TSAutoBox: A Localized Multi-Agent System for Industrial Time-Series Task Automation

Xinyu Xi
National University of Singapore
xinyu_xi@u.nus.edu

Yihao Ang*
National University of Singapore
yihao_ang@comp.nus.edu.sg

Yifan Bao
National University of Singapore
yifan_bao@comp.nus.edu.sg

Ye Xu†
Huazhong University of Science and Technology
xiwon@hust.edu.cn

Ruiyao Ma
National University of Singapore
ruiyaoma@u.nus.edu

Anthony K. H. Tung
National University of Singapore
atung@comp.nus.edu.sg

ABSTRACT

Time-series task automation remains difficult in operational domains such as manufacturing, cybersecurity, and healthcare, where users must turn raw data and high-level objectives into executable modeling pipelines. Existing benchmarking, recommendation, and LLM-based agentic systems help identify models or generate code, but still face limitations in execution-grounded ToolChain search, user-visible traceability, and localized deployment. We present **TSAutoBox**, a localized multi-agent system for industrial time-series task automation. Given a natural-language task description and dataset, **TSAutoBox** profiles the data, infers the task, constructs an initial ToolChain, and refines candidates through Search Tree expansion with typed messages, deterministic execution, atomic revisions, and multi-fidelity evaluation. The system is implemented as an interactive localized box that packages local LLM inference, a multi-agent control service, Ray-backed execution, shared storage and observability on an edge device. We evaluate **TSAutoBox** on manufacturing anomaly detection, cybersecurity forecasting, and healthcare time-series generation. The local Box configurations achieve better results for every reported task metric, relative to fixed-structure AutoML baselines. Compared with a server and API deployment, local inference improves the average LLM-call success rate by 27%, significantly reduces measured per-run cost, while staying within a comparable energy consumption. These results suggest that localized multi-agent automation can make time-series modeling more accessible to privacy-sensitive and resource-constrained organizations.

VLDB Workshop Reference Format:
Xinyu Xi, Yihao Ang, Yifan Bao, Ye Xu, Ruiyao Ma, and Anthony K. H. Tung. TSAutoBox: A Localized Multi-Agent System for Industrial Time-Series Task Automation. VLDB 2026 Workshop: ADS 2026: The Joint Workshop on Agentic Data Systems and Data-Centric AI (The 1st ADS & 3rd DATAI).

*Corresponding author.
†Work done during internship at the National University of Singapore.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

1 INTRODUCTION

Time-series data are a primary substrate for operational decision making in manufacturing plants [3, 8], network operation centers [30, 31, 43], and hospitals [17, 42]. Over the past decade, the community has developed a rich ecosystem of time-series models and toolkits for anomaly detection [13, 15, 16, 38, 39], forecasting [32, 48], and generation [7, 12], ranging from classical statistical methods [8, 10, 54] to deep neural architectures [3, 14, 51, 52] and task-specific AutoML systems [4, 23, 33]. These advances have made it increasingly feasible to identify competitive model families for a given benchmark. However, in real deployments, knowing which model works is only the beginning. A practitioner must still translate a domain question and raw multivariate time series into an executable data-science process, including data inspection, task inference, preprocessing, model selection, split and metric definition, debugging, tuning, result interpretation, and artifact deployment.

This is the persistent "last-mile" gap between domain expertise and model engineering. Model benchmarking and recommendation tools [5, 9] help answer questions such as *which models exist* and *which models are promising*, but they do not by themselves construct, execute, revise, and monitor a complete toolchain for the user. The gap is particularly acute for Small and Medium-sized Enterprises (SMEs) and privacy-sensitive organizations. Domain experts in these settings often understand the physical process, network behavior, or clinical signal much better than a generic machine-learning engineer, yet they might lack the engineering resources needed to build reliable pipelines. At the same time, cloud-based automation is not always acceptable. Industrial sensor traces, network logs, and physiological recordings could be sensitive, regulated, latency-critical, or simply too costly to move outside the local environment [24]. For such users, automation must be not only accurate, but also deployable, observable, and locally controllable.

Recent progress in Large Language Models (LLMs) and agentic systems [6, 11, 22] suggests a promising direction for closing this gap. A direct "LLM-as-script-writer" approach can often produce a plausible modeling idea or code skeleton, but it is difficult to make dependable: generated code can have inconsistent data dependencies, incompatible preprocessing and evaluation choices, hidden leakage, or failures that only appear after execution [6]. General agentic data-science systems improve over direct generation by adding planning, tool use, execution feedback, and revision [11], but they are not explicitly designed for temporal structure, time-series metrics, or fragmented time-series libraries. Recent time-series

