Beyond Static Tools: Test-Time Tool Evolution for Scientific Reasoning

Anonymous ACL submission

Abstract

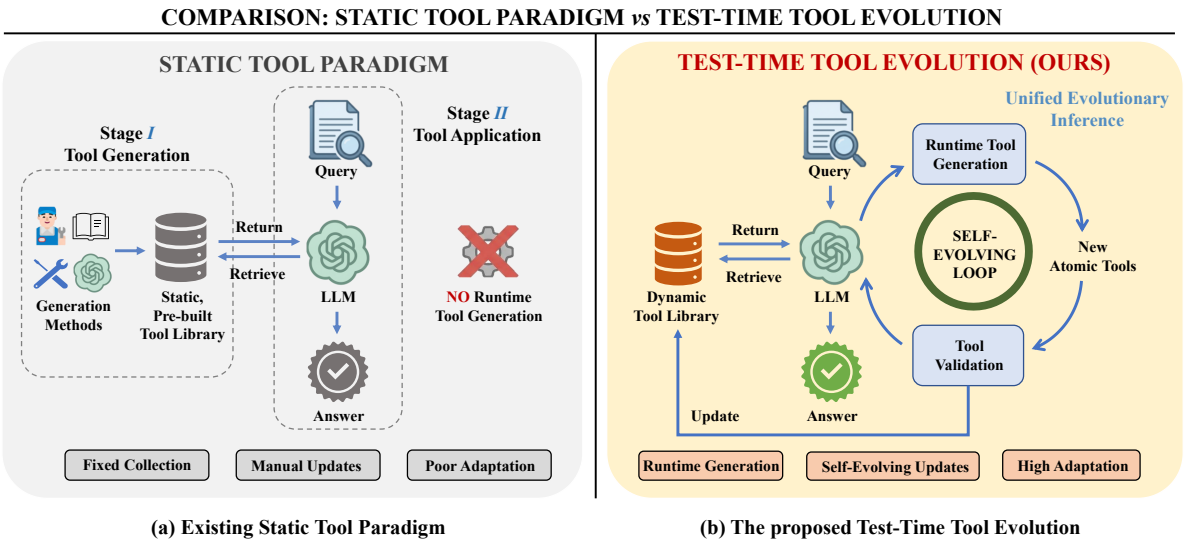
The central challenge of AI for Science is not reasoning alone, but the ability to create computational methods in an open-ended scientific world. Existing LLM-based agents rely on static, pre-defined tool libraries, a paradigm that fundamentally fails in scientific domains where tools are sparse, heterogeneous, and intrinsically incomplete. In this paper, we propose Test-Time Tool Evolution (TTE), a new paradigm that enables agents to synthesize, verify, and evolve executable tools during inference. By transforming tools from fixed resources into problem-driven artifacts, TTE overcomes the rigidity and long-tail limitations of static tool libraries. To facilitate rigorous evaluation, we introduce SciEvo, a benchmark comprising 1,590 scientific reasoning tasks supported by 925 automatically evolved tools. Extensive experiments show that TTE achieves state-of-the-art performance in both accuracy and tool efficiency, while enabling effective cross-domain adaptation of computational tools. The code and benchmark will be released to foster future research in evolutionary tool paradigms.

1 Introduction

The ultimate pursuit of "AI for Science" is to construct autonomous agents capable of navigating the unbounded complexity of the physical world, from discovering novel drug candidates to deriving governing equations of matter. While Large Language Models (LLMs) act as powerful reasoning engines (Brown et al., 2020), scientific research demands precise, executable rigor that inherently exceeds the probabilistic nature of LLMs (Miret and Krishnan, 2024). Without the mediation of computational tools, scientific LLMs demonstrate significantly limited performance (Yu et al., 2025), hallucinating on tasks requiring rigorous fidelity (Chen, 2021; Nijkamp et al., 2022).

Current paradigms attempt to bridge this gap through static tool libraries, *i.e.*, pre-defined functions constructed via manual curation or offline synthesis. While effective for standardized tasks (*e.g.*, weather, booking), this paradigm collapses in scientific reasoning. We identify two fatal bottlenecks in the static approach. First, scientific tools exhibit extreme sparsity and heterogeneity. Unlike the abundant ecosystems of general domains, scientific functions are scattered and non-standardized, rendering the manual curation of a comprehensive library computationally intractable. Second, and most critically, static libraries cannot anticipate the bespoke computational primitives required for novel inquiry, which confines agents to the role of passive selectors rather than active discoverers, imposing an artificial ceiling on their potential to solve unseen problems (Schick et al., 2023; Wan et al., 2025).

We contend that scientific reasoning is fundamentally unsuited for the static tool paradigm shown in Figure 1(a). For an agent to be a genuine scientist, it cannot merely *select* tools, it must *evolve* them. Unlike existing approaches using limited pre-defined tool libraries, we propose Test-Time Tool Evolution (TTE), a paradigm shift that transitions scientific reasoning from static retrieval to dynamic evolution. Instead of relying on a fossilized library, TTE synthesizes executable tools on demand during the inference phase. By dynamically decomposing complex problems into atomic functions and verifying them in real-time, TTE ensures that every tool in the library is intrinsically aligned with the problem space. We instantiate TTE in two fundamental tasks: Ab-initio Tool Synthesis (TTE-Zero) where the agent evolves a tool library from scratch to solve problems without prior knowledge, and Cross-Domain Tool Adaptation (TTE-Adapt) where the agent dynamically repurposes a source tool library (*e.g.*, Materials Science) to conquer a new domain (*e.g.*, Chemistry).



(a) Existing Static Tool Paradigm **(b) The proposed Test-Time Tool Evolution**

Figure 1: Paradigm comparison: Static Tool Paradigm (left) vs Test-Time Tool Evolution (right). Static approaches require pre-collected tool libraries, limiting coverage and domain adaptability. Our test-time evolution starts with an empty library and generates tools on-demand during problem-solving, enabling continuous evolution to new domains and problems.

Our contributions are summarized as follows:

1. We introduce **Test-Time Tool Evolution**, a novel framework that mirrors the iterative nature of the scientific method. By enabling tools to be generated, verified, and evolved during inference, TTE overcomes the inherent rigidity of static paradigms.

2. We release **SciEvo**, a comprehensive benchmark for evaluating tool evolution, comprising 1,590 scientific evaluation instances supported by a library of 925 evolved tools.

3. Extensive evaluations demonstrate that TTE establishes a new State-of-the-Art (SOTA) for scientific reasoning. Specifically, TTE-Zero outperforms existing baselines in both accuracy and tool utilization efficiency significantly, while TTE-Adapt enables effective cross-domain tool adaptation, demonstrating the transferability of computational primitives across scientific disciplines.

2 Related Work

2.1 Static Tool Paradigm

The paradigm of augmenting LLMs with external tools has expanded their capabilities beyond static parametric knowledge. Foundational works have established the mechanisms for this interaction, *e.g.*, ReAct (Yao et al., 2022) introduces the interleaving of reasoning traces with tool actions, while Toolformer (Schick et al., 2023) demonstrates that LLMs could teach themselves to use calculator and search APIs via self-supervised fine-tuning. Building on these execution frameworks, subsequent research have focused on scaling the tool space. Systems like Gorilla (Patil et al., 2024) and ToolLLM (Qin et al., 2024) employ instruction tuning and retrieval-based mechanisms to select appropriate tools from massive, pre-defined API libraries, *e.g.*, HuggingFace or RapidAPI, enabling models to address diverse general-domain queries.

The static tool paradigm has been widely adapted to specialized scientific domains to address the complexity of domain-specific tasks. In chemistry and materials science, systems like ChemCrow (Bran et al., 2024), CheMatAgent (Wu et al., 2025), and ChemMAS (Yang et al., 2025) integrate fixed sets of expert-curated tools ranging from simple calculators to complex synthesis planners to automate organic synthesis and drug discovery. Other approaches focus on enhancing domain capability through knowledge-base integration, such as HoneyComb (Zhang et al., 2024), or utilizing multi-agent frameworks to uncover hidden interdisciplinary relationships as explored in SCP (Jiang et al., 2025). Finally, recent works rigorously benchmark the impact of these static toolsets, as seen in ChemToolAgent (Yu et al., 2025) and MatTools (Liu et al., 2025). Despite their effectiveness in bounded scenarios, these systems share

a critical limitation, *i.e.*, they rely on pre-defined, static tool libraries, which fail to exhaustively cover the open-ended task space.

2.2 Dynamic Tool Synthesis

To address the coverage limitations of static libraries, recent research has shifted towards enabling LLMs to generate tools dynamically. Approaches such as CREATOR (Qian et al., 2023) and CRAFT (Yuan et al., 2024) leverage the code generation capabilities of LLMs to synthesize custom tools via abstract reasoning to solve specific problems. However, these methods typically treat tool generation as a one-off process or, as seen in LATM (Cai et al., 2024) and ToolMaker (Wölflein et al., 2025), adopt a decoupled paradigm where the tool-making phase is separated from inference, hindering real-time adaptation.

Moving beyond static generation, systems like Voyager (Wang et al., 2023) introduce the concept of an evolving skill library, allowing agents to accumulate executable code as tools through trial and error in embodied environments. Similarly, SEAgent (Sun et al., 2025) and ToolACE-DEV (Huang et al., 2025) investigate self-evolving mechanisms for operating system control. While promising, these evolutionary frameworks are designed for gamified or general computer tasks, lacking the rigor and domain-specific logic required for scientific reasoning. Parallelly, automated design approaches (Hu et al., 2024; Shang et al., 2024) explore searching for optimal agent architectures within modular design spaces, focusing on the arrangement of components rather than the evolution of the tools themselves.

3 Test-Time Tool Evolution

3.1 Problem Definition

We formalize *Test-Time Tool Evolution* (TTE) as a fundamentally new paradigm that addresses a critical gap in existing static tool paradigms. Unlike existing approaches where tools are prepared offline before problem-solving, TTE enables tools to be generated and evolved *during* problem-solving, representing a paradigm shift from static to dynamic tool ecosystems.

