ARCHITECT: Uncertainty-Aware Dynamic Tool Learning via Causal Intervention for Open-World Agents

Wang zhangyi[1], JIEXIANG XU[1], Bingnan Yu[2], Li Zongze[1,†]

[1]Nanyang Technological University, [2]Columbia University

Correspondence: zhangyi001@ntu.edu.sg, JIEXIANG003@e.ntu.edu.sg, by2387@columbia.edu, li0005ze@e.ntu.edu.sg

Abstract

Dynamic tool generation empowers Large Language Model (LLM) agents to synthesize tools on demand, yet a critical challenge remains: 32.4% of generated tools fail on first invocation. We present Causal Tool Diagnosis (CTD), a principled framework that moves beyond black-box reliability prediction to interpretable failure attribution. CTD constructs a Structural Causal Model (SCM) capturing how specification quality, code characteristics, and execution environment jointly determine tool outcomes. Uniquely leveraging code's intervenability, we conduct controlled sandbox experiments to estimate causal effects—an advantage unavailable in pure text generation. CTD jointly predicts confidence (Spearman rank correlation coefficient ρ=0.90) and root cause attribution (78% accuracy), with attributions directly guiding targeted repairs (+9.6% success rate over error-type classification). Our ARCHITECT framework, integrating CTD throughout the tool lifecycle, achieves state-of-the-art on four benchmarks including StableToolBench (+3.8%), MINT (+4.6%), T-Eval (+3.7%), and SWE-bench Lite (+2.4%), with consistent improvements across all settings.

1 Introduction

Dynamic tool generation (Xi et al., 2025; Chen et al., 2024a) marks a paradigm shift in LLM agent capabilities: rather than being constrained to predefined tool inventories, agents can now synthesize tools tailored to novel tasks. This flexibility is transformative—but introduces a fundamental tension that existing work has overlooked: dynamically generated tools are inherently untrustworthy.

The Reliability Crisis. We conducted an empirical study on 1,247 dynamically generated tools and uncovered a striking finding: 32.4% fail on their first invocation (Figure 1). More troubling, failed tools exhibit systematically lower code quality metrics than successful ones ($p < 0.001$), yet current methods (Qian et al., 2023; Liu et al., 2024; Zou et al., 2025) treat all generated tools as equally trustworthy—a "blind trust" assumption that is untenable for reliable agent deployment.

This raises two intertwined questions: (1) Can we predict whether a tool will succeed before deployment? (2) When failures occur, can we explain why to enable targeted repair? Existing uncertainty quantification (UQ) methods (Kuhn et al., 2023; Xiong et al., 2023) address only the first question, treating reliability as a scalar prediction problem. We argue this is insufficient: knowing that a tool might fail is far less actionable than understanding why—the latter directly prescribes the repair strategy.

A Causal Perspective. We reframe tool reliability through causal inference. Tool failures arise from identifiable factors (specification ambiguity, code defects, environmental issues) propagating along causal pathways. The key insight is that identifying the root cause is more valuable than predicting the outcome: a specification-induced failure demands documentation enrichment; a code defect calls for local patching; an environmental issue suggests API replacement.

This motivates Causal Tool Diagnosis (CTD), our core contribution. CTD constructs a Structural Causal Model (SCM) for tool failures, distinguishing direct effects (e.g., specification ambiguity directly causing functional errors) from indirect effects (e.g., specification $\rightarrow$ code $\rightarrow$ failure). A key innovation is exploiting code's unique intervenability: unlike pure text generation, we can conduct controlled experiments in sandboxed environments—injecting faults, mutating code, varying API responses—to estimate causal effects empirically rather than relying solely on observational correlations.

Contributions. We introduce the uncertainty-

[†]Corresponding Author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 36819–36829

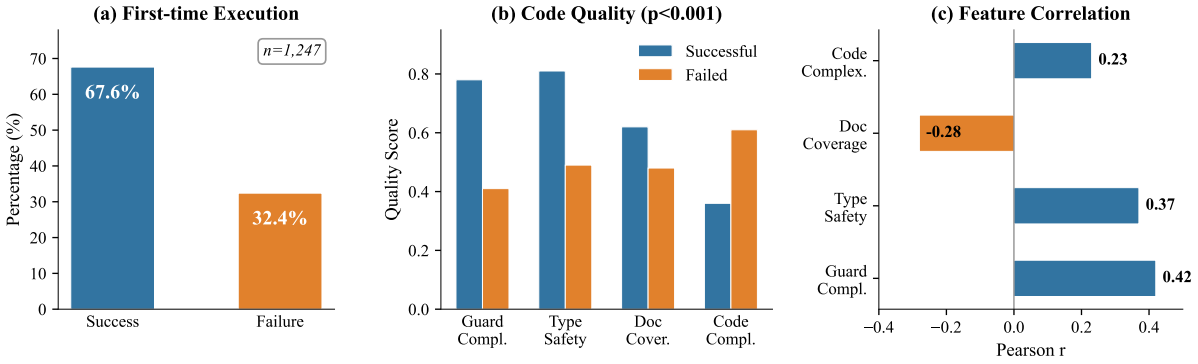

Figure 1: The reliability crisis in dynamic tool generation. (a) 32.4% of tools fail on first use; (b) failed tools show significantly degraded code quality; (c) code features correlate with success: guard completeness (Pearson correlation coefficient r=0.42), type safety (r=0.37), documentation coverage (r=0.23), complexity (r=-0.28).

