

Why Agentic Workflows Fail: A Taxonomy Derived from Literature and Empirical Data

Xin Li, Rami Bahsoon, and Tao Chen

School of Computer Science
University of Birmingham
Birmingham, United Kingdom
x.li.20@bham.ac.uk

Abstract—Agentic workflows combine large language models with planning, memory, tools, execution environments, and iterative feedback. Their failures are difficult to diagnose because the same user-visible symptom may originate from model behavior, workflow orchestration, external tools, infrastructure, or their interaction. This paper presents a source-aware taxonomy of agentic workflow failures and applies it to evidence from academic literature and GitHub. The taxonomy combines source attribution with three empirical dimensions: a concrete failure-type dictionary, workflow-stage localization, and agent-capability localization. Its central structure is a two-axis localization matrix: workflow stages form the horizontal process axis and agent capabilities form the vertical functional axis. This design moves beyond taxonomies that mainly classify final failure manifestations by localizing intermediate process states inside agent operation. Analysis of 397 literature-derived and 1,196 GitHub-derived failure records, followed by a separate temporal hold-out, shows that literature-derived failures emphasize planning, reasoning, learning, perception, and knowledge-related mechanisms, whereas GitHub-derived failures concentrate on execution-stage, actuation-related, tool, API, and infrastructure failures. The dominant hotspot in the collected GitHub evidence is iterative execution and actuation. These source-specific observations, which may also reflect reporting and observability differences, indicate that agentic workflow reliability should be treated as a systems problem rather than only a model-output problem. The taxonomy supports structured diagnosis and engineering triage, and motivates future research on evaluation, root cause analysis, and failure injection.

Index Terms—agentic workflow, LLM agent, failure taxonomy, empirical software engineering, GitHub mining, reliability

I. INTRODUCTION

Large language model (LLM) applications are increasingly organized as *agentic workflows*: systems in which an LLM interprets a goal, plans subtasks, invokes tools or external services, observes intermediate results, revises its behavior, and produces an outcome. Such workflows now appear in software engineering, web automation, data analysis, research assistance, and multi-agent collaboration. Prior work has studied foundation-model risks, LLM trustworthiness, and governance concerns [1]–[5]; tool-augmented and multi-agent architectures [6]–[12]; and benchmarks for web, software, operating-system, and tool-use agents [13]–[19]. These studies show that LLM agents can act beyond single-turn text generation, but they also expose new reliability problems: a failed workflow may result from goal misunderstanding, incomplete planning, wrong tool selection, invalid parameter

binding, unsafe code execution, stale memory, poor recovery, or infrastructure changes outside the model.

Existing taxonomies only partially address this problem. Traditional defect and dependability taxonomies standardize how software anomalies are recorded and analyzed [20]–[22], while LLM risk taxonomies describe harms such as hallucination, bias, privacy leakage, misuse, and brittleness [2], [23]–[25]. However, agentic workflows combine model reasoning, prompts, memory, tools, execution environments, APIs, and human feedback. A single observed symptom may therefore cross responsibility boundaries: an agent may choose the wrong action, an external tool may fail, or the two may interact in a way that produces the failure. Existing work often identifies *what* went wrong, but not consistently *where* the failure occurred in the workflow, *which capability* was implicated, or *whether* the primary responsibility lies in the agent, the surrounding system, or their interaction.

This paper addresses this gap by developing and applying a source-aware taxonomy of agentic workflow failures. The taxonomy combines source attribution with three empirical analysis dimensions: (1) a dictionary of recurring failure mechanisms, (2) workflow-stage localization, and (3) agent-capability localization. The stage and capability dimensions can also be inspected jointly through a stage–capability matrix. The goal is not to replace existing software or LLM risk taxonomies, but to connect their insights to concrete failure episodes in multi-step agentic workflows. This structure allows the same failure record to be described by its responsibility boundary, mechanism, process location, and implicated capability.

We construct the taxonomy from 397 literature-derived failure records and refine and empirically apply it using 1,196 GitHub-derived failure episodes. This literature–GitHub comparison is central to the study design; observed differences are source-specific visibility patterns and need not imply differences in the underlying population prevalence of failures. Literature provides conceptual coverage of failures discussed in prior research, while GitHub provides engineering evidence from issues, pull requests, discussions, commits, and related artifacts. The formal research questions are introduced in Section V, where we align the empirical analysis with three taxonomy dimensions: failure type, workflow stage, and agent capability. We compare literature and GitHub evidence on

each dimension and then integrate the results using the stage–capability matrix to identify cross-source failure hotspots. This paper makes three contributions:

- **A source-aware taxonomy.** We introduce a taxonomy for agentic workflow failures that combines responsibility attribution, failure type, workflow stage, and agent capability.
- **A literature–GitHub empirical comparison.** We apply the taxonomy to 397 literature-derived records and 1,196 GitHub-derived episodes, revealing source-specific visibility patterns rather than establishing differences in underlying failure prevalence.
- **Evidence-based diagnostic insights.** We localize the prominent $S3 \times D$ hotspot in public GitHub evidence, test the source contrast with clustered uncertainty estimates, and illustrate how source attribution can guide engineering triage. Failure injection and monitoring remain future applications.

The remainder of this paper is organized as follows. Section II reviews related work. Section III describes the methodology. Section IV presents the taxonomy. Section V reports the empirical results. Section VI discusses implications, Section VII summarizes threats to validity, and Section VIII concludes the paper and outlines future work.

II. RELATED WORK

Research on agentic workflow failures builds on software defect taxonomies, LLM risk taxonomies, agent benchmarks, and empirical studies of LLM-supported software engineering, but these lines of work usually analyze defects, model-level risks, benchmark outcomes, or task-specific failures separately.

