Liu, Chenghao Liu, Silvio Savarese, Caiming Xiong, and Doyen Sahoo. 2024. Gift-eval: A benchmark for general time series forecasting model evaluation. *arXiv preprint arXiv:2410.10393*.
- Alibaba Group. 2018. Alibaba cluster trace v2018. <https://github.com/alibaba/clusterdata>.
- Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. 2024. Chronos: Learning the language of time series. *TMLR*.
- Alberto Ardid, David Dempsey, Corentin Caudron, Shane Cronin, Ben Kennedy, Társilo Girona, Diana Roman, Craig Miller, Sally Potter, Oliver D Lamb, and 1 others. 2025. Ergodic seismic precursors and transfer learning for short term eruption forecasting at data scarce volcanoes. *Nat. Commun.*
- Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small language models are the future of agentic ai. *arXiv preprint arXiv:2506.02153*.
- Verónica Bolón-Canedo, Laura Morán-Fernández, Brais Cancela, and Amparo Alonso-Betanzos. 2024. A review of green artificial intelligence: Towards a more sustainable future. *Neurocomputing*.
- Centers for Disease Control and Prevention. Flu-View: Outpatient illness surveillance (ILINet). <https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html>.
- Centers for Disease Control and Prevention, NCHS. National vital statistics system: Births data. <https://www.cdc.gov/nchs/nvss/births.htm>.
- Mouxian Chen, Lefei Shen, Zhuo Li, Xiaoyun Wang, Jianling Sun, and Chenghao Liu. 2024. Visions: Visual masked autoencoders are free-lunch zero-shot time series forecasters. *arXiv preprint arXiv:2408.17253*.
- Si-An Chen, Chun-Liang Li, Nate Yoder, Sercan O. Arik, and Tomas Pfister. 2023. Tsmixer: An all-mlp architecture for time series forecasting. *TMLR*.
- Sarkar Snigdha Sarathi Das, Palash Goyal, Mihir Parmar, Nanyun Peng, Vishy Tirumalashetty, Chun-Liang Li, Rui Zhang, Jinsung Yoon, and Tomas Pfister. 2026. Nexus: An agentic framework for time series forecasting. *arXiv preprint arXiv:2605.14389*.
- Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. 2021. Timevae: A variational auto-encoder for multivariate time series generation. *arXiv preprint arXiv:2111.08095*.
- Vijay Ekambaram, Arindam Jati, Nam Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. TSMixer: Lightweight MLP-mixer model for multivariate time series forecasting. In *KDD*.
- Jingru Fei, Kun Yi, Wei Fan, Qi Zhang, and Zhen-dong Niu. 2025. Amplifier: Bringing attention to neglected low-energy components in time series forecasting. In *AAAI*.

- Azul Garza and Renée Rosillo Garcia. 2025. Timecopilot. In *NeurIPS Workshop on Recent Advances in Time Series Foundation Models*.
- Yunfeng Ge, Jiawei Li, Yiji Zhao, Haomin Wen, Zhao Li, Meikang Qiu, Hongyan Li, Ming Jin, and Shirui Pan. 2025. T2s: High-resolution time series generation with text-to-series diffusion models. In *IJCAI*.
- Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I Webb, Rob J Hyndman, and Pablo Montero-Manso. 2021. Monash time series forecasting archive. In *NeurIPS Datasets and Benchmarks*.
- Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew Gordon Wilson. 2023. Large language models are zero-shot time series forecasters. In *NeurIPS*.
- Shuqi Gu, Chuyue Li, Baoyu Jing, and Kan Ren. 2025. Verbalts: Generating time series from texts. In *ICML*.
- Tong Guan, Zijie Meng, Dianqi Li, Shiyu Wang, Chao-Han Huck Yang, Qingsong Wen, Zuozhu Liu, Sabato Marco Siniscalchi, Ming Jin, and Shirui Pan. 2026a. Timeomni-1: Incentivizing complex reasoning with time series in large language models. In *ICLR*.
- Tong Guan, Sheng Pan, Johan Barthelemy, Zhao Li, Yujun Cai, Cesare Alippi, Ming Jin, and Shirui Pan. 2026b. Timeomni-v1: Unified models for time series understanding and generation. *arXiv preprint arXiv:2602.17149*.
- Keith W Hipel and A Ian McLeod. 1994. *Time Series Modelling of Water Resources and Environmental Systems*. Elsevier.
- Qihe Huang, Zhengyang Zhou, Kuo Yang, Zhongchao Yi, Xu Wang, and Yang Wang. 2025a. TimeBase: The power of minimalism in efficient long-term time series forecasting. In *ICML*.
- Yu-Hao Huang, Chang Xu, Yueying Wu, Wu-Jun Li, and Jiang Bian. 2025b. Timedp: Learning to generate multi-domain time series with domain prompts. In *AAAI*.
- Brian Kenji Iwana and Seiichi Uchida. 2021. An empirical survey of data augmentation for time series classification with neural networks. *PLOS ONE*.
- Gunjan Jalori, Preetika Verma, and Serkan Ö. Arık. 2025. Flairr-ts: Forecasting llm-agents with iterative refinement and retrieval for time series. In *Findings of EMNLP*.
- Jinsung Jeon, Jeonghak Kim, Haryong Song, Seunghyeon Cho, and Noseong Park. 2022. Gt-gan: General purpose time series synthesis with generative adversarial networks. In *NeurIPS*.
- Yushan Jiang, Kanghui Ning, Zijie Pan, Xuyang Shen, Jingchao Ni, Wenchao Yu, Anderson Schneider, Haifeng Chen, Yuriy Nevmyvaka, and Dongjin Song. 2025. Multi-modal time series analysis: A tutorial and survey. In *KDD*.
- Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y. Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, and Qingsong Wen. 2024. Time-llm: Time series forecasting by reprogramming large language models. In *ICLR*.
- Yaxuan Kong, Yiyuan Yang, Yoontae Hwang, Wenjie Du, Stefan Zohren, Zhangyang Wang, Ming Jin, and Qingsong Wen. 2025. Time-mqa: Time series multi-task question answering with context enhancement. In *ACL*.
- Irena Koprinska, Dongsong Wu, and Zheng Wang. 2018. Convolutional neural networks for energy time series forecasting. In *IJCNN*.
- Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*.
- Junjie Li, Xi Xiao, Yunbei Zhang, Chen Liu, Lin Zhao, Xiaoying Liao, Yingrui Ji, Janet Wang, Jianyang Gu, Yingqiang Ge, Weijie Xu, Xi Fang, Xiang Xu, Tianchen Zhao, Youngeun Kim, Tianyang Wang, Jihun Hamm, Smita Krishnaswamy, Jun Huan, and Chandan Reddy. 2026a. Agent harness engineering: A survey. <https://openreview.net/pdf?id=eONq7FdiHa>.
- Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, and 1 others. 2026b. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*.
- Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *ICLR*.
- Zhe Li, Shiyi Qi, Yiduo Li, and Zenglin Xu. 2023. Revisiting long-term time series forecasting: An investigation on linear mapping. *arXiv preprint arXiv:2305.10721*.
- Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, and Tao Gui. 2026. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*.
- Shengsheng Lin, Weiwei Lin, Xinyi Hu, Wentai Wu, Ruichao Mo, and Haocheng Zhong. 2024a. Cyclenet: Enhancing time series forecasting through modeling periodic patterns. In *NeurIPS*.
- Shengsheng Lin, Weiwei Lin, Wentai Wu, Haojun Chen, and Junjie Yang. 2024b. Sparsetsf: Modeling long-term time series forecasting with 1k parameters. In *ICML*.
- Peiyuan Liu, Hang Guo, Tao Dai, Naiqi Li, Jigang Bao, Xudong Ren, Yong Jiang, and Shu-Tao Xia. 2025a. Calf: Aligning llms for time series forecasting via cross-modal fine-tuning. In *AAAI*.

