TSCOMP-AGENT: AUTOMATED DECOMPOSITION AND REASSEMBLY OF TIME SERIES FORECASTING MODELS VIA LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

The time-series forecasting literature is expanding rapidly, with new methods spanning an increasingly broad range of architectural paradigms. Many advances arise from specific mechanisms ranging from series normalization and decomposition to specialized architecture design, whose effectiveness often varies across diverse forecasting tasks. This motivates the question of whether effective mechanisms from different models can be identified as reusable components and composed into stronger forecasting architectures. Recent studies investigate this modular perspective, yet remain confined to manual expert curation, rendering continuous expansion unsustainable as the forecasting literature rapidly evolves.

We introduce TSCOMP-AGENT, an LLM-based multi-agent framework that automatically decomposes forecasting models into *verified* and *reusable* components for automated reassembly. Building on a taxonomy-structured knowledge base, TSCOMP-AGENT synthesizes proposals from independent exploration agents into an executable plan. The implementation agent then translates this plan into modular components through iterative review and repair. Global consolidation finally verifies their compatibility and reusability before admission into the shared component pool. Experimental results verify that TSCOMP-AGENT reliably extends the component pool from diverse forecasting models across varied LLM instantiations. Models reassembled from this pool match or outperform human-designed baselines on multiple datasets, even under historical-cutoff and backbone-constrained settings, providing a scalable mechanism to accumulate design knowledge of time-series forecasting and facilitate automated architecture discovery.

1 INTRODUCTION

Multivariate time-series forecasting (MTSF) underpins critical decision-making for energy systems, financial markets, healthcare, and environmental monitoring (Deb et al., 2017; Sezer et al., 2020; Kaushik et al., 2020; Du et al., 2021). Recent advances in these domains span specialized deep architectures (Nie et al., 2023; Wang et al., 2024a; Lin et al., 2024) and emerging large time-series models (Zhou et al., 2023; Jin et al., 2024; Liu et al., 2024b; Shi et al., 2025). Conventionally, models from both families are introduced as holistic architectures and evaluated through model-level comparisons. However, the "no-free-lunch" theorem and extensive empirical benchmarks indicate that no monolithic architecture consistently dominates (Qiu et al., 2024; Liu et al., 2025b; Liang et al., 2026). Instead, forecasting gains typically stem from stage-specific components, such as series tokenization or attention, which remain obscured when models are evaluated as indivisible units.

These limitations have driven an emerging shift from holistic model evaluation toward component-level analysis of forecasting systems. Recent work has pursued this perspective through complementary objectives, including systematic benchmarking of component choices (Liang et al., 2026; Zhao et al., 2026), performance attribution across modular pipelines (Wang et al., 2026), and automated model adaptation through component or transformation selection (Qiu et al., 2026; Lyu et al., 2025). While promising, existing component spaces remain static and confined to manual expert curation. To incorporate each new model, researchers must manually reverse-engineer disparate codebases, decoupling intertwined mechanisms to re-implement them under unified interfaces. As new fore-

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

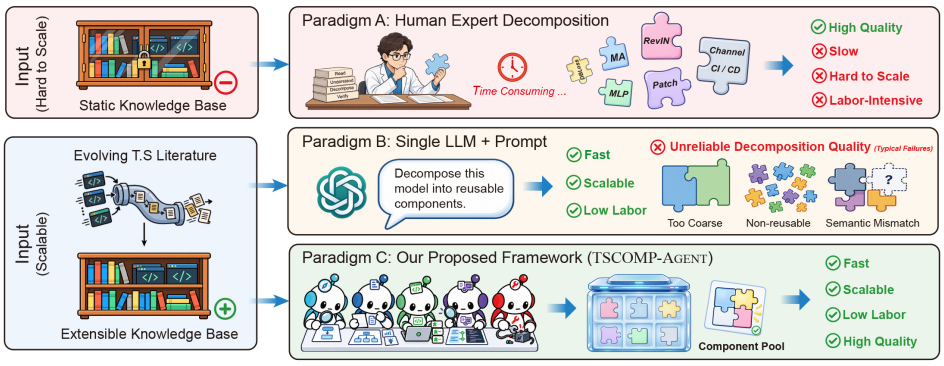


Figure 1: Model decomposition paradigms.

casting architectures emerge continuously, this labor-intensive process cannot scale, leaving existing component spaces lagging behind the evolving literature.

However, expanding such component spaces via autonomous agents remains non-trivial (Fig. 1): neither advanced coding agents nor generic multi-agent systems (MAS) can reliably deconstruct heterogeneous forecasting models. Algorithmic modules are tightly coupled with framework-specific implementations, preventing existing agents from identifying well-defined component boundaries. Without domain-specific tensor contracts and verification protocols, extracted modules would suffer from interface mismatches and semantic drift during cross-model recombination.

We introduce TSCOMP-AGENT, an LLM-based multi-agent framework for the automated decomposition and reassembly of heterogeneous time-series forecasting models. For each model, multiple exploration agents independently probe candidate boundaries, followed by a plan synthesizer that refines them to guide the implementer with plan-consistency repair. TSCOMP-AGENT schedules these pipelines in parallel batches, systematically accumulating reusable components drawn from diverse model families. Finally, global consolidation resolves interface conflicts via workspace preflight and combination smoke tests, establishing an extensible component space for automated reassembly. In summary, our principal contributions are as follows:

- We propose TSCOMP-AGENT, an LLM-based multi-agent framework that decomposes heterogeneous forecasting models by jointly reasoning over papers, source code, and execution evidence. TSCOMP-AGENT orchestrates multi-agent exploration, review-guided implementation, and global assembly verification to establish an extensible component space for automated model assembly.
- We systematically validate TSCOMP-AGENT through stage-wise ablations and agent harness comparisons. Our collaborative workflow consistently outperforms manual curation and single-agent baselines across diverse LLMs, while generalizing to held-out unseen models.
- Experimental results show that reassembled architectures match or outperform SOTA baselines, maintaining this superiority under historical cutoff constraints and across backbone-constrained search. To support community research, we open-source TSCOMP-AGENT and its extensible component library to support continuous modular benchmarking and automated discovery.

2 Related Work

2.1 Time-Series Forecasting and Model Reassembly

Multivariate time-series forecasting continues to evolve rapidly, with recent models spanning lightweight MLP models (Zeng et al., 2023; Wang et al., 2024a; Lin et al., 2024; Yue et al., 2025), Transformer-based architectures (Wang et al., 2024b; Liu et al., 2025a), graph-based methods (Li et al., 2025; Hu et al., 2025b), and state-space approaches (Peng et al., 2025; Dama et al., 2025). Concurrently, large time-series models have gained prominence, including LLM-based methods (Zhou et al., 2023; Jin et al., 2024) and pretrained time-series foundation models (TSFMs) (Liu et al., 2024b; Das et al., 2024; Shi et al., 2025). However, the "no-free-lunch" theorem indicates that no monolithic architecture consistently dominates across diverse time-series forecasting tasks (Wang

et al., 2024c; Shao et al., 2024; Qiu et al., 2024). Instead, forecasting gains typically stem from localized mechanisms tailored to specific pipeline stages, including tokenization, channel interaction, periodic modeling, spatial correlation, and orthogonal transformation (Nie et al., 2023; Liu et al., 2024a; Lin et al., 2024; Si et al., 2025; Yue et al., 2025). This suggests that effective forecasting knowledge resides in modular components rather than rigid, monolithic model boundaries.

Motivated by this insight, recent studies investigate forecasting models through fine-grained modular analysis. TSCOMP (Liang et al., 2026) decomposes models into reusable components for analysis and data-adaptive model construction; TimeRecipe (Zhao et al., 2026) benchmarks module combinations for architecture recommendation; and CombinationTS (Wang et al., 2026) evaluates architectural choices through probabilistic attribution. TATO (Qiu et al., 2026) and AutoSTF (Lyu et al., 2025) further automate design within predefined operator spaces. However, these component spaces remain static and manually curated, making them increasingly costly to maintain as new forecasting models continue to emerge. TSCOMP-AGENT addresses this limitation with an LLM Agent-driven workflow that automatically extracts, verifies, and integrates reusable components from newly emerging models, allowing the shared component pool to continually expand and support cross-model reassembly.

2.2 AGENTIC SYSTEMS FOR COMPLEX TASK AUTOMATION

LLM-based agentic systems have been widely adopted to automate complex tasks that require multi-step planning, tool coordination, and iterative refinement. **General-purpose** agentic systems enable agents to generalize across diverse coding tasks by combining standardized interfaces for code editing, command execution, and environment interaction (Yang et al., 2024; Wang et al., 2025b) with ReAct-style reasoning and action (Yao et al., 2023). However, complex long-horizon tasks often require explicit workflow orchestration. Orchestration frameworks such as LangGraph (LangChain Team, 2024) organize agent interactions and tool calls into structured workflows, while subsequent work explores dynamic workflow adaptation and automatic workflow design (Anthropic, 2026; Niu et al., 2025; Hu et al., 2025a; Zhang et al., 2025). Together, these efforts improve the flexibility, controllability, and adaptability of agentic systems for complex task execution.

In complex downstream scenarios, however, general-purpose frameworks are often too unconstrained to consistently deliver stable and reproducible results. **Task-specific** agentic systems address this need through structured workflow design tailored to the target task. For example, MetaGPT (Hong et al., 2024) and ChatDev (Qian et al., 2024) assign specialized agent roles that mirror established software development processes. Other systems organize work into domain-specific stages with explicit dependencies among planning, experimentation, and reporting (Hong et al., 2025; Schmidgall et al., 2025; Seo et al., 2026). Further work leverages execution-driven feedback, where test outcomes or runtime states guide iterative refinement (Ridnik et al., 2024; Zhong et al., 2024; Han et al., 2026). TSCOMP-AGENT applies this principle to forecasting model decomposition as a *stage-wise verified component decomposition framework*, constructing a reusable forecasting component space that grows as new models emerge.

3 TSCOMP-AGENT FRAMEWORK

3.1 PROBLEM FORMULATION

Following standard multivariate time-series forecasting formulations (Zhou et al., 2021; Nie et al., 2023; Liu et al., 2024a), let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_L]^\top \in \mathbb{R}^{L \times C}$ denote a historical sequence of L lookback steps across C variates. Given $\mathbf{X}$, the forecasting objective is to predict future values $\mathbf{Y} = [\mathbf{x}_{L+1}, \ldots, \mathbf{x}_{L+H}]^\top \in \mathbb{R}^{H \times C}$ over an H-step horizon. A conventional model M implements a predictor $f_M : \mathbb{R}^{L \times C} \to \mathbb{R}^{H \times C}$ with parameters Θ_M, trained via empirical risk minimization:

$$\widehat{\mathbf{Y}} = f_M(\mathbf{X}; \Theta_M), \qquad \min_{\Theta_M} \mathbb{E}_{(\mathbf{X},\mathbf{Y})\sim\mathcal{D}} \left[\mathcal{L}(f_M(\mathbf{X}; \Theta_M), \mathbf{Y}) \right], \tag{1}$$

where $\mathcal{L}(\cdot, \cdot)$ evaluates forecasting error, typically instantiated as Mean Squared Error (MSE).

Rather than treating f_M as an indivisible holistic architecture, we formalize the forecasting pipeline via a standardized modular schema (detailed in Appendix C). Each consecutive stage (i.e., from series normalization and tokenization to the final prediction) defines a functional slot $s_k \in \mathcal{S} = \{s_1, \ldots, s_K\}$ governed by a standardized tensor contract. Given N heterogeneous source models

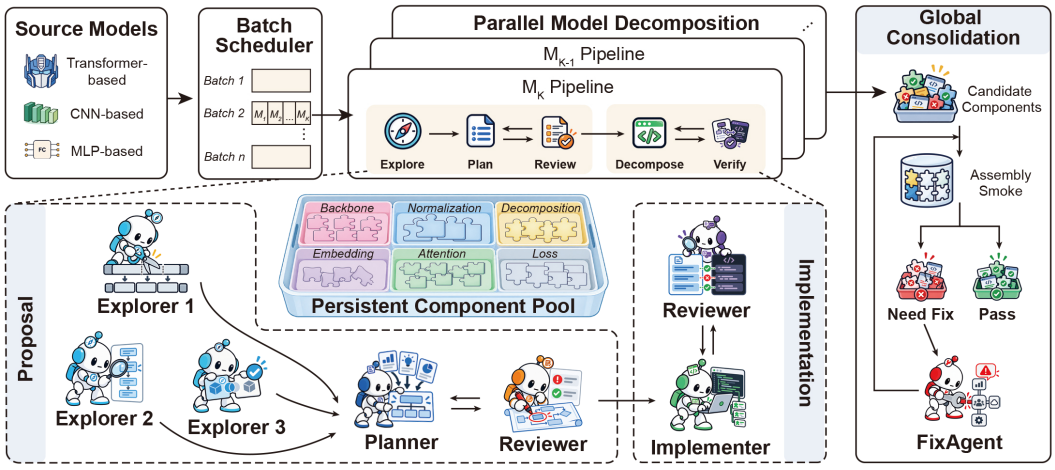


Figure 2: TSCOMP-AGENT framework. Each source model passes through decomposition proposal, component implementation, and global consolidation, yielding the final pool of reusable components.

$\mathcal{M} = \{M_1, \ldots, M_N\}$, our agentic framework parses raw code, maps extracted components to functional slots, and verifies contract alignment. This extraction yields a verified candidate pool $\mathcal{C}_k$ for each slot s_k, defining an extensible component space $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_K$. An assembled predictor $f_{\mathbf{c}}$ is parameterized by a configuration $\mathbf{c} = (c_1, \ldots, c_K) \in \mathcal{C}$, formalizing model discovery as combinatorial optimization:

$$\mathbf{c}^\star = \arg\min_{\mathbf{c}\in\mathcal{C}} \mathcal{L}_{\text{val}}(f_{\mathbf{c}}(\mathbf{X}; \Theta_{\mathbf{c}}), \mathbf{Y}), \tag{2}$$

where $\Theta_{\mathbf{c}}$ denotes the trainable weights of the assembled architecture. Because $\mathcal{C}$ is exponentially large and evaluating $\mathcal{L}_{\text{val}}$ requires full training, Eq. 2 is typically resolved via sample-efficient search such as Bayesian optimization. This modular formulation transforms monolithic models into the extensible space $\mathcal{C}$, enabling component-level analysis and automated cross-model recombination.

3.2 TSCOMP-AGENT DECOMPOSITION ARCHITECTURE

TSCOMP-AGENT constructs an extensible component pool via three stages: decomposition proposal, component implementation, and global consolidation (Fig. 2), as formalized in Algorithm 1. Source models are drawn from a curated knowledge base linking forecasting literature with verified codebases (see Appendix A for details). To automate this pipeline, TSCOMP-AGENT orchestrates specialized **LLM Agents** with deterministic verification harnesses. For each source model, exploration agents analyze papers and code, guiding plan proposal to establish explicit component boundaries. Subsequently, component implementation translates the proposed design into modular code through iterative self-repair. Finally, global consolidation verifies all registered candidates in assembled architectures, admitting verified components to the component pool.

Algorithm 1 Verification-Guided Decomposition

Require: Source models $\{M_i\}_{i=1}^N$; initial component pool $\mathcal{C}_0$; proposal-version, implementation-attempt, and global-repair limits R_p, R_{impl}, and $R_{\mathcal{G}}$, respectively.

$\mathcal{W} \leftarrow \text{Initialize}(\mathcal{C}_0)$
for $i = 1, \ldots, N$ **do**
  $\mathcal{C}_i \leftarrow \text{Snapshot}(\mathcal{W})$;
  $E_i \leftarrow \textbf{Agent}_{\text{exp}}(M_i)$
  $P_i \leftarrow \text{RefinePlan}(M_i, E_i, \mathcal{C}_i; R_p)$ ▷ Eq. 3
  $D_i \leftarrow \text{RefineCode}(P_i; R_{\text{impl}})$ ▷ Eq. 4
  $\text{Register}(D_i, \mathcal{W})$
end for
$\mathcal{C}_{\text{cand}} \leftarrow \bigcup_{i=1}^N \text{Registered}(D_i)$
$\mathcal{S} \leftarrow \text{ExecCheck}(\mathcal{C}_{\text{cand}}; \mathcal{W})$
while $\mathcal{S}$ admits an eligible resolution **do**
  $(\mathcal{C}_{\text{cand}}, \mathcal{W}) \leftarrow \text{Resolve}(\mathcal{C}_{\text{cand}}, \mathcal{S}; \mathcal{W})$ ▷ Eq. 5
  $\mathcal{S} \leftarrow \text{ExecCheck}(\mathcal{C}_{\text{cand}}; \mathcal{W})$
end while
return $\mathcal{C}^\star = \mathcal{C}_0 \cup \text{Verified}(\mathcal{C}_{\text{cand}}; \mathcal{W})$

Decomposition Proposal. The proposal stage analyzes algorithmic mechanisms of the source model and formalizes their modular boundaries. For each model M_i, exploration agent $\textbf{Agent}_{\text{exp}}$

inspects its paper and codebase to gather architectural evidence $E_i = \mathbf{Agent}_{\mathsf{exp}}(M_i)$. Conditioned on E_i and current pool snapshot $\mathcal{C}_i$, the planner initializes proposal $P_i^{(0)} = \mathbf{Agent}_{\mathsf{plan}}(E_i, \mathcal{C}_i)$ and then iteratively refines it using feedback from plan reviewer $\mathbf{Agent}_{\mathsf{rev}}^{\mathrm{plan}}$:

$$F_{\mathrm{plan}}^{(r)} = \mathbf{Agent}_{\mathsf{rev}}^{\mathrm{plan}}(P_i^{(r)}, M_i, \mathcal{C}_i), \qquad P_i^{(r+1)} = \mathbf{Agent}_{\mathsf{plan}}(P_i^{(r)}, E_i, F_{\mathrm{plan}}^{(r)}). \tag{3}$$

Refinement terminates once reviewer concerns are resolved or the iteration limit is reached. The finalized proposal P_i specifies component boundaries and contracts for subsequent decomposition.