Formally, given a sequence of scientific problems $\mathcal{P} = \{P_1, P_2, \ldots, P_t\}$ arriving sequentially at test time, the goal of TTE is to maintain an evolving tool library L_t that balances capability and efficiency. We frame this task as an online optimization problem where the system seeks to maximize the cumulative utility:

$$\max_{\{L_t\}_{t=1}^{T}} \sum_{t=1}^{T} \left(\mathbb{I}(\text{Solved}(P_t, L_t)) - \lambda \cdot |L_t| \right), \quad (1)$$

where $\mathbb{I}(\cdot)$ is the indicator function for problem resolution, *e.g.*, accuracy, and λ is a regularization coefficient penalizing library expansion. parameter training. We instantiate TTE for two primary tasks: TTE-Zero for ab-initio tool synthesis ($L_0 = \emptyset$), and TTE-Adapt for cross-domain adaptation of a pre-defined tool library to a new target domain. Since finding the global optimum for Eq. 1 is computationally intractable due to the combinatorial nature of tool composition, our TTE framework adopts a greedy evolution strategy. At each step t, the system updates L_t to L_{t+1} via tool generation and pruning mechanisms to approximate the optimal trajectory without explicit parameter training.

3.2 Architecture Overview

Our framework implements a closed-loop evolutionary workflow comprising five integrated modules, as shown in Figure 2. Structured Task Decomposition decomposes complex queries into executable sub-goals. Dynamic Tool Retrieval queries the library for existing tools. Generative Tool Synthesis creates new tools on-demand when retrieval fails. Atomic Tool Refinement decouples, validates, and registers new tools to evolve the library. Runtime Execution Engine executes the tool sequence to derive the final answer. The proposed architecture enables continuous library growth from an empty state while solving real-world problems.

3.3 Structured Task Decomposition

The *Problem Analyzer* serves as the planning engine, decomposing scientific problems into a sequence of executable sub-goal operations. Given a problem P, it identifies the set of required operations $\mathcal{O}$:

$$\begin{aligned} \mathcal{O} &= \text{Analyze}(P) \\ &= \{\, O_i : O_i \text{ is required to solve } P \,\}. \end{aligned} \quad (2)$$

The decomposition is tool-aware, isolating specific sub-goals that require computational intervention, setting the stage for the retrieval process.

3.4 Dynamic Tool Retrieval

For each identified operation O_i, the system queries the *Dynamic Tool Registry*. We verify the existence

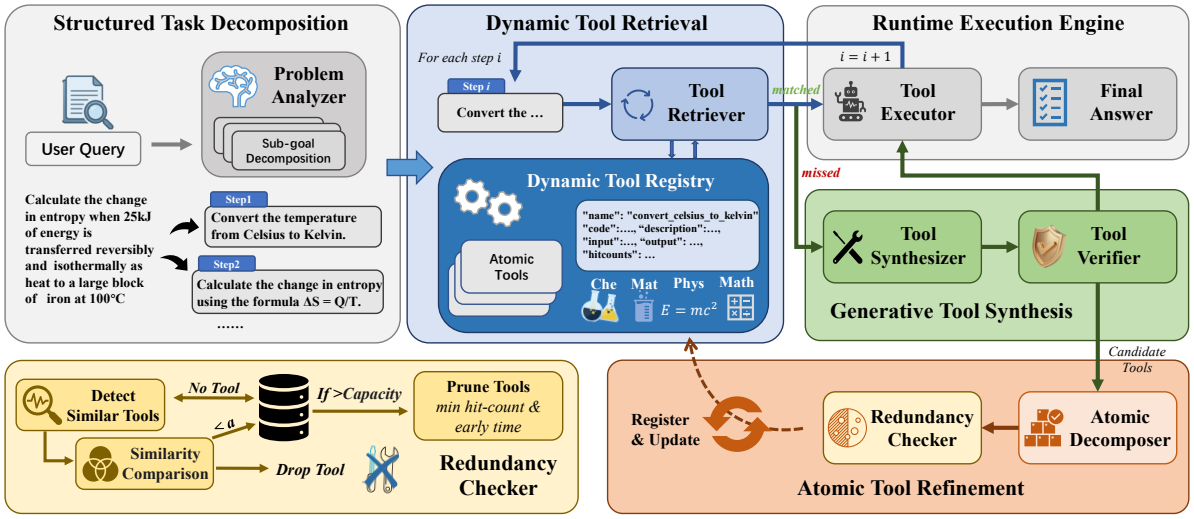


Figure 2: The architecture of the Test-Time Tool Evolution (TTE) framework. The system operates through a closed-loop workflow comprising five integrated stages. (1) Structured Task Decomposition: The Problem Analyzer decomposes complex scientific queries into a sequence of executable sub-goals. (2) Dynamic Tool Retrieval: The system queries the Dynamic Tool Registry for existing atomic tools. If retrieval fails, it triggers (3) Generative Tool Synthesis: The Tool Synthesizer creates candidate tools on-the-fly, which undergo strict verification by the Tool Verifier. (4) Atomic Tool Refinement: Validated tools are decoupled into reusable atomic units by the Atomic Decomposer, filtered by the Redundancy Checker, and registered to update the library. (5) Runtime Execution Engine: Once the required tools are successfully retrieved or generated for all the steps, the Tool Executor executes the sequence to synthesize the final answer.

of suitable tools using semantic similarity between their textual descriptions:

$$\text{sim}(O_i, T_j) = \cos(\text{embed}(d_{O_i}), \text{embed}(d_{T_j})), \tag{3}$$

where d denotes the functional description. The system makes a branching decision based on the maximum similarity score found in the current library L:

$$T^* = \begin{cases} \arg\max_{T_j \in L} \text{sim}(O_i, T_j), & \text{if } s_{\max} \geq \tau, \\ \text{Generate}(O_i, P), & \text{otherwise,} \end{cases} \tag{4}$$

where $s_{\max} = \max_{T_j \in L} \text{sim}(O_i, T_j)$, τ is the threshold maximizing F1 score. The *Tool Retriever* balances exploitation and exploration, ensuring efficient reuse of existing tools (the "matched" path) while automatically triggering the synthesis pipeline (the "missed" path) for novel requirements.

3.5 Generative Tool Synthesis

When retrieval fails, the Generative Tool Synthesis module creates a new tool through a rigorous generation-verification process. Given P and O_i, the *Tool Synthesizer* proposes a tool T_{proposed} via chain-of-thought reasoning:

$$P(T_{\text{proposed}} \mid P, O_i) = \prod_{k=1}^{K} P(f_k \mid P, O_i, f_{1:k-1}), \tag{5}$$

where f_k represents components such as function signature and implementation, and K denotes the total number of generation steps. The *Tool Verifier* ensures correctness through syntax checking, execution testing, and domain validation:

$$P(\text{valid} \mid T_{\text{proposed}}) = P_{\text{syntax}} \cdot P_{\text{exec}} \cdot P_{\text{domain}}. \tag{6}$$

Only tools that pass all validation checks proceed to the refinement stage.

3.6 Atomic Tool Refinement

To ensure the library evolves with high-quality, reusable assets, valid tools undergo atomic refinement before registration. The *Atomic Decomposer* first breaks complex generated tools into fundamental “cell tools”. The decomposition process is formalized as:

$$\{A_1, \ldots, A_k\} = \text{Decompose}(T), \tag{7}$$

which maximizes the expected reuse improvement $\mathbb{E}[R_{\text{atomic}}]$:

$$\mathbb{E}[R_{\text{atomic}}] \geq k \cdot \mathbb{E}[R(T)] \cdot p_{\text{partial}}, \tag{8}$$

where $R(\cdot)$ represents the reuse utility function, k denotes the number of atomic components derived from the decomposition, and p_{partial} denotes the probability that a future problem requires only a subset of functions. Intuitively, monolithic tools suffer from rigidity. Decomposing T into k atomic units unlocks partial reusability, allowing future queries to invoke specific sub-functions (p_{partial}) independently, which flexibility ensures the decomposed set yields higher cumulative utility than the single rigid tool.

The *Redundancy Checker* acts as a gatekeeper. New atomic functions A_{new} are compared against the library:

$$A_{new} \in L_{t+1} \Leftrightarrow \max_{A_i \in L_t} \text{sim}(A_{new}, A_i) < \tau. \quad (9)$$

Concurrently, the curator of the *Dynamic Tool Registry* maintains library efficiency by pruning low-usage tools when capacity C is exceeded, ensuring the library remains compact and relevant:

$$L_{t+1} = L_t \setminus \{A_i : u(A_i) < \theta_{min} \wedge |L_t| > C\}, \quad (10)$$

where $u(A_i)$ denotes the historical usage count of tool A_i, and θ_{min} is the minimum usage threshold.

3.7 Runtime Execution Engine

Once the required tools are successfully retrieved or generated, the *Tool Executor* integrates them into the final reasoning process. The solution synthesis is formalized as $S = \text{Solve}(P, L_t)$. The whole framework closes the loop, applying the evolved capabilities of the library to synthesize the final answer S for the user query.

4 The SciEvo Benchmark

4.1 Benchmark Construction

A defining characteristic of SciEvo is its evolutionary construction paradigm. Unlike libraries curated from static codebases, tools from SciEvo are bootstrapped from scratch using the TTE framework, ensuring that every tool is pragmatically generated to address authentic scientific reasoning needs.

Seed Data Source. To construct a robust and diverse seed environment, we integrate high-quality scientific inquiries from three distinct sources, including SciEval (Sun et al., 2024), SciBench (Wang et al., 2024), and a proprietary materials science dataset focused on specialized domain calculations. We explicitly filter for computational problems that require multi-step reasoning and precise numerical solutions, filtering out purely knowledge-retrieval queries. To ensure the selected questions cover a comprehensive spectrum of scientific scenarios, we employ a semantic clustering-based stratified sampling strategy. Specifically, we embed all candidate questions using the embedding model (Reimers and Gurevych, 2019) and perform K-Means clustering, subsequently sampling instances uniformly from each cluster to maximize problem diversity within the seed set. These pairs provide the problem contexts (Q) and ground-truth validation signals (A) required for reliable tool verification.

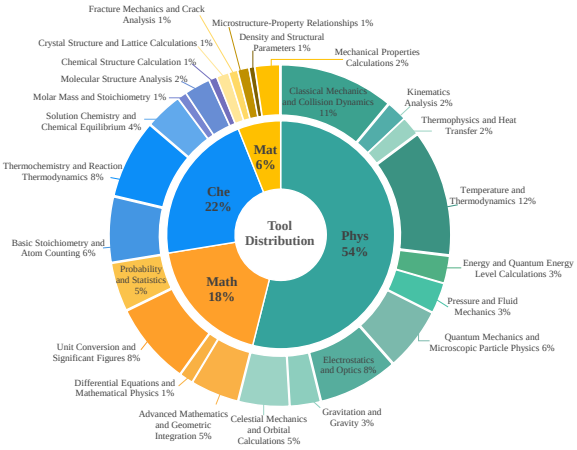


Figure 3: Tool distribution of the curated SciEvo benchmark. SciEvo covers 25 sub-disciplines across four major scientific fields: Physics (499 tools), Chemistry (192), Mathematics (171), and Materials (63), demonstrating comprehensive coverage of diverse scientific computational needs.