- Peiyuan Liu, Beiliang Wu, Yifan Hu, Naiqi Li, Tao Dai, Jigang Bao, and Shu-Tao Xia. 2025b. TimeBridge: Non-stationarity matters for long-term time series forecasting. In *ICML*.
- Yong Liu, Guo Qin, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. 2024. Autotimes: Autoregressive time series forecasters via large language models. In *NeurIPS*.
- Yong Liu, Guo Qin, Zhiyuan Shi, Zhi Chen, Caiyin Yang, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. 2025c. Sundial: A family of highly capable time series foundation models. In *ICML*.
- Dongsheng Luo, Wei Cheng, Yingheng Wang, Dongkuan Xu, Jingchao Ni, Wenchao Yu, Xuchao Zhang, Yanchi Liu, Yuncong Chen, Haifeng Chen, and 1 others. 2023. Time series contrastive learning with information-aware augmentations. In *AAAI*.
- Aitian Ma, Dongsheng Luo, and Mo Sha. 2024. MixLinear: Extreme low resource multivariate time series forecasting with 0.1K parameters. *arXiv preprint arXiv:2410.02081*.
- Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2020. The M4 competition: 100,000 time series and 61 forecasting methods. *Int. J. Forecast.*
- Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2022. The M5 competition: Background, organization, and implementation. *Int. J. Forecast.*
- Mike A Merrill, Mingtian Tan, Vinayak Gupta, Thomas Hartvigsen, and Tim Althoff. 2024. Language models still struggle to zero-shot reason about time series. In *Findings of EMNLP*.
- Mohammad Amin Morid, Olivia R Liu Sheng, and Joseph Dunbar. 2023. Time series prediction using deep learning methods in healthcare. *ACM TMIS*.
- Ilan Naiman, Nimrod Berman, Itai Pemper, Idan Arbiv, Gal Fadlon, and Omri Azencot. 2024. Utilizing image transforms and diffusion models for generative modeling of short and long time series. In *NeurIPS*.
- National Renewable Energy Laboratory. 2006. Solar power data for integration studies. <https://www.nrel.gov/grid/solar-power-data.html>.
- Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A time series is worth 64 words: Long-term forecasting with transformers. In *ICLR*.
- Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, and 1 others. 2026. Code as agent harness. *arXiv preprint arXiv:2605.18747*.
- Zijie Pan, Yushan Jiang, Sahil Garg, Anderson Schneider, Yuriy Nevmyvaka, and Dongjin Song. 2024. S2ip-llm: Semantic space informed prompt learning with llm for time series forecasting. In *ICML*.
- ChengAo Shen, Wenchao Yu, Ziming Zhao, Dongjin Song, Wei Cheng, Haifeng Chen, and Jingchao Ni. 2025a. Multi-modal view enhanced large vision models for long-term time series forecasting. In *NeurIPS*.
- ChengAo Shen, Ziming Zhao, Hanghang Tong, Dongjin Song, Dongsheng Luo, Qingsong Wen, and Jingchao Ni. 2025b. Svtime: Small time series forecasting models informed by "physics" of large vision model forecasters. *arXiv preprint arXiv:2510.09780*.
- Siqi Shen, Vincent van Beek, and Alexandru Iosup. 2015. Statistical characterization of business-critical workloads hosted in cloud datacenters. In *CCGrid*.
- Haotian Si, Changhua Pei, Jianhui Li, Dan Pei, and Gaogang Xie. 2025. CMoS: Rethinking time series prediction through the lens of chunk-wise spatial correlations. In *ICML*.
- Artyom Stitsyuk and Jaesik Choi. 2025. xPatch: Dual-stream time series forecasting with exponential seasonal-trend decomposition. In *AAAI*.
- Mingtian Tan, Mike A Merrill, Vinayak Gupta, Tim Althoff, and Thomas Hartvigsen. 2024. Are language models actually useful for time series forecasting? *NeurIPS*.
- Peiwan Tang and Weitai Zhang. 2025. Unlocking the power of patch: Patch-based MLP for long-term time series forecasting. In *AAAI*.
- Artur Trindade. 2015. Electricity-LoadDiagrams20112014. <https://archive.ics.uci.edu/dataset/321/electricityloadaddiagrams20112014>.
- University of Washington STAR Lab. Seattle inductive loop detector dataset. <https://github.com/zhilyongc/Seattle-Loop-Data>.
- Chengsen Wang, Qi Qi, Jingyu Wang, Haifeng Sun, Zirui Zhuang, Jinming Wu, Lei Zhang, and Jianxin Liao. 2025. Chattime: A unified multimodal time series foundation model bridging numerical and textual data. In *AAAI*.
- Shiyu Wang, Haixu Wu, Xiaoming Shi, Tengge Hu, Huakun Luo, Lintao Ma, James Y. Zhang, and Jun Zhou. 2024. Timemixer: Decomposable multiscale mixing for time series forecasting. In *ICLR*.
- Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. 2021. Time series data augmentation for deep learning: A survey. In *IJCAI*.
- Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. 2024. Unified training of universal time series forecasting transformers. In *ICML*.
- Xingjian Wu, Junkai Lu, Zhengyu Li, Xiangfei Qiu, Jilin Hu, Chenjuan Guo, Christian S. Jensen, and Bin Yang. 2026. Timeart: Towards agentic time series reasoning via tool-augmentation. *arXiv preprint arXiv:2601.13653*.

- Mingyuan Xia, Chunxu Zhang, Zijian Zhang, Hao Miao, Qidong Liu, Yuanshao Zhu, and Bo Yang. 2025. TimeEmb: A lightweight static-dynamic disentanglement framework for time series forecasting. *arXiv preprint arXiv:2510.00461*.
- Zhijian Xu, Ailing Zeng, and Qiang Xu. 2024. Fits: Modeling time series with 10k parameters. In *ICLR*.
- Hao Xue and Flora D Salim. 2023. Promptcast: A new prompt-based learning paradigm for time series forecasting. *IEEE TKDE*.
- Kun Yi, Jingru Fei, Qi Zhang, Hui He, Shufeng Hao, Defu Lian, and Wei Fan. 2024. FilterNet: Harnessing frequency filters for time series forecasting. In *NeurIPS*.
- Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. 2019. Time-series generative adversarial networks. In *NeurIPS*.
- Fangxu Yu, Hongyu Zhao, and Tianyi Zhou. 2025. Ts-reasoner: Aligning time series foundation models with llm reasoning. *arXiv preprint arXiv:2510.03519*.
- Xinyu Yuan and Yan Qiao. 2024. Diffusion-ts: Interpretable diffusion for general time series generation. In *ICLR*.
- Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. 2022. Ts2vec: Towards universal representation of time series. In *AAAI*.
- Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by back-propagating language model feedback. *Nature*.
- Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are transformers effective for time series forecasting? In *AAAI*.
- Boya Zhang, Shuaijie Yin, Huiwen Zhu, and Xing He. 2026. FreqCycle: A multi-scale time-frequency analysis method for time series forecasting. *arXiv preprint arXiv:2603.09661*.
- Tianping Zhang, Yizhuo Zhang, Wei Cao, Jiang Bian, Xiaohan Yi, Shun Zheng, and Jian Li. 2022. Less is more: Fast multivariate time series forecasting with light sampling-oriented mlp structures. *arXiv preprint arXiv:2207.01186*.
- Xiaohan Zhang, Tian Gao, Mingyue Cheng, Bokai Pan, Ze Guo, Yaguo Liu, Xiaoyu Tao, and Qi Liu. 2025. Alphacast: A human wisdom-llm intelligence co-reasoning framework for interactive time series forecasting. *arXiv preprint arXiv:2511.08947*.
- Haokun Zhao, Xiang Zhang, Jiaqi Wei, Yiwei Xu, Yuting He, Siqi Sun, and Chenyu You. 2025. Timeseries-scientist: A general-purpose ai agent for time series analysis. *arXiv preprint arXiv:2510.01538*.
- Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2020. T-GCN: A temporal graph convolutional network for traffic prediction. *IEEE TITS*.
- Siru Zhong, Weilin Ruan, Ming Jin, Huan Li, Qingsong Wen, and Yuxuan Liang. 2025. Time-vlm: Exploring multimodal vision-language models for augmented time series forecasting. In *ICML*.
- Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI*.
- Pengfei Zhou, Yunlong Liu, Junli Liang, Qi Song, and Xiangyang Li. 2025. CrossLinear: Plug-and-play cross-correlation embedding for time series forecasting with exogenous variables. In *KDD*.
- Tian Zhou, Peisong Niu, Xue Wang, Liang Sun, and Rong Jin. 2023. One fits all: Power general time series analysis by pretrained lm. In *NeurIPS*.

A Algorithm

The meta-harness optimization algorithm of METACASTER is summarized in Algorithm 1. The notations are consistent with §3.

Algorithm 1: Meta-harness optimization of METACASTER

Input: (1) Corprurs $\mathcal{C}_{\text{har}} = \{\mathcal{D}^m, \mathcal{C}_m\}_{m=1}^M$; (2) LT-LIB $\mathcal{F} = \{f_1, \dots, f_L\}$; (3) number of epochs E ; (4) batch size B

Output: Optimized harness θ^*

```

1  $\ell^* \leftarrow +\infty$  // Initialize loss function value
2  $\theta \leftarrow \theta^0$  // Initialize harness
3 for  $i = 1, \dots, E$  do
    /* Optimization loop */
4      $K \sim \text{Uniform}([10, 50])$ 
5     Draw a batch  $\mathcal{C}_b = \{\mathcal{D}^m, \mathcal{C}_m\}_{m=1}^B$ 
    /* Forward passes across datasets in a batch */
6     for  $\{\mathcal{D}^m, \mathcal{C}_m\} \in \mathcal{C}_b$  do
7          $\{\mathcal{D}_{\text{tr}}^m, \mathcal{D}_{\text{val}}^m, \mathcal{D}_{\text{te}}^m\} \leftarrow \text{Split}(\mathcal{D}^m)$  // train/validation/test split
8          $\mathcal{D}_{\text{sup}}^m \leftarrow K \text{ samples from } \mathcal{D}_{\text{tr}}^m$  // K-shot support set
9         /* Run MGagent (Eq. (2)) */
10         $\bar{\mathcal{D}}^m \leftarrow \text{MGAGENT}_{\theta}(\mathcal{D}_{\text{sup}}^m, \mathcal{C}_m)$ 
11         $\{\bar{\mathcal{D}}_{\text{tr}}^m, \bar{\mathcal{D}}_{\text{val}}^m\} \leftarrow \text{Split}(\bar{\mathcal{D}}^m)$ 
12        /* Run FTagent (Eq. (3)) */
13         $\{\{f_l^m, \bar{f}_l^m, \omega(f_l^m), \omega(\bar{f}_l^m)\}_{l=1}^L, \mathbf{R}\} \leftarrow \text{FTAGENT}(\{\mathcal{D}_{\text{tr}}^m, \mathcal{D}_{\text{val}}^m, \mathcal{D}_{\text{te}}^m\}, \{\bar{\mathcal{D}}_{\text{tr}}^m, \bar{\mathcal{D}}_{\text{val}}^m\})$ 
14    end
15     $\ell \leftarrow \frac{1}{BL} \sum_{m=1}^B \sum_{l=1}^L \delta(\omega(f_l^m), \omega(\bar{f}_l^m))$  // The optimization loss Eq. (1)
16    /* Run HPagent */
17     $\theta \leftarrow \text{HPAGENT}(\theta, \ell, \ell^*, \mathbf{R})$ 
18    if  $\ell < \ell^*$  then
19         $\ell^* \leftarrow \ell$ 
20         $\theta^* \leftarrow \theta$ 
21    end
22 end

```

B Datasets and Baselines

B.1 Datasets

Pretraining-style train/test separation. Both the training and evaluation datasets are drawn from GIFT-Eval (Aksu et al., 2024), which is released under the Apache 2.0 license and permits research use. To prevent dataset leakage, we follow the data-isolation paradigm adopted by foundation-model pre-training and draw them from two disjoint GIFT-Eval collections, so that no source series ever appears in both phases.