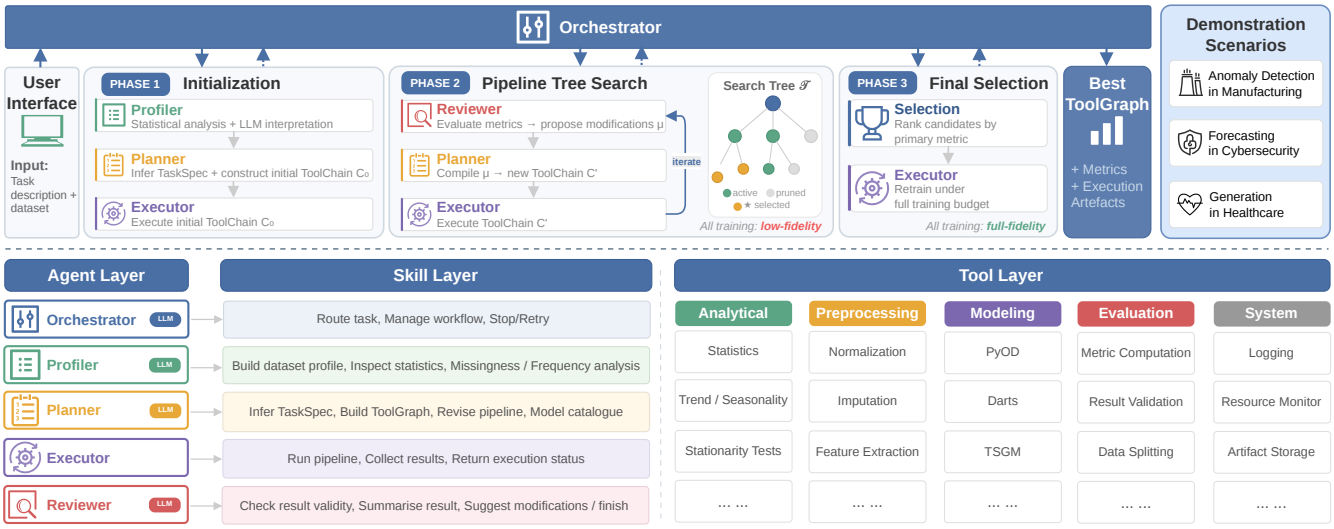

Figure 1: End-to-end multi-agent automation loop (top) and layered agent design (bottom).

agents, such as DCATS [50], TimeCopilot [20], and TS-Agent [6], further demonstrate the promise of LLM-based automation, but typically focus on a narrower task family or do not make edge-resident local execution a primary design target.

Taken together, existing approaches leave three limitations. **L1**: they do not construct and refine complete executable ToolChains across preprocessing, modeling, evaluation, and final selection. **L2**: they provide limited interactivity and traceability for domain users who need to inspect decisions, artifacts, and failures. **L3**: they often rely on cloud APIs or large-server deployments whose cost, reliability, and data-transfer requirements could be prohibitive for SMEs. We therefore present **TSAutoBox**, a localized multi-agent system for automating industrial time-series tasks from natural-language task descriptions and datasets, with three key contributions:

- **C1: An end-to-end multi-agent automation loop for industrial time-series tasks.** To address **L1**, **TSAutoBox** coordinates an Orchestrator, Profiler, Planner, Executor, and Reviewer to transform a task description and dataset into an optimized ToolChain. Rather than relying on a single generated script, the loop uses typed structured messages, execution-grounded review, atomic revisions, and a Search Tree whose nodes are candidate ToolChains and whose edges are interpretable modifications such as changing a model family, adding preprocessing, removing a step, or tuning a hyperparameter.
- **C2: An interactive system design with localized LLMs.** To tackle **L2**, **TSAutoBox** separates the Agent layer, Skill layer, and Tool layer so that decision logic, task knowledge, and callable operations can evolve independently. The system interface allows users to submit datasets and task descriptions, inspect data previews, monitor the Search Tree, review execution traces, and view the ToolChain summary without writing machine-learning code. A local LLM service is deployed with the agents, while the Executor runs deterministic calls to off-the-shelf time-series libraries for anomaly detection, forecasting, and generation.
- **C3: A localized system running on an edge device.** To handle **L3**, **TSAutoBox** packages the user interface, local LLM service, multi-agent control service, Ray-backed execution, shared storage, and observability components into a containerized deployment that can run on an edge device.

We evaluate **TSAutoBox** on three demonstration scenarios using three local LLM backends on a localized box and a server-based GPT-5.4 API deployment, and task-specific AutoML baselines. Across all reported task metrics, the best performance is achieved by a local box configuration. Compared with fixed-structure AutoML, the best local runs improve manufacturing Aff-F1 from 0.810 to 0.909 and reduce cybersecurity forecasting MAE by 83%. Operationally, local inference raises the average LLM-call success rate from 0.74 to 0.94 and reduces total per-run cost, from 0.72 to 0.006, while maintaining comparable energy consumption. A case study on manufacturing anomaly detection further illustrates the complete run, from user submission and ToolChain search to full-fidelity retraining, live monitoring, and final reporting. These results show that multi-agent time-series task automation can be both accurate and deployable when data, model reasoning, execution, and observability remain within the same local system boundary.

2 AGENTIC AUTOMATION LOOP

At the core of **TSAutoBox** is an end-to-end agentic automation loop which automates time-series workflow construction and optimization, with sufficient flexibility to generalize across diverse real-world domains. In particular, we first describe the three-phase workflow that coordinates the agents (§2.1), then detail the layered agent design (§2.2), and finally introduce the ToolChain representation together with the pipeline tree search mechanism that constitutes the core of our approach (§2.3).

2.1 Multi-Agent Workflow

The upper half of Figure 1 shows the end-to-end agentic automation loop: given a task and a dataset, **TSAutoBox** runs in three phases to produce an optimized, executable time-series solution.

Phase 1 (Initialization). The Orchestrator first forwards the natural-language task description and the dataset to the Profiler, which

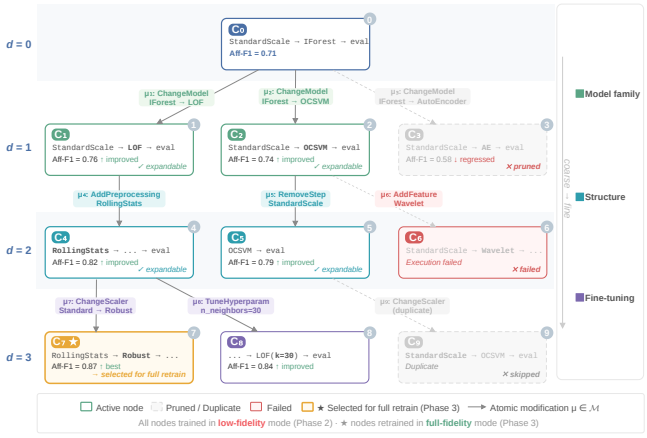

Figure 2: An example of Search Tree $\mathcal{T}$.

produces a structured characterization of the input data (see Figure 5(b.1)–(b.2)). The resulting profile is then passed to the Planner, which infers a formal task specification and constructs an initial ToolChain. The initial ToolChain is handed to the Executor for execution, which marks the beginning of Phase 2.