Tool Library Synthesis. We utilize the TTE-Zero framework to bootstrap the SciEvo tool library. By initializing the agent with an empty tool library and sequentially exposing it to the seed questions, we simulated a "Tabula Rasa" learning process. The agent generate, execute, and validate Python functions dynamically. Only atomic functions that successfully contributed to deriving the correct ground-truth answers are permanently inducted into the repository. The whole process yielded a verified library of 925 atomic tools, ensuring 100% alignment between the toolset and the problem space.

4.2 Taxonomy and Statistics

To facilitate fine-grained analysis, we organize the synthesized tools into a hierarchical taxonomy using a hybrid classification strategy.

Domain Classification. We apply Principal Component Analysis (PCA) on the vector embeddings of the generated tool descriptions to identify latent semantic clusters. These clusters are subsequently reviewed and refined by PhD-level domain experts to establish a precise taxonomy comprising 25 sub-disciplines across Physics (10), Chemistry (6), Materials Science (5), and Mathematics (4) as shown in Figure 3, which ensures the classification captures both computational semantics and canonical scientific distinctions.

Data Distribution. The complete SciEvo benchmark encompasses 1,590 evaluation instances supported by a library of 925 evolved tools. The domain-specific tool distribution spans four primary disciplines: Physics contains the largest subset with 499 tools, followed by Chemistry (192 tools), Mathematics (171 tools), and Materials (63 tools). As illustrated in Figure 3, the diverse composition ensures robust coverage of scientific computational primitives.

4.3 Evaluation Metrics

To simulate realistic resource constraints, all evaluations are conducted with a maximum tool library capacity of $C = 500$. Under this setting, we assess performance using Accuracy (Acc), following standard protocols (Wang et al., 2024; Sun et al., 2024). To quantify the utility and generalizability of the evolved library $\mathcal{T}$, we additionally define Tool Reuse Rate (TRR@k) for TTE-Zero as the proportion of tools that have been successfully reused at least k times:

$$\text{TRR@}k = \frac{|\{t \in \mathcal{T} \mid h(t) \geq k\}|}{|\mathcal{T}|}, \quad (11)$$

where $h(t)$ denote the hit-count for the tool t. We report TRR@k at several increasing thresholds to capture different levels of utility, *i.e.*, TRR@1 measures the fraction of non-redundant tools used at least once, while TRR@5 and TRR@10 identify the emergence of core scientific primitives.

For cross-domain evaluations, *i.e.*, TTE-Adapt, we decompose the total tool library $\mathcal{T}$ into the predefined set $\mathcal{T}_{pre}$ and the newly evolved set $\mathcal{T}_{new}$. We introduce two stratified metrics to disentangle the sources of competence:

$$\text{TRR}_{evol}\text{@}k = \frac{|\{t \in \mathcal{T}_{new} \mid h(t) \geq k\}|}{|\mathcal{T}_{new}|}, \quad (12)$$

$$\text{TRR}_{trans}\text{@}k = \frac{|\{t \in \mathcal{T}_{pre} \mid h(t) \geq k\}|}{|\mathcal{T}_{pre}|}, \quad (13)$$

where $\text{TRR}_{evol}\text{@}k$ serves as the primary benchmark metric for adaptation efficiency. A higher TRR_{evol} indicates superior performance, signifying that the system has successfully consolidated novel domain knowledge into high-quality, reusable primitives rather than generating disposable scripts. Conversely, TRR_{trans} monitors the substitution of prior knowledge. In cross-domain settings, a lower TRR_{trans} is generally preferred as it reflects the mitigation of negative transfer, *i.e.*, discarding irrelevant tools, provided it remains non-zero to ensure the retention of fundamental, domain-agnostic capabilities.

5 Experiments

5.1 Experimental Setup

Datasets. We evaluate our framework on three distinct benchmarks to assess both problem-solving accuracy and tool evolution efficiency, including SciBench (Wang et al., 2024), SciEval (Sun et al., 2024), and the curated SciEvo dataset.

Baselines. We compare TTE-Zero against five representative baselines categorized into two paradigms. To evaluate fundamental reasoning capabilities without external tool support, we employ Basic-COT (Chain-of-Thought) and Basic-POT (Program-of-Thought). For agentic frameworks that utilize tools, we compare against Creator (Qian et al., 2023), KTCE (Ma et al., 2025), and CheMatAgent (Wu et al., 2025). In the TTE-Adapt setting, we compare against a "No Tool" baseline and a "Source Only" baseline to isolate the performance gains attributed to domain-specific tool evolution.

5.2 Implementation Details

Model Architecture. We evaluate our framework using three LLMs, including GPT-4o, Qwen2.5-7B-Instruct, and GPT-3.5-turbo. Unless otherwise specified, the main experimental results are reported based on GPT-3.5-turbo with a sampling temperature of 0.3 to balance diversity and determinism.

Retrieval and Ranking. We implement a dense retrieval pipeline using bge-m3 (Chen et al., 2024) for embedding and bge-reranker-v2-m3 for reranking. For each sub-goal, the system retrieves the top-k ($k = 3$) relevant tools to provide focused context.

Method	SciBench				SciEval				SciEvo			
	TRR@1	TRR@2	TRR@5	TRR@10	TRR@1	TRR@2	TRR@5	TRR@10	TRR@1	TRR@2	TRR@5	TRR@10
KTCE	0.17	0.10	0.05	0.02	0.06	0.04	0.03	0.02	0.31	0.20	0.09	0.04
Creator	0.12	0.06	0.02	0.01	0.03	0.02	0.02	0.01	0.17	0.07	0.04	0.02
CheMatAgent	0.43	0.27	0.18	0.13	0.20	0.16	0.09	0.05	0.62	0.42	0.28	0.17
TTE-Zero	**0.89**	**0.71**	**0.40**	**0.21**	**0.48**	**0.35**	**0.15**	**0.05**	**0.99**	**0.94**	**0.66**	**0.41**

Table 1: Hierarchical analysis of tool reuse (TRR@k). We report the Tool Reuse Rate at thresholds $k = \{1, 2, 5, 10\}$. TTE-Zero achieves near-perfect utilization (TRR@1 ≈ 1.0) on SciEvo and consistently maintains high reuse rates at stricter thresholds ($k = 5, 10$), whereas baselines fail to generate high-frequency core primitives.

Method	SciBench	SciEval	SciEvo
Basic-COT	0.27	0.18	0.33
Basic-POT	0.31	0.21	0.36
Creator	0.27	0.22	0.49
KTCE	0.37	0.24	0.55
CheMatAgent	0.34	0.23	0.56
TTE-Zero	**0.45**	**0.30**	**0.62**

Table 2: Accuracy comparison across benchmarks. TTE-Zero consistently outperforms all baselines.

Tool Evolution and Deduplication. To maintain a compact and efficient library constrained to a maximum capacity of $C = 500$, we employ strict semantic deduplication. We utilize CodeBERT (Feng et al., 2020) to compute semantic similarity between candidate tools and existing library entries. A new tool is strictly rejected if its maximum cosine similarity with any existing tool exceeds the threshold $\tau = 0.8$.

Evaluation Protocol. Final answer correctness is verified by a GPT-4.1-nano judge. We apply a relative tolerance of 10^{-5} for numerical results and require exact canonical matches for symbolic expressions. As for evaluation metrics, we report Accuracy (Acc) for solution correctness and the proposed Tool Reuse Rate (TRR@k, TRR$_{trans}$@k, and TRR$_{evol}$@k) to quantify the evolutionary quality of the tool library.

6 Results and Analysis

6.1 Performance for TTE-Zero

In this setting, the agent starts with an empty library ($L_0 = \emptyset$) to evaluate its capability to synthesize scientific primitives and solve real problems.

Comparative Analysis on Scientific Benchmarks. We first evaluate the performance of TTE-Zero against SOTA baselines across three benchmarks. As shown in Table 2, TTE-Zero consistently establishes a new SOTA performance. On the SciBench dataset, TTE-Zero achieves an accuracy of 0.45, significantly surpassing the strongest baseline KTCE (0.37) and the domain-specific CheMatAgent (0.34). The performance advantage is further amplified on the proposed SciEvo benchmark, where TTE-Zero reaches 0.62 accuracy compared to 0.56 for CheMatAgent and 0.55 for KTCE. The results demonstrate that evolving tools at test time provides a distinct advantage over static or retrieval-based paradigms, particularly for complex scientific problems requiring multi-step reasoning. Notably, TTE-Zero outperforms standard prompting strategies (Basic-COT and Basic-POT) by a wide margin, *e.g.*, +0.29 improvement over Basic-COT on SciEvo, validating the necessity of external tool support.

Analysis of Tool Evolution Quality. To understand whether the performance gain stems from efficient tool utilization or mere brute-force generation, we analyze the Tool Reuse Rate (TRR). Table 1 presents the hierarchical reuse statistics. A critical observation is the near-perfect utilization rate of TTE-Zero on the SciEvo dataset, achieving a TRR@1 of 0.99, which indicates that almost every generated tool was successfully reused to solve the target problem, minimizing computational waste. In contrast, baselines such as Creator (TRR@1 = 0.17) and KTCE (TRR@1 = 0.31) exhibit severe redundancy, where a vast majority of offline-generated tools are never used. Furthermore, TTE-Zero demonstrates superior capability in consolidating "scientific primitives". At the stricter threshold of $k = 10$, it maintains a reuse rate of 0.41 on SciEvo and 0.21 on SciBench, whereas Creator drops to near zero (0.02 and 0.01, respectively), which confirms that TTE-Zero does not simply flood the library but actively evolves high-utility, reusable tools.