Component Implementation. To translate proposal P_i into verified modular code, implementer $\mathbf{Agent}_{\mathsf{imp}}$ generates initial candidate $D_i^{(0)} = \mathbf{Agent}_{\mathsf{imp}}(P_i)$. Each attempt pairs deterministic checks with semantic review: syntax and contract checks verify execution, while reviewer $\mathbf{Agent}_{\mathsf{rev}}^{\mathrm{code}}$ diagnoses plan-code consistency to guide iterative repair:

$$F_{\mathrm{code}}^{(r)} = \mathbf{Agent}_{\mathsf{rev}}^{\mathrm{code}}(D_i^{(r)}, P_i), \qquad D_i^{(r+1)} = \mathbf{Agent}_{\mathsf{imp}}(D_i^{(r)}, P_i, F_{\mathrm{code}}^{(r)}). \tag{4}$$

Iterative repair proceeds until the candidate satisfies all contract checks or the maximum iteration limit is reached. Verified implementations enter the shared workspace across parallel batches of source models, updating pool snapshot $\mathcal{C}_i$ to guide subsequent decompositions.

Global Consolidation. Once all source models complete decomposition, their registered component candidates form initial candidate set $\mathcal{C}_{\mathrm{cand}}^{(0)}$. Global consolidation then verifies these candidates within complete assembled architectures and resolves cross-model compatibility failures. At check round r, execution checks in shared workspace $\mathcal{W}^{(r)}$ generate runtime failure signatures $\mathcal{S}^{(r)}$, guiding conflict resolution by integration agent $\mathbf{Agent}_{\mathsf{integ}}$:

$$\mathcal{S}^{(r)} = \mathrm{ExecCheck}(\mathcal{C}_{\mathrm{cand}}^{(r)}, \mathcal{W}^{(r)}), \quad (\mathcal{C}_{\mathrm{cand}}^{(r+1)}, \mathcal{W}^{(r+1)}) = \mathbf{Agent}_{\mathsf{integ}}(\mathcal{C}_{\mathrm{cand}}^{(r)}, \mathcal{S}^{(r)}, \mathcal{W}^{(r)}). \tag{5}$$

Repair proceeds up to a maximum iteration limit, halting early if consecutive checks yield identical failure signatures. When a shared error causes collective failure, failure attribution isolates the offending component without consuming the repair budget (Appendix D.5). Upon termination, verified components are admitted to the global pool $\mathcal{C}^\star = \mathcal{C}_0 \cup \mathrm{Verified}(\mathcal{C}_{\mathrm{cand}}; \mathcal{W})$, excluding candidates with unresolved failures from model reassembly.

3.3 Component Assembly and Model Discovery

With verified component space $\mathcal{C}^\star$ established, we recombine extracted modules across functional slots to instantiate novel forecasting architectures. Each configuration $\mathbf{c} = (c_1, \ldots, c_K) \in \mathcal{C}^\star$ assigns active component $c_k \in \mathcal{C}_k^\star$ to slot s_k. The framework automatically assembles the selected modules into an end-to-end trainable predictor $f_{\mathbf{c}}(\mathbf{X}; \Theta_{\mathbf{c}})$. Because evaluating validation loss requires full parameter training, exhaustive search over the component pool is computationally prohibitive. Therefore, we randomly sample components to prune underperforming model combinations, initializing sequential Bayesian optimization via Tree-structured Parzen Estimators (TPE) (Bergstra et al., 2011). At trial t, history $\mathcal{H}_{t-1} = \{(\mathbf{c}^{(\tau)}, e^{(\tau)})\}_{\tau=1}^{t-1}$ tracks past validation errors $e^{(\tau)}$. Given threshold $e^\star$, TPE partitions trials into two likelihood distributions using slot-wise estimators ℓ_k and g_k:

$$p(\mathbf{c} \mid e) = \begin{cases} \ell(\mathbf{c}) = \prod_{k=1}^{K} \ell_k(c_k), & \text{if } e < e^\star, \\ g(\mathbf{c}) = \prod_{k=1}^{K} g_k(c_k), & \text{if } e \geq e^\star. \end{cases} \tag{6}$$

Under the Expected Improvement criterion (Brochu et al., 2010), each trial t samples candidate configuration $\mathbf{c}^{(t)} = \arg\max_{\mathbf{c} \in \mathcal{C}_{\mathrm{valid}}^\star} \ell(\mathbf{c})/g(\mathbf{c})$. Here, valid subspace $\mathcal{C}_{\mathrm{valid}}^\star \subseteq \mathcal{C}^\star$ restricts candidate choices to interface-compatible architectures prior to execution, while the likelihood ratio favors components prevalent in superior models. After search budget T terminates, we select the top-M configurations with the lowest validation errors from $\mathcal{H}_T$ and ensemble their predictions for final evaluation on testing data.

4 Evaluating Automated Decomposition

4.1 Settings and Metrics

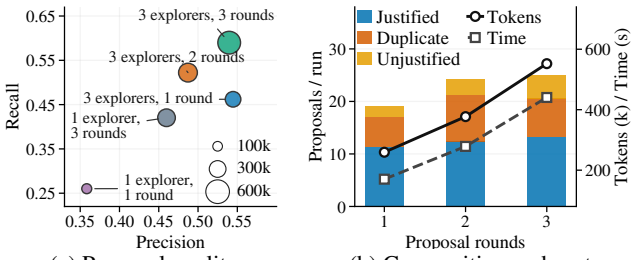
Settings. We evaluate automated decomposition by examining how effectively it extends an existing component pool with reusable components from forecasting models. We initialize $\mathcal{C}_0$ with the human-curated component pool from Liang et al. (2026), constructed through expert decomposition of forecasting models. We evaluate decomposition on ten architectures spanning diverse modeling paradigms, including linear, Transformer, and state-space models. Appendix D.2 details the specifications, selection criteria, and source codebases of each model.

Metrics. We measure decomposition quality along two dimensions: what the pipeline extracts and how well it extracts it. **Count** and **Novelty** measure the number of accepted components and the proportion of mechanisms new to $\mathcal{C}_0$. **Consistency** and **Compatibility** then ask whether each component maintains consistency with its source logic and executes successfully across functional slots under tensor shape contracts. Computational cost is measured via raw **Token** consumption and wall-clock **Time**. Formal definitions for all six metrics appear in Appendix D.1.

4.2 Stage-Wise Design Analysis

To understand how design choices affect each stage, we compare alternative configurations using metrics tailored to its role. Appendix D.3.1 details these stage-specific diagnostic metrics.

Proposal Stage. The proposal stage identifies components through independent exploration and iterative refinement. Proposal quality is evaluated using *Recall* for target coverage and *Precision* the proportion of proposals matching the reference targets. To assess **exploration breadth**, we compare one and three explorers under matched refinement budgets. As shown in Figure 3a, three explorers consistently achieve higher recall and precision than a single explorer, indicating that independent exploration improves component coverage. The effect of **refinement depth** is examined by fixing three explorers and increasing the proposal budget from one to three rounds. Recall rises from 0.46 to 0.59, showing that iterative refinement recovers additional target components. However, Figure 3b shows that the additional proposals also include more duplicates of components already in the pool, while token consumption and wall-clock time increase substantially. Despite these costs, we adopt three explorers with up to three refinement rounds to achieve more complete component coverage.



(a) Proposal quality (b) Composition and cost

Figure 3: Proposal stage quality and cost analysis.

Implementation Stage. The implementation stage translates decomposition plans into executable components and iteratively refines them through review and repair. We use *Plans Implemented Rate* to measure the fraction of planned components successfully implemented. In our default setting, **Standard Review** employs a reviewer agent to check whether the implementation follows the proposed design and satisfies the required contracts, and the resulting feedback guides iterative repair by the implementer. The comparison with **No Review** shows that review mainly improves *Consistency* rather than implementation success. Without review, *Implemented Rate* increases slightly from 0.32 to 0.36, while *Consistency* decreases from 0.69 to 0.63, suggesting that the resulting components are more likely to deviate from the source model design. The comparison with **Exhaustive Review** shows that stricter verification improves component quality but makes it harder to retain enough components. Extending review to engineering validity, source model coverage, component reusability, and runtime pluggability increases *Consistency*, yet substantially lowers *Implemented Rate* and *Novelty* while requiring much

Table 1: Implementation-stage analysis (5-run means).

Configuration	Impl. Rate	Consistency	Count	Novelty	Tokens (k/model)
Standard Review	0.32	0.69	7.0	0.50	41
No Review	0.36	0.63	7.8	0.52	11
Exhaustive Review	0.16	0.87	3.4	0.22	133
Local Memory	0.36	0.79	6.8	0.40	45

more *Token*. This suggests that overly strict verification can filter out valid and novel components along with low-quality ones. **Local Memory** reveals a different trade-off between reliability and novelty. Reusing experience from previous decomposition runs increases *Implemented Rate* and *Consistency*, while *Novelty* decreases from 0.50 to 0.40. Given our emphasis on component-space expansion, we therefore omit Local Memory from the main setting, while retaining it as an alternative configuration when reliability is prioritized.

Global Consolidation Stage. Global integration verifies implemented components in a shared workspace and repairs failures that emerge during integration. Our default setting uses full-context repair for up to two rounds, where the integration agent receives complete failure information after deterministic checks and fixes. The effect of **global repair** is evident from the comparison between no repair and one or two repair rounds. As shown in Figure 4, agent repair recovers components that fail the initial integration check, allowing them to pass verification, while additional repair rounds recover further components. The importance of **failure disclosure** is revealed by restricting the failure information available to the Fix Agent under the same two-round budget. This substantially reduces the *Agent-Rescued* share, indicating that complete failure information is important for effective diagnosis and repair.

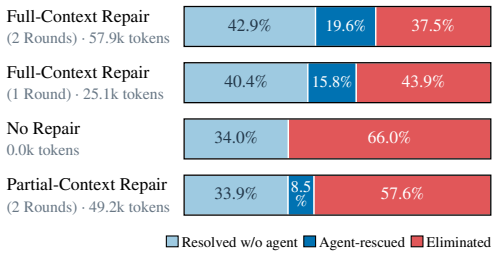


Figure 4: Composition of integration outcome.

4.3 Framework Effectiveness Across LLM Backbones

Figure 5 compares three decomposition methods across different LLM backbones. TSCOMP-Agent shows **strong overall decomposition effectiveness** across all backbones, extracting more components with stronger consistency and novelty while maintaining high compatibility at comparable Token and Time costs. The advantage remains pronounced for a weaker backbone (MiniMax-M2.7): Prompt Only and Dynamic Workflow extract few reusable components, whereas TSCOMP-Agent remains effective, demonstrating **robustness to weaker LLM backbones**. More importantly, comparing the weaker backbone with TSCOMP-Agent against a stronger backbone (GPT-5.6-sol) with simpler workflows shows competitive or stronger outcomes in Count, Consistency, and Novelty with fewer tokens and comparable wall-clock time, suggesting that **framework design can partially offset differences in backbone capability**.

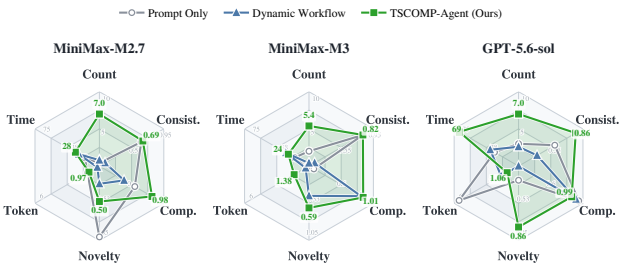


Figure 5: Workflow comparisons across LLM backbones. Token and Time measure cost; lower is better.

4.4 Generalization to Newly Introduced Models

We further evaluate whether TSCOMP-Agent can support the intended continual expansion of the component space as new forecasting models emerge. The newly introduced model set is described in Appendix D.2. As shown in Table 2, newly introduced models are more challenging to decompose, with lower *Consistency* and *Compatibility*. Despite this increased difficulty, they contribute more accepted components with higher Novelty, indicating that new models continue to introduce mechanisms beyond those already represented in the existing component space. These results show that TSCOMP-Agent can continue extracting compatible and novel components from newly emerging models, supporting **continual expansion of the component space**.

Table 2: Comparison with newly introduced models.

Models	Count	Novelty	Consistency	Compatibility	Token (M)	Time (min)
Previously analyzed	7.0	0.498	0.693	0.980	1.02	27
Newly introduced	9.0	0.578	0.583	0.970	1.19	31

5 EVALUATING COMPONENT REASSEMBLY

Having established reliable component extraction, we evaluate whether the resulting component pool supports cross-model reassembly. The component pool is constructed from the 30 models, and we evaluate full-pool, historical-cutoff, and backbone-constrained settings, details in Appendices E.

5.1 EXPERIMENTAL SETUP

Datasets and Comparison Baselines. We evaluate on eight widely used time-series forecasting datasets: ETTh1, ETTh2, ETTm1, ETTm2, Weather, ECL, Traffic, and ILI, using MSE and MAE. Following standard forecasting protocols, we use prediction horizons $H \in \{96, 192, 336, 720\}$ for all datasets except ILI, for which $H \in \{24, 36, 48, 60\}$. Dataset statistics are provided in Appendix E.1.

For 13 reference models reproducible from the component pool, including OLinear (Yue et al., 2025), RAFT (Han et al., 2025), TimeMixer (Wang et al., 2024a), iTransformer (Liu et al., 2024a), DLinear (Zeng et al., 2023), Mamba (Gu & Dao, 2023), PatchTST (Nie et al., 2023), and SegRNN (Lin et al., 2025), we reproduce their component configurations and evaluate them under the same experimental settings as automated reassembly. We further compare against automated forecasting methods: PatchLinear from CombinationTS (Wang et al., 2026), TATO (Qiu et al., 2026), AutoGluon (Shchur et al., 2023), and AutoTS (Catlin, 2020). Details are provided in Appendix E.2.

Assembly Search. Direct search over the full component pool is challenging because validation performance may not reliably rank the strongest configurations. Figure 6 illustrates this issue on ILI with $H = 48$: although stronger configurations are present in the component pool, validation-guided TPE does not reliably recover them, whereas test-guided TPE, used only as an oracle reference, can. We therefore adopt a two-stage search strategy: 100 random configurations are used for screening and pruning, followed by 100 TPE trials over the reduced pool. The top-M configurations ($M = 5$ by default) with lowest validation MSE are ensembled for final evaluation. All search and selection decisions use validation performance only; details are provided in Appendix E.3.

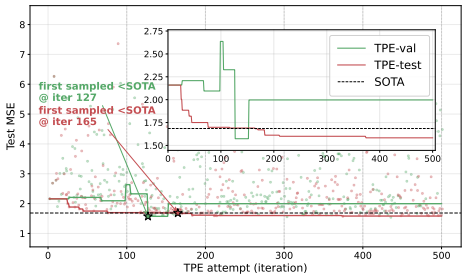


Figure 6: Diagnostic TPE trajectories.

5.2 EFFECTIVENESS OF CROSS-MODEL REASSEMBLY

Table 3 compares TSCOMP-AGENT with reassembled forecasting references, with results averaged across prediction horizons. *Ours* denotes the configuration selected by validation performance, whereas *Ours (Oracle)* reports the lowest test MSE among all successfully evaluated configurations and is reported only to characterize the performance potential of the evaluated component pool. The validation-selected TSCOMP-AGENT achieves lower MSE and MAE than all reference models on ETTh2, ETTm1, ETTm2, Weather, ECL, and Traffic, while ETTh1 and ILI remain more challenging. These results show that cross-model reassembly can produce configurations that outperform their corresponding reassembled references across diverse forecasting settings.

Table 3: Performance comparison with reassembled forecasting baselines.

	Ours (oracle)		**Ours**		**OLinear**		**RAFT**		**TimeMixer**		**iTransformer**		**DLinear**		**Mamba**		**PatchTST**		**SegRNN**	
Dataset	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
ETTh1	**0.419**	**0.432**	0.475	0.478	0.458	0.451	0.465	0.450	0.541	0.502	0.464	0.456	0.504	0.486	0.488	0.467	0.450	0.442	**0.446**	**0.439**
ETTh2	**0.348**	**0.392**	**0.363**	**0.392**	0.394	0.414	0.384	0.406	0.439	0.439	0.382	0.406	0.587	0.540	0.428	0.489	0.375	0.399	0.378	0.402
ETTm1	**0.343**	**0.375**	**0.356**	**0.381**	0.405	0.407	0.417	0.409	0.400	0.415	0.409	0.413	0.420	0.423	0.418	0.420	0.391	0.400	0.395	0.405
ETTm2	**0.249**	**0.309**	**0.251**	**0.307**	0.292	0.332	0.287	0.331	0.326	0.359	0.291	0.336	0.374	0.415	0.293	0.334	0.280	0.325	0.281	0.328
Weather	**0.220**	**0.262**	**0.220**	**0.264**	0.246	0.273	0.276	0.293	0.257	0.289	0.268	0.288	0.260	0.304	0.250	0.279	0.262	0.282	0.266	0.284
ECL	**0.167**	**0.263**	**0.168**	**0.261**	0.195	0.283	0.230	0.312	0.242	0.337	0.227	0.311	0.226	0.317	0.250	0.341	0.218	0.306	0.221	0.309
Traffic	**0.420**	**0.297**	**0.433**	**0.284**	0.503	0.342	0.670	0.412	0.747	0.434	0.624	0.410	0.684	0.429	0.780	0.459	0.569	0.368	0.588	0.377
illness	**1.613**	**0.810**	2.691	1.061	**2.177**	**1.005**	3.409	1.313	4.086	1.476	2.695	1.136	5.151	1.608	4.099	1.464	2.292	1.013	2.853	1.182

A similar pattern is observed against automated forecasting methods in Table 4. TSCOMP-AGENT achieves the best MSE and MAE on ETTh2, ETTm1, ETTm2, and Weather, but trails the strongest baseline on ETTh1 and ILI. The *Ours (Oracle)* results further confirm that the component pool contains configurations with substantial performance potential across all evaluated datasets.