Phase 2 (Pipeline Tree Search). Then, **TSAutoBox** enters an iterative search loop. At each iteration, the Reviewer examines the evaluation metrics produced by the Executor and either declares the current result satisfactory or proposes atomic modifications. Each modification proposal is compiled by the Planner into a new candidate ToolChain and executed under a low-fidelity training regime. This search loop progressively expands a Search Tree (see Figure 2 and 5(b.3)) where each node is a ToolChain (see Figure 5(b.4)) and each edge is one modification. Details are provided in §2.3.

Phase 3 (Final Selection). Upon reaching the search budget, **TSAutoBox** selects top-performing candidates from the Search Tree and retrains each under a full-fidelity regime. Finally, the best ToolChain, together with its evaluation metrics and execution artifacts, is returned as the output (see Figure 5(b.6)).

All inter-agent communication is routed through the Orchestrator via typed, structured messages carrying session and trace identifiers, ensuring provenance tracking and deterministic replay, in contrast to free-form natural language communication, which is prone to instability and irreproducibility [26]. Agent decisions and resource consumption are continuously surfaced to the user via a live activity log (Figure 5(b.5)) and resource monitor (Figure 5(b.7)).

2.2 Layered Agent Design

TSAutoBox is organized into three layers with distinct responsibilities, as shown in the lower half of Figure 1. The *Agent Layer* specifies what each agent decides, the *Skill Layer* equips each agent with its role and domain knowledge, and the *Tool Layer* exposes a shared pool of callable functions. This loose coupling ensures that each layer can be modified independently, enabling scalability and adaptability to varying task types and domain requirements.

Agent Layer. The Agent Layer comprises five specialist agents. The *Orchestrator* maintains global session state and coordinates all agent interactions. The *Profiler* characterizes the input dataset via statistical analysis and LLM-based interpretation. The *Planner*

Algorithm 1: ToolChain Tree Search

Input: Initial ToolChain C_0 with metrics $\mathbf{r}_0$; maximum depth D; beam width K; branching factor B
Output: Set of top-performing candidate ToolChains $\mathcal{C}_{\text{top}}$

Initialize tree $\mathcal{T}$ with root node $n_0 \leftarrow \langle C_0, \mathbf{r}_0 \rangle$;
for $d \leftarrow 1$ **to** D **do**
  $\mathcal{F} \leftarrow \textsc{SelectFrontier}(\mathcal{T}, d{-}1, K)$;
  for *each parent* $n \in \mathcal{F}$ **do**
    $\mathcal{P} \leftarrow \textsc{Reviewer}(n, \textsc{AllowedTypes}(d), B)$;
    for *each modification* $\mu \in \mathcal{P}$ **do**
      $C' \leftarrow \textsc{Planner.Revise}(C_n, \mu)$;
      $C' \leftarrow \textsc{RepairBindings}(C')$;
      if $\textsc{IsInvalid}(C')$ *or* $\textsc{Duplicate}(C', \mathcal{T})$ **then**
        continue;
      $\mathbf{r}' \leftarrow \textsc{Executor.Run}(C', \textit{low-fidelity})$;
      $n' \leftarrow \langle C', \mathbf{r}' \rangle$; `// Create child node`
      $\mathcal{T}.\textsc{AddEdge}(n, n', \mu)$;
      if $\mathbf{r}'.\textit{status} = \textsc{Fail}$ *or* $\textsc{ShouldPrune}(n', n)$ **then**
        mark n' as non-expandable;
$\mathcal{C}_{\text{top}} \leftarrow \textsc{SelectTopCandidates}(\mathcal{T})$;

infers the task specification and constructs or revises ToolChains. The *Executor* runs ToolChains deterministically without invoking an LLM, ensuring reproducibility. The *Reviewer* assesses execution metrics and, when unsatisfied, produces structured modification proposals that feed back into the Planner.

Skill Layer. The Skill Layer defines a self-contained capability module for each agent, encapsulating its role specification and domain knowledge as prompt-augmented procedures. For instance, the Planner's skill module includes a model catalogue and pipeline templates tailored to each task type.

Tool Layer. The Tool Layer exposes a shared registry of atomic, stateless functions with typed I/O contracts, organized into five categories: *analytical tools* (statistical tests, trend and seasonality detectors) serving primarily the Profiler; *preprocessing tools* (normalisation, imputation, feature extraction) used within ToolChain pipelines; *modeling tools* wrapping established time-series libraries such as Darts [23], PyOD [54], and TSGM [33] to support forecasting, anomaly detection, and generation tasks respectively; *evaluation tools* (metric computation, result validation) invoked during pipeline execution; and *system tools* (logging, resource monitoring) that support observability across all agents. New tools are discovered and registered automatically at system startup, making the tool pool extensible without configuration changes.