Addressing Data Leakage. Three successive layers guard against test-source contamination. First, the training and evaluation datasets are drawn from two disjoint GIFT-Eval collections, Salesforce/GiftEvalPretrain for training and Salesforce/GIFT_Eval for evaluation, so no source series ever appears in both phases. Second, the context $\mathcal{C}_m$ delivered to MGAGENT is stripped of dataset identifiers, source URLs, and benchmark names before reaching the agent, leaving only the domain label and a short semantic description, so the agent cannot use a dataset name as a hook to retrieve or imitate the

true test data from any pretraining corpus it may have memorised. Third, we manually audit the runtime traces of MGAGENT on held-out runs and confirm that no external web request is issued and no real test data is touched; synthesis is driven only by the few-shot support set $\mathcal{D}_{\text{sup}}^m$ and the de-identified context.

Training collection. We take 8 datasets covering 6 domain categories (energy, traffic, weather, health, retail, cloud) from Salesforce/GiftEvalPretrain (Aksu et al., 2024); per-dataset frequency, channel count, length, domain label, and source are listed in Table 3. Two of them (cdc_fluview_ilinet, alibaba_cluster_trace_2018) are multivariate ($D = 5$ and $D = 2$); the remaining six are sampled from populations of parallel univariate series ($D = 1$).

Dataset	Freq.	D	N	T	Domain	Source
australian_electricity_demand	30min	1	5	231k	energy	(Godaheva et al., 2021)
solar_power	4s	1	1	7.4M	energy	(Godaheva et al., 2021)
PEMS_BAY	5min	1	325	52k	traffic	(Li et al., 2018)
traffic_hourly	1h	1	862	17.5k	traffic	(Godaheva et al., 2021)
weather	1d	1	3,010	var	weather	(Godaheva et al., 2021)
cdc_fluview_ilinet	1w	5	5	1.2k	health	(Centers for Disease Control and Prevention)
m5	1d	1	30,490	1.9k	retail	(Makridakis et al., 2022)
alibaba_cluster_trace_2018	1min	2	58k	2k+	cloud	(Alibaba Group, 2018)

Table 3: Overview of Training Dataset.

Test collection. We take 10 datasets from Salesforce/GIFT_Eval (Aksu et al., 2024), split into 7 IND (big-domain label appears in the training collection) and 3 OOD (entirely absent from training); per-dataset frequency, channel count, length, domain label, and source are listed in Table 4. Among the OOD set, M4 is a mixed-domain daily subset; we include it to probe whether the model still works when domain information is ambiguous or absent.

	Dataset	Freq.	D	N	T	Domain	Source
IND	ETTM1	15min	7	1	50k	energy	(Zhou et al., 2021)
	Electricity	1h	1	370	35.1k	energy	(Trindade, 2015)
	Seattle	1h	1	323	8.8k	traffic	(University of Washington STAR Lab)
	SZTaxi	15min	1	156	3.0k	traffic	(Zhao et al., 2020)
	Sales	1d	1	118	1.8k	retail	(Godaheva et al., 2021)
	Bitbrains	5min	2	1,250	8.6k	cloud	(Shen et al., 2015)
	Solar	1h	1	137	8.8k	energy	(National Renewable Energy Laboratory, 2006)
OOD	Saugeen	1d	1	1	23.7k	hydrology	(Hipel and McLeod, 1994)
	USbirths	1d	1	1	7.3k	demographics	(Centers for Disease Control and Prevention, NCHS)
	M4	1d	1	4,227	9.9k	mixed	(Makridakis et al., 2020)

Table 4: Overview of Test Dataset.

Coverage of the difficulty spectrum. The 18 datasets jointly span the axes that govern few-shot forecasting difficulty on GIFT-Eval. Sampling frequency ranges from 4-second photovoltaic telemetry on solar_power to weekly health indicators on cdc_fluview_ilinet, covering six orders of magnitude. The number of variates D takes values in $\{1, 2, 5, 7\}$, exposing both univariate and small-multichannel regimes. Per-series length ranges from 1.2k steps on the shortest health series to 7.4M steps on the densest telemetry feed, a five-order-of-magnitude span that subjects the synthesis pipeline to highly heterogeneous regimes within a single benchmark.

Forecasting protocol on each dataset. On every dataset the raw series is split chronologically into train / validation / test partitions at an 80/10/10 ratio, z-score-normalised per channel using statistics estimated on the train partition, and sliced into $(\mathbf{X}, \mathbf{Y})$ pairs with a look-back of $T = 336$ steps and a horizon of $H = 192$ steps; this contract is shared by every method evaluated in this paper. The few-shot support set $\mathcal{D}_{\text{sup}}^m$ fed to METACASTER and to every external baseline is obtained by drawing K windows from the train partition of dataset m , with $K \in \{10, 30, 50\}$ in the main results (Table 1) and $K \in \{10, 20, 30, 50, 100\}$

in the few-shot scaling study (Appendix F). Each dataset is paired with a textual context C_m summarising its domain and semantic cues, bundled with the support set and passed to any method that consumes textual conditioning.

B.2 Baselines

This subsection consolidates every baseline reported in the main table (Table 1), grouped into deep generative synthesisers and classical augmentation procedures, together with two reference levels that delineate the dynamic range of each row. The reference levels are $\mathcal{D}_{\text{sup}}^m$, which trains each forecaster directly on the few-shot support set with no augmentation and acts as the no-synthesis floor, and $\mathcal{D}_{\text{tr}}^m$, which trains on the full historical training split and acts as the data-rich oracle that any few-shot method aspires to match. For completeness we additionally describe the foundation forecasters compared in §4.2 and the closest agent-driven reference TimeScientist (Fig. 3), neither of which fits the few-shot synthesis cell of the main table but both of which appear in the rest of the experimental study.

Deep generative baselines.

- TimeDP (Huang et al., 2025b): a diffusion model conditioned on domain prompts assembled from learned time-series prototype vectors, whose weights are inferred from few-shot samples of the target domain.
- VerbalTS (Gu et al., 2025): a diffusion model that maps unstructured textual descriptions to time series through a multi-focal alignment module bridging text tokens and temporal latents.
- T2S (Ge et al., 2025): a text-conditioned Diffusion Transformer trained with Flow Matching over a length-adaptive VAE latent space, enabling variable-length text-to-series synthesis.
- Diffusion-TS (Yuan and Qiao, 2024): a denoising diffusion model with an encoder-decoder Transformer that decomposes samples into trend and seasonal components and predicts the clean signal under a Fourier-domain loss.
- TimeVAE (Desai et al., 2021): a variational autoencoder whose decoder composes outputs from interpretable level, trend, and seasonality blocks sampled from a Gaussian latent prior.

Classical augmentation baselines.

- Repeat: cycles deterministically through the few-shot pool until the target training-set size is reached.
- Bootstrap: resamples windows with replacement and adds small zero-mean Gaussian noise scaled per channel.
- Jitter (Iwana and Uchida, 2021): adds independent zero-mean Gaussian noise to every time step and channel of each resampled window.
- MagWarp (Iwana and Uchida, 2021; Wen et al., 2021): multiplies each channel by a smooth scaling curve obtained by cubic-spline interpolation through random knots.

Foundation forecasters.

- VisionTS (Chen et al., 2024): recasts forecasting as image inpainting by arranging the look-back into a 2D grid and decoding the masked horizon with a frozen ImageNet-pretrained MAE.
- Chronos (Ansari et al., 2024): quantises scaled time-series values into a fixed vocabulary and trains a T5 encoder-decoder on tokenised sequences for autoregressive probabilistic forecasting.
- Moirai (Woo et al., 2024): a masked encoder Transformer with any-variate attention and multi-patch projections, pretrained on the LOTSA corpus to handle arbitrary frequencies and channel counts.
- Time-LLM (Jin et al., 2024): freezes a pretrained GPT-2 backbone and reprograms patch embeddings into text-prototype tokens, prepending dataset descriptions as a prompt prefix.

TimeScientist. TimeScientist (Zhao et al., 2025) is a multi-agent pipeline driven by a local Qwen2.5 backbone that diagnoses each series and ensembles classical forecasters (ARIMA, ETS, Random Forest) per query. We separate it from the foundation forecasters above because it does not ship a single pretrained weight matrix; instead, the LLM agent is invoked at every test window to compose and tune the classical predictors, making it the closest published reference to METACASTER’s agent-driven paradigm.