Ablation Study. We investigate the contribution of the sub-goal decomposition module by com-

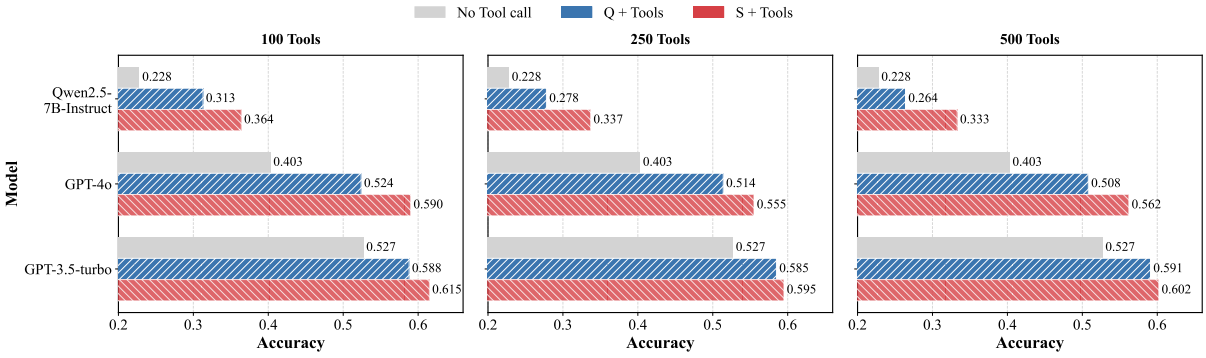


Figure 4: Accuracy comparison on SciEvo. We compare the "No Tool call" baseline against our TTE-Zero method using direct queries ("Q + Tools") and Sub-goal Decomposition ("S + Tools").

Method	Adaptation: Materials → Chemistry									Adaptation: Materials → Physics								
	Acc ↑	TRR_{trans} (↓)				TRR_{evol} (↑)				Acc ↑	TRR_{trans} (↓)				TRR_{evol} (↑)			
		1	2	5	10	1	2	5	10		1	2	5	10	1	2	5	10
No Tool	0.535	-	-	-	-	-	-	-	-	0.535	-	-	-	-	-	-	-	-
Source Only	0.561	0.26	0.13	0.04	0.02	-	-	-	-	0.585	0.38	0.20	0.03	0.01	-	-	-	-
TTE-Adapt	**0.595**	**0.23**	**0.10**	**0.01**	**0.00**	**0.24**	**0.11**	**0.02**	**0.01**	**0.618**	**0.25**	**0.11**	**0.01**	**0.01**	**0.32**	**0.16**	**0.03**	**0.01**

Table 3: Performance on cross-domain adaptation (Source: Materials). We report Accuracy and Tool Reuse Rates (TRR) at $k \in \{1, 2, 5, 10\}$. TRR_{trans} tracks retained source tools (lower is preferred to mitigate negative transfer), while TRR_{evol} tracks new target tools (higher is better for knowledge consolidation).

paring two TTE variants: "Q+Tools" (using the original query) and "S+Tools" (using sub-goal decomposition) against the "No Tool call" baseline. As illustrated in Figure 4, both tool-augmented settings outperform the "No Tool call" baseline across all evaluated models, including Qwen2.5-7B, GPT-4o, GPT-3.5-turbo. Crucially, the "S+Tools" strategy consistently yields the highest accuracy. For instance, with a library size of 100 on Qwen2.5-7B, "S+Tools" achieves clear gains over "Q+Tools" (0.364 vs 0.313), which validates that breaking down complex scientific queries into granular sub-goals is essential for precise tool retrieval and execution, thereby maximizing the efficacy of the evolved tool library.

6.2 Performance for TTE-Adapt

We assess the plasticity of the TTE-Adapt framework by initializing it with a pre-defined tool library (*e.g.*, Materials) and adapting it to novel target domains (*e.g.*, Chemistry and Physics).

Cross-Domain Adaptation. Table 3 presents the adaptation performance. TTE-Adapt consistently outperforms the "No Tool" and "Source Only" baselines, achieving obvious accuracy gains in both settings. The performance improvement is driven by an adaptive substitution mechanism, where the system effectively mitigates negative transfer by pruning irrelevant source tools, *i.e.*, reducing $\text{TRR}_{trans}@1$ from 0.26 to 0.23 in Chemistry, while simultaneously consolidating new knowledge into reusable primitives, which is evidenced by the substantial contribution of evolved tools ($\text{TRR}_{evol}@1 = 0.24$ in Chemistry and 0.32 in Physics). The dynamic adjustment process confirms that TTE-Adapt successfully reshapes the tool distribution to align with the specific reasoning patterns of the target domain.

7 Conclusion

In this work, we identify and address the fundamental limitations of static tool paradigms in scientific reasoning. By introducing Test-Time Tool Evolution (TTE), we shift the role of LLM agents from passive tool selectors to active tool creators. TTE empowers agents to synthesize, verify, and evolve computational primitives during inference, ensuring that the tool space remains intrinsically isomorphic to the unbounded scientific problem space. Our extensive evaluations confirm that TTE not only establishes a new SOTA in reasoning accuracy but also enables robust tool adaptation across diverse domains. We believe that equipping agents with the capacity for autonomous tool evolution is a prerequisite for realizing the next generation of general-purpose scientific AI.

8 Limitations

While Test-Time Tool Evolution (TTE) introduces a promising paradigm for scientific reasoning, we acknowledge several limitations inherent to our current framework.

Inference Latency and Computational Cost. Unlike static retrieval-based methods, TTE requires synthesizing and verifying tools during inference. The dynamic evolution process inevitably incurs higher computational overhead and increased latency compared to simple tool selection. Future work could investigate lightweight "meta-models" to predict tool necessity, thereby skipping evolution for trivial queries.

Dependency on Base LLM Coding Capability. The efficacy of TTE is intrinsically bounded by the code generation capability of the backbone LLM. Our experiments demonstrate SOTA performance using the high-capacity models. However, performance degradation is observed with smaller, less capable open-source models (*e.g.*, <7B parameters) that struggle with generating syntactically correct Python primitives.

Safety and Sandboxing in Open-Ended Evolution. Allowing an agent to generate and execute arbitrary code at test time introduces potential safety risks, particularly in open-ended scientific exploration where generated scripts might inadvertently consume excessive resources or attempt unsafe operations. While our experiments are conducted in a strictly sandboxed environment with timeout constraints, scaling TTE to autonomous real-world systems will require more robust, semantic-level safety verification protocols beyond simple syntactic checks.

9 Ethical Statement

We recognize that the dynamic generation of scientific tools introduces potential dual-use risks, particularly in domains such as chemistry or materials science where code could be misused for harmful applications (*e.g.*, toxin synthesis). To mitigate these risks, we conducted a rigorous manual review of the entire evolved tool library prior to its public release. We strictly adhere to responsible disclosure practices and ensure that no tools enabling harmful applications are included in the released artifacts.

Regarding the artifacts released with this work, *i.e.*, SciEvo benchmark, we confirm that all data sources are from public scientific repositories and are consistent with their intended research use. We have conducted a screening process to ensure no personally identifiable information (PII) or offensive content is included. The code and data are released under the MIT License to promote open research while restricting malicious use. We acknowledge that the system may reflect biases present in the underlying LLMs and scientific literature, and users should verify critical calculations before commercial deployment. Finally, we acknowledge the use of AI assistants (*e.g.*, ChatGPT) solely for linguistic polishing. All scientific claims, experimental designs, and data analyses remain our original work.

References

Tamer Alkhouli, Katerina Margatina, James Gung, Raphael Shu, Claudia Zaghi, Monica Sunkara, and Yi Zhang. 2025. CONFETTI: Conversational function-calling evaluation through turn-level interactions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 7993–8006.

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, and 1 others. 2025. Nestful: A benchmark for evaluating llms on nested sequences of api calls. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 33526–33535.

M. Bran, A. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller. 2024. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6:525–535.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers. In *International Conference on Learning Representations*.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multilinguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335.

Mark Chen. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for program-ming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*.

Shengran Hu, Cong Lu, and Jeff Clune. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.

Xu Huang, Weiwen Liu, Xingshan Zeng, Yuefeng Huang, Xinlong Hao, Yuxian Wang, Yirong Zeng, Chuhan Wu, Yasheng Wang, Ruiming Tang, and 1 others. 2025. Toolace-dev: Self-improving tool learning via decomposition and evolution. *arXiv preprint arXiv:2505.07512*.

Yankai Jiang, Wenjie Lou, Lilong Wang, Zhenyu Tang, Shiyang Feng, Jiaxuan Lu, Haoran Sun, Yaning Pan, Shuang Gu, Haoyang Su, and 1 others. 2025. Scp: Accelerating discovery with a global web of autonomous scientific agents. *arXiv preprint arXiv:2512.24189*.

Kiran Kate, Tejaswini Pedapati, Kinjal Basu, Yara Rizk, Vijil Chenthamarakshan, Subhajit Chaudhury, Mayank Agarwal, and Ibrahim Abdelaziz. 2025. Longfunceval: Measuring the effectiveness of long context models for function calling. *arXiv preprint arXiv:2505.10570*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116.

Siyu Liu, Jiamin Xu, Beilin Ye, Bo Hu, David J Srolovitz, and Tongqi Wen. 2025. Mattools: Benchmarking large language models for materials science tools. *arXiv preprint arXiv:2505.10852*.

Zhiyuan Ma, Zhenya Huang, Jiayu Liu, Minmao Wang, Hongke Zhao, and Xin Li. 2025. Automated creation of reusable and diverse toolsets for enhancing llm reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23):24821–24830.

Santiago Miret and Nandan M Krishnan. 2024. Are llms ready for real-world materials discovery? *arXiv preprint arXiv:2402.05200*.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. A conversational paradigm for program synthesis. *arXiv preprint arXiv:2203.13474*, 30.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Qiwei Peng, Yekun Chai, and Xuhong Li. 2024. HumanEval-XL: A multilingual code generation benchmark for cross-lingual natural language generalization. In *Proceedings of the Joint International Conference on Computational Linguistics, Language Resources and Evaluation*, pages 8383–8394.

Cheng Qian, Chi Han, Yi Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6922–6939.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sijie Zhao, Lauren Hongyu Ruan, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, pages 3982–3992.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. 2024. Agentsquare: automatic llm agent search in modular design space. 2024. *arXiv preprint arXiv:2410.06153*.

Liangtai Sun, Yang Han, Zihan Zhao, Da Ma, Zhennan Shen, Baocai Chen, Lu Chen, and Kai Yu. 2024. Scieval: A multi-level large language model evaluation benchmark for scientific research. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19053–19061.

Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. 2025. Seagent: Self-evolving computer use agent with autonomous learning from experience. *arXiv preprint arXiv:2508.04700*.

Haiyuan Wan, Chen Yang, Junchi Yu, Meiqi Tu, Jiaxuan Lu, Di Yu, Jianbao Cao, Ben Gao, Jiaqing Xie, Aoran Wang, and 1 others. 2025. Deepresearch arena: The first exam of llms' research abilities via seminar-grounded tasks. *arXiv preprint arXiv:2509.01396*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. In *Conference on Neural Information Processing Systems*.