2.3 ToolChain and Search Tree

During Phase 2, **TSAutoBox** must repeatedly generate, execute, compare, and revise candidate pipelines. To make this process reproducible, we represent each candidate pipeline as a structured executable object called a ToolChain. A ToolChain is an ordered sequence $C = (s_1, \ldots, s_L)$ of L processing steps, where each step s_ℓ encapsulates its complete executable configuration, including tool

identity, hyperparameters, and input/output bindings. The order of C defines the execution schedule, e.g., `load_data` → `preprocess` → `train_model` → `evaluate`. Because ToolChains are generated and revised through LLM calls, they might contain missing inputs, inconsistent outputs, or tool choices that do not match the inferred task. Before execution, **TSAutoBox** checks each ToolChain against the tool specifications, fills in simple missing bindings when possible, and rejects candidates that still cannot be executed.

ToolChain Tree Search. To explore ToolChains systematically, **TSAutoBox** constructs a tree $\mathcal{T}$. The root stores the initial ToolChain C_0 from Phase 1. Each node n stores a state tuple $\langle C_n, \mathbf{r}_n \rangle$, where C_n is a candidate ToolChain and $\mathbf{r}_n$ contains its execution status and evaluation metrics. An edge from parent n to child n' records the atomic modification μ applied to C_n to obtain $C_{n'}$, such as replacing a model, adding preprocessing, or tuning a hyperparameter.

At tree depth $d = 1, \ldots, D$, where D is the maximum search depth, SelectFrontier selects up to K expandable nodes from the previous depth based on their evaluation metrics. For each selected parent, the Reviewer proposes up to B modifications allowed by AllowedTypes(d), the Planner materializes each proposal into a new ToolChain, and the Executor runs valid candidates under a low-fidelity budget. The policy AllowedTypes(d) implements a coarse-to-fine schedule: shallow depths explore coarse structural choices such as model family, while deeper depths focus on local refinements such as preprocessing and hyperparameters.

Duplicate ToolChains are skipped before execution to avoid redundant runs. Candidates that fail during execution or strongly regress relative to their parent are marked as non-expandable, but remain in $\mathcal{T}$ to preserve the complete search trace. The search follows a train-cheap, validate-expensive strategy: candidates are evaluated under a low-fidelity budget during Phase 2, and once the search reaches depth D, the top-performing candidates are retrained under a full-fidelity budget in Phase 3. Algorithm 1 summarizes the expansion process used to construct the ToolChain search tree.

3 LOCALIZED SYSTEM DEPLOYMENT

Existing agentic systems for time-series analysis typically rely on cloud-hosted LLM APIs [6, 22] for reasoning and decision-making. Such dependence introduces reliability issues: API calls may fail due to network instability or firewall restrictions [53], making these systems unsuitable for air-gapped or latency-sensitive environments. **TSAutoBox** addresses this as a fully localized system, independent of external cloud infrastructure or API services. Furthermore, **TSAutoBox** is deployed as a *localized box*: a fully containerized, self-contained deployment on a single edge device. Compared to conventional servers, it significantly reduces per-run cost while maintaining comparable energy consumption (§4).

3.1 System Topology

The system consists of four functional tiers packaged as Docker containers and orchestrated via Docker Compose [19] (Figure 3).

- **User Interface** provides a web-based interface through which users submit time-series tasks and monitor execution progress.
- **Control Service** hosts the Orchestrator, Profiler, Planner, and Reviewer agents. It receives task submissions from User Interface, coordinates the three-phase workflow described in §2.1, and writes all artifacts to a shared persistent volume. All LLM inference is served locally via vLLM [25, 47] with an OpenAI-compatible API, eliminating dependence on external calls and ensuring reliable operation under network restrictions.
- **Executor** is backed by a Ray [29, 46] cluster consisting of a head node and one or more worker nodes. Control Service submits complete ToolChains as remote tasks to Executor, which handles resource allocation, queuing, and fault recovery. Each ToolChain is treated as an atomic unit: steps run sequentially on a single worker, while multiple ToolChains from the same search tree layer run concurrently across workers.
- **Storage and Observability tier** comprises a PostgreSQL [45] instance for session persistence and a shared artifact volume accessible to the User Interface, Control Service, and Executor. Prometheus [1] and Grafana [21] are deployed for resource monitoring. All session outputs, including search trees, execution traces, and resource summaries, are written to the shared volume and accessible to the User Interface for real-time visualization.

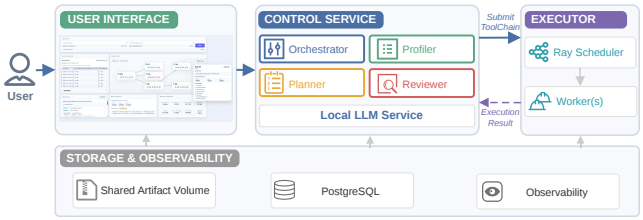

Figure 3: Deployment topology of TSAutoBox.

This four-tier decomposition allows each concern to evolve independently. For instance, the User Interface can be replaced with a domain-specific frontend, and the Executor can be scaled horizontally by adding Ray nodes without modifying application logic.

3.2 Resource Management

TSAutoBox can be deployed entirely within a localized edge device (e.g., NVIDIA DGX Spark), with all compute managed on-device. The separation between the Control Service and the Executor enables differentiated GPU allocation across two training fidelities: low-fidelity search tasks use fractional GPU shares so that multiple candidates run concurrently, while full-fidelity retraining receives dedicated GPU access, enforced by the Ray scheduler. On a single GPU, this supports up to four concurrent low-fidelity evaluations or one full-fidelity training run. Compared to conventional cloud servers, this self-contained edge deployment achieves significantly lower total cost, as quantified in §4.

4 DEMONSTRATION AND EVALUATION

We evaluate **TSAutoBox** across three representative time-series tasks in diverse application domains, comparing against both classical AutoML baselines and a cloud-hosted deployment to validate the system's effectiveness and the viability of on-premises operation.