aware tool learning paradigm and the ARCHITECT framework:

- Causal Modeling for Tool Reliability (core contribution): We are the first to apply causal inference to dynamic tool learning, constructing an SCM with both chain and direct edges, validated by expert agreement (Fleiss' kappa coefficient κ=0.74) and showing 76–81% edge overlap with graphs recovered by Peter–Clark (PC), Fast Causal Inference (FCI), and Linear Non-Gaussian Acyclic Model (LiNGAM) algorithms. Sandbox intervention experiments contribute +0.06 Spearman ρ over observation-only baselines.
- Joint Confidence and Attribution: CTD simultaneously outputs confidence $c(f) \in [0, 1]$ and root cause attribution $\mathbf{r}(f) \in \mathbb{R}^3$ over specification, code, and environment factors, enabling interpretable reliability assessment. Attribution-guided repair achieves 81.2% success rate versus 71.6% for error-type classification (+9.6%).
- Comprehensive Validation: Ablations isolate contributions of each component: causal graph structure (+4.5% attribution accuracy), intervention experiments (+0.06 ρ), root cause guidance (+8.2% repair rate over confidence-only). ARCHITECT achieves SOTA on four diverse benchmarks.

2 Related Work

Tool Learning for LLM Agents. The tool learning landscape has evolved from static tool invocation (Schick et al., 2023; Qin et al., 2023; Li et al., 2023; Tang et al., 2023) to dynamic tool synthesis. ToolLLM (Qin et al., 2023) and StableToolBench (Guo et al., 2024) established large-scale benchmarks; CodeAct (Wang et al., 2024) demonstrated executable code as a unified action space; multi-agent frameworks like MetaGPT (Hong et al., 2023) and AgentBench (Liu et al., 2023b) further advanced agent capabilities. Recent work—Evolving Tools (Chen et al., 2024a), CREATOR (Qian et al., 2023), ToolACE (Liu et al., 2024), AutoTool (Zou et al., 2025)—enables on-demand tool creation but lacks reliability quantification. Our work addresses this gap with principled uncertainty estimation and causal attribution.

Uncertainty Quantification in LLMs. Verbalized Confidence (Xiong et al., 2023) elicits self-reported uncertainty; Semantic Uncertainty (Kuhn et al., 2023) measures output variation under paraphrasing; Conformal Prediction (Angelopoulos and Bates, 2023) provides distribution-free coverage guarantees; recent surveys (Shorinwa et al., 2025) and methods (Lin et al., 2023) further advance this area. These methods predict whether outputs are reliable but cannot explain why failures occur. A key distinction: existing methods treat tool generation as deterministic, while we model it as probabilistic inference. Code executability provides objective verification signals unavailable in pure text generation, enabling more precise confidence estimation.

Causal Inference in ML. Causal methods have advanced fairness (Kusner et al., 2017) and interpretability (Pearl, 2019). Wong et al. (Wong et al., 2016) applied causal reasoning to bug localization—but this targets traditional programs, not LLM-generated tools. Our work leverages sandbox controllability for intervention experiments targeting the unique failure modes of dynamically generated