C Implementation Details

C.1 Forecaster Pool

Twenty-three lightweight forecasters across four families:

Family	Forecaster	Params	MACs (M)	Latency (ms)	Peak VRAM (MB)	Ref.
Linear	Vanilla Linear	65K	0.45	0.03	8.4	(Zeng et al., 2023)
	DLinear	129K	0.90	0.10	8.7	(Zeng et al., 2023)
	NLinear	65K	0.45	0.04	8.4	(Zeng et al., 2023)
	RLinear	65K	0.45	0.17	8.4	(Li et al., 2023)
	CrossLinear	2.5M	28.45	0.72	23.1	(Zhou et al., 2025)
	MixLinear	243	0.08	0.43	8.3	(Ma et al., 2024)
MLP	TSMixer	153K	1.66	0.42	10.1	(Chen et al., 2023)
	LightTS	74K	0.70	0.75	10.1	(Zhang et al., 2022)
	PatchMLP	2.5M	18.12	1.21	20.1	(Tang and Zhang, 2025)
	xPatch	770K	8.16	1.44	13.6	(Stitsyuk and Choi, 2025)
	CMoS	13K	0.36	1.09	9.3	(Si et al., 2025)
	PatchTSMixer	553K	20.83	1.12	14.4	(Ekambaram et al., 2023)
Freq/Filter	FITS	925	0.01	0.32	8.3	(Xu et al., 2024)
	CycleNet	65K	0.45	0.32	8.5	(Lin et al., 2024a)
	PaiFilter	136K	0.95	0.35	9.8	(Yi et al., 2024)
	TexFilter	179K	0.94	1.62	9.9	(Yi et al., 2024)
	FreqCycle	64K	0.43	0.78	9.5	(Zhang et al., 2026)
Mixing	TimeMixer	377K	50.08	0.80	65.0	(Wang et al., 2024)
	TimeBase	146	0.02	0.29	9.2	(Huang et al., 2025a)
	TimeBridge	36K	3.01	2.09	10.4	(Liu et al., 2025b)
	TimeEmb	308K	1.89	0.47	10.5	(Xia et al., 2025)
	Amplifier	327K	1.06	0.73	10.6	(Fei et al., 2025)
	SparseTSF	137	0.08	0.20	8.2	(Lin et al., 2024b)

Table 5: The 23 forecasters in LT-LIB, grouped by family.

Per-architecture descriptions. For completeness we list the core architectural idea of each forecaster in LT-LIB, grouped by family; bibliographic references match Table 5. All 23 forecasters are wrapped by a uniform training interface in LT-LIB so that FTAGENT can launch any of them via the same (forecaster, dataset, hyperparameter) call signature.

Linear family.

- Vanilla Linear (Zeng et al., 2023): a single fully-connected temporal layer maps each variate’s look-back window directly to its horizon via a weighted sum, with weights shared across variates.
- DLinear (Zeng et al., 2023): series decomposition splits the input into trend and seasonal components, each forecast by a separate linear layer whose outputs are summed.
- NLinear (Zeng et al., 2023): the last look-back value is subtracted before a linear projection and added back after, providing simple distribution-shift compensation.
- RLinear (Li et al., 2023): a single linear projection wrapped by reversible instance normalisation (RevIN), with weights shared across variates.
- CrossLinear (Zhou et al., 2025): a plug-and-play cross-correlation embedding is fused with patch embeddings of the endogenous variable before a global linear forecasting head produces the horizon under parameter-free RevIN.

- MixLinear (Ma et al., 2024): segment-wise intra/inter linear mixing in the time domain is fused with low-pass-filtered complex-valued linear compression and reconstruction in the frequency domain, reducing parameters from $O(n^2)$ to $O(n)$ on the downsampled length.

MLP family.

- TSMixer (Chen et al., 2023): stacks interleaved time-mixing and feature-mixing MLP blocks that alternately mix along the temporal axis and across covariates.
- LightTS (Zhang et al., 2022): applies MLP blocks on top of two downsampled views of the input (interval and continuous sampling) before recombining them.
- PatchMLP (Tang and Zhang, 2025): embeds channel-independent multi-scale patches, splits them via moving-average decomposition, and routes smooth and residual components through intra- and inter-variable MLPs.
- xPatch (Stitsyuk and Choi, 2025): decomposes the patched input via exponential moving average into trend and seasonal components processed by parallel MLP-linear and CNN-nonlinear streams.
- CMoS (Si et al., 2025): replaces shape embeddings with a Correlation Mixing layer that uses parameter-shared basis matrices combined via channel-specific weights to model relative positional correlations between input and output chunks.
- PatchTSMixer (Ekambaram et al., 2023): adapts the MLP-Mixer to patched series with successive inter-patch, intra-patch, and inter-channel mixing blocks, each augmented with a gated-attention module.

Frequency / filter family.

- FITS (Xu et al., 2024): applies rFFT, low-pass truncation, and a single complex-valued linear layer for amplitude-and-phase interpolation in the frequency domain before inverse FFT.
- CycleNet (Lin et al., 2024a): subtracts a per-channel learnable recurrent cycle of user-specified length, then forecasts the residual with a linear or two-layer MLP head.
- PaiFilter (Yi et al., 2024): multiplies the input spectrum by a single universal learnable frequency kernel shared across all sequences, acting as a fixed shaping filter.
- TexFilter (Yi et al., 2024): generates the frequency-domain filter conditionally from each input's spectrum, adapting the shaping kernel per sample for context-dependent dependency learning.
- FreqCycle (Zhang et al., 2026): couples a filter-enhanced cyclic component for low-frequency periodicity with a segmented frequency-domain branch that reweights mid- and high-frequency bands.

Mixing family.

- TimeMixer (Wang et al., 2024): decomposes series at multiple sampling scales and mixes seasonal components fine-to-coarse and trend components coarse-to-fine before ensembling scale-specific predictors.
- TimeBase (Huang et al., 2025a): a sub-0.4k-parameter network that extracts orthogonal full-rank typical-period bases and reformulates point-level forecasting as segment-level prediction over learned cycles.
- TimeBridge (Liu et al., 2025b): a patch-based Transformer that applies Integrated Attention within each variate's patches to mitigate short-term non-stationarity and Cointegrated Attention across variates to model long-term cointegration.
- TimeEmb (Xia et al., 2025): disentangles series into a time-invariant component handled by a learnable global embedding bank and a time-varying component processed via frequency-domain filtering.

- Amplifier (Fei et al., 2025): amplifies low-energy frequency components before seasonal-trend modelling and restores original energy afterward, paired with a semi-channel interaction block for cross-channel dependencies.
- SparseTSF (Lin et al., 2024b): a sub-1k-parameter design that downsamples the input by its period into subsequences and predicts each via cross-period sparse linear projection, reducing forecasting to cross-period trend extrapolation.

C.2 Training Environment

Hardware. All experiments reported in this paper are executed on a single node equipped with 4×NVIDIA RTX 6000 Ada GPUs, each with 48 GB of memory.

Software stack. All code runs under Python 3.12.8 with PyTorch 2.5.1, CUDA 12.4, and cuDNN 9.1. Auxiliary libraries include NumPy 2.1.3, Pandas 2.2.3, SciPy 1.15.1, scikit-learn 1.6.1, Matplotlib 3.10.0, and torchvision 0.20.1. All three METACASTER agents (HPAGENT, MGAGENT, FTAGENT) are driven by an OpenAI GPT-5.4 model invoked through the OpenAI API.

C.3 Compute Budget

METACASTER’s compute cost decomposes into three stages that operate on distinct natural units, and Table 6 summarises each on the 4×NVIDIA RTX 6000 Ada node described in Appendix C.2. LLM-token counts cover both prompt and completion against GPT-5.4 via the OpenAI API.

Stage 1: Harness Optimisation (per Phase A run). Phase A trains the Meta-Generator harness once. All three agents are active: HPAGENT proposes edits, MGAGENT generates synthetic data under each candidate harness, and FTAGENT trains a panel of forecasters that scores the candidate. The resulting $\mathcal{H}^*$ is reused for every subsequent deployment, so this stage is paid once per project rather than per query.

Stage 2: Generation and Forecaster Training (per dataset). At deployment time, MGAGENT produces the synthetic training tensor $\mathbf{X}'$ from the few-shot input, and FTAGENT then trains the lightweight forecasters of LT-LIB on $\mathbf{X}'$ and selects the one shipped to the user. The LLM is invoked only inside MGAGENT’s generation step; the downstream-forecaster training that follows runs without any LLM call.

Stage 3: Post-deployment Inference (per query). Once a forecaster has been selected and deployed, only its forward pass runs at user query time. The deployed forecaster is a lightweight architecture drawn from LT-LIB (Appendix C.1), so the forward pass takes a few milliseconds per look-back window and can run on either a GPU or a CPU; no LLM call sits on this path.

	Pre-training	Deployment	Inference
Active agents	All	MGAGENT + FTAGENT	None
LLM-driven	Yes	Yes	No
GPU	Required	GPU or CPU	GPU or CPU
Time-Cost	5–7 h	30–40 min	a few ms
LLM tokens	~46 M	~150 K	0

Table 6: Computational cost of different stages under METACASTER framework.

C.4 Use of AI Assistants

During the paper writing process, we used LLMs to check and polish the grammar and typos. Beyond this, LLMs were not used extensively elsewhere.

D System Prompts

This section reproduces the System Prompts that constitute the system-prompt slot of θ for MGAGENT and the hand-authored Harnesses of HPAGENT and FTAGENT. Each prompt is rendered in the notation of the main text; runtime tokens supplied by the driver appear as `<placeholder>`.

D.1 Harness Proposer (HPAGENT)

HPAGENT optimises MGAGENT’s Harness θ over R epochs (§3.6); its System Prompt directs every epoch of the propose–evaluate–accept loop and is reproduced verbatim below.