Table 4: Performance comparison with AutoML and external baselines.

	Ours (oracle)		Ours		PatchLinear		TATO		AutoGluon		AutoTS	
Dataset	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	**0.419**	**0.432**	0.475	0.478	**0.442**	**0.435**	0.698	0.521	0.503	0.473	0.981	0.610
ETTh2	**0.348**	**0.392**	**0.363**	**0.392**	0.399	0.416	0.433	0.430	0.419	0.430	0.589	0.488
ETTm1	**0.343**	**0.375**	**0.356**	**0.381**	0.420	0.413	0.807	0.579	0.482	0.408	0.744	0.546
ETTm2	**0.249**	**0.309**	**0.251**	**0.307**	0.287	0.329	0.328	0.358	0.273	0.337	0.392	0.389
Weather	**0.220**	**0.262**	**0.220**	**0.264**	0.271	0.291	0.292	0.310	0.236	0.270	0.519	0.372
ILI	**1.613**	**0.810**	2.691	1.061	**2.148**	0.996	2.631	1.057	2.271	**0.979**	2.533	1.049

5.3 REASSEMBLY UNDER HISTORICAL CUTOFFS

For each cutoff year $Y \in \{2021, \ldots, 2025\}$, we restrict both the component pool $\mathcal{C}_{\leq Y}$ and the comparison models to methods available no later than Y. This year-matched setting evaluates whether effective reassembled models can be discovered without access to components introduced in later years. Table 5 reports horizon-averaged test MSE and MAE on ILI. Here, Best Baseline denotes the strongest reference model satisfying the corresponding year constraint. The validation-selected TSCOMP-AGENT outperforms the corresponding Best Baseline in MSE for the 2021, 2022, and 2024 cutoffs, while performance is mixed for 2023 and 2025. In contrast, *Ours (Oracle)* consistently outperforms the Best Baseline, showing that stronger configurations exist in the restricted pools but are not always recovered by validation-based selection. As the component pool grows with later cutoffs, this gap highlights the increasing importance of search and model selection.

Table 5: Comparison under historical cutoffs.

	Ours (oracle)		Ours		Best Baseline	
≤Year	MSE	MAE	MSE	MAE	MSE	MAE
≤2021	**2.868**	**1.128**	**4.093**	**1.339**	5.578	1.637
≤2022	**1.855**	**0.882**	**2.582**	**1.033**	3.704	1.371
≤2023	**1.958**	**0.911**	2.527	1.045	**2.292**	**1.013**
≤2024	**1.624**	**0.797**	**1.803**	**0.840**	2.292	1.013
≤2025	**1.598**	**0.806**	2.197	**0.969**	**2.169**	0.998

5.4 REASSEMBLY UNDER BACKBONE CONSTRAINTS

We evaluate reassembly under backbone-family constraints, where the backbone family is fixed and all remaining component dimensions are optimized within the shared component pool. The resulting validation-selected models are compared with the Best Baseline from the same backbone family. Table 6 reports horizon-averaged test MSE and MAE on ILI. Under the same backbone-family constraint, automated reassembly outperforms the Best Baseline for RNN, CNN, GNN, Transformer, and StateSpace, while MLP shows no consistent gain. These results show that the gains of TSCOMP-AGENT are not driven solely by switching between backbone families: recombining the remaining components can improve forecasting performance within the same family, although the benefit varies across backbones.

Table 6: Comparison across fixed backbone families.

	Ours (oracle)		Ours		Best Baseline	
Backbone	MSE	MAE	MSE	MAE	MSE	MAE
RNN	**1.626**	**0.814**	**2.768**	**1.104**	2.853	1.182
CNN	**1.588**	**0.787**	**2.242**	**0.981**	3.607	1.345
GNN	**1.643**	**0.798**	**2.140**	**0.932**	3.424	1.305
MLP	**1.637**	**0.823**	2.501	1.057	**2.177**	**1.005**
Transformer	**1.574**	**0.788**	**1.907**	**0.875**	2.292	1.013
StateSpace	**1.663**	**0.834**	**2.399**	**1.007**	4.099	1.464

6 CONCLUSION AND DISCUSSION

In this work, we introduced TSCOMP-AGENT, an LLM Agent-driven framework for the automated decomposition and reassembly of time-series forecasting models. Extensive evaluations indicate that collaborative LLM Agents match human-expert decomposition quality while providing automated scalability, eliminating the coarse boundaries and semantic drift of single-LLM prompting. Furthermore, automated reassembly discovers novel architectures that outperform SOTA baselines across various datasets. Shifting from monolithic model invention to modular component exploration thus provides a cumulative and reproducible design paradigm for the time-series forecasting community.

Limitations and Future Work. While TSCOMP-AGENT reliably decomposes standard architectures, automated decomposition remains challenging for models with complex code semantics and highly coupled components. Future work will explore recursive self-improvement (RSI) (Huang et al., 2023; Hu et al., 2025a), enabling LLM Agents to better resolve complex dependencies and even discover novel forecasting components beyond existing literature.

AI USE STATEMENT

In this work, LLMs serve as a core methodological component, where an LLM Agent automates paper parsing and component decomposition. Additionally, we used generative AI tools for text polishing and preliminary literature search. We manually checked and verified all suggested references against primary publications to guarantee bibliographic accuracy. We also reviewed all polished text to preserve our intended technical semantics. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide complete reproduction paths for both stages of this work. For decomposition, Section 3.2 formalizes the workflow, and Appendix B lists the normalized prompt template of every LLM-facing stage together with the deterministic validation gates that own workspace writes, registration, runtime smoke checks, and metric reporting; Appendix C records the full component space, the newly extracted components, and the enforced compatibility contracts. For reassembly, Section 5 and Appendix E.3 specify the full evaluation protocol (random screening, marginal-effect pruning, TPE search, budgets, and final selection), while Appendices E.1 and E.2 describe the datasets, the baseline roster, and the per-model component signatures; Appendix E.4 reports the complete per-cell results. An anonymized repository containing the decomposition workflow, the component framework, and the assembly and analysis code is provided in the supplementary materials.

REFERENCES

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the language of time series. *Transactions on Machine Learning Research (TMLR)*, 2024.

Anthropic. A harness for every task: Dynamic workflows in claude code. `https://claude.com/blog/a-harness-for-every-task-dynamic-workflows-in-claude-code`, 2026.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, 2011. URL `https://papers.nips.cc/paper_files/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html`.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

Colin Catlin. Autots: Automated time series forecasting for python. `https://github.com/winedarksea/AutoTS`, 2020.

Tao Dai, Beiliang Wu, Peiyuan Liu, Naiqi Li, Jigang Bao, Yong Jiang, and Shu-Tao Xia. Periodicity decoupling framework for long-term series forecasting. In *The Twelfth International Conference on Learning Representations*, 2024.

Srinath Dama, Kevin L. Course, and Prasanth B. Nair. Shifting time: Time-series forecasting with khatri-rao neural operators. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *International Conference on Machine Learning (ICML)*, 2024.

Chirag Deb, Fan Zhang, Junjing Yang, Siew Eang Lee, and Kwok Wei Shah. A review on time series forecasting techniques for building energy consumption. *Renewable and Sustainable Energy Reviews*, 74:902–924, 2017.

Shengdong Du, Tianrui Li, Yan Yang, and Shi-Jinn Horng. Deep air quality forecasting using hybrid deep learning framework. *IEEE Transactions on Knowledge and Data Engineering*, 33(6): 2412–2424, 2021. doi: 10.1109/TKDE.2019.2954510.

Wei Fan, Peng Peng, Tiankuo Wang, Dong Wang, Jingang Wang, et al. Dish-ts: A general paradigm for alleviating distribution shift in time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. Moment: A family of open time-series foundation models. In *International Conference on Machine Learning (ICML)*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Kevin Han, Siddharth Maddikayala, Tim Knappe, Om Patel, Austen Liao, and Amir Barati Farimani. TDFlow: Agentic workflows for test driven development. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1511–1527, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.70. URL https://aclanthology.org/2026.eacl-long.70/.

Lu Han, Xu-Yang Chen, Han-Jia Ye, and De-Chuan Zhan. Softs: Efficient multivariate time series forecasting with series-core fusion. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Sungwon Han, Seungeon Lee, Meeyoung Cha, Sercan O Arik, and Jinsung Yoon. Retrieval augmented time series forecasting. In *International Conference on Machine Learning (ICML)*. PMLR, 2025.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Yongxin Ni, Zhibin Gou, Zongze Xu, Yuyu Luo, and Chenglin Wu. Data interpreter: An LLM agent for data science. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19796–19821, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1016. URL https://aclanthology.org/2025.findings-acl.1016/.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Yifan Hu, Guibin Zhang, Peiyuan Liu, Disen Lan, Naiqi Li, Dawei Cheng, Tao Dai, Shu-Tao Xia, and Shirui Pan. Timefilter: Patch-specific spatial-temporal graph filtration for time series forecasting. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025b.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Songtao Huang, Zhen Zhao, Can Li, and Lei Bai. Timekan: Kan-based frequency decomposition learning architecture for long-term time series forecasting. In *The Thirteenth International Conference on Learning Representations*, 2025.

Romain Ilbert, Ambroise Odonnat, Vasilii Feofanov, Aladin Virmaux, Giuseppe Paolo, Themis Palpanas, and Ievgen Redko. Samformer: Unlocking the potential of transformers in time series forecasting with sharpness-aware minimization and channel-wise attention. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Y Zhang, Xiaoming Shi, Pin-Yu Chen, Bernard Lim, Shirui Pan, et al. Time-LLM: Time series forecasting by reprogramming large language models. In *International Conference on Learning Representations*, 2024.

Shruti Kaushik, Abhinav Choudhury, Pankaj Kumar Sheron, Nataraj Dasgupta, Sayee Natarajan, Larry A. Pickett, and Varun Dutt. AI in healthcare: Time-series forecasting using statistical, neural, and ensemble architectures. *Frontiers in Big Data*, 3:4, 2020.

Dongbin Kim, Jinseong Park, Jaewook Lee, and Hoki Kim. Are self-attentions effective for time series forecasting? In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2022.

Dilfira Kudrat, Zongxia Xie, Yanru Sun, Tianyu Jia, and Qinghua Hu. Patch-wise structural loss for time series forecasting. *arXiv preprint arXiv:2503.00877*, 2025. URL https://arxiv.org/abs/2503.00877.

LangChain Team. Langgraph: Multi-agent workflows. https://www.langchain.com/blog/langgraph-multi-agent-workflows, 2024.

Boyuan Li, Yicheng Luo, Zhen Liu, Junhao Zheng, Jianming Lv, and Qianli Ma. Hyperimts: Hypergraph neural network for irregular multivariate time series forecasting. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Shuang Liang, Chaochuan Hou, Xu Yao, Shiping Wang, Hailiang Huang, Songqiao Han, and Minqi Jiang. Beyond holistic models: Systematic component-level benchmarking of deep multivariate time-series forecasting. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, 2026.

Shengsheng Lin, Weiwei Lin, Xinyi Hu, Wentai Wu, Ruichao Mo, and Haocheng Zhong. Cyclenet: Enhancing time series forecasting through modeling periodic patterns. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Shengsheng Lin, Weiwei Lin, Wentai Wu, Feiyu Zhao, Ruichao Mo, and Haotong Zhang. Segrnn: Segment recurrent neural network for long-term time series forecasting. *IEEE Internet of Things Journal*, 2025.

Peiyuan Liu, Beiliang Wu, Yifan Hu, Naiqi Li, Tao Dai, Jigang Bao, and Shu-Tao Xia. Timebridge: Non-stationarity matters for long-term time series forecasting. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 39815–39840, 2025a.

Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Non-stationary transformers: Exploring the stationarity in time series forecasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. iTransformer: Inverted transformers are effective for time series forecasting. In *International Conference on Learning Representations*, 2024a.

Yong Liu, Haoran Zhang, Chenyu Li, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer: Generative pre-trained transformers are large time series models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 32369–32399, 2024b.

Zhining Liu, Ze Yang, Xiao Lin, Ruizhong Qiu, Tianxin Wei, Yada Zhu, Hendrik Hamann, Jingrui He, and Hanghang Tong. Breaking silos: Adaptive model fusion unlocks better time series forecasting. *arXiv preprint arXiv:2505.18442*, 2025b.

Tengfei Lyu, Weijia Zhang, Jinliang Deng, and Hao Liu. Autostf: Decoupled neural architecture search for cost-effective automated spatio-temporal forecasting. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 985–996, 2025.

Lukas Moosbrugger, Valentin Seiler, Philipp Wohlgenannt, and Peter Kepplinger. xlstm-mixer: Multivariate time series forecasting by mixing via scalar memories. *arXiv preprint arXiv:2410.17180*, 2024.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations*, 2023.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized agentic workflow automation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 74949–74977, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/ba84da6921f3040b74ee163aa7451f53-Paper-Conference.pdf.

Jing Peng, Meiqi Yang, Qiong Zhang, and Xiaoxiao Li. S4m: S4 for multivariate time series forecasting with missing values. In *The Thirteenth International Conference on Learning Representations*, 2025.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Mu Qiao. Awesome time series forecasting. https://github.com/qiao-mu/awesome-time-series-forecasting, 2024. GitHub repository.

Xiangfei Qiu, Jilin Hu, Lekui Zhou, Xingjian Wu, Jun Du, Buang Zhang, Chenjuan Guo, Aoying Zhou, Christian S. Jensen, Zhenli Sheng, and Bin Yang. TFB: Towards comprehensive and fair benchmarking of time series forecasting methods. *Proceedings of the VLDB Endowment*, 17(9): 2363–2377, 2024.

Xiangfei Qiu, Xingjian Wu, Hanyin Cheng, Xvyuan Liu, Chenjuan Guo, Jilin Hu, and Bin Yang. Dbloss: Decomposition-based loss function for time series forecasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025a.

Xiangfei Qiu, Xingjian Wu, Yan Lin, Chenjuan Guo, Jilin Hu, and Bin Yang. Duet: Dual clustering enhanced multivariate time series forecasting. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '25, pp. 1185–1196, New York, NY, USA, 2025b. Association for Computing Machinery. ISBN 9798400712456. doi: 10.1145/3690624.3709325. URL https://doi.org/10.1145/3690624.3709325.

Yunzhong Qiu, Zhiyao Cen, Zhongyi Pei, Chen Wang, and Jianmin Wang. Adapt data to model: Adaptive transformation optimization for domain-shared time series foundation models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=APb0lLptDL.

Tal Ridnik, Dedy Kredo, and Itamar Friedman. Code generation with alphacodium: From prompt engineering to flow engineering, 2024. URL https://arxiv.org/abs/2401.08500.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using LLM agents as research assistants. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 5977–6043, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.320. URL https://aclanthology.org/2025.findings-emnlp.320/.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2Code: Automating code generation from scientific papers in machine learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3DcaUTjdKc.

Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing*, 90:106181, 2020.

Zezhi Shao, Fei Wang, Yongjun Xu, Wei Wei, Chengqing Yu, Zhao Zhang, Di Yao, Tao Sun, Guangyin Jin, Xin Cao, et al. Exploring progress in multivariate time series forecasting: Comprehensive benchmarking and heterogeneity analysis. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

Oleksandr Shchur, Ali Caner Turkmen, Nick Erickson, Huibin Shen, Alexander Shirkov, Tony Hu, and Bernie Wang. AutoGluon-TimeSeries: Automl for probabilistic time series forecasting. In *Proceedings of the Second International Conference on Automated Machine Learning*, 2023.

Xiaoming Shi, Shiyu Wang, Yuqi Nie, Dianqi Li, Zhou Ye, Qingsong Wen, and Ming Jin. Time-moe: Billion-scale time series foundation models with mixture of experts. In *International Conference on Learning Representations (ICLR)*, 2025.

Haotian Si, Changhua Pei, Jianhui Li, Dan Pei, and Gaogang Xie. Cmos: Rethinking time series prediction through the lens of chunk-wise spatial correlations. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Mad Sun. Time-series-forecast-papers. `https://github.com/madsun-cloud/Time-Series-Forecast-Papers`, 2024. GitHub repository.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Hao Wang, Lichen Pan, Yuan Shen, Zhichao Chen, Degui Yang, Yifei Yang, Sen Zhang, Xinggao Liu, Haoxuan Li, and Dacheng Tao. Fredf: Learning to forecast in the frequency domain. In *International Conference on Learning Representations (ICLR)*, 2025a.

Shiyu Wang, Haixu Wu, Xiaoming Shi, et al. Timemixer: Decomposable multiscale mixing for time series forecasting. In *International Conference on Learning Representations*, 2024a.

Xiaorui Wang, Fanda Fan, Chenxi Wang, Yuxuan Yang, Rui Tang, Kuoyu Gao, simiao pang, Yuanfeng Shang, Zhipeng Liu, Wanling Gao, Lei Wang, and Jianfeng Zhan. CombinationTS: A modular framework for understanding time-series forecasting models. In *Forty-third International Conference on Machine Learning*, 2026. URL `https://openreview.net/forum?id=CwHRT46VmC`.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL `https://openreview.net/forum?id=OJd3ayDDoF`.

Xue Wang, Tian Zhou, Qingsong Wen, Jinyang Gao, Bolin Ding, and Rong Jin. Card: Channel aligned robust blend transformer for time series forecasting. In *The Twelfth International Conference on Learning Representations*, 2024b.

Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Yong Liu, Mingsheng Long, and Jianmin Wang. Deep time series models: A comprehensive survey and benchmark. *arXiv preprint arXiv:2407.13278*, 2024c.