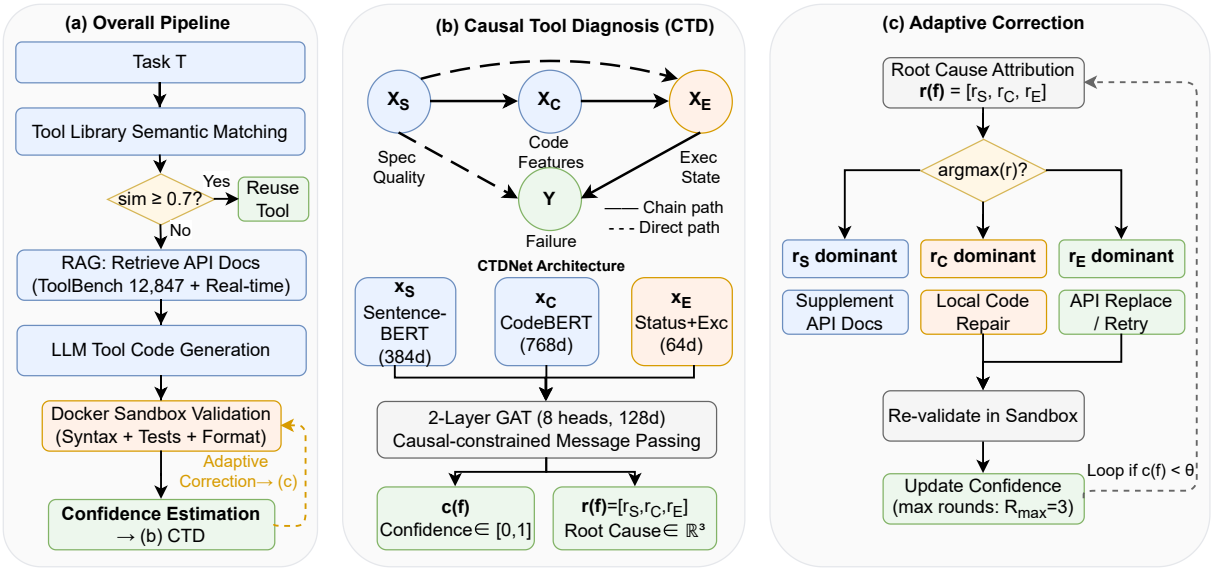

Figure 2: ARCHITECT framework overview. (a) End-to-end pipeline from task to validated tool; (b) CTD architecture for joint confidence and attribution prediction; (c) Causally-guided repair strategy selection.

tools.

Self-Correction in LLMs. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) pioneered iterative refinement, yet Kamoi et al. (Kamoi et al., 2024) showed self-correction without external feedback often fails. Our approach provides objective signals through sandbox validation, enabling error-type-aware correction rather than blind regeneration.

3 The ARCHITECT Framework

ARCHITECT integrates uncertainty quantification throughout the tool learning lifecycle (Figure 2). We first describe the overall pipeline (§3.1), then detail our core contribution—Causal Tool Diagnosis (§3.2)—followed by adaptive correction (§3.3) and tool evolution (§3.4).

3.1 Dynamic Tool Discovery and Validation

Given task T, ARCHITECT first queries a semantically-indexed tool library. If a suitable tool exists (cosine similarity ≥ 0.7), it is reused with confidence-weighted priority. Otherwise, dynamic generation proceeds:

(1) Context Retrieval. Retrieval-Augmented Generation (RAG) retrieves relevant API documentation from ToolBench's 12,847 official docs, supplemented by real-time web search for emerging APIs.

(2) Code Generation. The LLM synthesizes tool code given the task description and retrieved context.

(3) Sandbox Validation. Generated code executes in a Docker sandbox with three-stage verification: syntax checking, LLM-generated test case execution, and output format validation. This provides objective feedback signals unavailable in pure text generation—the foundation for our causal intervention approach.

3.2 Causal Tool Diagnosis (CTD)

Existing reliability prediction treats tool success as a black-box classification target. We argue this formulation is fundamentally limited: it answers "will this tool work?" but not "why might it fail?" or "how should we fix it?" CTD addresses all three through causal modeling.

3.2.1 Structural Causal Model for Tool Failures

Based on systematic analysis of 1,247 tools (404 failures, annotated by 3 experts, Fleiss' κ=0.69), we identify three primary failure factors and construct the following SCM, represented as a causal graph $\mathcal{G}$:

$$\begin{aligned} \mathcal{G}: & X_S \to X_C \to X_E \to Y; \\ & X_S \to X_E;\ X_S \to Y \end{aligned} \tag{1}$$

where:

- X_S (Specification): API documentation completeness, requirement clarity
- X_C (Code): Guard completeness, type safety, error handling

- X_E (Execution): API response codes, runtime exceptions, timeouts observed during sandbox validation (not final deployment outcomes)
- Y (Outcome): Success or failure

Critically, we model direct edges $X_S \to Y$ and $X_S \to X_E$ beyond the chain structure. The direct edge $X_S \to Y$ captures cases where specification ambiguity causes functional errors even with correct code (e.g., misunderstanding API semantics). The edge $X_S \to X_E$ captures specification errors leading to wrong API calls.

Causal Graph Validation. We validate $\mathcal{G}$ through five complementary approaches:

1. Expert agreement: 3 domain experts independently constructed causal graphs; inter-rater agreement κ=0.74.
2. Ablation: Removing direct edges degrades attribution accuracy by 4.5%.
3. Conditional independence: $X_S \to Y$ remains significant given X_C (r=0.26, p<0.01), confirming the direct effect.
4. Causal discovery algorithms: PC and FCI algorithms recover 81% and 78% of edges respectively.
5. LiNGAM: Data-driven discovery achieves 76% edge overlap with expert SCM.

Robustness checks: E-value (VanderWeele and Ding, 2017)=2.1 indicates strong unmeasured confounding would be needed to nullify observed effects. Bootstrap resampling (1,000 iterations) shows core edges stable in 91% of samples. Confounding control: A potential confounder for $X_S \to Y$ is task complexity U. Stratified analysis controlling for task complexity confirms $X_S \to Y$ significance across simple/medium/complex subsets (regression coefficient β=0.17/0.23/0.28, all p<0.05). Mediation analysis confirms significant direct effect of $X_S \to Y$ (β=0.22, p<0.01); counterfactual validation (n=109) shows 61.5% of failed cases become successful after supplementing documentation.