You are HPagent of MetaCaster. Your job is to grow a library of validated synthesis skills θ^* that MGagent will load at deployment time. You do NOT generate data yourself – you author the recipes that MGagent executes.

You are currently executing epoch `<round_n>` of `<max_rounds>`.

Workspace. The Harness you edit lives at `<harness_root>/`:

```
harness/  
|- core/router.md           classification rules + skill manifest  
\- skills/<name>/SKILL.md   one self-contained recipe per regime
```

- You may edit ONLY files under `<harness_root>/`.
- The Harness starts EMPTY at epoch 0 – author every artefact from scratch.

State digest (pre-loaded each epoch). The first user message contains:

- Pinned 3-forecaster panel + the full 23-model held-out pool.
- Per-dataset metadata for the `<n_train>` training datasets in `<train_datasets>`.
- Current Harness body + skill-health report.
- Diagnostic figures from the previous epoch.
- Full history of prior epochs’ summary.json + current best.

Per-epoch workflow.

- Turns 1–2: read_image on ≥ 2 diagnostic figures; identify gaps.
- Turn 3: Brief proposal: which SKILL.md to add or edit, and why.
- Turns 4–7: edit_file / write_file – author the change (each new skill ≥ 150 lines, with ≥ 1 generation function and ≥ 1 validation function).

Skill schema. Every SKILL.md must declare YAML frontmatter:

```
---  
name: <snake_case_id>  
description: <one-line summary>  
version: <int>  
---
```

Decision rule for is_new_best. Compare against the current best epoch (not epoch 0):

- Median downstream hinge $\delta(\cdot, \cdot)$ decreases (audit-only excluded): High.
- 8-dim distribution-metric portfolio improves on the majority of datasets: High.
- Visual evidence on diagnostic figures: Tie-break.
- Catastrophic regression on any non-audit dataset ($\Delta\delta \geq +2.0$): Hard veto.

Tools. read_file, write_file, edit_file, bash, read_image, run_round_evaluation, finalize_round, etc.

End of epoch `<round_n>` spec. Begin work.

D.2 Meta-Generator (MGAGENT)

MGAGENT produces the synthetic dataset $\bar{\mathcal{D}}$ from the few-shot input via Eq. (2) (§3.4); its System Prompt is the slot of θ that HPAGENT optimises during Harness training and is reproduced verbatim below.

You are MGagent of MetaCaster. Your single deliverable is `<output_dir>/dataset.npy`: a float32 array of synthetic windows that, when used as training data for a lightweight forecaster, will yield the lowest possible test MSE.

You execute a frozen Harness θ^* produced offline by HPagent, located at `<harness_root>/`:

```
harness/  
|- core/router.md  
\- skills/<name>/SKILL.md
```

You may NOT train any forecaster yourself – FTagent will do that on your output.

Inputs you may read.

- `<input_dir>/few_shot.npy` – shape (N_few_shot, L+H, C) float32.
- `<input_dir>/meta.json` – seq_len=L, pred_len=H, C_eff=C, freq, domain.
- `<input_dir>/context.txt` – optional natural-language context C.

Three-stage pipeline.

1. **Analyse**: compute statistics (mean, std, quantiles), ACF at lag-1, 24, 48, 168, FFT peaks, channel correlation, zero_frac, lower_tail_mass. Emit a Fingerprint JSON describing regime, priorities, and challenges.
2. **Retrieve and Execute**: read `core/router.md`; match the fingerprint to one or more `SKILL.md`; run_python on the synthesis code inside that `SKILL.md`; write the candidate to `<output_dir>/dataset.npy.tmp`.
3. **Validate**: run the gates declared inside the `SKILL.md` (shape, NaN, per-channel mean/std drift, ACF preservation, quantile match, range coverage, sample-mean diversity ratio $\in [0.5, 2.0]$). On PASS, move to `dataset.npy`; on FAIL, return to Stage 2 with a different recipe.

Output contract (non-negotiable).

- Shape exactly (N, L+H, C) float32, all finite, $N \geq 100$.
- $L = \text{meta}["\text{seq_len}"]$, $H = \text{meta}["\text{pred_len}"]$, $C = \text{meta}["\text{C_eff}"]$.
- Always preserve the trailing channel dim, even when $C = 1$.
- Save exactly once to `<output_dir>/dataset.npy`.

Tools. read_file, run_python, read_image, web_search, etc.

D.3 Forecaster Trainer (FTAGENT)

FTAGENT trains every $f_i \in \mathcal{F}$ on the synthetic data and returns the best forecaster to the user via Eq. (3) (§3.5); its System Prompt is hand-authored and not optimised by HPAGENT, and is reproduced verbatim below.

You are FTagent of MetaCaster. Your job is to supervise the training of every forecaster $f_i \in \mathcal{F}$ from the lightweight library LT-Lib on the synthetic dataset $\mathcal{D}$ produced by MGagent, and to return the single best forecaster (lowest validation MSE) as the deliverable for the user.

Inputs.

- `<synth_dir>/dataset.npy` – MGagent output, shape (N, L+H, C) float32.
- `<input_dir>/meta.json` – shape and frequency metadata.
- `<test_dir>/test.npy` – held-out evaluation windows.
- Model library at `<model_pool>` – the 23 lightweight forecasters of LT-Lib.

Responsibilities.

1. **Plan and dispatch.** Enumerate (forecaster, hyperparameter, split) training jobs from `<model_pool>` and queue them onto the available `<gpu_pool>` for maximally parallel execution.
2. **Supervise in real time.** Monitor every running job’s loss curve and resource usage; reallocate GPUs as jobs finish, resolve errors as they arise, and recover interrupted jobs without human intervention.
3. **Select and deliver.** Rank the trained forecasters by validation MSE, evaluate the chosen Top-1 on `<test_dir>/test.npy`, and ship it as the deliverable for the user.

Hard constraints.

- Never modify `dataset.npy` or `test.npy`.
- Use only the registered factory architectures from `<model_pool>`.
- Every queued job MUST terminate (success or explicit failure); no silent skips.

Tools. `bash`, `read_file`, `write_file`, `read_json`, etc.

E Supplementary Results

MAE counterpart of the main table. Table 7 reports per-dataset MAE for the nine methods compared on MSE in Table 1.

Foundation models on all benchmarks. Foundation forecasters — VisionTS (Chen et al., 2024), Chronos (Ansari et al., 2024), Moirai (Woo et al., 2024), Time-LLM (Jin et al., 2024), and TimeScientist (Zhao et al., 2025) — replace the lightweight downstream forecaster with a 91 M–7 B-param backbone invoked at every test window. Figure 7 plots the METACASTER forecaster against each foundation method across all 10 evaluation datasets, with bubble area $\propto \log_{10}(\text{params})$.

Comparison with TimeScientist under the shared forecaster pool. We substitute TimeScientist’s original candidates with forecasters drawn from LT-LIB, so that both methods select from the same pool. Fig. 8 contrasts the per-dataset MSE of the best forecaster selected by METACASTER against that by TimeScientist under this shared pool. METACASTER outperforms TimeScientist on 26 of 30 cells.