Yuxuan Wang, Haixu Wu, Jiaxiang Dong, Guo Qin, Haoran Zhang, Yong Liu, Yunzhong Qiu, Jianmin Wang, and Mingsheng Long. Timexer: Empowering transformers for time series forecasting with exogenous variables. In *The Twelfth International Conference on Learning Representations*, 2024d.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, volume 34, pp. 22419–22430, 2021.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Wenzhen Yue, Yong Liu, Hao Wang, Haoxuan Li, Xianghua Ying, Ruohao Guo, Bowei Xing, and Ji Shi. Olinear: A linear model for time series forecasting in orthogonally transformed domain. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 11121–11128, 2023.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, XiongHui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5492ecbce4439401798dcd2c90be94cd-Paper-Conference.pdf.

Zhiyuan Zhao, Juntong Ni, Shangqing Xu, Haoxin Liu, Wei Jin, and B. Aditya Prakash. Timerecipe: A time-series forecasting recipe via benchmarking module level effectiveness. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=CsoR8ztROC.

Li Zhong, Zilong Wang, and Jingbo Shang. Debug like a human: A large language model debugger via verifying runtime execution step by step. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 851–870, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.49. URL https://aclanthology.org/2024.findings-acl.49/.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 11106–11115, 2021.

Pengfei Zhou, Yunlong Liu, Junli Liang, Qi Song, and Xiangyang Li. Crosslinear: Plug-and-play cross-correlation embedding for time series forecasting with exogenous variables. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, 2025.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, pp. 27268–27286. PMLR, 2022.

Tian Zhou, PeiSong Niu, Liang Sun, Rong Jin, et al. One fits all: Power general time series analysis by pretrained lm. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

A Knowledge Base Construction Details

TSCOMP-Agent aggregates peer-reviewed papers, source repositories, community-maintained paper indexes, and unified forecasting libraries spanning Transformer, MLP, convolutional, and state-space model families (Sun, 2024; Qiao, 2024; Wang et al., 2024c). Each candidate is curated through taxonomy, identity, and implementation checks so that similarly named, incomplete, or unverifiable resources are not treated as reliable decomposition targets.

For each verified model, the knowledge base retains its paper assets, parsed documentation, available source code, model card, taxonomy metadata, and provenance. Taxonomy-guided extraction additionally produces a compact core-code snapshot that preserves the implementation paths most

Table 7: Taxonomy and source-level statistics of the knowledge-base snapshot.

Taxonomy path	Models	PDF	Markdown	Code
backbone / CNN	3	3	3	3
backbone / GNN	8	8	8	8
backbone / MLP	18	18	18	18
backbone / RNN	5	5	5	5
backbone / state-space	7	7	7	7
backbone / Transformer	44	44	44	42
foundation / LLM-based	15	15	15	13
foundation / TSFM-based	15	15	15	14
paradigm / diffusion	10	10	10	9
paradigm / ensemble	2	2	2	2
paradigm / probabilistic	3	3	3	2
technique / normalization	5	5	5	4
Total	135	135	135	127 (94.1%)

Table 8: Standardized record schema with an illustrative iTransformer entry.

Field	Stored information	iTransformer example
Identity	Model, paper, venue, year	iTransformer, ICLR 2024
Evidence	Paper, parsed document, LaTeX, repository	Paper and official implementation
Taxonomy	Primary model category	backbone / Transformer
Tensor flow	Input-to-output transformations	$(B, L, C) \rightarrow (B, P, C)$
Semantics	Core mechanism and design distinction	Variables as tokens
Implementation	Dependencies, parameters, and source locations	Inverted embedding over look-back dimension

relevant to model behavior. Together, these per-model records form the decomposition target pool and provide the multi-agent decomposition workflow with semantic evidence from the literature and executable evidence from the repository.

A.1 Corpus Construction and Taxonomy

We include forecasting methods whose paper identity and bibliographic metadata can be verified from the collected sources. For executable decomposition, an accessible implementation is additionally required; methods without usable source code are retained only as literature records. Duplicate or inconsistent entries are resolved during cross-source verification rather than treated as independent methods. The resulting snapshot contains 135 curated methods. All records include a paper PDF and normalized Markdown representation, 92 additionally contain LaTeX source, and 127 contain usable source code. Each method is assigned to a primary taxonomy path according to its main backbone or methodological role. Table 7 summarizes both taxonomy coverage and source availability.

A.2 Standardized Model Records

Each method is represented by a standardized record that aligns its conceptual description with the corresponding implementation. The record captures model identity, source evidence, taxonomy, tensor flow, distinguishing mechanisms, implementation dependencies, and concrete source locations. Paper and source code remain the primary evidence, while the model record provides a compact interface for downstream decomposition. Table 8 summarizes the metadata schema and gives iTransformer as an illustrative example. For iTransformer, the record captures its central design choice of treating variables rather than temporal positions as tokens, with model-level tensor flow $(B, L, C) \rightarrow (B, C, L) \rightarrow (B, C, d) \rightarrow (B, C, P) \rightarrow (B, P, C)$.

The internal record may additionally contain component and integration annotations used in later stages. These derived annotations are maintained separately from source-derived evidence so that their provenance remains explicit.

A.3 CORE-CODE EVIDENCE EXTRACTION

Complete repositories often contain substantial infrastructure unrelated to the forecasting model itself. We therefore construct a compact core-code view while retaining the complete repository for verification. Starting from the model mechanisms and expected tensor flow described in the paper, we locate the corresponding forecasting entry point, model definition, invoked modules, configuration parameters, and prediction head. We retain source files that directly determine forecasting behavior and omit dataset loaders, training utilities, metrics, and unrelated experiments unless they affect forward semantics. The resulting record preserves tensor transformations, dependencies, initialization parameters, and concrete source locations, and any retained code snippet is copied directly from the original implementation rather than rewritten.

B AGENT DECOMPOSITION PROTOCOL

The decomposition workflow follows the three-stage organization introduced in Section 3.2: decomposition proposal, component implementation, and global consolidation. Within this structure, LLM agents handle mechanism analysis, plan synthesis, component implementation, and semantic review, while deterministic code owns workspace preparation, component registration, runtime smoke checks, and metric reporting. Agent outputs are committed to the shared workspace only after passing the corresponding validation checks, preventing LLM agents from modifying shared state directly. This appendix lists the normalized prompt templates of the agent-facing stages.

B.1 ENGLISH PROMPT TEMPLATES

The workflow is implemented as a LangGraph state machine whose LLM-facing prompts live in external template files. Slot interface contracts and reference implementations are injected programmatically at execution time, together with run-specific paths, artifacts, and JSON schemas. For readability, the following boxes provide faithful English-normalized templates. Repeated runtime values are represented by placeholders, and shared safety and structural rules are factored into one common contract rather than repeated in every box. Several stages—run initialization, model-queue construction, manifest and status bookkeeping, and the closing measurement and reporting—are deterministic and LLM-free; they are described in prose alongside the corresponding phases rather than as prompt templates.

Shared Structural Rules (Factored Out of the Individual Templates)

Slot policy. New public components may be added only to the existing configurable non-backbone slots (attention, embedding, normalization, decomposition, and feature attention). Never introduce a new backbone or a new slot, and never repackage a complete model or forward pass as a component; plain reshaping, permutation, and other non-semantic glue is not a component.
Registry and naming. Every component is registered through its slot registry decorator, with the registry name equal to the component's `pool_value` verbatim (kebab-case). A name appearing only in comments or strings does not count as a registration.
Compatibility declarations. Declared allow-domains use only field names and values from the fixed compatibility vocabulary (backbone takes family names such as `CNN` or `Transformer`, never model names). Every declared constraint must be justified from algorithmic principle, tensor layout, input scale, or required invocation context, with a per-field rationale; implementations inherit the planner's `compatibilityIntent` verbatim and may not widen it.
Output. Return only the requested structured JSON object.

Run preparation. Two deterministic steps establish the execution context before any model-level reasoning begins, preventing later agents from silently changing the experimental scope. Run initialization verifies that the run configuration, source models, reference framework, and workspace exist and are readable; bootstraps the run-local workspace from the read-only canonical framework, including the registry-fallback patches that let factory functions resolve newly registered components; and snapshots the baseline component pool. Model queue construction then preserves any wrapper-created queue exactly—no model is added, removed, reordered, or rescanned—or, when no queue exists, enumerates eligible model files under {`SOURCE_MODEL_DIR`} or loads a curated model set, validating model counts, duplicate paths, file existence, and the consistency of `modelName`, `category`, and `sourcePath` fields before writing `MODEL_QUEUE.json`.

Per-model decomposition. For each queued model, three independent discussers first diversify the candidate decompositions, after which a planner produces one prioritized plan and refines it over a bounded number of review rounds with a plan reviewer. The implementer executes the finalized plan, while subsequent attempts switch to a targeted repair prompt conditioned on accumulated review evidence.

Agent 1: Decomposition Discusser (Three Independent Instances)

Role. You are Decomposition Discusser {`DISCUSSER_ID`} of three. You work independently and in parallel with the other two discussers, and the planner later fuses the three proposals rather than selecting one.
Inputs. The model identity, a source-code summary, your assigned analysis perspective, the current reusable component pool, and run-local memory.
Task.
- Propose how the model should be decomposed: existing components to reuse directly, and new public components, each with a name, a target slot, a one-line description of the mechanism, and a source location.
- Record the decomposition ideas you considered but rejected, with reasons, and the key mechanisms of the source model that no candidate covers.
- List the open questions that the planner or the implementer must decide.
- Reuse existing components whenever possible instead of re-implementing them.

Output. A JSON object with reuse candidates, proposed components, rejected ideas, coverage gaps, open questions, and an overall recommendation to the planner.

Agent 2: Plan Synthesizer

Role. You are the decomposition planner. Based on the discussers' proposals, the previous plan, the latest plan-review feedback, and the current component pool, decide which mechanisms should be implemented as pool components.
Task.
- Select only mechanisms that fall into existing slots, and prefer a small number of high-value, clearly bounded components (typically 1–3) over many half-formed ones, with a brief rationale for each.
- Give every discussed candidate an explicit disposition—`implement_new`, `reuse_existing`, `merge_into_parent`, `defer_no_slot`, or `reject_glue`; no candidate may be dropped silently.
- Bind every `implement_new` item to a real source location, a single primary mechanism, clear input/output boundaries, and a cross-model reuse argument; a model or paper name is not a mechanism justification.
- Place mechanisms that are semantically equivalent to an existing component in `rejected`, naming the component to reuse; a reuse judgment must compare mechanism semantics, tensor contracts, key configuration behavior, and boundary conditions—name or shape similarity is not sufficient evidence.
- Treat `missing_coverage` review feedback as mandatory: incorporate the design into the plan or justify in `rejected` why it cannot be implemented independently. In revision rounds, output a `changes` diff against the previous plan; never rewrite or empty the component list.

Output. A JSON plan with `components`, `rejected`, `changes`, and `dispositions`.

Agent 3: Plan Reviewer

Role. You are the reviewer of the planner's decomposition plan for model {`MODEL`}. Review only the current plan: do not re-run exploration and do not expand the implementation scope yourself.
Inputs. Source excerpts with original line numbers, the current component pool, the planner's current plan, the previous review round's feedback (empty on the first round), and the current round's proposal-stage semantic distinctiveness results.
Task.
- Check that planned components are not semantic duplicates of the existing pool, and that the plan faithfully covers the source model's most distinctive, component-worthy designs; report well-evidenced but uncovered designs as missing coverage.
- Verify that every explored candidate receives an explicit disposition, and that every new-component item has a concrete source location, a clear target mechanism, explicit input/output boundaries, and an independently reusable rationale.
- Check component granularity: do not pack multiple independent mechanisms, a complete backbone, or a large forward segment into one component, and do not split a single reshape, permutation, linear layer, or glue into a component.

- Reject renames, thin wrappers, and slight parameter variants of existing components; a reuse judgment must compare mechanism semantics, tensor contracts, key configuration behavior, and boundary conditions.
- Re-review every previous-round issue under its stable identifier, marking it resolved or open; pass only when all historical and new issues are resolved.

Output. A JSON verdict (`pass` or `revise`) with typed issues—duplicate, unfaithful, missing coverage, wrong granularity, invalid compatibility, invalid plan item, or missing disposition—each carrying concrete evidence and an actionable suggestion for the planner.

Agent 4: Initial Implementer

Role. You are the implementer for model {`MODEL`}. Following the decomposition plan, generate directly runnable PyTorch component code for each planned component and return the code together with its metadata; the framework writes your files into the workspace deterministically.
Inputs. The decomposition plan, the interface contracts and reference implementations of every slot, the compatibility-domain vocabulary, and run-local memory.
Task.

- Implement only the components listed in the plan, one standalone `nn.Module` per file under `modules/<slot>/`; if a planned component proves infeasible, place it in `rejected` with a reason instead.
- Follow the slot contracts exactly: contract-exact constructor signature names, `**kwargs` to absorb extra framework arguments, a registry decorator whose name equals the planned `pool_value` verbatim, and imports under the `models.tsgym_v2plus` namespace only.
- Do not guess PyTorch API arguments; use only constructor arguments that the installed API explicitly supports.
- Declare each component's compatibility allow-domain from the fixed vocabulary with a per-field rationale provable from algorithmic principle, and inherit the planner's `compatibilityIntent` verbatim.
- Cover every key computation segment of the source forward pass (norm/denorm, input projection, embedding, backbone body, branches, residual, reshape, decoder, head) in `forwardMappings`, and declare every reuse through `reuseDeclarations` with its source location and reuse justification.

Output. A JSON object with `components`, `files`, `rejected`, `forwardMappings`, and `reuseDeclarations`.

Agent 5: Iterative Repair by the Original Implementer

Role. You are the implementer of model {`MODEL`}, now inside the repair loop at attempt {`ATTEMPT`}. Your previous decomposition did not pass review; repair it yourself with targeted edits—do not restart open-ended design.
Inputs. The plan and slot contracts, your previous component declarations, file list, forward-coverage mappings, and reuse declarations, together with the structured reviewer failures (verdicts, issues, and required fixes) and the summary's repair instructions.
Task.

- Bind every repair action to a concrete reviewer issue, required fix, or summary instruction, declaring in `fixActions` what was done for which issue.
- Follow the round's output contract—an incremental patch or a full rewrite; all hard rules of the initial implementation still apply.
- Never downgrade an unresolved issue to `skip` or silently delete a component; components that genuinely cannot be repaired go to `rejected` with a reason.
- Implement only components listed in the plan; registry and pool names approved by the planner must be kept verbatim.

Output. The repaired decomposition result under the round's contract, including `fixActions` binding each repair to its triggering issue.

Review and consolidation. The default configuration pairs deterministic contract checks with a single advisory semantic review rather than a hierarchy of vetoing reviewers. Deterministic filtering first rejects path, syntax, registry-declaration, pool-collision, and unsupported compatibility-domain violations. The consistency reviewer then audits plan fidelity; its verdict is advisory—the issues it raises guide the repair loop, while final acceptance is settled by runtime validation at the batch-level global check (in the reported configuration, per-model runtime smoke is deferred to this batch-level check). A consolidation step then assigns the per-model status—`accepted`, `needs_fix`, or `skipped` for deterministically verified pure reuse—and converts the reviewer issues into a concrete

repair plan conditioning the next attempt, with components that exhaust the repair budget marked `failed`.

Agent 6: Plan–Implementation Consistency Reviewer

Role. You are the consistency reviewer for model {`MODEL`}. Verify that the implementer's components completely and accurately realize the planner's current plan.
Inputs. The decomposition plan, the declared components, and the candidate component source code.
Task.
- Check that every component in the plan has a corresponding implementation, with slot, mechanism, I/O contract, registry name, and rationale consistent with the plan.
- Flag omitted plan items, unplanned additions, and empty shells or wrappers masquerading as implementations.
- Report every issue with its severity, location, and a required fix that can be handed directly to the repair loop.

Output. A structured verdict (`pass`, `fail`, or `needs_fix`) with a one-line summary, the issue list, and the concrete required fixes.

Accepted artifacts. An accepted verdict is converted into machine-checkable artifacts before it can affect later experiments, and this conversion is deterministic: registration writes `COMPONENT_MANIFEST.json` recording each component's implementation path, registry name, pool value, compatibility decision, and evidence pointers, and closes the model pipeline with a status marker that settles to `accepted` or `rejected` once runtime validation completes.

Run-level consolidation. Once all per-model pipelines finish, a deterministic workspace preflight assembles the eligible-target ledger `GLOBAL_WORKSPACE_PREFLIGHT.json` from status, manifest, and runtime-smoke evidence. A deterministic global check then validates every eligible component in standalone and complete assembled-model forward passes on its declared compatible backbones. Components that fail enter a bounded engineering-repair loop with a single integration agent (Agent 7); repairs cannot revise component semantics or override the earlier decomposition reviews. An optional cross-model deduplication pass can additionally arbitrate mechanism-equivalent duplicates under deterministic guards, keeping one member per confirmed duplicate cluster and rolling back the rest.

Agent 7: Global Consolidation Fix

Role. You are the global engineering-fix agent of a decomposition run. The closing global validation has re-run every accepted component in the shared workspace and the final component pool, and the components listed below failed. You perform targeted engineering repairs within a bounded budget (round {`ROUND`} of {`BUDGET`}).
Inputs. The smoke contract of each slot, the failed validation cases with complete error traces, and the failing components' current metadata and file contents.
Task.
- Perform only engineering repairs: imports, registry wiring, constructor argument passing, obvious shape adapters, dtype/device errors, and compatibility declarations inconsistent with the real I/O contract.
- Never change a component's algorithm, mathematical semantics, boundaries, slot duty, or canonical registry name; if a component can only run by changing its algorithm, list it in `giveUp` rather than repairing it into a different component.
- Each repair overwrites only files declared by the failing component itself, returning the complete repaired file content; never touch `model.py`, backbones, or other components' files.
- Bind every repair action to its failure case, and never weaken shape/finite checks or delete candidates to mask failures.