4.1 Demonstration Scenarios

We demonstrate the applicability of **TSAutoBox** in three real-world domains where localized deployment is operationally motivated: manufacturing, cybersecurity, and healthcare.

- **Predictive quality control in the manufacturing domain**: The task is to detect abnormal operating periods that may indicate faults or quality deviations, supporting predictive maintenance and downtime reduction. The dataset is a multivariate sensor stream from a continuous manufacturing process (1 Hz sampling, 5 machines, 15 measurement channels) [44].
- **Network traffic forecasting in the cybersecurity domain**: The task is to forecast future traffic volumes given historical observations, supporting capacity planning and intrusion detection in network monitoring centers. The dataset is multivariate traffic volume data from the Abilene backbone network, where each variable corresponds to an origin–destination pair [49].
- **Synthetic EEG signal generation in the healthcare domain**: The task is to generate realistic synthetic EEG sequences that preserve temporal dynamics and cross-channel correlations, addressing the chronic scarcity of labeled physiological recordings in clinical research. The dataset is a 14-channel EEG recording from the Emotiv neuroheadset with a binary eye-state label [40].

4.2 Experimental Setup

We compare five configurations across all three scenarios:

- **AutoML baseline**: task-specific classical pipelines including Optuna+PyOD [4, 54] for anomaly detection, AutoTS [18] for forecasting, Optuna+TSGM [4, 33] for generation, representing single-pass hyperparameter search over a fixed pipeline.
- **Server + GPT-5.4 API**: **TSAutoBox** runs on a server with the GPT-5.4 [36] API, representing a high-resource, network-dependent configuration. The server has an NVIDIA H200 GPU, a 32-core AMD EPYC CPU, and 128 GB RAM.
- **Box + Local LLM** (×3): We deploy **TSAutoBox** on an NVIDIA DGX Spark as an edge AI box, using three local LLMs of varying sizes: Gemma-4-31B-it-NVFP4 [34], DeepSeek-R1-70B-Q4KM [35], and Qwen3.5-122B-A10B-NVFP4 (MoE, 10B active) [41].

Following established evaluation protocols [7, 27, 28, 33, 39], we first evaluate the performance using task-specific metrics: Affiliate-F1 (Aff-F1), AUC-ROC, and VUS-ROC [27, 39] for anomaly detection; MAE, RMSE, and MASE [28] for forecasting; Maximum Mean Discrepancy (MMD), Discriminative Score (DS), and downstream performance [7, 33] for generation. In addition to task quality, we evaluate operational efficiency and cost across deployment configurations by measuring three indicators:

- **Reliability**: the LLM-call success rate over the full search loop.
- **Energy**: the total energy consumption in Wh per run.
- **Cost**: total per-run cost, including electricity and API fees.

Electricity cost is calculated at the global average rate of $0.17/kWh [2]. API cost follows the GPT-5.4 pricing listed on the OpenAI platform [37] ($2.50 per 1M input tokens, $15.00 per 1M output tokens). All reported metrics are averaged over five independent runs to ensure statistical reliability.

4.3 Results Analysis

Table 1 reports task-level performance across three scenarios, and Figure 4 compares the operational profile of a GPT-5.4 cloud server against three localized box configurations running local LLMs.

Task-specific Performance. Across all nine metrics in Table 1, the best result is achieved by a local LLM in every case. The advantage over the non-LLM AutoML baseline is most visible on the cybersecurity forecasting task, where the average MAE of the three local LLMs is roughly 5× lower than AutoML's. Unlike AutoML, which optimizes only model choice and hyperparameters within a fixed pipeline topology, the ToolChain Tree Search revises every stage of the pipeline from preprocessing through evaluation, enabling end-to-end configurations that a narrower search cannot reach. Among the three, Qwen3.5-122B leads on five metrics, particularly on the manufacturing task, with DeepSeek-R1-70B leading on one, and Gemma-4-31B on three. The consistently strong performance together with the small cross-model variance (e.g., a coefficient of variation of only 2.5% on the manufacturing metrics), confirms that **TSAutoBox** is robust to LLM choice and generalizes across diverse domains, owing to its loosely coupled layered design.

Table 1: Task performance across three demonstration scenarios. Bold marks the best result per metric.

Method	Manufacturing			Cybersecurity			Healthcare		
	Aff-F1 ↑	AUC ↑	VUS ↑	MAE ↓	RMSE ↓	MASE ↓	MMD ↓	DS ↓	Down. ↑
Gemma-4-31B	0.902	0.837	0.886	**0.431**	**0.462**	0.697	0.006	**0.147**	-0.017
DeepSeek-R1-70B	0.903	0.909	0.938	0.435	0.467	0.704	0.003	0.208	**1.770**
Qwen3.5-122B	**0.909**	**0.928**	**0.946**	0.700	1.255	**0.244**	**0.001**	0.464	-3.835
GPT-5.4 API	0.903	0.921	0.945	0.688	1.229	0.798	0.057	0.179	0.006
AutoML (Non-LLM)	0.810	0.920	0.931	2.602	5.036	3.264	0.006	0.478	1.695

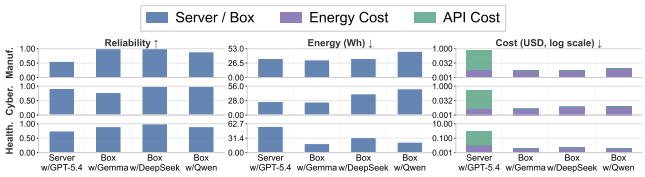

Figure 4: Operational comparisons across configurations.