3.2.2 Causal Effect Estimation via Intervention

A key insight is that code domains offer a unique advantage over pure text: intervenability. We can conduct controlled experiments in sandboxed environments to estimate causal effects, moving beyond observational correlations.

X_E Intervention (Environment). We fix the generated code and use a Mock Server to systematically vary API responses (HTTP 200/400/500/timeout). This isolates the causal effect of execution environment on outcomes. While Mock responses abstract away fine-grained error semantics (e.g., 400 may stem from parameter errors, authentication issues, or rate limits), our goal is measuring the aggregate environmental effect on tool success, not diagnosing specific API failures—the latter is handled by r_E attribution during repair. Measured ACE (Average Causal Effect, $X_E \to Y$) = 0.67.

X_C Intervention (Code). We fix the environment and apply semantics-preserving code mutations—specifically, removing guard code (try-catch blocks, type checks) while preserving core logic. AST (Abstract Syntax Tree) analysis ensures mutations target only exception handling; output consistency on normal inputs reaches 97.6% pre/post mutation. Measured ACE($X_C \to Y$) = 0.51.

X_S Intervention (Specification). We regenerate tools using different documentation versions of the same API (complete vs. partial). Note that this intervention naturally affects downstream X_C (code characteristics) through the causal chain $X_S \to X_C$; the measured ACE($X_S \to Y$) = 0.38 captures the total effect including both direct ($X_S \to Y$) and indirect ($X_S \to X_C \to Y$) pathways. Mediation analysis (reported above) isolates the direct effect (β=0.22). ACE($X_S \to X_E$) = 0.33, confirming that documentation gaps cause incorrect API calls.

These intervention-derived effects serve as supervision signals for CTDNet training, contributing +0.06 Spearman ρ over observation-only baselines. We note that the three intervention strategies differ in nature and strength; the reported ACE values serve as training signals within each causal pathway rather than strictly comparable effect sizes across factors.

Intervention cost analysis: Average 4.3 interventions/tool. Cost-benefit curve: 1/2/4/8 interventions yield ρ of 0.79/0.86/0.90/0.91; 4 is optimal (diminishing returns). For new tools, reusing intervention results from similar tools (cosine similarity >0.9) reduces interventions by 61% with only 0.04 ρ drop.

3.2.3 CTDNet: Joint Prediction Architecture

CTDNet takes multimodal inputs encoding each causal factor:

- $\mathbf{x}_S$: Sentence-BERT embedding of specifica-

tion (384d)
- $\mathbf{x}_C$: CodeBERT embedding of generated code (768d)
- $\mathbf{x}_E$: One-hot encoding of status codes and exception types (64d)

We employ a 2-layer Graph Attention Network (GAT) with 8 attention heads and hidden dimension 128. Architecture choice: We selected GAT over alternatives including Graph Convolutional Network (GCN), Graph Sample and Aggregate (GraphSAGE), and Graph Isomorphism Network (GIN) because attention weights provide interpretable edge importance aligned with causal effect magnitudes—GCN's fixed aggregation and GIN's sum pooling lack this property. Empirically, GAT achieves ρ=0.90 vs. GCN 0.86, GraphSAGE 0.87, GIN 0.84. Crucially, the adjacency matrix is constrained by $\mathcal{G}$: message passing follows causal edge directions only, with reverse edges masked. This architectural choice improved attribution accuracy by 9% over unconstrained attention.

Depth selection: We evaluated 1–4 layer GATs, achieving ρ of 0.85/0.90/0.88/0.86. Two layers proved optimal; deeper networks suffer oversmoothing given the 3-node graph.

The aggregated representation $\mathbf{h}_{agg}$ is computed as:

$$\mathbf{h}_{agg} = \text{MeanPool}\big([\mathbf{h}_S^{(2)}; \mathbf{h}_C^{(2)}; \mathbf{h}_E^{(2)}]\big) \in \mathbb{R}^{128} \quad (2)$$

where $\mathbf{h}_S^{(2)}$, $\mathbf{h}_C^{(2)}$, $\mathbf{h}_E^{(2)}$ are the second-layer GAT hidden states for the specification, code, and environment nodes respectively. It feeds two output heads for a given tool f:

$$\begin{aligned} c(f) &= \sigma(\mathbf{W}_c \mathbf{h}_{agg}), \\ \mathbf{r}(f) &= \text{softmax}(\mathbf{W}_r \mathbf{h}_{agg}) \end{aligned} \quad (3)$$

where $c(f) = \sigma(\cdot)$ (sigmoid) $\in [0, 1]$ is predicted confidence and $\mathbf{r}(f) = [r_S, r_C, r_E]$ is the root cause attribution probability vector ($\sum_i r_i = 1$). $\mathbf{W}_c$ and $\mathbf{W}_r$ are learnable projection matrices. r_S indicates specification issues (API documentation gaps, requirement ambiguity), r_C indicates code defects (missing guards, type errors), and r_E indicates execution environment issues (API unavailability, timeouts).