Output. A JSON object with `fixes` (per repair: the component, its declared files with complete repaired content, and the bound failure case) and `giveUp` (components judged unrepairable).

Mechanical measurement and reporting. The closing stages are deterministic and make no decomposition decisions. Batch metrics serialization writes `BATCH_METRICS.json` exactly as recorded by the orchestrator: input/output/total token usage, per-model and total durations, throughput, completion versus decomposition-success counts, and the result classification of every model. Token usage accounting is exact: the LLM client logs the input/output token counts and stop reason of every call, including structured retries, and the run-level state aggregates them into per-model and batch totals; no count is estimated from natural language. Finally, the run summary aggregates

per-model outcomes and completion/failure counts into `BATCH_STATUS.json`, while batch-level deduplication and global-check evidence are preserved in their own run-level artifacts.

B.2 COMPONENT INGESTION AND VERIFICATION

We now detail the per-model ingestion protocol. Given a source model, the agent (i) locates decomposable mechanisms in its implementation, (ii) maps them onto standardized component interfaces, and (iii) verifies the result through staged gates before any component joins the shared pool.

For each candidate model, the agent performs **Logic Parsing and Semantic Anchoring**. By analyzing the source implementation, the agent identifies "semantic anchor points" within the codebase. It traces the `forward()` function to locate blocks corresponding to normalization, decomposition, or attention mechanisms.

These identified segments then undergo **Component Mapping and SCI Normalization**. The agent decouples the code blocks and maps them into the current component pool, extending an existing configurable slot. Crucially, each extracted module is normalized to adhere to a **Standardized Component Interface (SCI)**, which ensures seamless interoperability across diverse model families.

Finally, TSCOMP-AGENT executes a **Staged Verification Protocol** to guard against semantic drift. Deterministic filtering first enforces static contract checks (registry visibility, signature conformance, and compatibility domains), and each decomposition must supply a forward-coverage mapping that accounts for every core computation segment of the original `forward()` as a new component, a reuse declaration, or non-core glue. An advisory consistency review then audits whether the implementation faithfully realizes the plan, with its issues driving an incremental repair loop carried out by the original implementer. Runtime smoke validation instantiates each surviving component inside the assembled model and verifies output shape and finiteness, and a batch-level global check re-validates the consolidated pool before experiments, rolling back components that fail.

At any ingestion step t, the component library is represented as $\mathcal{C}^{(t)} = \bigcup_{k=1}^{K} \mathcal{C}_k^{(t)}$, where $\mathcal{C}_k^{(t)}$ denotes the verified component pool associated with slot s_k (Section 3.1), whose modules share a standardized tensor contract. The slot set remains fixed, while each $\mathcal{C}_k^{(t)}$ may expand as new reusable mechanisms are identified and mapped to the corresponding slot.

To ensure interoperability, we establish a **Standardized Execution Flow** for all assembled models. This 11-step pipeline, detailed in Table 9, formalizes interactions from normalization to retrieval augmentation. This standardized skeleton provides the necessary infrastructure for cross-model component exchange.

Table 9: Standardized execution flow of an assembled model in TSCOMP-AGENT. B: batch size; L: lookback length; C: variates; L': tokenized length; d: hidden size; H: forecast horizon.

Step	Dimension	I/O Shape	Role
1	Normalization	$(B, L, C) \to (B, L, C)$	Instance normalization (e.g., RevIN, DishTS)
2	Feature Attention	$(B, L, C) \to (B, L, C)$	Channel-wise dependency modeling (optional)
3	Channel Adaptation	$CD \leftrightarrow CI$	Conversion between dependent and independent modes
4	Decomposition	$(B, L, C) \to$ Trend + Season	Multi-scale decomposition (e.g., MA, MoEMA)
5	Embedding	$(B, L, C) \to (B, L', d)$	Tokenization (e.g., Patch, Inverted, Ortho)
6	Backbone	$(B, L', d) \to (B, L', d)$	Core temporal/channel encoder blocks
7	Output Head	$(B, L', d) \to (B, H, C)$	Mapping latent states to prediction horizons
8	Reverse Adaptation	$CI \to CD$	Restoring the original channel dimension
9	Attention Fusion	Residual	Optional cross-channel fusion for CD mode
10	RAG Fusion	Retrieval	Augmentation via historical memory (optional)
11	Reverse Norm	$(B, H, C) \to (B, H, C)$	Restoring original statistics to predictions

The decoupled modules extracted by the agent are integrated into the candidate pools to fill this execution skeleton. The resulting component pool $\mathcal{C}$ spans slots such as series normalization, decomposition, tokenization, and network backbone. In the current protocol, the agent extracts modules only into existing configurable non-backbone slots, while the backbone remains a fixed assembly slot. The current registry is detailed in Table 10 in Appendix C.

Table 10: Current seed and observed implementations in the extensible component pool.

Pipeline	Functional Slot	Components
Series Preprocessing	Series normalization	w/o Norm, Stat, RevIN (Kim et al., 2022), DishTS (Fan et al., 2023)
	Series decomposition	w/o Decomp, Moving Average, MoEMA (Zhou et al., 2022), DFT (Wang et al., 2024a)
	Series sampling/mixing	w/o Mixing, Multi-Scale Mixing (Wang et al., 2024a)
Series Encoding	Channel adaptation	Channel-dependent, channel-independent
	Series tokenization	Point Encoding, Series Patching (Nie et al., 2023), Inverted Encoding (Liu et al., 2024a), Ortho Encoding (Yue et al., 2025)
	Timestamp embedding	w/o Embedding, calendar/time-feature embedding
Network Architecture	Network backbone	**MLP**: DNN, NormLin (Yue et al., 2025); **RNN**: GRU, xLSTM (Moosbrugger et al., 2024); **StateSpace**: StateSpace; **GNN**: GNN; **CNN**: CNN; **Transformer**: w/o Attn, SelfAttn, AutoCorr (Wu et al., 2021), SparseAttn (Zhou et al., 2021), FrequencyAttn (Zhou et al., 2022), DestationaryAttn (Liu et al., 2022); **LLM**: GPT4TS (Zhou et al., 2023), TimeLLM (Jin et al., 2024); **TSFM**: Timer (Liu et al., 2024b), Moment (Goswami et al., 2024), TimeMoE (Shi et al., 2025), Chronos (Ansari et al., 2024)
	Feature attention	w/o Attn, SelfAttn, SparseAttn
	Retrieval augmentation	w/o RAG, historical-memory RAG fusion (Han et al., 2025)
Network Optimization	Sequence length	48, 96, 192, 512
	Loss function	MSE, MAE, Huber, DBLoss (Qiu et al., 2025a), PSLoss (Kudrat et al., 2025), FreDFLoss (Wang et al., 2025a)

A complete forecasting architecture is a configuration $\mathbf{c} = (c_1, \ldots, c_K)$ selecting one compatible candidate $c_k \in \mathcal{C}_k$ from each slot s_k, i.e., a point in the induced assembly space:

$$\mathbf{c} = (c_1, \ldots, c_K) \in \mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_K, \tag{7}$$

where K is the fixed slot count of the component space (cf. Eq. 2). This unified formulation allows TSCOMP-AGENT to assemble diverse forecasting pipelines from the current registry.

To prevent "Frankenstein" models (e.g., pairing a channel-dependent fusion module with a channel-independent backbone), we implement a **Compatibility Contract** system. Each component c explicitly declares the discrete configuration domains it supports,

$$D_c = \{\text{field}_j : [\text{allowed values}]\}, \tag{8}$$

together with a per-field rationale grounded in the algorithmic mechanism, tensor layout, or required calling context (e.g., `channel_mode`: [`dependent`], `backbone`: [`Transformer`]).

Our **Compatibility Validator** statically checks these declarations at ingestion, rejecting unknown fields, unsupported values, or missing rationales. The assembly sampler then filters the product space to physically compatible architectures, so an incompatible combination is never instantiated.

C COMPONENT SPACE DETAILS

Table 10 records the seed and observed implementations of each functional slot s_k in the extensible component pool $\mathcal{C}$ (Section 3.1). Table 11 lists the components added by TSCOMP-AGENT, together with the source model each was extracted from and the existing slot it is mapped to. Table 12 summarizes the incompatibility rules enforced by the Compatibility Validator, covering both the built-in tensor-contract rules and the compatibility domains declared by newly added components.

Table 11: Components added by TSCOMP-AGENT, their source models, and their runtime mappings.

Added component	Decomposed from	Pipeline	Dimension
alpha-beta-blend	CrossLinear	Network architecture	Feature attention
cross-channel-correlation	CrossLinear	Network architecture	Feature attention
graph-block	TimeFilter	Network architecture	Feature attention
mask-moe-gating	TimeFilter	Network architecture	Feature attention
basic-conv-block	TimeKAN	Series encoding	Series tokenization
data-embedding-wo-pos	TimeMixer	Series encoding	Series tokenization
dilated-inception	MTGNN	Series encoding	Series tokenization
dummy-queries	CATS	Series encoding	Series tokenization
external-context-embedding	TimeXer	Series encoding	Series tokenization
linear-extractor	DUET	Series encoding	Series tokenization
patch-embed-v2	CrossLinear	Series encoding	Series tokenization
patch-global-embedding	TimeXer	Series encoding	Series tokenization
patch2d_encoding	PDF	Series encoding	Series tokenization
recurrent-cycle	CycleNet	Series encoding	Series tokenization
time-moe-input-embedding	Time-MoE	Series encoding	Series tokenization
distill-conv	Informer	Series preprocessing	Series decomposition
multi-scale-merge	MICN	Series preprocessing	Series decomposition
series-decomp	DLinear	Series preprocessing	Series decomposition
series-decomp-multi	MICN	Series preprocessing	Series decomposition
channel-stat-norm	CrossLinear	Series preprocessing	Series normalization
instance-stat-norm	TimeFilter	Series preprocessing	Series normalization
itransformer-norm	iTransformer	Series preprocessing	Series normalization
normalize	TimeLLM	Series preprocessing	Series normalization
period-norm	TimeBridge	Series preprocessing	Series normalization
rms-norm	Time-MoE	Series preprocessing	Series normalization
timer-stat-norm	Timer	Series preprocessing	Series normalization

Table 12: Runtime component incompatibility rules.

Component pair	Rejected combination	Validator rule
Tokenization –channel	inverted-encoding or ortho-encoding + channel-independent	Both encodings require channel-dependent mode.
Tokenization –backbone	inverted-encoding + backbone $\in$ GRU, MLP	These backbones are incompatible with inverted encoding.
Tokenization –decomposition	ortho-encoding + decomposition $\neq$ None	Ortho encoding requires no decomposition.
Tokenization –sampling	inverted-encoding + use_sampling=True	Inverted encoding cannot be combined with sampling/mixing.
Tokenization –timestamp	series-patching, ortho-encoding + use_x_mark=True	These encodings reject timestamp marks.
Tokenization –attention	inverted-encoding + attention $\notin$ self-attention, sparse-attention, null	Inverted encoding has a restricted attention set.
Attention–normalization	destationary-attention + normalization $\neq$ Stat	Destationary attention requires Stat normalization.
Attention–Tokenization	destationary-attention + Tokenization $\neq$ series-encoding	Destationary attention requires series encoding.
Channel–feature attention	channel-independent + feature_attn $\neq$ null	Independent channels disable feature attention.
Retrieval–sampling	use_rag=True + use_sampling=True	RAG and sampling cannot be enabled together.
Retrieval–loss	use_rag=True + loss=PSLoss	RAG rejects PSLoss.
Tokenization –backbone	recurrent-cycle + backbone $\notin$ Transformer, CNN, MLP, GRU	The added recurrent-cycle component declares this allowed backbone set.

D DECOMPOSITION EVALUATION DETAILS

This appendix collects the evaluation protocol and detailed evidence for the decomposition stage: the decomposition-run metrics, the model suites, the shared stage-wise evaluation setup, and the proposal-stage and global-consolidation evaluation details.

Algorithm 2 Count final new components

Require: Frozen run pool and terminal acceptance artifacts
1: **for all** models m in the run **do**
2: $\mathcal{N}_m \leftarrow$ components present in the frozen pool and terminally accepted
3: $N_m \leftarrow |\mathcal{N}_m|$
4: **end for**
5: Return all N_m and their mean

D.1 Six-Dimensional Decomposition Metrics

We evaluate a decomposition run after its workspace has been frozen. A *legacy component* belongs to the canonical TSGym V2+ pool before the run. A *new component* is produced by the run, has a terminal accepted status, and remains in the frozen run-local component pool. Only such new components enter the component-quality metrics.

Let $\mathcal{M}$ be the processed source-model set and $\mathcal{S}$ the set of component slots. For model $m \in \mathcal{M}$, let $\mathcal{N}_m$ be its final new-component set and $N_m = |\mathcal{N}_m|$. Later revisions replace the candidate set; outputs from different revisions are never added together.

D.1.1 Count

Count records the number of new components that survive the complete decomposition and validation process. For model m, it is

$$\mathrm{Count}(m) = N_m = |\mathcal{N}_m|. \tag{9}$$

For a run, we report both the per-model counts and their mean

$$\mathrm{Count} = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} N_m. \tag{10}$$

This is a descriptive quantity rather than a monotonic quality score: a larger pool is useful only when its components are also compatible, distinctive, and faithful.

D.1.2 Component Quality

Component Quality contains Compatibility, Novelty, and Consistency. All three are computed over the final new-component set only.

Compatibility. For a new component i in slot s, let $\mathcal{C}_u^{\mathrm{old}}$ be the legacy values of every other slot $u \neq s$. We construct a deterministic strength-t covering array, $t \in \{2, 3\}$, as

$$\mathcal{Q}_s \subseteq \prod_{u \neq s} \mathcal{C}_u^{\mathrm{old}}. \tag{11}$$

The test pool is a controlled subset, not the full Cartesian product. It covers every assignment to every t-slot subset at least once; coverage is recorded and checked by the generator. Each row replaces only slot s with i. New components are not combined with one another.

For $q \in \mathcal{Q}_s$, let $\mathbf{1}_{i,q} = 1$ if construction and forward execution succeed, the output is a tensor with the expected shape, and all output values are finite. The raw score is

$$c_i = \frac{1}{|\mathcal{Q}_s|} \sum_{q \in \mathcal{Q}_s} \mathbf{1}_{i,q}. \tag{12}$$

Validator diagnostics do not remove cases from the denominator. If c_j^{old} is the same raw score for legacy component j in slot s, define

$$\mu_s = \frac{1}{|\mathcal{L}_s|} \sum_{j \in \mathcal{L}_s} c_j^{\mathrm{old}}, \qquad \widetilde{c}_i = \frac{c_i}{\mu_s + \varepsilon}. \tag{13}$$

The run-level score is

$$\mathrm{Compatibility} = \frac{1}{\sum_s |\mathcal{N}_s|} \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{N}_s} \widetilde{c}_i. \tag{14}$$

This slot-relative score is not clipped and may exceed one.

Algorithm 3 Evaluate component quality

Require: Final new components, legacy pool, source-code mappings
1: Generate or load each strength-t covering pool $\mathcal{Q}_s$ and verify coverage
2: Compute legacy baselines μ_s under the same compatibility protocol
3: **for all** new components i **do**
4: Evaluate i on every row of its slot's $\mathcal{Q}_s$; compute c_i and $\widetilde{c}_i$
5: Rank same-slot legacy components by $\rho(i, j)$
6: Compare ranked components in batches; set $r_i = 1$ and stop at the first semantic duplicate
7: Compare i with its mapped original code; obtain f_i and set $\bar{f}_i = f_i/4$
8: **end for**
9: Return the means in Eqs. 14, 15, and 16

Novelty. For each new component i in slot s, we rank all legacy components in the same slot by a deterministic surface-similarity function $\rho(i, j)$ based on names and code. The ranking only determines comparison order. An independent evaluator compares the ranked candidates, typically in small batches, and marks redundancy only when the same or nearly the same core mechanism is implemented. Interface, shape, import, and other engineering differences are ignored. Comparison stops at the first redundant legacy component.

Let $r_i = 1$ if a redundant same-slot legacy component is found and $r_i = 0$ otherwise. Then

$$\text{Novelty} = 1 - \frac{\sum_i r_i}{\sum_i 1}. \tag{15}$$

New components are not compared with one another. A failed component evaluation makes this aggregate unavailable rather than changing its denominator.

Consistency. Consistency evaluates the degree of alignment between each implemented component and its original source code. For each new component i, an independent evaluator receives its code, the corresponding source-code evidence from the original model, and the forward mapping between them. The component need not cover the entire source model; non-semantic interface, shape, dtype, and registry adaptations are allowed. The evaluator assigns $f_i \in \{0, 1, 2, 3, 4\}$: 4 means full consistency with faithful preservation and implementation of the core mechanism, 3 substantial consistency, 2 partial or ambiguous consistency, 1 weak relation, and 0 no evidence of provenance. With $\bar{f}_i = f_i/4$,

$$\text{Consistency} = \frac{1}{\sum_i 1} \sum_i \bar{f}_i. \tag{16}$$

An evaluation failure makes the aggregate unavailable.

D.1.3 Efficiency

For each processed model m, we sum input and output tokens over all LLM calls in discussion, planning, decomposition, review, revision, and summary, but exclude Phase 6 training. If these sums are I_m and O_m, then

$$T_m = I_m + O_m, \qquad \text{TokenEfficiency} = \frac{1}{|\mathcal{M}_{\text{ok}}|} \sum_{m \in \mathcal{M}_{\text{ok}}} T_m. \tag{17}$$

Let τ_m be the wall-clock time from starting model m to persisting its final status, including its review and revision rounds but excluding Phase 6. Then

$$\text{TimeEfficiency} = \frac{1}{|\mathcal{M}_{\text{ok}}|} \sum_{m \in \mathcal{M}_{\text{ok}}} \tau_m. \tag{18}$$

These are direct resource quantities. Reference-based transformations are outside the metric definition.