	Dataset	Agent	Generation					Augmentation				Ref	
		METACASTER	TimeDP	VerbalTS	T2S	DiffTS	TimeVAE	Repeat	Bootstrap	Jitter	MagWarp	Sample	Full
IND ($K = 10$)	ETTm1	0.457	0.568	0.567	<u>0.531</u>	0.582	0.620	0.650	0.637	0.636	0.930	0.625	0.400
	Electricity	0.413	<u>0.415</u>	0.860	0.866	0.474	0.436	0.468	0.466	0.464	1.652	0.652	0.243
	Seattle	<u>0.849</u>	0.844	0.989	0.989	0.971	0.948	0.995	0.962	0.986	4.395	1.127	0.663
	SZTaxi	0.255	0.277	0.274	0.276	<u>0.260</u>	0.261	0.285	0.266	0.272	0.814	0.339	0.246
	Sales	0.661	0.795	0.771	1.027	1.510	0.830	0.945	0.766	<u>0.763</u>	1.169	0.866	0.868
	Bitbrains	<u>0.102</u>	0.085	0.114	0.110	0.135	0.149	0.172	0.150	0.156	0.408	0.208	0.148
	Solar	0.253	0.351	0.633	0.661	0.378	<u>0.349</u>	0.381	0.375	0.376	0.454	0.514	0.255
OOD	Saugeen	0.802	0.666	<u>0.675</u>	0.743	1.003	1.042	1.108	1.044	1.053	1.184	0.816	0.548
	USbirths	0.669	1.387	1.100	1.097	1.145	0.605	0.665	<u>0.649</u>	0.649	2.420	0.894	0.365
	M4 [†]	1.447	<u>1.476</u>	1.527	1.563	1.639	1.730	1.803	1.761	1.748	9.298	1.621	1.527
IND ($K = 30$)	ETTm1	0.424	0.570	0.572	<u>0.531</u>	0.592	0.565	0.606	0.583	0.583	0.675	0.565	0.400
	Electricity	0.357	0.416	0.855	0.862	0.472	<u>0.385</u>	0.411	0.398	0.401	0.824	0.616	0.243
	Seattle	0.740	<u>0.842</u>	0.980	0.980	0.983	1.085	0.933	0.903	0.891	3.241	0.974	0.663
	SZTaxi	<u>0.271</u>	0.279	0.274	0.274	0.263	0.300	0.315	0.295	0.290	0.999	0.309	0.246
	Sales	0.637	<u>0.737</u>	0.793	1.093	1.528	0.769	0.933	0.820	0.832	0.999	0.769	0.868
	Bitbrains	0.119	0.087	0.116	<u>0.113</u>	0.143	0.159	0.214	0.190	0.186	0.327	0.197	0.148
	Solar	0.243	0.363	0.636	0.675	0.380	<u>0.340</u>	0.356	0.353	0.349	0.405	0.478	0.255
OOD	Saugeen	0.647	<u>0.654</u>	0.677	0.748	1.005	0.886	0.980	0.932	0.937	0.983	0.733	0.548
	USbirths	0.570	1.218	1.104	1.090	1.132	0.533	0.569	<u>0.556</u>	0.558	1.561	0.716	0.365
	M4 [†]	1.453	<u>1.499</u>	1.547	1.588	1.632	1.845	1.768	1.763	1.780	10.04	1.718	1.527
IND ($K = 50$)	ETTm1	0.378	0.572	0.574	0.533	0.615	0.533	0.557	0.535	<u>0.532</u>	0.558	0.496	0.400
	Electricity	0.328	0.409	0.855	0.885	0.465	<u>0.352</u>	0.366	0.358	0.357	0.736	0.509	0.243
	Seattle	0.730	0.845	0.984	0.979	1.003	1.206	0.866	<u>0.836</u>	0.848	2.455	0.889	0.663
	SZTaxi	<u>0.265</u>	0.275	0.274	0.269	0.265	0.451	0.386	0.379	0.373	0.454	0.301	0.246
	Sales	0.884	<u>0.793</u>	0.797	1.080	1.534	0.833	0.860	<u>0.793</u>	0.789	0.985	0.742	0.868
	Bitbrains	0.144	0.100	0.119	<u>0.113</u>	0.136	0.186	0.317	0.281	0.292	0.381	0.180	0.148
	Solar	0.240	0.377	0.640	0.660	0.388	0.593	0.343	0.340	<u>0.338</u>	0.393	0.414	0.255
OOD	Saugeen	0.615	<u>0.642</u>	0.678	0.747	1.007	0.803	0.970	0.940	0.939	0.950	0.716	0.548
	USbirths	0.501	1.318	1.105	1.089	1.128	0.459	0.476	0.464	<u>0.461</u>	1.601	0.584	0.365
	M4 [†]	1.384	<u>1.469</u>	1.567	1.563	1.702	1.811	1.825	1.816	1.812	8.318	1.600	1.527
Wins (of 30)		19	5	0	0	2	3	0	0	1	0	–	–

Table 7: MAE counterpart of Table 1 on the same nine methods. [†]M4 uses instance-normalised MAE.

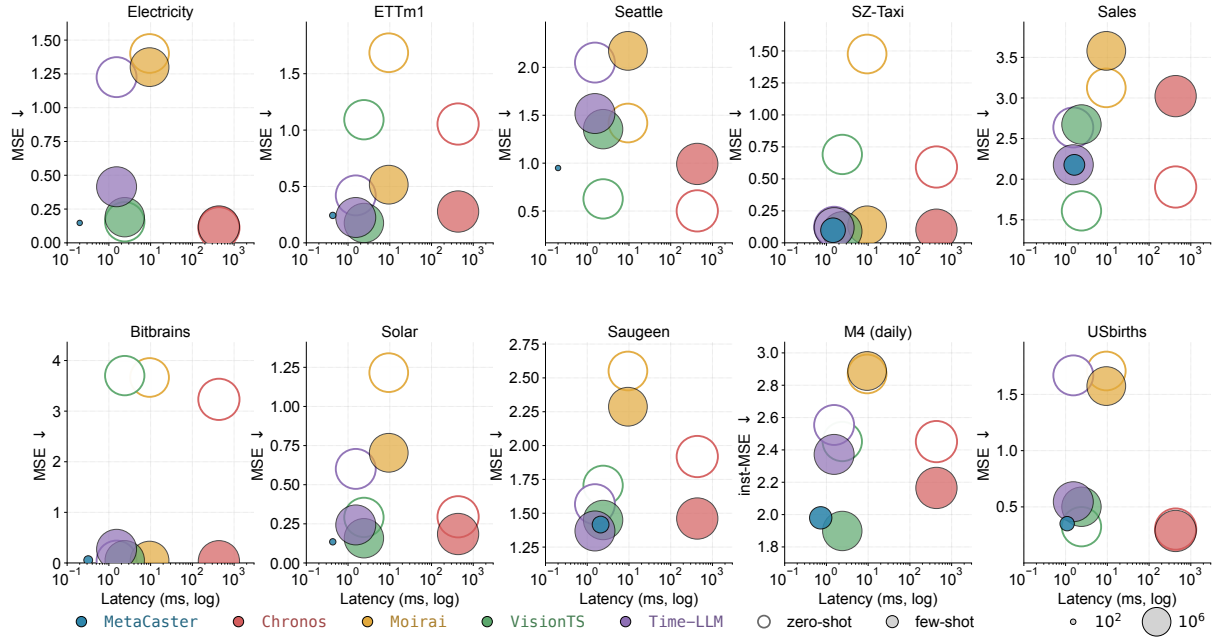


Figure 7: Comparing METACASTER with TSFMs on all evaluation datasets.

F Further Study

Top-1 / Top-3 / Top-5 vs. baselines. At deployment, only a single forecaster is shipped per dataset, so we additionally compare each method using only its best-performing forecasters rather than the full pool. Table 8 reports, per dataset, the mean MSE of the K best forecasters trained under each method (for $K = 1, 3, 5$), evaluated within the 20 main-pool forecasters. METACASTER retains the lead across most cells.

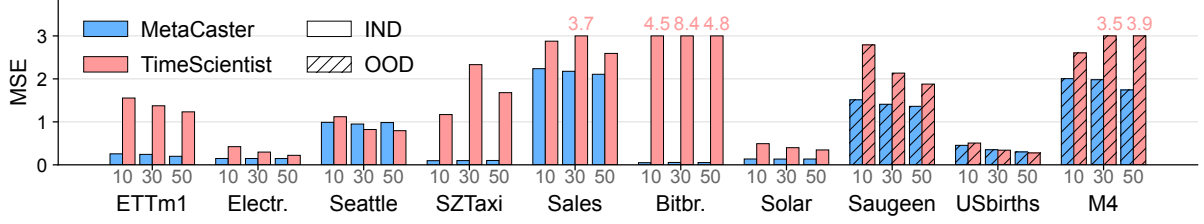


Figure 8: METACASTER vs. TimeScientist under the shared LT-LIB forecaster pool.

Dataset	Ours METACASTER	Generation Models						Augmentation Methods				References	
		TimeDP	VerbalTS	T2S	DiffTS	TimeVAE	Repeat	Bootstrap	Jitter	MagWarp		$\mathcal{D}_{sup}^m$	$\mathcal{D}_{tr}^m$
Top-1													
ETTM1	0.243	0.967	0.982	0.886	0.921	0.437	0.382	0.382	0.385	<u>0.358</u>		0.489	0.300
Electricity	0.148	0.209	0.947	1.017	0.237	0.128	0.125	0.125	0.125	<u>0.125</u>		0.274	0.107
Seattle	0.949	1.271	1.658	1.674	1.288	1.678	1.097	<u>1.097</u>	1.114	1.647		1.295	0.953
SZTaxi	0.099	0.091	0.103	0.105	0.091	0.095	0.092	<u>0.092</u>	<u>0.091</u>	0.088		0.090	0.075
Sales	2.176	2.265	<u>2.205</u>	2.416	2.483	2.223	2.410	2.382	2.416	2.512		2.202	2.270
Bitbrains	0.056	<u>0.077</u>	0.079	0.079	0.082	0.082	0.144	0.142	0.143	0.154		0.083	0.102
Solar	0.135	<u>0.173</u>	0.453	0.503	<u>0.145</u>	0.148	0.149	0.149	0.150	0.149		0.222	0.136
Saugeen	1.416	1.444	1.429	1.512	1.658	1.489	1.372	<u>1.371</u>	1.380	1.359		1.292	1.089
USbirths	0.351	1.376	1.425	1.412	1.364	0.266	<u>0.250</u>	0.249	0.255	0.502		0.487	0.106
M4 [†]	<u>1.980</u>	1.932	2.299	2.164	2.190	2.204	2.121	2.086	2.089	13.182		2.139	1.853
Top-3													
ETTM1	0.256	0.974	0.986	0.892	0.944	0.491	0.435	0.431	0.432	<u>0.391</u>		0.496	0.303
Electricity	0.158	0.219	0.974	1.026	0.271	0.149	0.154	<u>0.152</u>	0.154	0.172		0.308	0.107
Seattle	0.985	1.295	1.698	1.696	1.363	1.697	1.177	<u>1.151</u>	1.192	2.052		1.351	0.966
SZTaxi	0.101	0.095	0.105	0.106	0.092	0.098	0.097	<u>0.094</u>	0.095	0.115		0.106	0.077
Sales	2.201	2.295	<u>2.225</u>	2.464	2.634	2.248	2.504	2.437	2.486	2.597		2.221	2.324
Bitbrains	0.068	<u>0.077</u>	0.079	0.079	0.085	0.083	0.178	0.171	0.159	0.173		0.085	0.104
Solar	0.137	0.181	0.470	0.509	<u>0.163</u>	0.165	0.172	0.167	0.172	0.183		0.227	0.137
Saugeen	1.422	1.450	1.435	1.535	1.712	1.572	1.395	<u>1.396</u>	1.404	1.572		1.306	1.096
USbirths	0.369	1.452	1.457	1.417	1.396	0.326	0.336	<u>0.332</u>	0.334	0.695		0.513	0.123
M4 [†]	<u>2.006</u>	1.974	2.311	2.168	2.278	2.506	2.219	2.208	2.280	34.704		2.186	1.974
Top-5													
ETTM1	0.282	0.979	0.989	0.894	0.970	0.539	0.493	0.467	0.469	<u>0.466</u>		0.499	0.304
Electricity	0.183	0.229	0.998	1.040	0.293	<u>0.192</u>	0.200	0.194	0.196	0.263		0.337	0.107
Seattle	1.046	1.314	1.730	1.716	1.403	1.704	1.280	<u>1.249</u>	1.268	5.347		1.403	0.971
SZTaxi	0.102	<u>0.098</u>	0.106	0.107	0.095	0.102	0.102	<u>0.098</u>	0.098	0.185		0.112	0.079
Sales	2.213	2.311	<u>2.234</u>	2.563	2.787	2.273	2.537	2.481	2.518	2.721		2.241	2.381
Bitbrains	0.074	<u>0.077</u>	0.079	0.082	0.090	0.085	0.195	0.188	0.181	0.211		0.089	0.118
Solar	0.141	0.185	0.477	0.517	0.187	<u>0.183</u>	0.190	0.186	0.190	0.226		0.234	0.138
Saugeen	1.425	1.455	<u>1.438</u>	1.543	1.767	1.595	1.623	<u>1.598</u>	1.619	1.839		1.320	1.104
USbirths	0.394	1.478	1.467	1.429	1.429	0.358	0.372	<u>0.359</u>	0.368	1.051		0.537	0.133
M4 [†]	<u>2.013</u>	2.000	2.319	2.181	2.331	2.667	2.273	2.267	2.358	52.066		2.253	2.021