Reliability, Energy Consumption, and Cost. Figure 4 shows that the box configuration is both more reliable and cheaper than the cloud server. Local inference via vLLM eliminates rate limits, network timeouts, and content-filter rejections, raising the average LLM-call success rate from 0.74 to 0.94. It also removes API fees entirely: the server path costs $0.72 per run on average, while each box run costs only $0.006 in electricity, a roughly 120× reduction. Energy consumption remains comparable across configurations (34.2 Wh vs. 39.4 Wh on average), though some local configurations consume slightly more energy due to the longer inference time of on-device LLMs, confirming that the localized box is a practical, cost-effective alternative to cloud-based deployment.

4.4 Case Study: Manufacturing Anomaly Detection on Box with Qwen3.5-122B

We now illustrate an end-to-end run on the time series anomaly detection task in a manufacturing factory using the Box deployment with Qwen3.5-122B as the local LLM backend in Figure 5.

Deployment and Data Ingestion. In this scenario, production-line sensors stream temperature, pressure, RPM, flow, and humidity readings through a PLC and SCADA system to the localized box, which runs **TSAutoBox** locally without cloud connectivity (Figure 5(a); physical prototype in (c)). The operator reviews the ingested time series data via the dataset preview panel (Figure 5(b.2)), specifies an anomaly-detection objective in natural language (Figure 5(b.1)), and launches the run.

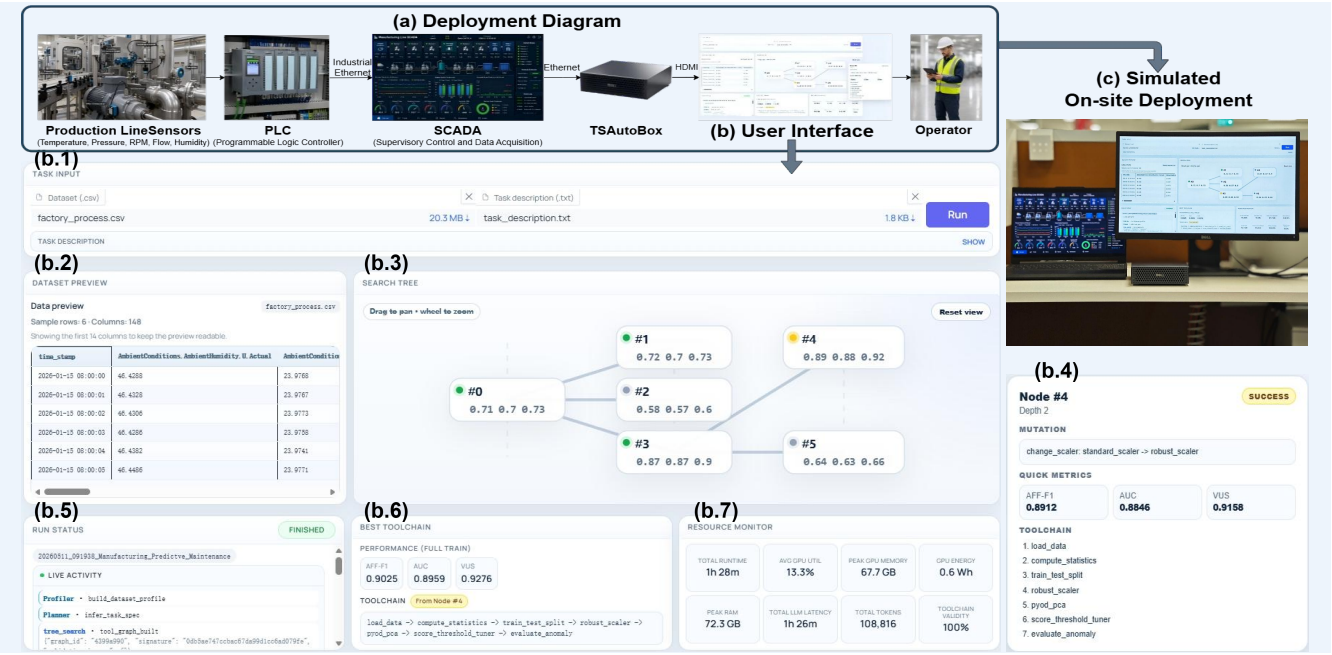

Figure 5: A case study on the time series anomaly detection task in a manufacturing factory. (a) Deployment: sensor data flows from the production line through PLC and SCADA to the localized box. (b) User interface showing the overall workflow. (c) Simulated on-site deployment of the TSAutoBox on an edge device.

Search and Execution. Once the run starts, the Orchestrator coordinates the three-phase workflow: the Profiler characterizes the input and the Planner compiles an initial ToolChain rooted in ECOD anomaly scoring. This root node (Node #0 in the Search Tree, Figure 5(b.3)) already achieves reasonable low-fidelity scores. The Reviewer diagnoses low recall as the main weakness and triggers detector substitution at the first expansion layer: Nodes #1-3 each swap in a different detector while keeping the preprocessing backbone fixed. Node #3, using PCA-based scoring, emerges as the strongest branch. Following the coarse-to-fine schedule described in §2.3, the second layer shifts from model-level changes to preprocessing refinements. Node #4 replaces the standard scaler with a robust scaler, better accommodating the heavy-tailed distributions typical of industrial sensor data, and achieves the highest low-fidelity scores (0.891, 0.885, 0.916, shown in Figure 5(b.4)). Node #5 instead inserts a differencing step, which degrades the score and is pruned. In total, six candidates are executed across two layers, producing a compact and interpretable search trace.