D.2 Model Curation

This section details the curation rationale and architecture specifications for the models evaluated in Section 4.

Algorithm 4 Aggregate decomposition efficiency

Require: Completed model records
1: **for all** completed models m **do**
2: Sum all decomposition input and output tokens to obtain T_m
3: Read the elapsed time from model start to final status as τ_m
4: **end for**
5: Return the means of T_m and τ_m over completed models

Table 13: Overview of the 30 forecasting architectures evaluated across the decomposition stages and used to induce the assembly component pool.

Suite	Model	Paradigm	Key Modeling Mechanism	Initial Manual Curation
Previously analyzed	DLinear (Zeng et al., 2023)	Linear / MLP	Trend-seasonal linear decomposition	✓
	TimeMixer (Wang et al., 2024a)	Multi-scale MLP	Deconstructed multi-scale mixing	✓
	GRU	Recurrent	Gated recurrent transitions	✓
	MICN	Convolutional	Multi-scale isometric convolutions	✓
	MTGNN	Spatiotemporal Graph	Adaptive graph structure learning	✓
	Informer (Zhou et al., 2021)	Transformer	ProbSparse self-attention mechanism	✓
	PatchTST (Nie et al., 2023)	Transformer	Subseries patching with channel-independence	✓
	Mamba (Gu & Dao, 2023)	State-Space Model	Selective state-space sequence modeling	✓
	TimeLLM (Jin et al., 2024)	Foundation Model	Frozen LLM reprogramming for time series	✓
	Timer (Liu et al., 2024b)	Foundation Model	Pretrained autoregressive sequence transformer	✓
Newly introduced	iTransformer (Liu et al., 2024a)	Transformer	Inverted temporal tokens across variates	✓
	TimeXer (Wang et al., 2024d)	Transformer	Patch tokens with exogenous-informed global token	×
	CARD (Wang et al., 2024b)	Transformer	Channel-aligned attention with token blending	×
	CATS (Kim et al., 2024)	Transformer	Cross-attention with constructed auxiliary series	×
	PDF (Dai et al., 2024)	Transformer	Periodicity-decoupled attention blocks	×
	SAMformer (Ilbert et al., 2024)	Transformer	Sharpness-aware channel-wise attention	×
	SOFTS (Han et al., 2024)	Transformer	Series-core centralized channel aggregation	×
	CrossLinear (Zhou et al., 2025)	Linear / MLP	Cross-variable lightweight linear mapping	×
	CycleNet (Lin et al., 2024)	Linear / Cyclic	Learnable recurrent cycle residuals	×
	S4M (Peng et al., 2025)	State-Space Model	State-space modeling robust to missing values	×
Unannotated 2025	Time-MoE (Shi et al., 2025)	Foundation Model	Billion-scale sparse mixture-of-experts pretraining	✓
	OLinear (Yue et al., 2025)	Linear / MLP	Linear mapping in an orthogonally transformed domain	✓
	DUET (Qiu et al., 2025b)	Linear / Clustering	Dual temporal-channel clustering with distribution routing	×
	HyperIMTS (Li et al., 2025)	Graph	Hypergraph message passing under irregular sampling	×
	CMoS (Si et al., 2025)	MLP	Chunk-wise cross-variate correlation mixing	×
	ShiftingTime (Dama et al., 2025)	Neural Operator	Khatri–Rao neural operators over temporal shifts	×
	FreDF (Wang et al., 2025a)	Frequency-domain	Frequency-domain forecast learning	×
	TimeBridge (Liu et al., 2025a)	Transformer	Non-stationarity-aware patch bridging	×
	TimeFilter (Hu et al., 2025b)	Graph / Filtering	Patch-specific spatio-temporal graph filtration	×
	TimeKAN (Huang et al., 2025)	KAN / Frequency	KAN-based frequency decomposition blocks	×

✓ indicates that the model's architecture is manually represented in the original seed component registry (Appendix C), either as a hand-written configuration template or through manually implemented core components; × indicates that the model is decomposed by TSCOMP-AGENT without such prior manual encoding.

Forecasting Curation Rationale. To evaluate automated decomposition objectively, the model set must balance paradigm diversity with authentic research workflows:

- We curate ten canonical architectures spanning diverse modeling paradigms (linear operators, recurrent cells, convolutions, spatiotemporal graphs, self-attention, state-space representations, and foundation models). These models balanced structural coverage to diagnose stage-wise workflow decisions and LLM backbone capabilities.

- We curate ten recently published SOTA architectures whose rich, decomposable mechanisms are largely absent from the seed registry, and construct manually verified ground-truth decompositions for them. Together with the canonical suite, these twenty models support quantitative fidelity evaluation against human annotations and simulate a realistic multi-model decomposition scenario.

- We further curate ten models published in 2025 that carry neither ground-truth annotation nor—except for OLinear and Time-MoE—any prior manual encoding of their core mechanisms, so that decomposition on this suite proceeds without human reference. This suite guards against overfitting the workflow to the models it was developed on and probes generalization to uncurated codebases.

All 30 models are decomposed by TSCOMP-AGENT, and the resulting extracted components induce the pool used for cross-model reassembly (Appendix E).

D.3 Stage-Wise Evaluation Setup and Diagnostics

The stage-wise comparisons of Section 4.2 share a common evaluation setup. All configurations are compared on the same ten-model evaluation set: DLinear, GRU, Informer, Mamba, MICN, MTGNN, PatchTST, TimeLLM, TimeMixer, and Timer. Each configuration is executed in five independent runs, and reported values are means with sample standard deviations across these runs. Unless stated otherwise, MiniMax-M2.7 serves as the backbone LLM for the decomposition agents and for LLM-assisted evaluation judgments.

D.3.1 Stage-Specific Diagnostic Metrics

Proposal recall and precision. We evaluate the final available planner output for each source model, without taking the union or the best score over earlier revisions. For model i, let $\widehat{\mathcal{P}}_i$ be the final proposal set and $\mathcal{T}_i$ the reference mechanisms absent from the existing component pool. Let $J_i \subseteq \widehat{\mathcal{P}}_i$ contain justified proposals that identify these reference mechanisms. Coverage $c_i(t) \in \{1, \frac{1}{2}, 0\}$ denotes complete, partial, or absent coverage of reference target t. We define

$$\mathrm{Recall}_i = \frac{\sum_{t \in \mathcal{T}_i} c_i(t)}{|\mathcal{T}_i|}, \qquad \mathrm{Precision}_i = \frac{|J_i|}{|\widehat{\mathcal{P}}_i|}. \tag{19}$$

Recall is undefined when no eligible reference target exists, and precision is undefined when no component is proposed. Each run reports the arithmetic mean over models with defined scores. Appendix D.4 describes the reference annotations, proposal categories, and evaluation scope.

Plans Implemented Rate. Plans Implemented Rate measures the fraction of planned components that are successfully implemented. Let N_{planned} denote the number of planned components and $N_{\mathrm{implemented}}$ the number successfully implemented from those plans. The diagnostic is

$$\mathrm{PlansImplementedRate} = \frac{N_{\mathrm{implemented}}}{N_{\mathrm{planned}}}. \tag{20}$$

It measures completion relative to the proposed work; the final pool's Count and component-quality metrics are reported separately.

Agent-rescued components. This diagnostic counts candidates that fail the initial global verification and subsequently pass after repair by the integration agent. Candidates that pass initially or recover through deterministic failure attribution without agent repair are excluded. For run r, let $\mathcal{R}_r^{\mathrm{agent}}$ denote this set; then

$$\mathrm{AgentRescued}_r = |\mathcal{R}_r^{\mathrm{agent}}|. \tag{21}$$

Reported values are averaged over independent runs. Appendix D.5 details the repair records and attribution procedure used to distinguish agent repair from deterministic recovery.

D.3.2 Implementation-Stage Results

Table 14 provides the means and sample standard deviations underlying the implementation-stage comparisons in Section 4.2.

Table 14: Full implementation-stage results on Balanced-10, reported as mean±std over five runs. Tokens cover only the implementation stage.

Configuration	Plans Implemented Rate ↑	Consistency ↑	Count ↑	Novelty ↑	Tokens ↓ (k/model)
Default	0.32 ± 0.12	0.69 ± 0.07	7.0 ± 3.0	0.50 ± 0.15	41 ± 6
w/o Review & Repair	0.36 ± 0.08	0.63 ± 0.16	7.8 ± 2.4	0.52 ± 0.21	11 ± 1
Exhaustive Review	0.16 ± 0.12	0.87 ± 0.14	3.4 ± 2.6	0.22 ± 0.24	133 ± 8
Repair Handoff	0.32 ± 0.06	0.84 ± 0.06	7.0 ± 1.6	0.48 ± 0.11	48 ± 7
Local Memory	0.36 ± 0.13	0.79 ± 0.06	6.8 ± 1.9	0.40 ± 0.14	45 ± 11

D.4 Proposal Evaluation and Additional Ablations

Reference annotations and evaluation scope. We use the frozen v0.2-draft reference annotations associated with the September 6 proposal experiment protocol. Mechanisms are annotated with source locations, slot assignments, and whether an equivalent mechanism already exists in the pool. Alignment uses LLM-assisted judgments; these references should not be interpreted as an exhaustive, independently validated ground truth. Eligible slots are attention, embedding, normalization, decomposition, and feature attention. Only mechanisms absent from the existing pool enter recall denominators. Complete, partial, and absent coverage receive scores of 1, 0.5, and 0, respectively.

Aggregation and metric definitions. Appendix D.3.1 defines proposal recall and precision and their aggregation. The final proposal set $\widehat{\mathcal{P}}_i$ is partitioned into disjoint justified, duplicate, unjustified, and unknown subsets J_i, D_i, U_i, and X_i. Justified proposals identify reference mechanisms absent from the pool; duplicate proposals match mechanisms already in the pool; unjustified proposals match neither; unknown proposals remain unclassified. We additionally measure redundancy as

$$\text{Redundancy} = \frac{\sum_i |D_i|}{\sum_i |\widehat{\mathcal{P}}_i|}. \tag{22}$$

Unlike the model-averaged precision, redundancy pools proposals across all ten models before dividing, so the two metrics use different aggregation weights. Within each model, the four category fractions sum to one. Tables report the mean and sample standard deviation across five independent runs. No significance claim is inferred from these descriptive statistics.

Cost accounting. Proposal token counts sum recorded input and output tokens for exploration, planning, and online plan review, including the token usage of online distinctiveness evaluation when enabled. Offline quality evaluation and component implementation are excluded. Time is the mean across ten models of the sum of recorded proposal-node durations. Exploration duration already includes its parallel branches; it is not summed again across explorers. This is accumulated per-model stage latency, not the concurrent batch makespan. The original experiment guide instead reports full decomposition-batch costs; those totals are retained in the accompanying audit export but are not used for the proposal-cost bubbles.

D.4.1 Detailed Results and Distinctiveness Feedback

Table 15: Proposal-stage ablations on a ten-model set without distinctiveness feedback. Three explorers with three rounds is the default configuration. Values are five-run means ± sample standard deviations. Precision is the justified fraction of all proposals (Appendix D.4). Token usage is per ten-model run; time is per-model proposal latency. Redundancy measures duplication against the existing pool.

Explorers	Max. rounds	Precision ↑	Recall ↑	Redundancy ↓	Proposals	Tokens (k)	Time (s/model)
3	1	0.545 $\pm_{0.102}$	0.462 $\pm_{0.086}$	0.285 $\pm_{0.076}$	19.0 $\pm_{3.4}$	259.3 $\pm_{52.7}$	170.2 $\pm_{17.5}$
3	2	0.487 $\pm_{0.052}$	0.522 $\pm_{0.133}$	0.365 $\pm_{0.045}$	24.2 $\pm_{2.9}$	377.1 $\pm_{68.0}$	278.6 $\pm_{38.6}$
3	3	0.540 $\pm_{0.077}$	0.590 $\pm_{0.096}$	0.300 $\pm_{0.077}$	25.0 $\pm_{2.8}$	552.3 $\pm_{44.7}$	440.5 $\pm_{64.3}$
1	3	0.460 $\pm_{0.127}$	0.420 $\pm_{0.116}$	0.344 $\pm_{0.098}$	23.8 $\pm_{2.8}$	332.8 $\pm_{19.6}$	350.0 $\pm_{33.2}$
1	1	0.358 $\pm_{0.050}$	0.260 $\pm_{0.041}$	0.452 $\pm_{0.124}$	18.2 $\pm_{1.3}$	106.1 $\pm_{5.4}$	94.7 $\pm_{23.4}$

Table 15 reports the per-configuration values behind Figure 3. We additionally provide the plan reviewer with explicit evidence about semantic duplication against the existing component pool, and compare enabled and disabled distinctiveness feedback at the default three-explorer, three-round budget. A one-round budget has no review step, so feedback is inapplicable and is not duplicated as an additional condition. Duplicate evidence for proposal evaluation comes from online records when available and offline evaluation otherwise.

At the default three-explorer, three-round budget, distinctiveness feedback decreases redundancy from 0.300 to 0.200 and the mean proposal count from 25.0 to 21.8, but recall falls from 0.590 to 0.490 while precision is essentially unchanged (0.540 vs. 0.536) (Table 16). Recorded proposal-stage

Table 16: Distinctiveness feedback on a ten-model set. Only the three-round pair is a matched enabled/disabled comparison; the two-round disabled row is retained for budget context. Values are five-run means ± sample standard deviations. Precision is the justified fraction of all proposals (Appendix D.4). Token usage is per ten-model run; time is per-model proposal latency. Redundancy measures duplication against the existing pool.

Explorers	Max. rounds	Distinctiveness feedback	Precision ↑	Recall ↑	Redundancy ↓	Proposals	Tokens (k)	Time (s/model)
3	2	Disabled	0.487 $\pm_{0.052}$	0.522 $\pm_{0.133}$	0.365 $\pm_{0.045}$	24.2 $\pm_{2.9}$	377.1 $\pm_{68.0}$	278.6 $\pm_{38.6}$
3	3	Disabled	0.540 $\pm_{0.077}$	0.590 $\pm_{0.096}$	0.300 $\pm_{0.077}$	25.0 $\pm_{2.8}$	552.3 $\pm_{44.7}$	440.5 $\pm_{64.3}$
3	3	Enabled	0.536 $\pm_{0.108}$	0.490 $\pm_{0.114}$	0.200 $\pm_{0.097}$	21.8 $\pm_{4.0}$	706.4 $\pm_{123.6}$	482.3 $\pm_{77.0}$

tokens increase by approximately 27.9%. The feedback therefore suppresses duplication against the existing pool, but at the cost of coverage and tokens, so we do not enable it by default.

The redundancy reduction from distinctiveness feedback comes with lower coverage and a higher token cost at the default budget. These results concern proposal redundancy; they do not measure redundancy among final implemented components.

D.5 Global Consolidation Evaluation Details

Data sources. The ten-model set is the shared evaluation set of Appendix D.3. All consolidation metrics are computed from the per-run consolidation record (`GLOBAL_CHECK.json`) and the batch status record (`BATCH_STATUS.json`); run directories are frozen before evaluation.

Metric denominators.

- *Survivors*: the length of the `survivors` list.
- *Rollback*: the `rolledBack` count divided by `candidateCount`, the number of pooled candidates entering consolidation.
- *Rescued*: the length of the `rescued` list, i.e. components that appear in the first (round-0) verification failures and pass a later verification, whether repaired by the fix loop or revived by the attribution step.
- *Agent-rescued*: the subset credited to agent repair, as defined in Appendix D.3.1; the attribution and rescue-timing rules are detailed below.
- *Global-stage tokens*: the `tokenUsage` field (input + output) of the consolidation record, in thousands of tokens. Only the fix agent consumes LLM tokens at this stage; the smoke check, the attribution step, and elimination are deterministic and add none.

A repair is marked no-progress when two consecutive rounds yield the identical failure signature, compared on the trailing 80 characters of the trace, in which case the component is rolled back without consuming further budget.

Attribution step. The step triggers when a verification round fails for *every* candidate and the error traces share a common origin. It rolls back the suspected failure-responsible component first and re-verifies the remaining candidates, consuming no repair budget; components recovered this way are recorded as rescued. A rescue is credited to this step when the component never appears in any fix-loop action of the run. Without the fix loop the attribution step remains active, which is why the w/o-repair-agent arm still records 0.8 rescues per run, all of them attribution rescues.

Rescue timing. For Figure 4, verification events are processed as an ordered sequence, since the same round number can appear twice (round 0 re-runs after an attribution rollback). A rescued component is assigned to the repair round of the verification in which it first passes after its last recorded failure; components touched by no fix action are assigned to the attribution step. In the figure, candidates passing the round-0 verification and attribution rescues are shown merged as resolved without the agent, and the two repair rounds are shown merged as agent-rescued. For each setting, Figure 4 normalizes the mean counts of these final outcomes across five runs to percentages. Counts use final survivor and rollback records so duplicate candidate entries do not inflate the totals. Token annotations report the mean global-stage cost.

Failure taxonomy. Each fix-loop action records a natural-language description of the failure and the applied repair. We strip the component path prefix and classify the description with a fixed keyword rule set into five classes: *shape* (tensor-shape mismatches, e.g. size/broadcast/reshape errors), *signature* (constructor-signature mismatches, e.g. unexpected or missing arguments), *import/attribute*, *numeric* (NaN/Inf), and *other* for ambiguous cases. Across the 153 actions of the four fix-enabled arms (default, partial-context, per-model closure, budget 2) the counts are 105 shape, 25 signature, 13 import/attribute, and 10 other; 20 further components are abandoned by the fix agent (give-up), for 173 repair outcomes in total. The rule set and the classification script are released with the analysis code.