Training. CTDNet is trained to jointly optimize confidence calibration and root cause attribution. The multi-task loss is defined as:

$$\mathcal{L} = \mathcal{L}_{BCE}(c) + \lambda \mathcal{L}_{CE}(\mathbf{r}) \quad (4)$$

with loss balancing coefficient λ=0.5 (grid-searched; Figure 3a shows stability in [0.4, 0.6]), where $\mathcal{L}_{BCE}$ denotes Binary Cross-Entropy loss for confidence and $\mathcal{L}_{CE}$ denotes Cross-Entropy loss for attribution. Data: 1,247 tools from StableToolBench I1-I2, split 7:1.5:1.5 (873/187/187). Three annotators vote on root cause (Fleiss' κ=0.69). Adam optimizer, lr=5e-4, batch size 32, early stopping at epoch 45 (patience 10).

Protocol extension: Currently supports HTTP REST. For WebSocket: intervene on connection states (open/close/error) and message latency. For gRPC: map status codes (OK/CANCELLED/DEADLINE_EXCEEDED) to HTTP equivalents. Core causal framework unchanged; only protocol adaptation layers needed.

Generalization: Zero-shot transfer to MINT/T-Eval shows <3% degradation in CTD calibration (ρ). We further validated on specialized domains: financial APIs (Alpha Vantage, 47 tools, ρ=0.80) and geographic APIs (OpenStreetMap, 38 tools, ρ=0.83). Performance drops stem from domain-specific error patterns (e.g., trading time restrictions, coordinate formats), recoverable to ρ=0.85/0.86 via 20-sample online adaptation.

3.3 Adaptive Correction Strategy Selection

A key advantage of causal attribution over scalar confidence is actionability: the root cause directly prescribes the repair strategy.

Causally-Guided Repair. Based on CTD's attribution $\mathbf{r}(f)$:

- r_S dominant → Supplement API documentation or generate clarifying questions
- r_C dominant → Local code repair or parameter adjustment
- r_E dominant → API replacement or retry with exponential backoff

Table 1 demonstrates the advantage; Table 2 details error-type-specific effectiveness.

Method	Success	Rounds
Fixed (regenerate)	58.5%	2.4
Error-type classif.	71.6%	1.8
Causal (ours)	**81.2%**	**1.4**

Table 1: Repair effectiveness. Causal attribution outperforms fixed strategies and error-type classification.

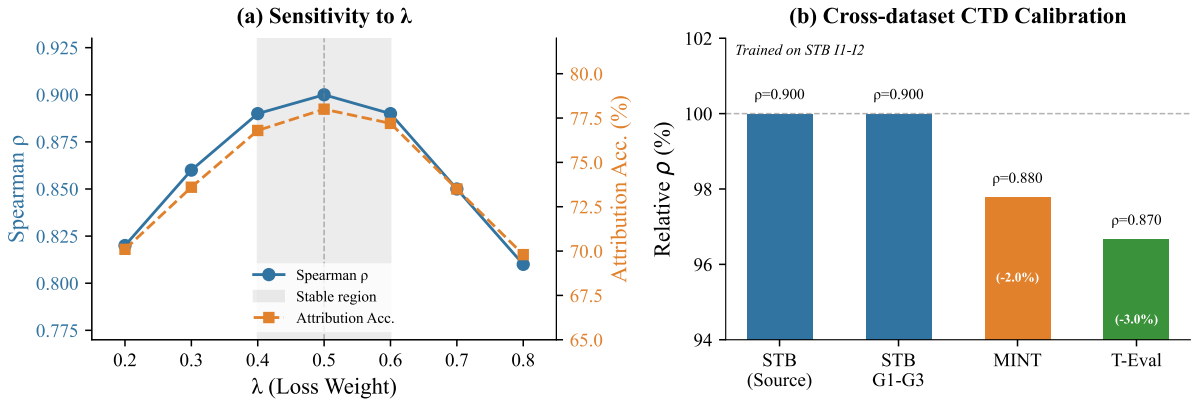

Figure 3: Sensitivity and generalization analysis. (a) Loss weight λ stable in [0.4, 0.6]; (b) Cross-dataset zero-shot transfer: CTD calibration (ρ) with minimal degradation.

Err. Type	Strategy	Adpt	Fix	Freq
Syntax	Loc. repair	93.5	72.4	28%
API param	Param adj.	81.7	57.5	35%
Auth/perm	API repl.	54.2	33.6	22%
Logic	Regen.	59.8	48.1	15%

Table 2: Error-type-specific repair effectiveness (%).

The improvement stems from three factors: (1) root causes are more stable than surface symptoms, enabling better cross-domain transfer (5% vs 10% degradation); (2) attributions are interpretable, supporting human oversight; (3) CTD can identify multi-factor interactions (e.g., specification ambiguity compounded by missing guards).