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R. Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. 2024. SCIBENCH: Evaluating college-level scientific problem-solving abilities of large language models. In *Proceedings of the International Conference on Machine Learning, ICML 2024*, volume 235, pages 2072–2099.

Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelovic, and Jakob Nikolas Kather. 2025. Llm agents making agent tools. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 26092–26130.

Mengsong Wu, YaFei Wang, Yidong Ming, Yuqi An, Yuwei Wan, Wenliang Chen, Binbin Lin, Yuqiang Li, Tong Xie, and Dongzhan Zhou. 2025. Chemagent: Enhancing llms for chemistry and materials science through tree-search based tool learning. *arXiv preprint arXiv:2506.07551*.

Cheng Yang, Jiaxuan Lu, Haiyuan Wan, Junchi Yu, and Feiwei Qin. 2025. From what to why: A multi-agent system for evidence-based chemical reaction condition reasoning. *arXiv preprint arXiv:2509.23768*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Botao Yu, Frazier N Baker, Ziru Chen, Garrett Herb, Boyu Gou, Daniel Adu-Ampratwum, Xia Ning, and Huan Sun. 2025. Tooling or not tooling? the impact of tools on language agents for chemistry problem solving. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 7620–7640.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. 2024. Craft: Customizing LLMs by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations*.

Huan Zhang, Yu Song, Ziyu Hou, Santiago Miret, and Bang Liu. 2024. HoneyComb: A flexible LLM-based agent system for materials science. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 3369–3382.

A Complete Algorithmic Workflow

The appendix provides the end-to-end algorithmic details of Test-Time Tool Evolution (TTE) that are omitted from the main paper due to space constraints, including the full closed-loop evolution procedure, and the failure-handling logic that ensures robustness at test time.

A.1 End-to-End Test-Time Tool Evolution

Algorithm 1 presents the full TTE pipeline. The same procedure covers both TTE-Zero (starting from an empty library) and TTE-Adapt (starting from a pre-defined source library), since the evolution loop is identical except for the initialization of the registry. For clarity and reproducibility, we explicitly distinguish: (i) τ_{ret} for retrieval acceptance (whether to reuse a retrieved tool), and (ii) τ_{dup} for deduplication (whether a new atomic tool is considered redundant). These two thresholds need not be identical and can be tuned independently.

A.2 Failure Handling and Fallback

TTE is designed to be robust under imperfect tool synthesis. When verification fails (syntax / runtime / domain constraints), the proposed tool is *not* registered. Downstream execution can either attempt to solve the sub-goal via reasoning-only mode or proceed with partial tool chains when intermediate values are still available. In our implementation, we primarily use a conservative strategy: only verified tools are registered, and failed tool generation triggers a lightweight fallback to direct reasoning or Program-of-Thought, ensuring the system degrades gracefully rather than accumulating faulty tools.

B Prompts for Each Agent Module

This section details the system prompts designed for the three core modules of the TTE framework. We enforce strict JSON or XML-based output formats to ensure robust parsing and seamless integration with the Python execution environment.

B.1 Structured Task Decomposition

The *Problem Analyzer* translates high-level scientific queries into linear execution plans using the prompt in Figure 5.

B.2 Dynamic Tool Retrieval

The *Tool Retriever* selects existing primitives from the library using the prompt in Figure 6, which enforces a "no-hallucination" policy.

Algorithm 1 Complete Test-Time Tool Evolution.

Require: User problem P, initial tool library L, library capacity C
Ensure: Solution S (or FAIL)

```
O ← DECOMPOSE(P)
chain ← [ ]
for each operation O_i ∈ O do
    T_i ← RETRIEVETOPK(L, O_i, k)
    (T*, s*) ← arg max_{T∈T_i} SIM(T, O_i)
    if s* ≥ τ_ret then
        chain.APPEND(T*); u(T*) += 1
    else
        T_new ← SYNTHESIZETOOL(P, O_i)
        ok ← VERIFYTOOL(T_new)
        if ok = false then
            continue
        end if
        A ← ATOMICDECOMPOSE(T_new)
        for each atomic tool A ∈ A do
            if max_{T∈L} SIM_dup(A, T) < τ_dup then
                L ← L ∪ {A}; u(A) ← 1
            else
                T_match ← arg max_{T∈L} SIM_dup(A, T)
                u(T_match) += 1
            end if
        end for
        L ← PRUNEIFNEEDED(L, C)
        chain.APPEND(T_new)
    end if
end for
S ← EXECUTECHAIN(P, chain)
if S = FAIL then
    S ← FALLBACK(P)
end if
return S
```

B.3 Tool Synthesis and Reasoning

Figure 7 displays the hybrid prompt used by the generative tool synthesis module. It handles both tool synthesis and final answer generation.

C Subject-wise Results on SciEvo

Table 4 reports subject-wise performance on SciEvo under three settings: (i) “No Tools” (direct inference), (ii) “Q+Tools” (retrieve tools using the original question as query), and (iii) “S+Tools” (retrieve tools using decomposed sub-questions as queries).

Across all model–subject pairs, tool augmentation provides clear benefits over direct inference. More importantly, sub-question driven retrieval (S+Tools) tends to be more robust and yields higher peak performance than retrieving tools using the original question (Q+Tools), especially in Chemistry and Physics, where tool selection is sensitive to units, constants, and domain-specific formulas.

A key pattern is that Chemistry exhibits the largest gain from structured decomposition: Chem-

Prompt for Problem Analyzer

```
[SYSTEM]
You are an expert computational scientist with broad interdisciplinary expertise.
You excel at decomposing complex scientific problems into a sequence of programmable computation steps that can be
executed by predefined atomic tools.

[TASK]
Decompose the user problem into a list of concrete computational subtasks.
Each subtask must be a specific computation step (not analysis, discussion, or description).
The subtasks must form a coherent execution order: outputs of earlier steps can be used as inputs to later steps.
Decompose until the full computation pipeline is covered, but do NOT provide the final numerical answer.

[STRICT OUTPUT FORMAT]
Return STRICT JSON only (no extra text, no Markdown, no comments):
{
  "original_problem": "...",
  "subtasks": [
    {"step": 1, "description": "..."},
    {"step": 2, "description": "..."},
    ...
  ]
}

[USER]
{query}
```

Figure 5: The prompt used by the Problem Analyzer to decompose user queries into structured execution plans.

Prompt for Tool Executor

```
[SYSTEM]
You are a strict tool-calling agent. You MUST respond only by calling one predefined AtomicTool.
Your goal is to select the single most relevant tool from the provided tool catalog and call it.

[CORE RULES]
1) Do NOT answer the question directly. Do NOT do calculations in natural language.
2) Call exactly ONE tool. If no tool matches, return an empty tool-call list.
3) Do NOT invent tools. Only use tools provided by the system.
4) Output MUST be in standard OpenAI tool-call JSON format. No extra text.

[USER]
Problem: {sub_question_or_operation}
Tool catalog: {tool_catalog_with_signatures}
```

Figure 6: The prompt used by the Tool Executor to invoke primitives from the library.

istry queries often mix multiple operations (unit conversions, ideal gas relations, stoichiometry), where decomposed sub-questions provide a sharper semantic signal for retrieval and reduce the chance of selecting irrelevant tools. Materials science, in contrast, often has higher baseline performance and may require fewer distinct atomic operations per problem, resulting in relatively smaller incremental gains from decomposition.

We also observe that the best configuration depends on both the model and the subject. This motivates the library-size analysis in Appendix F, which explains why increasing the tool inventory does not always monotonically improve performance under question-level retrieval.

D Analysis of Tool Reusability

We investigate the reusability of generated tools by analyzing their invocation frequency (hit-count) across benchmarks. As visualized in the histograms (Figure 8) and Kernel Density Estimation curves (Figure 9), the baseline methods exhibit a severe left-skewed distribution, where the vast majority of tools are concentrated in the lowest frequency bins ($1 \sim 2$ uses). The heavy reliance on "disposable" tools suggests that baselines tend to overfit specific queries with monolithic scripts, resulting in high redundancy and poor transferability.

In contrast, TTE demonstrates a significant right-shift in probability density, effectively redistributing the mass towards moderate-to-high frequency ranges (*e.g.*, $10 \sim 50+$). This phenomenon indi-

Prompt for Tool Synthesizer and Tool Executor

```
[SYSTEM]
You are an expert team composed of specialists from multiple scientific disciplines.
You can perform rigorous interdisciplinary computation by combining step-by-step sub-questions with executable Python
code.

[INPUT]
Main question:
{main_question}

Sub-questions (process in order). Each item provides a sub-question and its corresponding code.
If the code is empty / null, treat it as "missing".
{(step_i, sub_question_i, code_i)}  # repeated for i=1..n

[EXECUTION RULES]
1) If a sub-question provides non-empty and valid code, simulate its execution and write down the structured result.
2) If a sub-question has missing code or the provided code cannot solve the sub-question, generate ONE runnable Python
function:
   - snake_case function name
   - include a clear docstring with I/O specification and units
   - include a minimal test example
   Then simulate executing your function on the test example.

[NUMBERS AND UNITS]
- Use scientific notation with 6 significant digits for floating-point numbers.
- For probabilities/ratios in [0,1], round to 4 decimal places.
- All physical quantities must include SI units. If units are missing, explicitly state the default assumption.

[STRICT OUTPUT FORMAT]
Part 1: For missing-code sub-questions ONLY, output a JSON list wrapped in <code> ... </code>:
<code>
[
  {
    "sub_question": "...",
    "name": "function_name",
    "code": "full Python function as a string",
    "text_description": "brief function summary",
    "io_description": {"input": "...", "output": "..."},
    "test_example": {"input": {...}, "result": "... or null"},
    "error": "optional"
  }
]
</code>

Part 2: Output the final answer wrapped in <answer> ... </answer>:
<answer>
Plain natural-language conclusion (do not mention "code" or "function").
</answer>

[CONCISENESS]
No extra text is allowed outside <code> and <answer>.
```

Figure 7: The hybrid prompt used for synthesizing new tools and deriving the final scientific conclusion.

cates a qualitative transition from generating ad-hoc solutions to discovering atomic computational primitives. By evolving tools that capture fundamental operations (*e.g.*, canonical formulas or unit conversions), TTE reduces redundancy and ensures that the learned tool library consists of generalized modules capable of solving diverse scientific problems through composition.