Table 8: Per-dataset top- K best-forecaster MSE at $n_{\text{few}} = 30$. [†]M4 uses instance-normalised MSE.

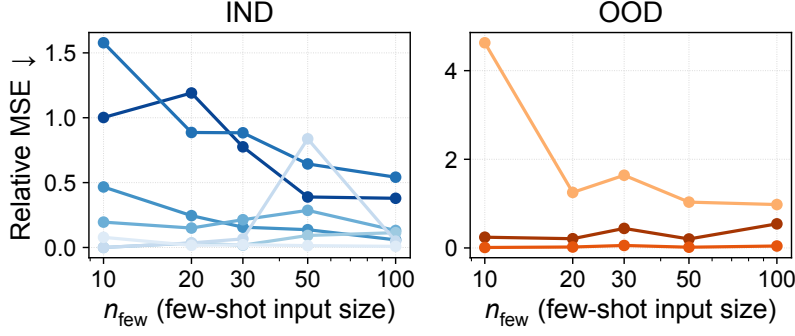
Few-shot input scaling. Figure 9 reports the per-dataset Relative MSE of METACASTER at $n_{\text{few}} \in \{10, 20, 30, 50, 100\}$ on both IND and OOD subsets. On most datasets, METACASTER improves as n_{few} grows.

Multi-seed sensitivity at $n_{\text{few}} = 30$. Table 9 reports the per-dataset MSE of METACASTER across three independent LLM-sampling seeds at $n_{\text{few}} = 30$. METACASTER is seed-insensitive on most datasets.

Per-forecaster MSE distribution: violin plots. Figure 10 renders the per-method Relative MSE distribution across the $20 \times \{7, 3\}$ (forecaster, dataset) cells, split IND / OOD. METACASTER sits below every non-Full baseline at every quartile in both subsets, with the narrowest non-trivial distribution.

Heatmap across forecasters and methods. Figure 11 renders the per-(forecaster, method) Relative MSE as a dual-panel heatmap, with the 20 main forecasters as rows and the 10 methods of Table 1 as columns. METACASTER is the lowest column on both panels.

Qualitative synthetic samples. Figure 12 compares one synthetic window per method on four representative datasets (ETTM1 ch. 0, Solar, Electricity, Saugeen), against a real window on the top row for



Dataset	MSE ($\pm$ std)
ETTM1	0.345 ± 0.023
Electricity	0.226 ± 0.000
Seattle	1.177 ± 0.041
SZTaxi	0.114 ± 0.003
Sales	2.362 ± 0.411
Bitbrains	0.125 ± 0.025
Solar	0.152 ± 0.000
Saugeen	1.464 ± 0.523
USbirths	0.533 ± 0.254
M4 [†]	2.112 ± 0.044

Figure 9: Few-shot input scaling of METACASTER on IND and OOD.

Table 9: Per-dataset MSE of METACASTER, mean $\pm$ std across three independent LLM-sampling seeds ([†]M4 in instance-normalised MSE).

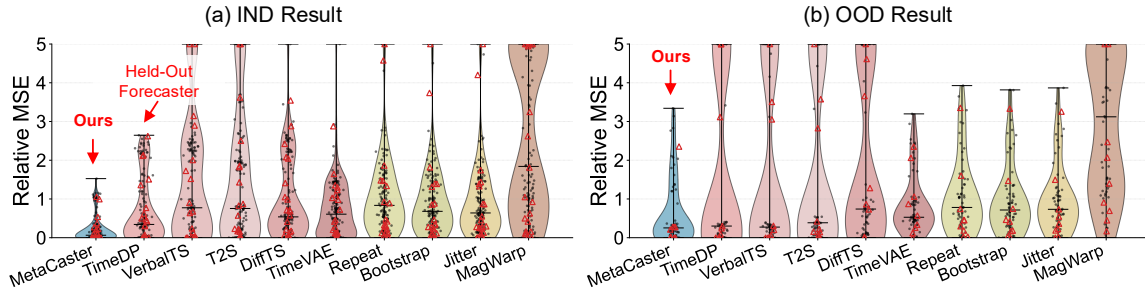


Figure 10: Relative MSE distributions at $n_{\text{few}} = 30$ on IND (left) and OOD (right).

reference. METACASTER’s window tracks both the dominant period and the local fluctuations of the real series across all four datasets, while the deep generative baselines either drift in amplitude, miss the period, or collapse to near-constant trajectories.

Distributional alignment via t-SNE. Figure 13 runs t-SNE separately on each of the 10 evaluation datasets. METACASTER-generated windows overlap the real manifold on most datasets, indicating that the synthetic distribution is well aligned with the real one.

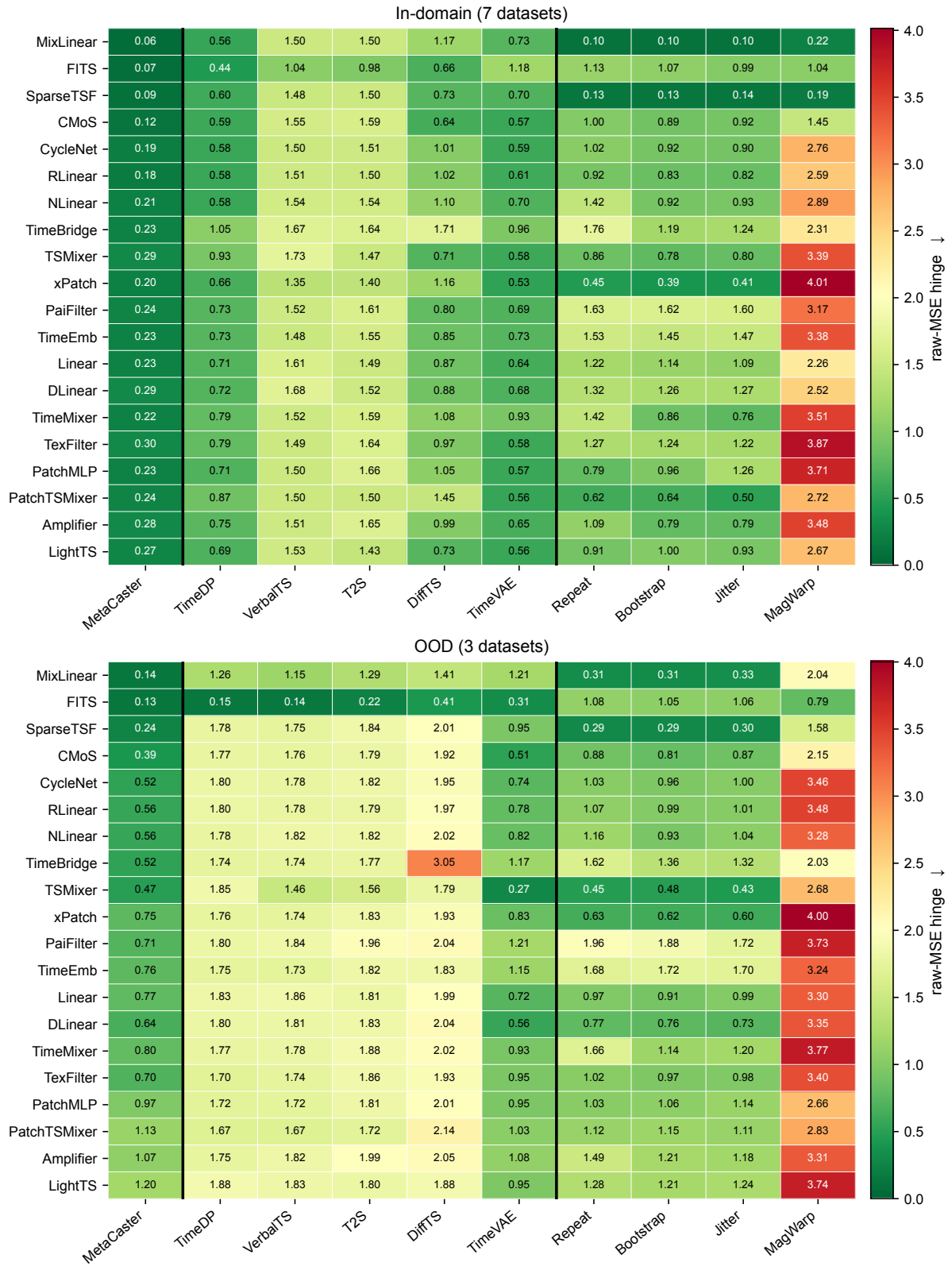


Figure 11: Per-(forecaster, method) Relative MSE heatmap (top: IND; bottom: OOD).