Component quality and cost accounting. Fidelity, Compatibility, and Distinctiveness follow the six-dimension evaluation protocol: Fidelity and Distinctiveness are judged by MiniMax-M2.7 over the frozen run artifacts regardless of the backbone LLM that produced the run, and Compatibility is computed deterministically by forward-pass execution. Table 17 reports the Compatibility and Distinctiveness sub-metrics omitted from the main table. End-to-end tokens are the recorded pipeline total in millions (already including consolidation; the offline quality evaluation is never charged to this ledger). Wall-clock minutes exclude the offline evaluation duration. All values are mean$\pm$std over five independent runs per cell.

Table 17: Remaining component-quality sub-metrics for the global-integration ablation (mean$\pm$std over five runs). Comp: Compatibility (deterministic forward-pass check); Nov: Novelty (judged non-redundancy). None of the consolidation switches moves either sub-metric beyond run-level noise.

Arm	Comp	Nov
Default (final system)	0.980$\pm$0.013	0.638$\pm$0.148
w/o engineering repair	0.982$\pm$0.011	0.483$\pm$0.149
w/o full error context	0.980$\pm$0.004	0.512$\pm$0.287
Per-model closure	0.990$\pm$0.009	0.659$\pm$0.223
Fix budget 2†	0.974$\pm$0.023	0.617$\pm$0.247

Run-level semantic consolidation (alternative regime). We also implemented and preliminary-tested a variant in which the consolidation stage additionally performs *semantic* review (hierarchical LLM judges with a dedicated repair loop and cross-model deduplication), rather than the purely engineering closure evaluated above. Under the current default configuration, semantic screening is already applied per model before consolidation, and our preliminary runs did not show an advantage of repeating it at the run level; engineering closure is also substantially cheaper. We therefore adopt the engineering regime as the default and leave a systematic study of semantic consolidation to future work.

D.6 Robustness to the Backbone LLM

Table 18 reports the full matrix underlying Figure 5: three backbone LLMs (MiniMax-M2.7, MiniMax-M3, and GPT-5.6-sol) crossed with the three decomposition workflows (Prompt Only, Dynamic Workflow, and TSCOMP-Agent), evaluated on the shared ten-model set (Appendix D.3) over five independent runs under the metric protocol of Appendix D.1.

E Reassembly Evaluation Details

Component Pool Construction. All assembly experiments operate on the component pool induced by running the decomposition protocol of Section 4 on the 30 models of Table 13 (Appendix D.2): the ten previously analyzed architectures, the ten newly introduced SOTA architectures, and ten unannotated models published in 2025. The pool consists of the seed registry implementations together with the 26 *new* components extracted by TSCOMP-Agent during this run (Table 11, Appendix C); no component outside this pool is used in screening, search, or final assembly.

Table 18: Backbone × workflow matrix on the ten-model set (mean±std over five runs). Novelty is not evaluable by design (—) when a workflow produces no judgeable components for the corresponding slot.

Backbone	Workflow	Count	Consistency	Compatibility	Novelty	Token(M)	Time(min)
MiniMax-M2.7	Prompt Only	$0.4 \pm .9$	$.67 \pm .00$	$.66 \pm .00$	$1.00 \pm .00$	1.11 ± 1.46	17 ± 16
	Dynamic Workflow	0.8 ± 1.3	$.22 \pm .31$	$.46 \pm .48$	$.25 \pm .35$	$.19 \pm .06$	24 ± 19
	TSCOMP-AGENT	7.0 ± 3.0	$.69 \pm .07$	$.98 \pm .01$	$.50 \pm .15$	$.97 \pm .10$	28 ± 4
MiniMax-M3	Prompt Only	2.0 ± 4.5	$.82 \pm .00$	$.09 \pm .00$	—	$.23 \pm .03$	17 ± 7
	Dynamic Workflow	$0.4 \pm .9$	$.22 \pm .31$	$.97 \pm .00$	$.42 \pm .12$	$.31 \pm .07$	27 ± 4
	TSCOMP-AGENT	5.4 ± 3.1	$.82 \pm .12$	$1.01 \pm .03$	$.59 \pm .24$	$1.38 \pm .10$	24 ± 3
GPT-5.6-sol	Prompt Only	3.0 ± 4.5	$.60 \pm .14$	$1.12 \pm .95$	$.20 \pm .00$	5.54 ± 4.76	28 ± 20
	Dynamic Workflow	2.6 ± 3.6	$.38 \pm .27$	$1.07 \pm .33$	—	$1.62 \pm .90$	33 ± 8
	TSCOMP-AGENT	7.0 ± 1.2	$.86 \pm .07$	$.99 \pm .00$	$.86 \pm .14$	$1.06 \pm .03$	69 ± 17

E.1 DATASET DETAILS

Table 19 summarizes the statistics of the eight forecasting datasets used in our experiments.

Table 19: Statistics of the forecasting datasets used in our experiments.

Dataset	Domain	Frequency	Lengths	Dim	Description
ETTh1	Electricity	1 hour	14,400	7	Power transformer 1, comprising seven indicators such as oil temperature and useful load
ETTh2	Electricity	1 hour	14,400	7	Power transformer 2, comprising seven indicators such as oil temperature and useful load
ETTm1	Electricity	15 mins	57,600	7	Power transformer 1, comprising seven indicators such as oil temperature and useful load
ETTm2	Electricity	15 mins	57,600	7	Power transformer 2, comprising seven indicators such as oil temperature and useful load
Weather	Environment	10 mins	52,696	21	Recorded every for the whole year 2020, which contains 21 meteorological indicators
ECL	Electricity	1 hour	26,304	321	Electricity records the electricity consumption in kWh every 1 hour from 2012 to 2014
Traffic	Traffic	1 hour	17,544	862	Road occupancy rates measured by 862 sensors on San Francisco Bay area freeways
ILI	Health	1 week	966	7	Recorded indicators of patients data from Centers for Disease Control and Prevention

E.2 BASELINE DETAILS

We compare against two groups of baselines, all trained and evaluated under the identical protocol as automated reassembly. The first group comprises 13 reference models: eleven published architectures reproducible from the component pool—Transformer (Vaswani et al., 2017), Informer (Zhou et al., 2021), Nonstationary Transformer (Liu et al., 2022), PatchTST (Nie et al., 2023), DLinear (Zeng et al., 2023), SegRNN (Lin et al., 2025), Mamba (Gu & Dao, 2023), iTransformer (Liu et al., 2024a), TimeMixer (Wang et al., 2024a), OLinear (Yue et al., 2025), and RAFT (Han et al., 2025)—plus two in-space backbone-family baselines (CNN and GNN). The second group comprises automated forecasting methods and external baselines: PatchLinear from CombinationTS (Wang et al., 2026), TATO (Qiu et al., 2026), AutoGluon (Shchur et al., 2023), and AutoTS (Catlin, 2020).

SOTA Reference Composite Specifications Table 20 details the exact component signatures for the reference models reproducible from the component pool. All models share identical training parameters, loss objectives, and evaluation pipelines to ensure strictly controlled comparisons.

E.3 ASSEMBLY SEARCH DETAILS

Random Screening. For each dataset–horizon setting, we sample 100 structurally valid configurations uniformly at random from the component pool using a fixed random seed of 1. Each configuration is trained for up to five epochs with Adam and a cosine learning-rate schedule, using validation MSE for early stopping with a patience of three epochs. The validation score associated with a configuration is the MSE at its best validation epoch. Only successfully completed trials are retained for subsequent screening and search.

Screening and Pruning. We use the successfully evaluated random configurations to identify component choices that are consistently associated with poor validation performance. For each choice v, we compute its marginal effect as

$$\Delta(v) = \text{mean}\left\{\text{MSE}_{\text{val}}(c) : v \in c\right\} - \text{median}\left\{\text{MSE}_{\text{val}}(c) : c \in \mathcal{R}\right\},$$

Table 20: The 13 reference models used in the assembly experiments: 11 published models reproducible from the component pool, plus two backbone-family baselines (CNN, GNN).

Model	Year	Component signature
OLinear	2025	RevIN+no-decomp+ortho-encoding+MLP+NormLin
RAFT	2025	Stat+no-decomp+CI+series-encoding+MLP+DNN+RAG
TimeMixer	2024	x_mark+sampling+RevIN+MA+series-encoding+MLP+DNN
iTransformer	2024	x_mark+Stat+no-decomp+inverted-encoding+Transformer+self-attention
DLinear	2023	no-norm+MA+CI+series-encoding+MLP+DNN
SegRNN	2023	Stat+no-decomp+CI+series-patching+GRU+GRU
Mamba	2023	x_mark+Stat+no-decomp+series-encoding+StateSpace+null
PatchTST	2023	Stat+no-decomp+CI+series-patching+Transformer+self-attention
Nonstationary	2022	x_mark+Stat+no-decomp+series-encoding+Transformer+destationary-attention
Informer	2021	x_mark+no-norm+no-decomp+series-encoding+Transformer+sparse-attention
GNN	2019	Stat+no-decomp+dilated-inception+GNN+null
CNN	2018	Stat+no-decomp+series-encoding+CNN+null
Transformer	2017	no-norm+no-decomp+series-encoding+Transformer+self-attention

where $\mathcal{R}$ denotes the successfully evaluated random configurations. A larger $\Delta(v)$ therefore indicates worse validation performance.

A component choice is eligible for pruning only if (i) it appears in at least three evaluated configurations and (ii) its effect remains above the 40th-percentile effect threshold after subtracting one standard error. As a safeguard, any choice appearing in at least one top-decile configuration ranked by validation MSE is retained. The total number of removed choices is capped at 40% of all component choices, with candidates removed in descending order of marginal effect.

Trial Timeouts. To bound search cost, each trial is subject to a wall-clock limit determined by dataset scale. The default limit is 600 seconds for standard-scale datasets and 2400 seconds for larger datasets. For standard-scale datasets, configurations with input lengths of 192 and 512 are allowed up to 750 and 900 seconds, respectively. Timed-out or failed trials are excluded from the evaluated pool and do not contribute validation or test metrics. Consequently, the number of successfully evaluated configurations can be smaller than the nominal sampling or search budget.

TPE Search. After pruning, we optimize validation MSE using the Optuna implementation of the Tree-structured Parzen Estimator (TPE). Successfully evaluated random configurations that are still valid under the pruned component pool are imported as historical observations to warm-start the study. We then perform 100 new TPE trials, updating the surrogate model every five observations. Failed or timed-out trials are reported to the study with a large constant penalty, steering the surrogate away from failing regions; only successfully completed trials are retained for screening, selection, and ensembling.

Final Selection and Ensemble. After random screening and TPE search, we merge all successfully evaluated configurations from both stages and rank them by validation MSE. The five highest-ranked configurations are retained, and their test-set predictions are averaged with equal weights. The resulting ensemble is used to compute the reported test MSE and MAE. All pruning, search, and final-selection decisions are based exclusively on validation performance.

Test Oracle. For diagnostic purposes, we additionally report the *Test Oracle*, defined as the lowest test MSE among all successfully evaluated configurations. The Test Oracle is never used for pruning, search, model selection, or ensemble construction.

E.4 Additional Assembly Experimental Results

This appendix provides extended experimental records, detailed per-baseline performance matrices, and comprehensive breakdowns supporting Section 5.

Full Benchmark Matrix Table 21 expands Table 3 with the complete 32-cell dataset–horizon matrix against the 13 reference models, and Table 22 expands Table 4 against automated forecast-

Table 21: Main results (full): per-cell test MSE/MAE on all 32 dataset–horizon cells against the 13 reference models, expanding Table 3.