E Explanation of Evaluation Metrics

E.1 Metrics for TTE-Zero

In the TTE-Zero setting, where the system evolves a tool library $\mathcal{T}$ entirely from scratch, we employ the Tool Reuse Rate ($\text{TRR@}k$) to quantify the utility and generalizability of the synthesized functions.

$$\text{TRR@}k = \frac{|\{t \in \mathcal{T} \mid h(t) \geq k\}|}{|\mathcal{T}|}. \quad (14)$$

We interpret $\text{TRR@}k$ across increasing thresholds to capture distinct dimensions of evolutionary quality. TRR@1 measures the fraction of non-redundant tools that are successfully executed at least once. A value approaching 1.0 indicates minimal computational waste, signifying that the generation process is precise and avoids creating "dead code" or hallucinated functions. TRR@2 reflects immediate transferability, identifying tools that are robust enough to address multiple distinct queries rather than overfitting a single instance. Crucially, higher-order metrics like TRR@5 and TRR@10

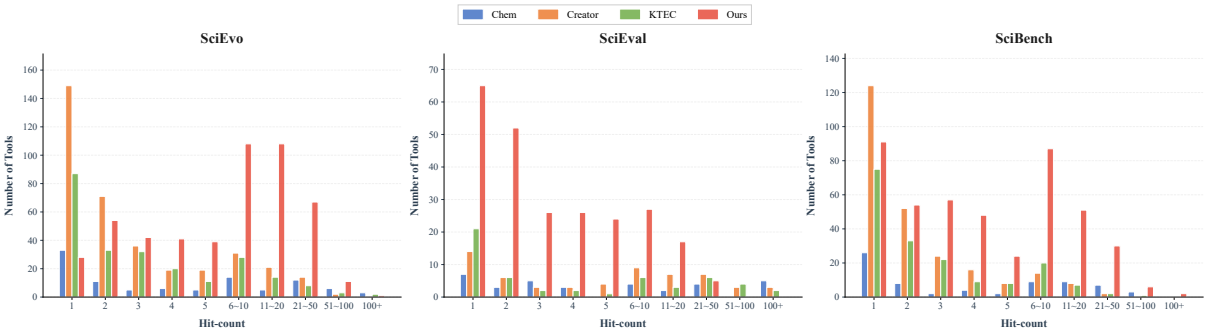


Figure 8: Histogram of tool usage frequency (Hit-count) across three benchmarks. The x-axis represents the reuse frequency of tools, and the y-axis denotes the number of tools.

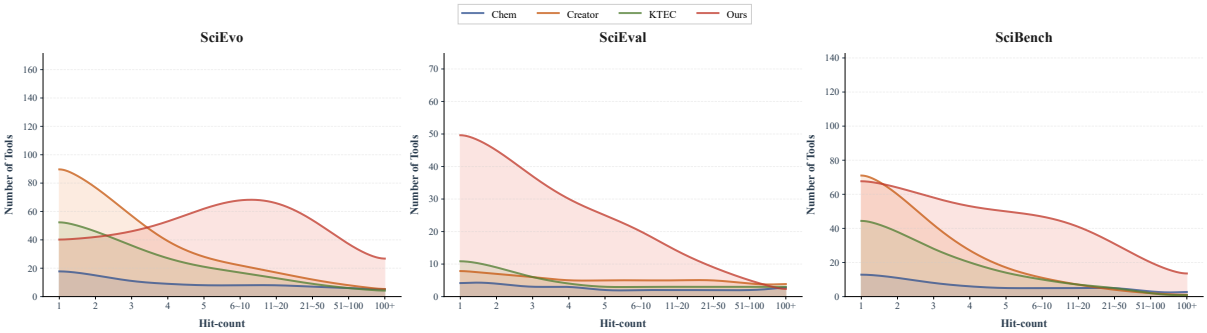


Figure 9: Kernel Density Estimation (KDE) of tool utilization rates. The distribution curves visualize the distributional shift in tool reusability.

serve as indicators for the emergence of core scientific primitives. A high value at these thresholds suggests that the system has autonomously discovered and consolidated fundamental domain operators (*e.g.*, specific unit converters or thermodynamic equation solvers) that are essential for solving a broad class of problems.

E.2 Metrics for TTE-Adapt

In the cross-domain adaptation setting, *i.e.*, TTE-Adapt, the system must balance stability (retaining useful prior knowledge) and plasticity (acquiring new domain-specific capabilities). To rigorously evaluate this dynamic, we decompose the final tool library $\mathcal{T}$ into two disjoint sets: (i) the pre-defined source subset $\mathcal{T}_{\text{pre}}$, transferred from the source domain, and (ii) the newly evolved subset $\mathcal{T}_{\text{new}}$, synthesized autonomously during target-domain inference. We introduce two stratified metrics $\text{TRR}_{\text{trans}}@k$ and $\text{TRR}_{\text{evol}}@k$ to disentangle the sources of competence.

$\text{TRR}_{\text{evol}}@k$. Metric for Knowledge Consolidation (Higher is Better). This is the primary metric for evaluating the quality of adaptation. In standard code generation approaches, models often generate "disposable" scripts, *i.e.*, one-off solutions that solve a single query but lack generalizability. A high $\text{TRR}_{\text{evol}}@k$ (especially for $k \geq 5$) indicates that the system has successfully distilled the "physical laws" or "core primitives" of the new domain into reusable atomic functions. It confirms that the library growth is efficient: the system solves many problems with a compact set of high-quality new tools, rather than overfitting with a bloated library of redundant scripts.

$\text{TRR}_{\text{trans}}@k$. Metric for Negative Transfer Mitigation. This metric monitors the utility of prior knowledge. In cross-domain settings (*e.g.*, Materials $\rightarrow$ Chemistry), we expect $\text{TRR}_{\text{trans}}$ to decrease compared to in-domain settings. A lower value implies the system correctly identifies and prunes source tools that are irrelevant or harmful to the target domain (*e.g.*, discarding a specific material property calculator that is invalid for molecules). However, this value should remain non-zero. A non-zero retention rate signifies the preservation of domain-agnostic capabilities (*e.g.*, basic algebra, statistical functions, unit conversion) that are universally applicable.

Model	Subject	No Tools	Q + Tools			S + Tools		
			100	250	500	100	250	500
GPT-3.5-turbo	Che	0.548	0.580	0.540	0.551	**0.647**	0.589	0.619
	Math	0.503	0.548	0.585	**0.598**	0.564	0.552	0.546
	Phy	0.524	0.600	0.629	**0.642**	0.634	0.609	0.639
	Mat	0.535	0.625	0.589	0.607	0.617	**0.631**	0.607
GPT-4o	Che	0.412	0.536	0.539	0.539	**0.638**	0.594	0.578
	Math	0.341	0.376	0.366	0.366	**0.502**	0.482	0.487
	Phy	0.346	0.533	0.534	0.487	**0.615**	0.557	0.576
	Mat	0.515	**0.652**	0.618	0.642	0.605	0.588	0.608
Qwen2.5-7B	Che	0.211	0.317	0.291	0.312	**0.400**	0.358	0.352
	Math	0.175	0.246	0.246	0.201	0.256	0.241	**0.286**
	Phy	0.227	0.332	0.302	0.304	**0.368**	0.365	0.360
	Mat	0.301	0.358	0.274	0.240	**0.433**	0.384	0.334

Table 4: Subject-wise performance on the SciEvo benchmark based on TTE framework. Che: Chemistry, Math: Mathematics, Phy: Physics, Mat: Materials. Sub-question decomposition ("S+Tools") consistently outperforms main question input ("Q+Tools") across all subjects.

The Substitution Effect. By analyzing the joint trajectory of $(\mathrm{TRR}_{\mathrm{trans}}, \mathrm{TRR}_{\mathrm{evol}})$, we can diagnose the adaptation strategy. Our empirical results shown in Table 3 demonstrate a Substitution Effect: as the domain gap increases, TTE autonomously reduces its reliance on $\mathcal{T}_{\mathsf{pre}}$ (lower $\mathrm{TRR}_{\mathrm{trans}}$) and compensates by up-regulating the synthesis of $\mathcal{T}_{\mathsf{new}}$ (higher $\mathrm{TRR}_{\mathrm{evol}}$). This contrasts with static baselines that suffer from "forced fit", attempting to solve new problems with mismatched old tools.

F The Tool Overload Phenomenon

The empirical analysis of Table 4 reveals a counter-intuitive non-monotonic trend: expanding the tool inventory from 100 to 500 atomic primitives does not consistently translate to performance gains. In specific configurations, particularly those relying on direct query-to-tool matching, increasing the library size paradoxically degrades problem-solving accuracy. We term this observation the "Tool Overload Phenomenon".

Theoretical Analysis. We attribute this degradation to the inherent tension between library richness and retrieval robustness. As the tool library expands, the semantic density of the vector space increases, inevitably shrinking the distance between the optimal tool and high-similarity distractors. This leads to retrieval collisions, where semantically adjacent but functionally distinct tools, *e.g.*, two variations of a thermodynamic calculator with slightly different input assumptions, crowd out the correct candidate during nearest-neighbor search.

Furthermore, even when the correct tool is successfully retrieved, the presence of these high-similarity distractors within the context window introduces significant contextual interference. The language model is forced to perform fine-grained discrimination among subtly different function signatures, which increases the cognitive load of the selection process. The "choice paralysis" consumes the model's reasoning capacity, increasing the likelihood of selecting a suboptimal tool or hallucinating parameters, thereby neutralizing the theoretical benefits of a larger capability set.

Implications for Scalable Agent Systems. These findings suggest that scaling tool libraries requires more than simply accumulating functions. It demands architectural innovations in the retrieval mechanism. To mitigate the noise introduced by library expansion, future systems must move beyond flat similarity search. Potential solutions include hierarchical indexing strategies that first isolate the relevant tool domain before selecting specific atomic functions, or uncertainty-aware retrieval mechanisms that dynamically adjust the number of retrieved candidates based on the semantic ambiguity of the query.

G Case Studies

We provide a detailed examination of two scientific reasoning scenarios to illustrate the specific failure modes of static tool libraries and how the Test-Time Tool Evolution (TTE) framework resolves them.

G.1 Case 1: Molar Mass Estimation

Problem Definition. The task requires estimating the molar mass of a gaseous compound given its density ($1.23\ \mathrm{kg\ m^{-3}}$), temperature ($330\ \mathrm{K}$),

Setting	Outcome and Diagnosis
No Tools	Predicted 76.9 g mol^{-1} (incorrect). The model likely applies an incorrect rearrangement or loses unit consistency without executable validation.
Q+Tools	Predicted 1.73×10^2 g mol^{-1} (incorrect). However, the tool context may contain many irrelevant functions, increasing selection noise and making the success less reliable across instances.
S+Tools	Predicted 1.69×10^2 g mol^{-1} (correct). Decomposition isolates the missing operation "compute molar volume via $V_m = RT/P$" and triggers targeted tool synthesis, producing an explicit executable function with unit handling.