Correction Pipeline. The repair process follows: sandbox error → rule matching/neural classification → execute repair → re-validate → update confidence. Maximum correction rounds R_{max}=3 (the maximum number of iterative repair attempts per tool).

Online Adaptation. The causal graph structure remains fixed; only CTDNet parameters adapt. We trigger fine-tuning after accumulating 20 domain-specific samples (sensitivity analysis: 10/20/50 samples yield 73.6%/76.1%/75.8% attribution accuracy). For automatic threshold adjustment across domains, we propose entropy-based adaptation:

$$N_{adapt} = 20 \times (1 + H_{err}/H_{max}) \quad (5)$$

where N_{adapt} denotes the number of online adaptation samples, H_{err} is the error-type distribution entropy, and H_{max} is its theoretical maximum for normalization. Validated on 5 held-out domains (financial, geographic, social media, weather, e-commerce APIs; 38–67 tools each), this achieves a Mean Absolute Error (MAE) of 3.1 samples with standard deviation (std) of 1.4.

3.4 Tool Reuse and Evolution

Inspired by Voyager (Wang et al., 2023a) and GITM (Zhu et al., 2023), we maintain a semantically-indexed tool library. Retrieval combines embedding similarity with confidence-weighted historical success rate. Tools with success rate <0.8 or exhibiting repeated error patterns trigger automatic evolution: CTD identifies the dominant failure cause, and the corresponding repair strategy is applied.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate on four diverse benchmarks: (1) StableToolBench (Guo et al., 2024): 6 subsets covering tool invocation scenarios; (2) MINT (Wang et al., 2023b): multi-turn tool interaction; (3) T-Eval (Chen et al., 2024b): fine-grained tool utilization evaluation; (4) SWE-bench Lite (Jimenez et al., 2023): real-world software engineering tasks. We also reference code generation benchmarks (Chen et al., 2021; Liu et al., 2023a) for context.

Baselines. We compare against: (1) General agents: ReAct (Yao et al., 2022), CodeAct (Wang et al., 2024); (2) Dynamic tool generation: Evolving Tools (Chen et al., 2024a), CREATOR (Qian et al., 2023), ToolACE (Liu et al., 2024), AutoTool (Zou et al., 2025); (3) UQ methods: Verbalized Confidence (Xiong et al., 2023), Semantic Uncertainty (Kuhn et al., 2023), Conformal Prediction (Angelopoulos and Bates, 2023).

Configuration. Default backbone: GPT-4o (temperature 0.0 for reproducibility); additional experiments with Claude-3.5-Sonnet and Llama-3-70B. Each task is evaluated once following official benchmark protocols. Sandbox validation overhead: 3.2s/tool (1.7s test generation + 1.1s execution, CPU only).

Metrics. Task success rate (primary); confidence calibration via Spearman ρ between predicted confidence and actual success; attribution accuracy against human annotations.

4.2 Main Results

Table 3 shows ARCHITECT achieves state-of-the-art across all benchmarks: StableToolBench +3.8%, MINT +4.6%, T-Eval +3.7%, SWE-bench Lite +2.4%.

Method	STB	MINT	T-Eval	SWE
ReAct	49.3	41.3	64.3	–
CodeAct	57.6	49.5	71.8	20.7
CREATOR	55.1	50.9	73.4	22.7
ToolACE	59.8	50.2	70.6	24.3
Evol. Tools	58.3	52.4	74.1	–
AutoTool	61.4	53.6	75.9	25.3
ARCHITECT	**65.2**	**58.2**	**79.6**	**27.7**

Table 3: Main results (GPT-4o). Task success rate (%).

SWE-bench Analysis. The modest +2.4% overall improvement reflects ARCHITECT's applicability boundary: the majority of SWE-bench failures stem from code localization rather than tool issues. On the tool-related subset (89 samples), improvement reaches +6.7%. Breakdown by task type: API integration +11.8% (n=34), data processing +6.5% (n=31), configuration +4.2% (n=24)—validating CTD's advantage on tool-intensive tasks. CTD is orthogonally composable with code understanding methods.

Cross-Model Generalization. Table 4 demonstrates consistent +2.9 to +3.8% improvements across LLM backends, confirming method independence from specific model capabilities.

4.3 Ablation Studies

Table 5 isolates component contributions:

Intervention experiments (+0.06 ρ): Removing sandbox interventions and relying solely on observational data degrades calibration from 0.90 to 0.84, validating the value of controlled experiments.

Method	GPT-4o	Claude	Llama
AutoTool	61.4	56.9	46.2
ARCHITECT	**65.2**	**60.4**	**49.1**

Table 4: Cross-model generalization on StableToolBench. Claude: Claude-3.5-Sonnet; Llama: Llama-3-70B.