Result and Resource Profile. After full-fidelity retraining of the top candidates, the best pipeline (Node #4 in Figure 5(b.6)):

```
load_data → compute_statistics → train_test_split → robust_scaler
→ pyod_pca → score_threshold_tuner → evaluate_anomaly
```

achieves Aff-F1 = 0.903, AUC = 0.896, and VUS = 0.928. The resource monitor (Figure 5(b.7)) reports 100% pipeline validity. The live activity log (Figure 5(b.5)) surfaces every agent decision in real time, giving the operator full visibility into the automated process. The entire workflow runs on a single DGX Spark with no external API dependency, showing that **TSAutoBox** can deliver strong anomaly-detection pipelines in a privacy-sensitive industrial setting.

Interpretability and Traceability. Beyond the final pipeline, the interface exposes the search trace. Each Search Tree node records its ToolChain, mutation, and metrics, allowing the operator to understand what was selected and why alternatives were discarded. This transparency is essential in regulated industrial settings where operators must justify automated decisions to auditors or experts.

5 CONCLUSIONS

In this paper, we present **TSAutoBox**, a localized multi-agent system for industrial time-series task automation. It addresses the last-mile gap between model recommendation and deployable time-series solutions by combining typed ToolChain construction, Search Tree refinement, deterministic tool execution, and an interactive edge-resident deployment with local LLM inference. Across manufacturing anomaly detection, cybersecurity forecasting, and healthcare time-series generation, the localized box configurations achieve strong task quality against fixed-structure AutoML baselines while improving operational practicality. These results confirm that multi-agent automation is able to make time-series modeling more accurate, accessible, and deployable for privacy-sensitive and resource-constrained organizations.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Singapore, under its CyberSG R&D Programme (CRPO Award No: CRPO-GC5-NUS-006), the Ministry of Education, Singapore, under its MOE AcRF TIER 1 Grant (T1 251RES2517), and the National University of Singapore, School of Computing Seed Fund.

REFERENCES

[1] 2026. Prometheus: Monitoring system & time series database. https://github.com/prometheus/prometheus.

[2] 2026. World Average Electricity Prices. https://www.globalpetrolprices.com/electricity_prices/. Accessed: 2026-05-17.

[3] Ahmed Abdulaal and Tomer Lancewicki. 2021. Real-time synchronization in neural networks for multivariate time series anomaly detection. In *ICASSP*. 3570–3574.

[4] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-Generation Hyperparameter Optimization Framework. In *KDD*. 2623–2631.

[5] Yihao Ang, Yifan Bao, Qiang Huang, Anthony KH Tung, and Zhiyong Huang. 2024. TSGAssist: An interactive assistant harnessing LLMs and RAG for time series generation recommendations and benchmarking. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4309–4312.

[6] Yihao Ang, Yifan Bao, Lei Jiang, Jiajie Tao, Anthony KH Tung, Lukasz Szpruch, and Hao Ni. 2025. Structured Agentic Workflows for Financial Time-Series Modeling with LLMs and Reflective Feedback. In *ICAIF*.

[7] Yihao Ang, Qiang Huang, Yifan Bao, Anthony KH Tung, and Zhiyong Huang. 2023. TSGBench: Time Series Generation Benchmark. *Proceedings of the VLDB Endowment* 17, 3 (2023), 305–318.

[8] Yihao Ang, Qiang Huang, Anthony KH Tung, and Zhiyong Huang. 2023. A Stitch in Time Saves Nine: Enabling Early Anomaly Detection with Correlation Analysis. In *ICDE*. 1832–1845.

[9] Yihao Ang, Qiang Huang, Anthony KH Tung, and Zhiyong Huang. 2024. EADS: An early anomaly detection system for sensor-based multivariate time series. In *ICDE*. 5433–5436.

[10] Yihao Ang, Peicheng Yao, Yifan Bao, Yushuo Feng, Qiang Huang, Anthony KH Tung, and Zhiyong Huang. 2026. RFOD: Random Forest-based Outlier Detection for Tabular Data. In *ICDE*. 2192—2205.

[11] Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. 2025. Researchagent: Iterative research idea generation over scientific literature with large language models. In *NAACL*. 6709–6738.

[12] Yifan Bao, Yihao Ang, Qiang Huang, Anthony KH Tung, and Zhiyong Huang. 2024. Towards controllable time series generation. *arXiv preprint arXiv:2403.03698* (2024).

[13] Paul Boniol, Michele Linardi, Federico Roncallo, Themis Palpanas, Mohammed Meftah, and Emmanuel Remy. 2021. Unsupervised and scalable subsequence anomaly detection in large data series. *VLDBJ* 30, 6 (2021), 909–931.

[14] Paul Boniol, Mohammed Meftah, Emmanuel Remy, and Themis Palpanas. 2022. dCAM: Dimension-wise Class Activation Map for Explaining Multivariate Data Series Classification. In *SIGMOD*. 1175–1189.

[15] Paul Boniol and Themis Palpanas. 2020. Series2Graph: Graph-based Subsequence Anomaly Detection for Time Series. *Proc. VLDB Endow.* 13, 11 (2020), 1821–1834.

[16] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. 2021. SAND: Streaming Subsequence Anomaly Detection. *Proc. VLDB Endow.* 14, 10 (2021), 1717–1729.

[17] D. Campos and J. Bernardes. 2000. Cardiotocography. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C51S4N.

[18] Colin Catlin. 2025. Adaptive forecasting in dynamic markets: An evaluation of AutoTS within the M6 competition. *International Journal of Forecasting* (2025), 1485–1493.

[19] Docker, Inc. 2026. Define and run multi-container applications with Docker. https://github.com/docker/compose.