Table 5: Execution results for Case 1. Sub-question decomposition enables targeted tool synthesis and reduces retrieval ambiguity.

Step	Sub-Goal Description	Tool Action	Tool Status	Intermediate Result
1	Convert density from kg/m^3 to g/L	`convert_density(1.23)`	Retrieved	$1.23\ g/L$
2	Convert pressure from kPa to Pa	`convert_pressure(20)`	Retrieved	$20,000\ Pa$
3	Calculate molar volume V_m ($PV = nRT$)	`calculate_molar_volume(20000, 330)`	**Evolved**	$13.738\ L/mol$
4	Calculate molar mass $M = \rho \times V_m$	`calculate_molar_mass(1.23, 13.738)`	Retrieved	$\mathbf{169.0}\ g/mol$

Table 6: Step-by-step execution trace for Case 1. The *Tool Status* column highlights the system's adaptive behavior: it retrieves existing tools for standard operations (Steps 1, 2, 4) but autonomously evolves a new tool for the missing primitive in Step 3.

and pressure (20 kPa). The ground truth is 169 g mol^{-1}. This problem serves as a critical test of the system's ability to handle multi-step reasoning, strict unit consistency, and missing computational primitives.

Performance Comparison. As summarized in Table 5, baseline methods struggle with either hallucination or precision loss. The *No Tools* baseline relies on parametric knowledge, resulting in a physically plausible but numerically incorrect value (76.9 g mol^{-1}), likely due to a misapplication of the Ideal Gas Law rearrangement. The *Q+Tools* setting retrieves a generic density calculator but suffers from noise in the context window, yielding an approximate result (173 g mol^{-1}). In contrast, our *S+Tools* framework achieves the exact analytical solution (169 g mol^{-1}) by enforcing a structured execution path.

Evolutionary Execution Trace. The core advantage of TTE is visualized in Table 6, which details the step-by-step resolution process. The *Problem Analyzer* first decomposes the complex query into four atomic sub-questions. For Steps 1, 2, and 4, the system successfully identifies high-similarity matches in the existing library and retrieves the standard unit conversion and arithmetic tools. However, at Step 3, the system encounters a gap: the library contains generic gas law functions but lacks a specific primitive to calculate molar volume directly from pressure and temperature. Detecting this retrieval failure, the *Tool Synthesizer* is triggered. It generates a dedicated atomic function `calculate_molar_volume` (shown in Figure 10), which correctly handles the gas constant R and unit conversion from m^3 to L. The "evolved" tool is then immediately executed, bridging the computational gap that caused baselines to fail.

G.2 Case 2: Electroplating Stoichiometry

Problem Definition. The problem involves calculating the mass of silver deposited on a tray via electrolysis (current: 8.46 A, time: 8.0 h) and subsequently determining the tray's surface area given a plating thickness of 0.00254 cm and density of 10.5 g/cm^3. The ground truth values are 33.98 g for mass and 1275.6 cm^2 for area. This task requires chaining Faraday's laws of electrolysis with geometric volume-area relationships.

Performance Comparison. Table 7 contrasts the outcomes. The *No Tools* baseline typically fails to integrate the physics constants (Faraday's constant) correctly, leading to magnitude errors. The *Q+Tools* model generates a monolithic function that correctly identifies the physics formulas but likely misinterprets the time parameter or stoichiometry context, yielding a result of 285.6 g, which deviates significantly from the ground truth. In contrast, *S+Tools* achieves high precision (31.6 g and 1283 cm^2) by decomposing the problem into charge calculation, stoichiometric conversion, and geometric derivation, validating

```python
def calculate_molar_volume(pressure_pa, temperature_k):
    """
    Compute molar volume Vm under the ideal gas law: Vm = RT/P.

    Args:
        pressure_pa (float): pressure in Pa
        temperature_k (float): temperature in K

    Returns:
        float: molar volume in L/mol
    """
    R = 8.314462618  # J/(mol*K)
    vm_m3_per_mol = (R * temperature_k) / pressure_pa
    vm_L_per_mol = vm_m3_per_mol * 1000.0
    return vm_L_per_mol
```

Figure 10: Excerpt of a synthesized atomic function for Case 1 that enables correct molar mass computation.

each step independently.

Evolutionary Execution Trace. Table 8 illustrates the adaptive workflow. The system decomposes the physics problem into sequential logic. Steps 1 and 4 utilize existing library tools for basic unit conversion and mass-mole relations. However, for Step 2 (calculating moles of electrons from charge), Step 5 (calculating volume), and Step 6 (deriving area from volume and thickness), the retrieval system returned low-similarity matches. Consequently, the *Tool Synthesizer* evolved dedicated primitives: `calculate_moles_of_electrons` (incorporating Faraday's constant) and `calculate_area`. Figure 11 displays the evolved area calculation tool, which explicitly handles the geometric relationship $A = V/t$, bridging the gap between chemical output and geometric input.

H Dataset Comparison and Uniqueness

SciEvo fills a critical gap in current evaluation protocols by establishing a benchmark that simultaneously assesses scientific reasoning accuracy and the validity of the tool evolution process. As summarized in Table 9, existing benchmarks typically isolate these capabilities, whereas SciEvo couples them to simulate the open-ended nature of real-world scientific research.

H.1 Comparison with Existing Benchmarks

Current benchmarks can be categorized into three groups, none of which fully capture the test-time evolution paradigm. First, scientific reasoning benchmarks like SciBench (Wang et al., 2024) and SciEval (Sun et al., 2024) focus on problem-solving but assume a fixed setting. SciBench provides problem sets without executable tools, forcing models to rely on internal parametric knowledge or external calculators without a unified interface. SciEval offers multi-level evaluation but lacks a mechanism to assess tool generation. Second, function calling benchmarks such as ToolBench (Qin et al., 2024), API-Bank (Li et al., 2023), and BFCL (Patil et al., 2024) focus on the retrieval and invocation of static libraries. While recent works like CONFETTI (Alkhouli et al., 2025) and NESTFUL (Basu et al., 2025) explore complex nested calls, and LongFuncEval (Kate et al., 2025) assesses long-context retrieval, they all operate under the "Closed-World" assumption where the toolset is immutable. Third, code generation benchmarks, *e.g.*, HumanEval (Peng et al., 2024) focus on generating standalone code snippets. While they involve synthesis, they do not evaluate the generated code as reusable library components (tools) that must be maintained and retrieved for future tasks. SciEvo uniquely integrates these dimensions, requiring the agent to not only solve scientific problems but also to maintain and evolve a persistent library of atomic primitives.

H.2 Domain Coverage and Tool Modality

A distinct advantage of SciEvo is its comprehensive disciplinary coverage. Previous domain-specific agents rely on manually curated, static toolkits. For instance, ChemCrow (Bran et al., 2024) provides a specialized library of approximately 19 tools, categorized into general inference (4), molecule manipulation (8), safety checks (3), and reaction

Setting	Outcome and Diagnosis
No Tools	Predicted incorrect values due to lack of domain constants (Faraday's constant) and geometric formulas. Hallucination of intermediate values is common.
Q+Tools	Predicted 285.6 g and $10,800$ cm^2 (incorrect). The model attempted a single-step calculation. While the code logic was syntactically correct, the monolithic execution path failed to align the time units with the specific problem constraints (likely over-scaling the time duration), leading to a large deviation.
S+Tools (TTE)	Predicted 31.6 g and 1.28×10^3 cm^2 (correct). Decomposition enforced a step-by-step validation. The system evolved specific tools for electron mole calculation and area derivation, ensuring dimensional consistency at each interface.

Table 7: Execution results for Case 2. Step-by-step decomposition prevents error propagation in multi-stage physics problems.

Step	Sub-Goal Description	Tool Action	Tool Status	Intermediate Result
1	Calculate total charge $Q = I \times t$	`calculate_charge(8.46, ...)`	Retrieved	$30,500\ C$
2	Calculate moles of electrons $n = Q/F$	`calculate_moles_of_electrons(30500)`	**Evolved**	$0.316\ mol$
3	Stoichiometry (Ag oxidation +1)	[Logic Reasoning]	-	$0.316\ mol\ Ag$
4	Convert moles to mass (Molar Mass)	`calculate_moles(mass, ...)`	Retrieved	$31.63\ g$
5	Calculate volume $Volume = mass/$density	`calculate_volume(31.63,10.5)`	**Evolved**	$3.26\ cm^3$
6	Calculate tray area $A = V/$thickness	`calculate_area(3.26, 0.00254)`	**Evolved**	$\mathbf{1283.5}\ cm^2$

Table 8: Step-by-step execution trace for Case 2. The system evolves specific physics and geometry tools (Steps 2 and 6) when exact matches are missing, while reusing standard chemical tools (Step 4) where appropriate.

processing (4). Similarly, CheMatAgent (Wu et al., 2025) expands this to the materials domain, offering about 34 chemistry tools (*e.g.*, molar mass, solution concentration) and 95 materials science tools (*e.g.*, crystal structure analysis, phase diagram calculation). However, these libraries are static and domain-confined. They lack support for fundamental Mathematics and Physics, which are ubiquitous in interdisciplinary research. As shown in our analysis, SciEvo encompasses not only Chemistry and Materials but also fills the void in Math and Physics. Crucially, unlike the static definitions in ChemCrow or CheMatAgent, the tools in SciEvo are dynamically synthesized, which means the library coverage is theoretically unbounded, capable of evolving from basic arithmetic to complex thermodynamic simulations depending on the inference trajectory.

I Theoretical Analysis

This section provides a rigorous formalization of the mechanisms underpinning the Test-Time Tool Evolution framework. We derive theoretical bounds for tool reusability, analyze the impact of library scaling on retrieval fidelity, quantify error propagation in sequential reasoning, and prove the convergence of the library size under our pruning strategy.

I.1 Utility Gain from Atomic Decomposition

We analyze the expected utility of decomposing a monolithic tool into atomic functions. Let $\mathcal{Q}$ be the space of all possible future queries. A monolithic tool T is defined as a composition of k independent atomic operations $\{a_1, a_2, \ldots, a_k\}$. For any query $q \in \mathcal{Q}$, let $S(q)$ denote the set of atomic operations required to solve q.