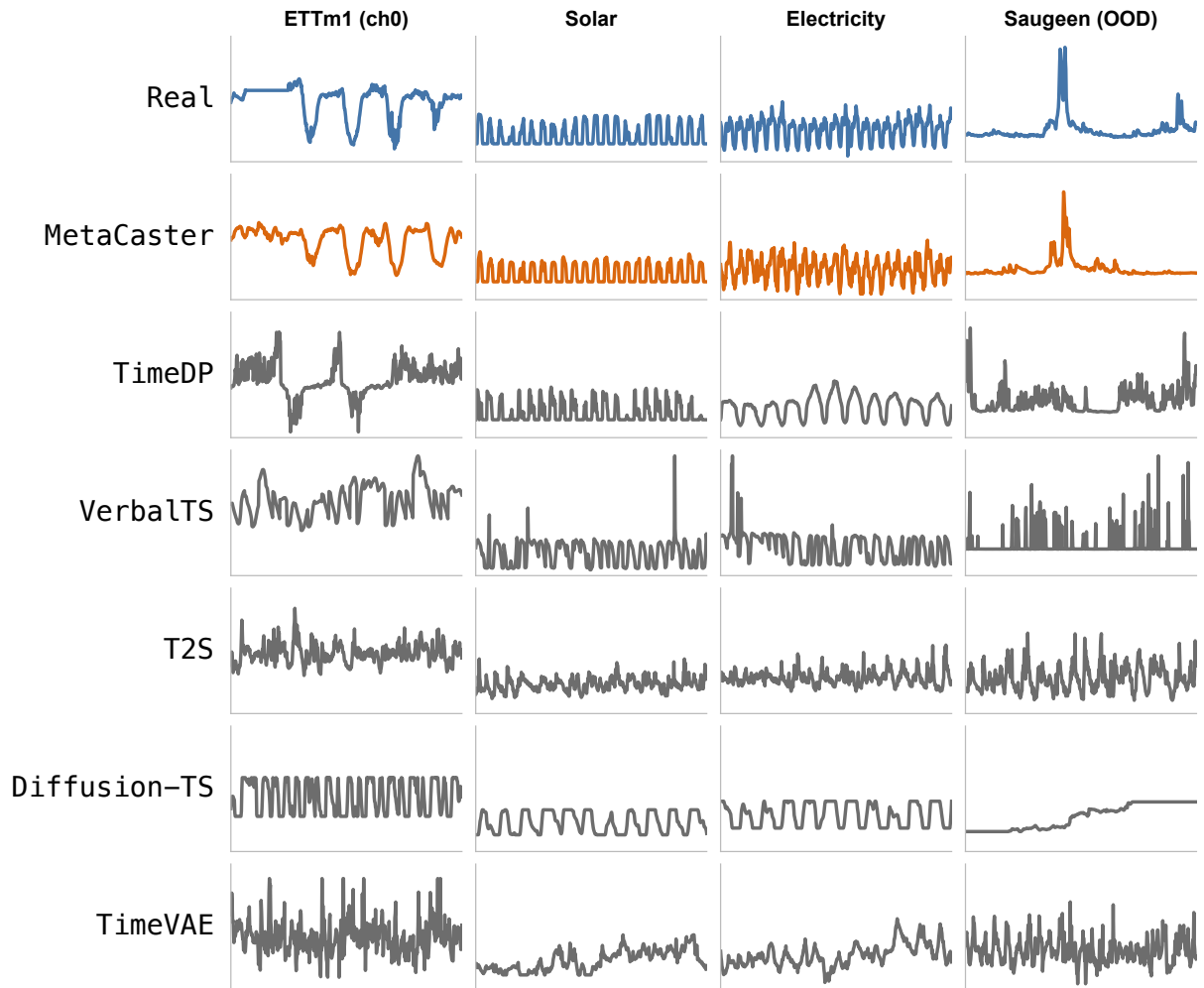
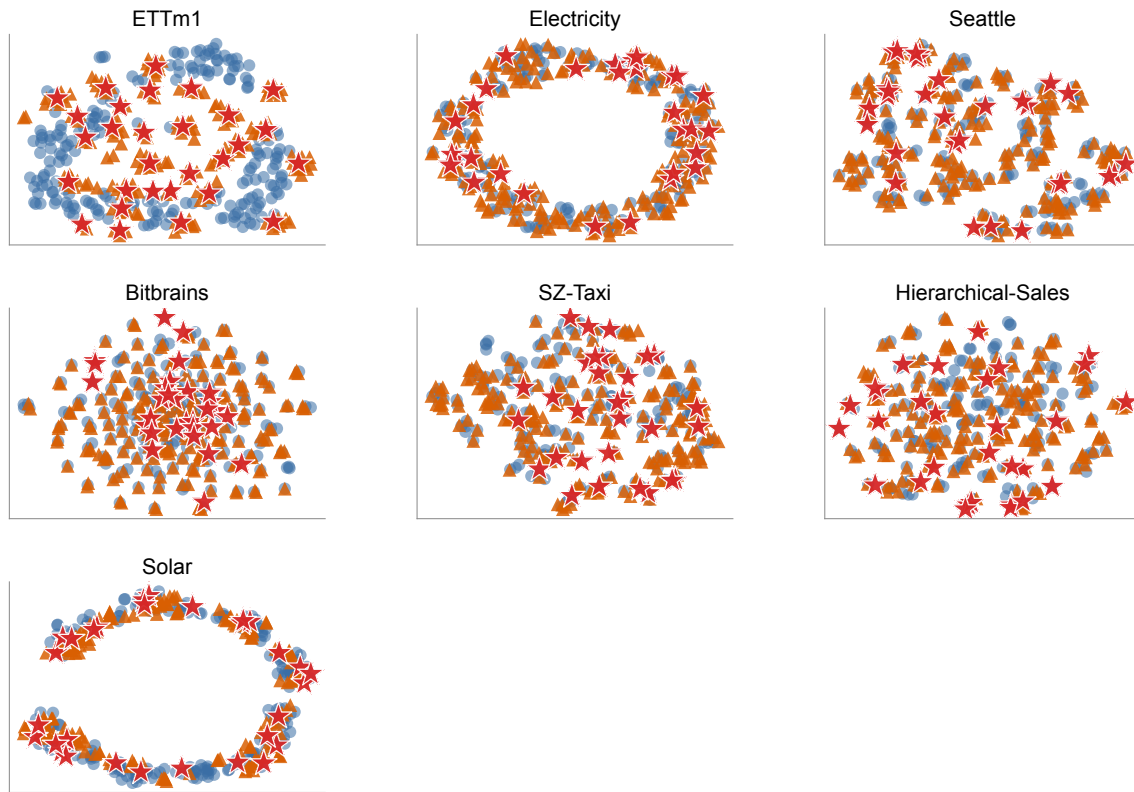


Figure 12: Synthetic windows on four datasets (channel 0): real on top, METACASTER, and five generative baselines.

In-Distribution



Out-of-Distribution

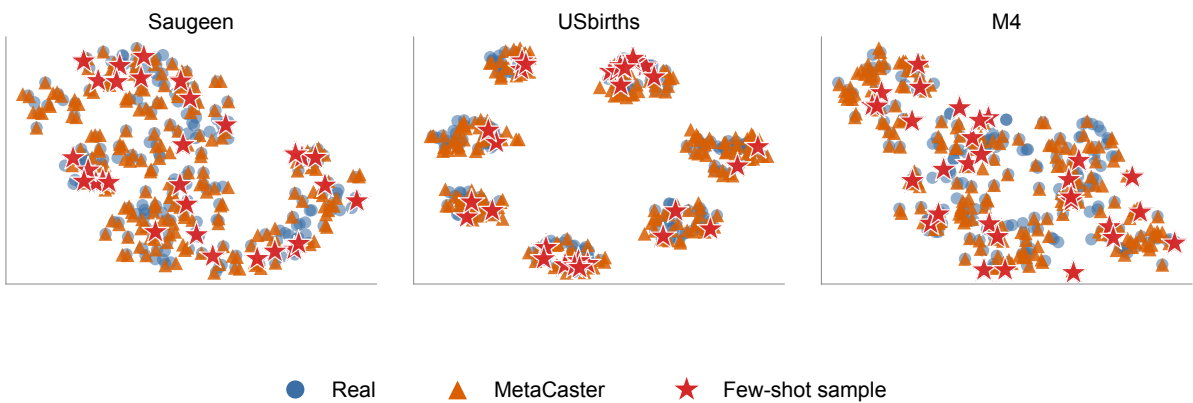


Figure 13: Per-dataset t-SNE of real, METACASTER-generated, and few-shot windows (top: IND; bottom: OOD).