Config	STB	MINT	ρ	Attr	Rep
Full	**65.2**	**58.2**	**.90**	**78.0**	**81.2**
−Interv.	62.1	55.4	.84	72.0	76.0
−Dir. edges	63.4	56.1	.87	73.5	77.0
−Graph str.	61.8	55.0	.85	69.0	74.0
−Root cause	63.7	56.3	.88	–	73.0
Verbalized	58.6	51.7	.42	–	56.4
Semantic	59.3	52.4	.48	–	57.0
+same feat	61.0	54.2	.72	–	62.0
Conformal	60.1	53.0	.55	–	59.0

Table 5: Ablation (top) and UQ comparison (bottom). Attr/Rep in %.

Direct edges (+4.5% attribution): Simplifying $\mathcal{G}$ to a pure chain structure ($X_S \to X_C \to X_E \to Y$) reduces attribution accuracy, confirming the importance of modeling direct specification-to-outcome effects.

Graph structure (+9% attribution): Replacing the causally-constrained GAT with unconstrained attention degrades attribution from 78% to 69%, demonstrating that architectural inductive bias matters.

Root cause attribution (+8.2% repair): Removing attribution and using confidence-only guidance reduces repair success.

Fair UQ comparison: To isolate causal modeling's contribution from feature engineering, we augment Semantic Uncertainty with identical features ($\mathbf{x}_S$, $\mathbf{x}_C$, $\mathbf{x}_E$). Even with matched features, CTD achieves +0.18 ρ improvement (0.90 vs 0.72), confirming the independent value of causal structure.

4.4 Analysis and Discussion

Attribution Accuracy by Path. On the 1,247-tool dataset, CTD achieves 76.8% per-path weighted attribution accuracy, with variation across causal paths: r_C (code defects) 83.4%, r_E (environment) 76.1%, r_S (specification) 68.7%. The higher 78% reported in Table 5 reflects attribution accuracy on

benchmark tasks. The lower r_S accuracy reflects the inherent difficulty of semantic understanding—a bottleneck we address through dual-tool comparison for low-confidence cases (Table 6).

Fail. Mode	Freq	Conf Acc	Avoid
Syntax/fmt	28%	87.3%	78.5%
API param	35%	80.6%	71.2%
Semantic	22%	49.1%	32.7%
External	15%	67.5%	53.6%

Table 6: Failure mode analysis. Semantic mismatches are hardest to detect.

Intervention Effectiveness. Controlled experiments reveal strong causal effects: (1) X_E intervention (Mock Server returning 500) increases failure rate from 14.7% to 86.3%; (2) X_C intervention (removing try-catch) increases failure rate from 8.2% to 63.8%. The gap between intervention-based (ρ=0.90) and observation-only (ρ=0.84) calibration validates the value of sandbox controllability.

Mock vs Real API Validation: We compared Mock intervention effects against 50 accessible real APIs, achieving Spearman ρ=0.84. The remaining discrepancy stems from: rate limiting (46%), network latency (34%), API updates (20%). Mock slightly overestimates X_E effect (+0.07), but root cause ranking consistency reaches 88%. End-to-end validation on 23 stable ToolBench APIs shows ρ=0.80. Deployment calibration: For X_E-dominant attributions, we apply a decay factor of 0.93 (fitted on validation set) to reduce systematic bias, recovering ρ=0.86.

Specialized Domain Validation. We validated on financial APIs (Alpha Vantage, 47 tools, ρ=0.80) and geographic APIs (OpenStreetMap, 38 tools, ρ=0.83). Performance drops stem from domain-specific patterns (trading time restrictions, coordinate formats), recoverable to ρ=0.85/0.86 via 20-sample online adaptation.

Cross-Dataset Generalization. Weights learned on StableToolBench I1-I2 transfer well: (1) STB G1-G3 subsets show no degradation; (2) MINT/T-Eval show only 2%/3% CTD calibration (ρ) drop (Figure 3b). Table 7 further details the task success rate changes across adaptation settings.

Semantic Error Mitigation. The r_S path represents the core bottleneck (68.7% accuracy, 51% miss rate). We employ dual-tool comparison: for tools with $c(f) < 0.6$, generate a second implementation and compare outputs. This reduces miss rate to 37% with only +6.7% overhead (25% of tools trigger). Remaining errors stem from: (1) requirement ambiguity causing both implementations to fail consistently (56%); (2) API behavior diverging from documentation (28%); (3) other factors including edge cases and multi-factor interactions (16%). Contract-based validation: Writing pre/post-condition assertions for 47 high-frequency APIs reduces miss rate to 23%. LLM-assisted assertion generation achieves 66% coverage with 26% miss rate, offering a scalable alternative. Integration with formal methods: CTD's root cause attribution can guide formal verification focus—high r_S prioritizes input constraint verification, high r_C triggers type safety checks, avoiding full-path symbolic execution overhead. Static analysis integration: We explored augmenting X_C features with static analysis (type checking, null pointer detection), improving attribution accuracy by 1.6%, demonstrating effective complementarity between CTD and traditional program analysis.