Dataset	Horizon	Ours (Oracle)		Ours		OLinear		RAFT		TimeMixer		iTransformer		DLinear		Mamba		PatchTST		SegRNN		Nonstationary		Informer		GNN		CNN		Transformer	
		MSE	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*
ETTh1	96	**0.369**	**0.395**	0.395	0.401	**0.386**	0.405	0.407	0.415	0.453	0.446	0.395	0.411	0.425	0.434	0.427	0.431	0.387	**0.400**	0.391	0.402	0.457	0.445	0.643	0.613	0.440	0.436	0.419	0.423	0.693	0.638
	192	**0.428**	**0.425**	0.482	0.486	0.454	0.445	0.456	0.442	0.485	0.469	0.450	0.444	0.480	0.467	0.473	0.456	**0.436**	**0.429**	0.442	0.432	0.575	0.514	0.844	0.717	0.476	0.463	0.468	0.449	0.744	0.668
	336	**0.438**	0.449	0.532	0.522	0.492	0.464	0.499	0.461	0.601	0.531	0.494	0.467	0.522	0.488	0.517	0.477	**0.476**	**0.449**	**0.476**	0.450	0.674	0.569	0.877	0.748	0.520	0.480	0.505	0.467	0.883	0.734
	720	**0.441**	**0.458**	0.490	0.503	0.501	0.489	0.497	0.480	0.627	0.562	0.517	0.502	0.588	0.554	0.535	0.505	0.502	0.488	**0.474**	**0.472**	0.699	0.587	0.976	0.806	0.551	0.516	0.516	0.492	0.953	0.799
	Avg	**0.419**	**0.432**	0.475	0.478	0.458	0.451	0.465	0.450	0.541	0.502	0.464	0.456	0.504	0.486	0.488	0.467	0.450	0.442	**0.446**	**0.439**	0.601	0.529	0.835	0.721	0.497	0.474	0.477	0.458	0.818	0.710
ETTh2	96	**0.272**	**0.334**	**0.280**	**0.337**	0.314	0.358	0.302	0.349	0.341	0.380	0.305	0.353	0.369	0.422	0.361	0.383	0.288	**0.337**	0.296	0.344	0.351	0.385	1.891	1.112	0.333	0.369	0.334	0.371	1.760	1.065
	192	**0.361**	0.395	0.368	**0.386**	0.378	0.400	0.385	0.398	0.461	0.441	0.378	0.398	0.554	0.530	0.433	0.658	**0.367**	0.388	0.375	0.392	0.481	0.449	3.641	1.637	0.414	0.421	0.431	0.428	4.895	1.968
	336	**0.376**	**0.409**	**0.406**	**0.419**	0.437	0.438	0.423	0.431	0.485	0.464	0.419	0.430	0.530	0.514	0.459	0.452	0.417	0.427	0.418	0.429	0.507	0.477	3.814	1.694	0.459	0.453	0.472	0.463	4.466	1.874
	720	**0.382**	**0.429**	**0.399**	**0.427**	0.445	0.459	0.428	0.444	0.470	0.471	0.425	0.444	0.896	0.693	0.458	0.463	0.427	0.444	0.425	0.443	0.521	0.498	4.480	1.827	0.521	0.500	0.455	0.462	3.646	1.651
	Avg	**0.348**	**0.392**	**0.363**	**0.392**	0.394	0.414	0.384	0.406	0.439	0.439	0.382	0.406	0.587	0.540	0.428	0.489	0.375	0.399	0.378	0.402	0.465	0.452	3.456	1.567	0.432	0.436	0.423	0.431	3.692	1.640
ETTm1	96	**0.288**	**0.334**	**0.287**	**0.332**	0.325	0.362	0.359	0.378	0.343	0.380	0.345	0.378	0.366	0.392	0.348	0.381	0.327	0.364	0.335	0.373	0.372	0.395	0.477	0.471	0.353	0.385	0.346	0.379	0.511	0.497
	192	**0.324**	**0.370**	**0.326**	**0.355**	0.378	0.391	0.395	0.395	0.379	0.404	0.384	0.396	0.398	0.407	0.404	0.409	0.373	0.388	0.376	0.392	0.441	0.422	0.508	0.510	0.424	0.417	0.387	0.398	0.514	0.514
	336	**0.351**	**0.379**	**0.358**	**0.384**	0.428	0.419	0.424	0.414	0.407	0.421	0.416	0.417	0.427	0.427	0.432	0.428	0.402	0.407	0.404	0.411	0.484	0.449	0.577	0.555	0.454	0.433	0.431	0.425	0.625	0.591
	720	**0.410**	**0.417**	**0.453**	0.453	0.487	0.456	0.489	0.449	0.472	0.455	0.492	0.458	0.490	0.468	0.488	0.460	0.461	**0.440**	0.464	0.444	0.609	0.518	0.737	0.661	0.487	0.464	0.481	0.450	0.769	0.684
	Avg	**0.343**	**0.375**	**0.356**	**0.381**	0.405	0.407	0.417	0.409	0.400	0.415	0.409	0.413	0.420	0.423	0.418	0.420	0.391	0.400	0.395	0.405	0.476	0.446	0.575	0.549	0.429	0.425	0.411	0.413	0.605	0.571
ETTm2	96	**0.160**	**0.250**	**0.159**	**0.246**	0.177	0.259	0.184	0.268	0.215	0.296	0.186	0.273	0.201	0.300	0.184	0.266	0.175	0.259	0.178	0.264	0.192	0.279	0.293	0.410	0.191	0.274	0.185	0.269	0.387	0.462
	192	**0.214**	**0.287**	**0.218**	**0.285**	0.247	0.305	0.248	0.308	0.286	0.337	0.252	0.312	0.296	0.373	0.252	0.309	0.241	0.302	0.242	0.305	0.272	0.332	0.606	0.617	0.254	0.313	0.252	0.310	0.685	0.661
	336	**0.268**	**0.322**	**0.273**	**0.322**	0.313	0.346	0.308	0.346	0.332	0.364	0.316	0.352	0.364	0.415	0.317	0.352	0.304	0.341	0.304	0.343	0.344	0.370	1.084	0.827	0.317	0.351	0.315	0.349	1.054	0.813
	720	**0.353**	**0.378**	**0.354**	**0.375**	0.432	0.419	0.405	0.401	0.472	0.437	0.411	0.406	0.636	0.570	0.418	0.408	0.400	0.397	0.399	0.399	0.432	0.416	4.526	1.704	0.417	0.411	0.411	0.405	2.649	1.364
	Avg	**0.249**	**0.309**	**0.251**	**0.307**	0.292	0.332	0.287	0.331	0.326	0.359	0.291	0.336	0.374	0.415	0.293	0.334	0.280	0.325	0.281	0.328	0.310	0.349	1.627	0.889	0.295	0.337	0.291	0.333	1.194	0.825
Weather	96	**0.145**	**0.187**	**0.146**	**0.188**	0.163	0.206	0.199	0.238	0.170	0.223	0.186	0.227	0.195	0.250	0.164	0.215	0.180	0.221	0.188	0.226	0.165	0.216	0.170	0.252	0.165	0.217	0.164	0.214	0.186	0.276
	192	**0.192**	**0.244**	**0.194**	0.258	0.209	**0.249**	0.245	0.273	0.223	0.269	0.235	0.267	0.232	0.283	0.212	0.255	0.229	0.261	0.231	0.263	0.225	0.270	0.215	0.298	0.214	0.463	0.212	0.256	0.245	0.321
	336	**0.234**	**0.269**	**0.234**	**0.268**	0.267	0.292	0.294	0.308	0.280	0.308	0.288	0.304	0.275	0.317	0.272	0.297	0.282	0.298	0.284	0.300	0.289	0.313	0.275	0.344	0.277	0.304	0.270	0.519	0.292	0.358
	720	**0.308**	0.347	**0.307**	0.344	0.346	**0.343**	0.367	0.355	0.354	0.354	0.364	0.354	0.338	0.365	0.353	0.349	0.357	0.346	0.360	0.349	0.367	0.360	0.353	0.391	0.359	0.355	0.348	0.348	0.357	0.398
	Avg	**0.220**	**0.262**	**0.220**	**0.264**	0.246	0.273	0.276	0.293	0.257	0.289	0.268	0.288	0.260	0.304	0.250	0.279	0.262	0.282	0.266	0.284	0.262	0.290	0.253	0.321	0.254	0.335	0.249	0.334	0.270	0.338
ECL	96	**0.130**	**0.223**	**0.126**	**0.219**	0.162	0.255	0.212	0.295	0.205	0.309	0.199	0.285	0.210	0.299	0.226	0.323	0.197	0.286	0.201	0.291	0.246	0.334	0.348	0.423	0.209	0.313	0.232	0.325	0.343	0.421
	192	**0.154**	**0.248**	**0.150**	**0.241**	0.178	0.267	0.212	0.298	0.229	0.324	0.209	0.297	0.211	0.303	0.226	0.324	0.201	0.291	0.204	0.294	0.226	0.329	0.337	0.419	0.233	0.330	0.229	0.325	0.336	0.417
	336	**0.166**	**0.270**	**0.165**	**0.260**	0.197	0.286	0.227	0.313	0.258	0.352	0.227	0.314	0.224	0.318	0.243	0.338	0.217	0.307	0.219	0.310	0.250	0.350	0.351	0.428	0.253	0.348	0.242	0.337	0.369	0.445
	720	**0.220**	**0.310**	**0.231**	**0.323**	0.245	**0.323**	0.268	0.343	0.275	0.361	0.272	0.348	0.260	0.349	0.304	0.380	0.258	0.338	0.260	0.341	0.297	0.378	0.354	0.427	0.289	0.370	0.282	0.368	0.344	0.422
	Avg	**0.167**	**0.263**	**0.168**	**0.261**	0.195	0.283	0.230	0.312	0.242	0.337	0.227	0.311	0.226	0.317	0.250	0.341	0.218	0.306	0.221	0.309	0.255	0.348	0.348	0.424	0.246	0.340	0.246	0.339	0.348	0.426
Traffic	96	**0.377**	**0.268**	**0.366**	**0.249**	0.470	0.327	0.693	0.421	0.714	0.424	0.578	0.389	0.712	0.441	0.737	0.444	0.562	0.368	0.592	0.381	0.742	0.433	0.766	0.451	0.781	0.451	0.766	0.449	0.799	0.467
	192	**0.403**	**0.280**	**0.408**	**0.275**	0.488	0.334	0.645	0.402	0.735	0.428	0.599	0.401	0.657	0.418	0.777	0.459	0.552	0.362	0.568	0.367	0.764	0.449	0.777	0.457	0.799	0.463	0.778	0.461	0.801	0.467
	336	**0.431**	**0.305**	**0.442**	**0.289**	0.510	0.343	0.652	0.404	0.734	0.426	0.628	0.413	0.663	0.422	0.785	0.464	0.561	0.363	0.578	0.370	0.781	0.454	0.795	0.464	0.820	0.471	0.789	0.464	0.810	0.469
	720	**0.470**	**0.334**	**0.514**	**0.321**	0.546	0.364	0.692	0.422	0.804	0.457	0.691	0.440	0.704	0.437	0.821	0.470	0.599	0.381	0.615	0.388	0.843	0.474	0.824	0.478	0.842	0.473	0.831	0.478	0.838	0.481
	Avg	**0.420**	**0.297**	**0.433**	**0.284**	0.503	0.342	0.670	0.412	0.747	0.434	0.624	0.410	0.684	0.429	0.780	0.459	0.569	0.368	0.588	0.377	0.783	0.452	0.790	0.462	0.811	0.465	0.791	0.463	0.812	0.471
illness	24	**1.634**	**0.827**	2.590	1.092	**1.975**	**0.938**	3.469	1.334	4.206	1.504	2.662	1.115	4.875	1.570	4.206	1.517	2.362	1.026	2.825	1.174	3.279	1.269	6.091	1.741	3.208	1.257	3.243	1.272	5.325	1.596
	36	**1.462**	**0.777**	**1.704**	**0.809**	2.283	1.035	3.369	1.307	3.992	1.464	2.608	1.115	4.787	1.543	3.902	1.407	2.315	1.020	2.802	1.174	3.471	1.315	6.136	1.743	3.304	1.278	3.800	1.392	5.333	1.601
	48	**1.834**	**0.875**	4.467	1.407	**2.186**	1.013	3.383	1.304	4.058	1.471	2.713	1.142	5.003	1.581	4.009	1.451	2.259	**1.001**	2.837	1.178	4.125	1.478	6.243	1.754	3.370	1.290	3.397	1.301	5.666	1.647
	60	**1.523**	**0.763**	**2.003**	**0.935**	2.265	1.036	3.415	1.305	4.090	1.464	2.796	1.172	5.938	1.739	4.277	1.482	2.233	1.005	2.950	1.204	3.943	1.423	6.459	1.795	3.812	1.394	3.986	1.417	5.988	1.705
	Avg	**1.613**	**0.810**	2.691	1.061	**2.177**	**1.005**	3.409	1.313	4.086	1.476	2.695	1.136	5.151	1.608	4.099	1.464	2.292	1.013	2.853	1.182	3.704	1.371	6.233	1.758	3.424	1.305	3.607	1.345	5.578	1.637
cross-line cells		**32/32**	**31/32**	**25/32**	**24/32**	0/32	1/32	0/32	1/32	0/32	1/32	0/32	1/32	0/32	0/32	0/32	1/32	0/32	6/32	0/32	2/32	0/32	1/32	0/32	0/32	0/32	1/32	0/32	1/32	0/32	0/32

Red bold = best, blue underline = second best among the non-oracle columns per (row, metric); oracle values are bold iff they beat every reference model. **cross-line cells** = number of cells where the method beats the best of the 13 reference models (0 by construction for the reference columns themselves).

ing methods and external baselines. Both list per-cell test MSE/MAE for our validation-selected configuration and the diagnostic test oracle.

Historical Back-Testing Breakdown Table 23 reports the full per-horizon metrics for year-restricted historical back-testing ($Y \in \{2021, \ldots, 2025\}$) on ILI. It compares our validation-selected assemblies and Oracle upper bounds against the best-performing SOTA model available at or before each cutoff year.

Backbone-Wise Reassembly Breakdown Table 24 presents the detailed per-horizon breakdown when freezing each of the 6 representation backbone families (RNN, CNN, GNN, MLP, Transformer, StateSpace) and searching over auxiliary component slots.

Table 22: Main results (full): per-cell test MSE/MAE against automated forecasting methods (AutoGluon, AutoTS, PatchLinear, TATO), expanding Table 4.

Dataset	Horizon	Ours (Oracle)		Ours		PatchLinear		TATO		AutoGluon		AutoTS	
		MSE	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*
ETTh1	96	**0.369**	0.395	0.395	0.401	**0.382**	**0.394**	0.632	0.481	0.410	0.419	0.604	0.507
	192	**0.428**	0.425	0.482	0.486	**0.433**	**0.424**	0.689	0.512	0.434	0.430	1.013	0.686
	336	**0.438**	0.449	0.532	0.522	**0.476**	**0.448**	0.735	0.536	0.518	0.474	1.564	0.651
	720	**0.441**	**0.458**	0.490	0.503	**0.475**	**0.471**	0.735	0.553	0.651	0.571	0.742	0.596
	Avg	**0.419**	**0.432**	0.475	0.478	**0.442**	**0.435**	0.698	0.521	0.503	0.473	0.981	0.610
ETTh2	96	**0.272**	**0.334**	**0.280**	**0.337**	0.297	0.346	0.336	0.372	0.311	0.356	0.386	0.405
	192	0.361	0.395	0.368	0.386	0.377	0.392	0.435	0.420	**0.340**	**0.383**	0.535	0.479
	336	**0.376**	**0.409**	**0.406**	**0.419**	0.428	0.439	0.479	0.457	0.418	0.454	0.778	0.524
	720	**0.382**	**0.429**	**0.399**	**0.427**	0.496	0.487	0.483	0.470	0.607	0.525	0.660	0.542
	Avg	**0.348**	**0.392**	**0.363**	**0.392**	0.399	0.416	0.433	0.430	0.419	0.430	0.589	0.488
ETTm1	96	**0.288**	**0.334**	**0.287**	**0.332**	0.362	0.380	0.791	0.561	0.310	0.353	0.565	0.483
	192	**0.324**	**0.370**	**0.326**	**0.355**	0.397	0.398	0.826	0.581	0.359	0.374	0.554	0.485
	336	**0.351**	**0.379**	**0.358**	**0.384**	0.431	0.421	0.803	0.583	0.416	0.398	0.989	0.569
	720	**0.410**	**0.417**	**0.453**	**0.453**	0.490	0.454	0.808	0.592	0.845	0.509	0.869	0.648
	Avg	**0.343**	**0.375**	**0.356**	**0.381**	0.420	0.413	0.807	0.579	0.482	0.408	0.744	0.546
ETTm2	96	**0.160**	**0.250**	**0.159**	**0.246**	0.182	0.265	0.225	0.295	0.176	0.262	0.312	0.339
	192	**0.214**	**0.287**	**0.218**	**0.285**	0.246	0.304	0.291	0.335	0.239	0.321	0.302	0.350
	336	**0.268**	**0.322**	**0.273**	**0.322**	0.307	0.342	0.350	0.375	0.285	0.345	0.467	0.425
	720	**0.353**	**0.378**	**0.354**	**0.375**	0.413	0.402	0.446	0.425	0.392	0.422	0.486	0.442
	Avg	**0.249**	**0.309**	**0.251**	**0.307**	0.287	0.329	0.328	0.358	0.273	0.337	0.392	0.389
Weather	96	**0.145**	**0.187**	**0.146**	**0.188**	0.193	0.234	0.206	0.249	0.163	0.204	0.318	0.290
	192	**0.192**	0.244	**0.194**	0.258	0.238	0.270	0.261	0.291	0.197	**0.237**	0.702	0.405
	336	**0.234**	**0.269**	**0.234**	**0.268**	0.293	0.310	0.313	0.327	0.266	0.314	0.679	0.432
	720	**0.308**	0.347	**0.307**	0.344	0.361	0.351	0.387	0.373	0.317	**0.325**	0.378	0.360
	Avg	**0.220**	**0.262**	**0.220**	**0.264**	0.271	0.291	0.292	0.310	0.236	0.270	0.519	0.372
illness	24	**1.634**	**0.827**	2.590	1.092	2.226	1.012	2.229	0.947	**1.948**	**0.877**	2.017	0.921
	36	**1.462**	**0.777**	**1.704**	**0.809**	2.110	0.972	2.602	1.042	2.292	0.966	2.421	1.013
	48	**1.834**	**0.875**	4.467	1.407	**2.087**	**0.974**	2.943	1.132	2.470	1.040	2.600	1.073
	60	**1.523**	**0.763**	**2.003**	**0.935**	2.169	1.027	2.750	1.108	2.374	1.033	3.095	1.188
	Avg	**1.613**	**0.810**	2.691	1.061	**2.148**	0.996	2.631	1.057	2.271	**0.979**	2.533	1.049
cross-line cells		**24/24**	**23/24**	**17/24**	**17/24**	5/24	7/24	0/24	0/24	11/24	9/24	0/24	3/24

Table 23: Comparison under historical cutoffs.

		Ours (Oracle)		Ours		Best Baseline	
≤Year	Horizon	MSE	MAE	MSE	MAE	MSE	MAE
≤2021	24	**2.498**	**1.040**	**4.407**	**1.404**	5.325	1.596
	36	**2.889**	**1.110**	**3.961**	**1.307**	5.333	1.601
	48	**3.050**	**1.212**	**3.950**	**1.309**	5.666	1.647
	60	**3.035**	**1.151**	**4.054**	**1.334**	5.988	1.705
	Avg	**2.868**	**1.128**	**4.093**	**1.339**	5.578	1.637
≤2022	24	**1.918**	**0.916**	4.406	1.391	**3.279**	**1.269**
	36	**1.988**	**0.872**	**2.319**	**0.982**	3.471	1.315
	48	**1.690**	**0.875**	**1.760**	**0.859**	4.125	1.478
	60	**1.823**	**0.865**	**1.843**	**0.901**	3.943	1.423
	Avg	**1.855**	**0.882**	**2.582**	**1.033**	3.704	1.371
≤2023	24	**2.051**	**0.909**	3.614	1.278	**2.362**	**1.026**
	36	**2.024**	**0.910**	**2.205**	**0.975**	2.315	1.020
	48	**1.877**	**0.947**	**2.244**	**0.992**	2.259	1.001
	60	**1.878**	**0.879**	**2.044**	**0.936**	2.233	1.005
	Avg	**1.958**	**0.911**	2.527	1.045	**2.292**	**1.013**
≤2024	24	**1.706**	**0.819**	**1.795**	**0.811**	2.362	1.026
	36	**1.610**	**0.789**	**1.695**	**0.796**	2.315	1.020
	48	**1.597**	**0.786**	**1.733**	**0.823**	2.259	1.001
	60	**1.583**	**0.793**	**1.990**	**0.930**	2.233	1.005
	Avg	**1.624**	**0.797**	**1.803**	**0.840**	2.292	1.013
≤2025	24	**1.556**	**0.801**	2.541	1.074	**1.975**	**0.938**
	36	**1.599**	**0.798**	**2.115**	**0.960**	2.283	1.035
	48	**1.508**	**0.774**	**1.927**	**0.907**	2.186	1.013
	60	**1.728**	**0.850**	**2.203**	**0.934**	2.233	1.005
	Avg	**1.598**	**0.806**	2.197	**0.969**	**2.169**	0.998

Table 24: Comparison across fixed backbone families.

		Ours (Oracle)		Ours		Best Baseline	
Backbone	Horizon	MSE	MAE	MSE	MAE	MSE	MAE
RNN	24	**1.582**	**0.800**	4.127	1.379	**2.825**	**1.174**
	36	**1.741**	**0.812**	**2.296**	**0.990**	2.802	1.174
	48	**1.579**	**0.816**	**2.249**	**0.996**	2.837	1.178
	60	**1.603**	**0.828**	**2.402**	**1.052**	2.950	1.204
	Avg	**1.626**	**0.814**	**2.768**	**1.104**	2.853	1.182
CNN	24	**1.555**	**0.786**	**3.089**	**1.161**	3.243	1.272
	36	**1.692**	**0.794**	**2.174**	**0.988**	3.800	1.392
	48	**1.541**	**0.785**	**1.751**	**0.835**	3.397	1.301
	60	**1.564**	**0.782**	**1.954**	**0.939**	3.986	1.417
	Avg	**1.588**	**0.787**	**2.242**	**0.981**	3.607	1.345
GNN	24	**1.881**	**0.869**	**2.275**	**1.012**	3.208	1.257
	36	**1.497**	**0.739**	**1.620**	**0.767**	3.304	1.278
	48	**1.545**	**0.758**	**1.629**	**0.793**	3.370	1.290
	60	**1.650**	**0.825**	**3.035**	**1.158**	3.812	1.394
	Avg	**1.643**	**0.798**	**2.140**	**0.932**	3.424	1.305
MLP	24	**1.520**	**0.784**	2.584	1.091	**1.975**	**0.938**
	36	**1.640**	**0.802**	**2.132**	**0.971**	2.283	1.035
	48	**1.730**	**0.879**	2.634	1.085	**2.186**	**1.013**
	60	**1.659**	**0.824**	2.656	1.083	**2.265**	**1.036**
	Avg	**1.637**	**0.823**	2.501	1.057	**2.177**	**1.005**
Transformer	24	**1.569**	**0.784**	**1.675**	**0.783**	2.362	1.026
	36	**1.634**	**0.800**	**1.969**	**0.912**	2.315	1.020
	48	**1.565**	**0.775**	**1.601**	**0.780**	2.259	1.001
	60	**1.529**	**0.792**	2.384	1.026	**2.233**	**1.005**
	Avg	**1.574**	**0.788**	**1.907**	**0.875**	2.292	1.013
StateSpace	24	**1.637**	**0.817**	**3.664**	**1.275**	4.206	1.517
	36	**1.609**	**0.791**	**1.940**	**0.914**	3.902	1.407
	48	**1.666**	**0.837**	**1.751**	**0.853**	4.009	1.451
	60	**1.740**	**0.891**	**2.242**	**0.985**	4.277	1.482
	Avg	**1.663**	**0.834**	**2.399**	**1.007**	4.099	1.464