Definitions. We define the applicability of a tool using indicator variables. The monolithic tool T is applicable to query q if and only if the query requires the complete set of operations implemented by T, or a set sufficiently similar that T is retrieved and executed monolithically. For strict analysis, we assume T is reused if $S(q) \supseteq \{a_1, \ldots, a_k\}$. Conversely, an atomic tool A_i (corresponding to operation a_i) is reused if $a_i \in S(q)$. Let $R(T)$ and $R(A_i)$ be random variables representing the reuse counts of the monolithic tool and the i-th atomic tool over a stream of M queries.

Theorem 1 (Decomposition Lower Bound). Let p_{partial} be the probability that a query requires a proper subset of operations, specifically that for any i, $P(a_i \in S(q) \mid \{a_1 \ldots a_k\} \not\subseteq S(q)) \geq \delta$ for some $\delta > 0$. The expected aggregate reuse of the decomposed atomic library strictly exceeds that of

```python
def calculate_area(volume_cm3, thickness_cm):
    """
    Calculate the area of the tray.

    Args:
        volume_cm3 (float): Volume of silver (cm^3)
        thickness_cm (float): Thickness of the silver coating (cm)

    Returns:
        float: Area (cm^2)
    """
    # Area = Volume / Thickness
    area = volume_cm3 / thickness_cm
    return area
```

Figure 11: Excerpt of a synthesized atomic function for Step 6 of Case 2. The tool was evolved on-the-fly to link the electrochemical result (volume) with the geometric requirement (area).

Dataset	Domain	Tool Modality	Evolution	Reasoning Type	Tool Coverage
SciBench (Wang et al., 2024)	Science (Gen)	None	No	Chain-of-Thought	-
SciEval (Sun et al., 2024)	Science (Gen)	None	No	Multi-level	-
ToolBench (Qin et al., 2024)	General	Static API	No	API Calling	General APIs
BFCL (Patil et al., 2024)	General	Static Function	No	AST Parsing	Python/Java/JS
ChemCrow (Bran et al., 2024)	Chemistry	Static Library	No	ReAct	19 (Chem)
CheMatAgent (Wu et al., 2025)	Chem & Mat	Static Library	No	ReAct	130 (Chem+Mat)
HumanEval (Peng et al., 2024)	CS/Coding	One-off Code	No	Code Gen	-
SciEvo (Ours)	**Science (Gen)**	**Dynamic/Evolved**	**Yes**	**TTE-Reasoning**	**925**

Table 9: Comparison of SciEvo with existing benchmarks. SciEvo is the only framework that supports Test-Time Evolution across multiple scientific domains, whereas others rely on static libraries or focus solely on code generation without library management.

the monolithic tool:

$$\mathbb{E}\left[\sum_{i=1}^{k} R(A_i)\right] \geq k \cdot \mathbb{E}[R(T)] + \Delta_{\text{flexibility}}, \quad (15)$$

where $\Delta_{\text{flexibility}}$ is a positive term representing the utility of partial reuse.

Proof. Consider a single query q. Let $X_T^{(q)}$ be the indicator that T is reused, and $X_i^{(q)}$ be the indicator that A_i is reused. By definition, if T is reused, all underlying operations are active, implying $X_T^{(q)} = 1 \implies \forall i, X_i^{(q)} = 1$. Therefore, $X_i^{(q)} \geq X_T^{(q)}$ almost surely. Taking expectations over the query distribution, we have $\mathbb{E}[X_i^{(q)}] = P(a_i \in S(q))$ and $\mathbb{E}[X_T^{(q)}] = P(\{a_1 \dots a_k\} \subseteq S(q))$. By the law of total probability, we expand the atomic reuse probability:

$$P(a_i \in S(q)) = P(a_i \in S(q) \mid \text{Mono})P(\text{Mono}) + P(a_i \in S(q) \mid \text{Partial})P(\text{Partial}), \quad (16)$$

Since $P(a_i \in S(q) \mid \text{Mono}) = 1$, the first term is exactly $\mathbb{E}[X_T^{(q)}]$. The second term represents the "partial match" scenario where the monolithic tool fails but the atomic tool succeeds. Summing over all k tools and all M queries, and applying the linearity of expectation, we obtain:

$$\sum_{i=1}^{k} \mathbb{E}[R(A_i)] = M \sum_{i=1}^{k} \Big(\mathbb{E}[X_T^{(q)}] + P(a_i \in S(q), \text{Partial}) \Big). \quad (17)$$

The term $M \sum \mathbb{E}[X_T^{(q)}]$ equals $k \cdot \mathbb{E}[R(T)]$. The remaining sum is strictly positive given that $p_{\text{partial}} > 0$ and the operations have non-zero marginal utility. Thus, decomposition guarantees higher expected reusability by capturing the marginal utility of sub-problems that monolithic tools miss. □

I.2 Retrieval Precision in Growing Libraries

We formally examine the "Tool Overload" phenomenon. Consider a library of size N. For a given query, let s_r be the similarity score of the unique

relevant tool, drawn from a distribution with PDF $f_r(s)$ and CDF $F_r(s)$. Let $\{s_i\}_{i=1}^{N-1}$ be the scores of $N-1$ irrelevant tools (distractors), independently drawn from a noise distribution with PDF $f_n(s)$ and CDF $F_n(s)$. The retrieval system selects the tool with the maximum score.

Theorem 2 (Monotonic Degradation). Assuming the support of the relevant and noise distributions overlap such that $F_n(s) < 1$ for some s where $f_r(s) > 0$, the probability of correctly retrieving the relevant tool, denoted as $P_N(\text{success})$, is a strictly decreasing function of the library size N.

Proof. The relevant tool is retrieved if its score s_r is greater than the maximum of all $N-1$ distractor scores. Let $M_{N-1} = \max\{s_1, \ldots, s_{N-1}\}$. The CDF of the maximum of independent variables is the product of their CDFs, so $P(M_{N-1} \leq x) = [F_n(x)]^{N-1}$. The probability of success is the probability that $s_r > M_{N-1}$. We integrate over all possible values of s_r:

$$\begin{aligned} P_N(\text{success}) &= \int P(M_{N-1} < s \mid s_r = s) f_r(s)\, ds \\ &= \int [F_n(s)]^{N-1} f_r(s)\, ds. \end{aligned} \tag{18}$$

To determine the trend with respect to N, we treat N as a continuous variable and differentiate under the integral sign using Leibniz's rule:

$$\frac{\partial}{\partial N} P_N(\text{success}) = \int_{-\infty}^{\infty} f_r(s) \frac{\partial}{\partial N} [F_n(s)]^{N-1}\, ds. \tag{19}$$

Calculating the derivative inside the integral:

$$\frac{\partial}{\partial N} [F_n(s)]^{N-1} = [F_n(s)]^{N-1} \ln(F_n(s)). \tag{20}$$

Since $F_n(s)$ is a cumulative distribution function, $0 \leq F_n(s) \leq 1$, which implies $\ln(F_n(s)) \leq 0$. For any region where the distributions overlap and retrieval is non-trivial, $F_n(s) < 1$, making the logarithm strictly negative. Thus, the integrand is non-positive everywhere and strictly negative on the set of overlap. Consequently, $\frac{\partial P_N}{\partial N} < 0$, proving that expanding the library without enhancing the retrieval mechanism (*e.g.*, via sub-question decomposition) inevitably increases the error rate.

I.3 Stability of Library Growth

We model the temporal dynamics of the tool library size $L(t)$ to prove that the proposed evolution mechanism leads to a stable system rather than unbounded growth.

Dynamics Model. The rate of change in library size is governed by two opposing forces: the generation of new tools upon retrieval failure and the pruning of low-utility tools. Let λ_g be the maximum potential generation rate. As the library grows, the probability of finding a match increases, suppressing new generation. We model this saturation with a logistic term $(1 - L(t)/K)$, where K represents the effective capacity of the semantic space. Let λ_p be the pruning rate, proportional to the current library size (assuming a constant fraction of tools falls below the usage threshold). The differential equation describing the system is:

$$\frac{dL}{dt} = \lambda_g \left(1 - \frac{L(t)}{K}\right) - \lambda_p L(t). \tag{21}$$

Theorem 3 (Convergence). For any non-negative initial condition $L(0) \geq 0$, the library size $L(t)$ converges asymptotically to a stable equilibrium point L^*.

Proof. We rearrange the differential equation into a standard linear form with constant coefficients:

$$\frac{dL}{dt} = \lambda_g - \left(\frac{\lambda_g}{K} + \lambda_p\right) L(t). \tag{22}$$

Let $A = \lambda_g$ and $B = \frac{\lambda_g}{K} + \lambda_p$. The equation becomes $\frac{dL}{dt} = A - BL(t)$. The equilibrium point is found by setting $\frac{dL}{dt} = 0$, yielding $L^* = \frac{A}{B} = \frac{\lambda_g K}{\lambda_g + \lambda_p K}$. The general solution to this first-order linear ordinary differential equation is:

$$L(t) = L^* + (L(0) - L^*)\, e^{-Bt}. \tag{23}$$

Since $B > 0$ (as generation rate, capacity, and pruning rate are all positive), the term e^{-Bt} decays to zero as $t \to \infty$. Therefore, regardless of whether the library starts empty or full, the system autonomously regulates itself towards the steady-state size L^*. This proves that the TTE framework is robust against explosion in library size, ensuring long-term computational efficiency.

J Future Directions and Broader Impact

The transition to evolutionary tool ecosystems offers a generalizable paradigm for intelligence in non-stationary environments. By treating tools as adaptive capabilities rather than static resources, the Test-Time Tool Evolution framework enables

agents to navigate open-ended challenges. We outline key directions to scale and robustify this paradigm.

Lifecycle Management. Unbounded library growth demands rigorous maintenance to preserve retrieval efficiency. Future research must address the trade-off between plasticity (acquiring new tools) and stability (retaining core competencies). Mechanisms such as intelligent pruning and hierarchical indexing will be critical to forget obsolete primitives while consolidating high-utility functions, preventing knowledge saturation.

Robustness and Calibration. Enhancing the reliability of synthesized tools is a priority. Future systems should incorporate formal verification or uncertainty-aware generation to guarantee code safety. Furthermore, we envision meta-cognitive calibration, where agents dynamically weight the cost of retrieval versus evolution based on confidence, alongside self-correction loops that refine tool logic iteratively upon execution failure.

Multi-Modal Frontiers. Real-world problems require interpreting diagrams or instrument readouts. Extending TTE to multi-modal contexts involves evolving tools for vision-based analysis or graph manipulation. Co-evolving perception and reasoning capabilities represents a key step toward fully autonomous agents capable of conducting end-to-end scientific research.