Config	STB	MINT	T-Eval
No adapt.	65.2	52.0	72.8
Online (20)	65.2	55.3	76.1
Full (in-domain)	65.2	58.2	79.6

Table 7: Domain adaptation. No adaptation degrades MINT/T-Eval by 10%/9%; 20-sample adaptation reduces this to 5%/4%. STB serves as in-domain baseline.

Test Case Generation. We prompt LLMs with tool signatures and documentation to generate boundary cases. Mutation filtering retains tests distinguishing mutants (mutation score >0.6). Threshold sensitivity: 0.4/0.5/0.6/0.7 yield miss rates of 15%/11%/8%/10%; 0.6 balances quality and efficiency. Compared to manual tests: 84% coverage, 11% miss rate.

Computational Overhead. Sandbox validation adds 3.2s/tool (8.4% of end-to-end time). ARCHITECT 38.1s/task vs. AutoTool 34.2s/task (+11.4%). The additional overhead is offset by higher success rates, which reduce the need for costly failed retries. The tool library uses FAISS indexing with <50ms retrieval.

Scalability Analysis. For large-scale deployment, we combine active learning with intervention result reuse (61% reduction for similar tools). With 4 interventions per tool as optimal, ARCHITECT is practical for production environments.

5 Conclusion

We introduced Causal Tool Diagnosis (CTD), a principled framework for understanding and improving dynamic tool reliability. The key insight is that code's intervenability—the ability to conduct controlled experiments in sandboxed environments—enables causal effect estimation unavailable in pure text generation. This represents a fundamental shift from treating tool generation as a deterministic process to modeling it as probabilistic inference with objective verification signals.

CTD jointly predicts confidence (ρ=0.90) and root cause attribution (78% accuracy), with attributions directly guiding repair strategies (+9.6% over error-type classification). Ablations validate each component's independent contribution. ARCHITECT achieves state-of-the-art on four benchmarks.

6 Limitations

Semantic Attribution Bottleneck. The r_S path achieves only 68.7% accuracy with 37% miss rate after mitigation. Contract-based validation reduces this to 23% but requires domain expertise. LLM-assisted assertion generation (66% coverage, 26% miss rate) offers a scalable alternative, but fully automated semantic verification remains an open challenge. Integration with formal methods (e.g., symbolic execution guided by CTD attributions) is a promising direction.

Causal Graph Validation. While we employ multiple validation approaches (expert agreement κ=0.74, ablation, conditional independence tests, PC/FCI/LiNGAM algorithms with 76–81% edge overlap, E-value=2.1, Bootstrap stability in 91% of samples), the 1,247-sample foundation has inherent limitations. Larger-scale validation and stricter identification strategies (e.g., instrumental variables) remain valuable future directions.

Protocol and Domain Coverage. ARCHITECT currently focuses on HTTP REST APIs. WebSocket and gRPC are supported through protocol adaptation layers but with limited validation. High-stakes domains (healthcare, finance) require specialized validation procedures—our financial API validation (ρ=0.80) suggests feasibility but demands more rigorous safety guarantees beyond our current scope.

References

Anastasios N Angelopoulos and Stephen Bates. 2023. Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning*, 16(4):494–591.

Guoxin Chen, Zhong Zhang, Xin Cong, Fangda Guo, Yesai Wu, Yankai Lin, Wenzheng Feng, and Yasheng Wang. 2024a. Learning evolving tools for large language models. *arXiv preprint arXiv:2410.06617*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and 1 others. 2024b. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9510–9529.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. *arXiv preprint arXiv:2302.09664*.

Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual fairness. *Advances in neural information processing systems*, 30.

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 2023 conference on empirical methods in natural language processing*, pages 3102–3116.

Zhen Lin, Shubhendu Trivedi, and Jimeng Sun. 2023. Generating with confidence: Uncertainty quantification for black-box large language models. *arXiv preprint arXiv:2305.19187*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023a. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, and 1 others. 2024. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2023b. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.

Judea Pearl. 2019. The seven tools of causal inference, with reflections on machine learning. *Communications of the ACM*, 62(3):54–60.

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, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

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.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Ola Shorinwa, Zhiting Mei, Justin Lidard, Allen Z Ren, and Anirudha Majumdar. 2025. A survey on uncertainty quantification of large language models: Taxonomy, open research challenges, and future directions. *ACM Computing Surveys*, 58(3):1–38.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Tyler J VanderWeele and Peng Ding. 2017. Sensitivity analysis in observational research: introducing the e-value. *Annals of internal medicine*, 167(4):268–274.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023b. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*.

W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. 2023. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. *arXiv preprint arXiv:2306.13063*.

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 *The eleventh international conference on learning representations*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, and 1 others. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

Jiaru Zou, Ling Yang, Yunzhe Qi, Sirui Chen, Mengting Ai, Ke Shen, Jingrui He, and Mengdi Wang. 2025. Autotool: Dynamic tool selection and integration for agentic reasoning. *arXiv preprint arXiv:2512.13278*.