[20] Azul Garza and Renée Rosillo. 2025. TimeCopilot. *arXiv preprint arXiv:2509.00616* (2025).

[21] Grafana Labs. 2026. Grafana OSS: Visualization and Dashboarding Technology. https://grafana.com/oss/.

[22] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024. DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning. In *ICML*. 16813–16848.

[23] Julien Herzen, Francesco Lässig, Samuele Giuliano Piazzetta, Thomas Neuer, Léo Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, Maxime Dumonal, Jan Kościsz, Dennis Bader, Frédérick Gusset, Mounir Benheddi, Camila Williamson, Michal Kosinski, Matej Petrik, and Gaël Grosch. 2022. Darts: User-Friendly Modern Machine Learning for Time Series. *JMLR* (2022).

[24] Linghe Kong, Jinlin Tan, Junqin Huang, Guihai Chen, Shuaitian Wang, Xi Jin, Peng Zeng, Muhammad Khan, and Sajal K Das. 2022. Edge-computing-driven internet of things: A survey. *CSUR* 55, 8 (2022), 1–41.

[25] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *SOSP*.

[26] Callie C. Liao, Duoduo Liao, and Sai Surya Gadiraju. 2025. AgentMaster: A Multi-Agent Conversational Framework Using A2A and MCP Protocols for Multimodal Information Retrieval and Analysis. In *EMNLP*. 52–72.

[27] Qinghua Liu and John Paparrizos. 2024. The elephant in the room: Towards a reliable time-series anomaly detection benchmark. In *NeurIPS*. 108231–108261.

[28] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2020. The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting* (2020), 54–74.

[29] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *OSDI*.

[30] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 military communications and information systems conference (MilCIS)*. 1–6.

[31] Nour Moustafa and Jill Slay. 2016. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal: A Global Perspective* 25, 1-3 (2016), 18–31.

[32] 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*.

[33] Alexander Nikitin, Letizia Iannucci, and Samuel Kaski. 2024. TSGM: a flexible framework for generative modeling of synthetic time series. In *NeurIPS*. 129042–129061.

[34] NVIDIA. 2026. nvidia/Gemma-4-31B-IT-NVFP4 Model Card. https://huggingface.co/nvidia/Gemma-4-31B-IT-NVFP4. Hugging Face; accessed 2026-05-17.

[35] Ollama. 2026. deepseek-r1:70b-llama-distill-q4_K_M Model Entry. https://ollama.ac.cn/library/deepseek-r1:70b-llama-distill-q4_K_M. Ollama model library; accessed: 2026-05-17.

[36] OpenAI. 2026. GPT-5.4 model page. https://developers.openai.com/api/docs/models/gpt-5.4.

[37] OpenAI. 2026. OpenAI API Pricing. https://openai.com/api/pricing/. Accessed: 2026-05-17.

[38] John Paparrizos, Paul Boniol, Themis Palpanas, Ruey S. Tsay, Aaron J. Elmore, and Michael J. Franklin. 2022. Volume Under the Surface: A New Accuracy Evaluation Measure for Time-Series Anomaly Detection. *Proc. VLDB Endow.* 15, 11 (2022), 2774–2787.

[39] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S. Tsay, Themis Palpanas, and Michael J. Franklin. 2022. TSB-UAD: An End-to-End Benchmark Suite for Univariate Time-Series Anomaly Detection. *Proc. VLDB Endow.* 15, 8 (2022), 1697–1711.

[40] Oliver Roesler. 2013. EEG Eye State. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C57G7J.

[41] Sehyo. 2026. Sehyo/Qwen3.5-122B-A10B-NVFP4 Model Card. https://huggingface.co/Sehyo/Qwen3.5-122B-A10B-NVFP4. Hugging Face; accessed: 2026-05-17.

[42] Jack W Smith, James E Everhart, William C Dickson, William C Knowler, and Robert Scott Johannes. 1988. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Applications and Medical Care*. 261–265.

[43] Salvatore Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip Chan. 1999. KDD Cup 1999 Data. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C51C7N.

[44] supergus. 2026. Multistage Continuousflow Manufacturing Process. https://www.kaggle.com/datasets/supergus/multistage-continuousflow-manufacturing-process. Kaggle dataset; accessed: 2026-05-17.

[45] The PostgreSQL Global Development Group. 2026. PostgreSQL: The World's Most Advanced Open Source Relational Database. https://www.postgresql.org/.

[46] The Ray Team. 2026. Ray. https://www.ray.io/.

[47] the vLLM Team. 2026. vLLM. https://github.com/vllm-project/vllm.

[48] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *NeurIPS*. 22419–22430.

[49] Ruotian xie. 2024. Traffic datsets: Abilene, GEANT, TaxiBJ. https://dx.doi.org/10.21227/7x3c-5p06

[50] Chin-Chia Michael Yeh, Vivian Lai, Uday Singh Saini, Xiran Fan, Yujie Fan, Junpeng Wang, Xin Dai, and Yan Zheng. 2025. Empowering Time Series Forecasting with LLM-Agents. *arXiv preprint arXiv:2508.04231* (2025).

[51] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. 2019. Time-series generative adversarial networks. In *NeurIPS*.

[52] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are transformers effective for time series forecasting?. In *AAAI*. 11121–11128.

[53] Yedi Zhang, Haoyu Wang, Xianglin Yang, Jin Song Dong, and Jun Sun. 2026. LLM-enabled Applications Require System-Level Threat Monitoring. *arXiv preprint arXiv:2602.19844* (2026).

[54] Yue Zhao, Zain Nasrullah, and Zheng Li. 2019. Pyod: A python toolbox for scalable outlier detection. *JMLR* 20, 96 (2019), 1–7.