Software and LLM risk taxonomies. Traditional software engineering uses taxonomies to standardize anomaly terminology and guide diagnosis. Orthogonal Defect Classification, the dependability taxonomy by Avizienis et al., and IEEE 1044 distinguish faults, errors, failures, and process feedback mechanisms [20]–[22]. These foundations remain useful, but agentic workflows distribute responsibility across models, prompts, memory, tools/APIs, orchestration, and human feedback. A parallel stream studies foundation-model and LLM risks, including hallucination, bias, misuse, privacy leakage, brittleness, coordination failure, and harmful social impact [1]–[5], [23]–[26]. However, these studies usually operate at the level of risks, governance principles, visible harms, or broad defect concepts rather than concrete failure episodes inside multi-step agentic workflows.

Agent architectures, benchmarks, and software-engineering evidence. Agent frameworks and benchmarks make workflow failures visible through reasoning traces, tool calls, messages, and environment interactions. ReAct, Toolformer, ToolLLM, AgentBench, AutoGen, CAMEL, MetaGPT, and ChatDev connect LLMs to tools, roles, conversations, and collaborative workflows [6]–[13]. WebArena, OSWorld, AppWorld, τ -bench, ToolSandbox, AgentBoard, and MAST evaluate agents in web, GUI, tool-use, long-horizon, and multi-agent settings [14]–[19],

[27]. Recent critiques of agent evaluation further argue that benchmark accuracy alone can obscure cost, overfitting, inadequate holdout sets, and reproducibility issues [28], reinforcing the need for reusable diagnostic views beyond final task success. Empirical software-engineering studies also expose failures through SWE-bench, SWE-agent, OpenHands, Agentless, ChatGPT-based bug fixing, DevGPT and shared developer–ChatGPT conversations, agent-issue benchmarks, and issue-resolution signals [29]–[37]. These works provide realistic evidence, but their analyses are usually tied to a benchmark, framework, or task outcome rather than a reusable process-level localization scheme.

Closest work and diagnostic gap. The closest line of work is recent failure analysis and attribution for LLM-based agent systems, especially Cemri et al.’s MAST study [27] and recent work on automated failure attribution in multi-agent systems [38]. MAST is highly relevant because it analyzes concrete failure patterns in multi-agent traces, including specification and system-design failures, inter-agent misalignment, and verification or termination failures; failure-attribution studies further ask which agent or step is responsible for a failed trace. Our taxonomy is complementary, but differs in scope and diagnostic objective. These studies mainly focus on multi-agent coordination, trace-level failure modes, or responsible-agent localization, whereas our unit of analysis is the broader *agentic workflow failure episode*, including single-agent tool use, orchestration, APIs, runtime environments, human feedback, infrastructure, and mixed agent–system interactions.

Existing taxonomy families provide important but partial coverage. Software defect and dependability taxonomies distinguish faults, errors, and failures, but do not localize failures inside LLM-agent workflow stages or capabilities. LLM risk taxonomies capture harms such as hallucination, bias, privacy leakage, and brittleness, but usually describe model-level risks or final user-visible outcomes. Agent benchmarks and multi-agent studies expose realistic traces, but their labels are often tied to benchmark outcomes, coordination patterns, or final trace-level manifestations. These schemes therefore do not directly provide a joint view of *who or what is responsible*, *what mechanism occurred*, *where in the workflow it occurred*, and *which agent capability was implicated*.

Our taxonomy addresses this gap through a source $\times$ type $\times$ stage $\times$ capability structure that adds diagnostic information beyond final-symptom classification. For example, an agent that keeps retrying a failed tool call without using the returned error signal can be coded as an agentic self-regulation failure at S3: Iterative Execution and Refinement and E: Self-Regulation, pointing to retry limits, monitoring, and recovery logic. In contrast, when an agent sends a valid request but the external API has changed or is unavailable, the same surface symptom is coded as a non-agentic External API/Tool Failure at S3 and D: Actuation, shifting remediation toward interface versioning, schema validation, and dependency monitoring.

Thus, MAST and related trace-level studies show that LLM-agent systems fail in recurring ways, while our taxonomy adds a broader process-localized layer for heterogeneous agentic

workflows. This layer enables direct comparison between literature-derived and GitHub-derived evidence and supports more specific engineering triage: whether a failure is primarily a reasoning problem, an execution problem, a monitoring problem, an external infrastructure problem, or a mixed interaction among these components.

Coverage comparison with MAST. To probe the distinct scope of our taxonomy, we mapped its 65 operational types against the 14 failure modes in MAST [27]. Under this mapping, 63% of GitHub episodes and 67% of literature records fall in types without a MAST counterpart. A complementary item-level inspection of 50 randomly sampled episodes judged 39/50 (78%) not expressible in MAST, compared with 33/50 (66%) suggested by the type-level mapping for those same episodes; the judgments agreed for 36/50 items. Conversely, all 14 MAST modes can be associated with at least one type in our taxonomy, while MAST provides finer distinctions for inter-agent coordination. These comparisons concern expressibility under the chosen mapping, not superior performance of one taxonomy on every diagnostic task.

III. METHODOLOGY

This study follows a source-aware mixed-method design. Literature evidence captures failure concepts studied in prior work, whereas GitHub evidence captures failures observed when agentic workflow systems are built, integrated, and maintained. Both streams are transformed into the same unit of analysis—a *failure episode*—and then coded with the taxonomy defined in Section IV. Figure 1 summarizes the workflow.

A. Evidence Collection Protocol

We used two complementary evidence streams. A mid-2025 literature search covered the preceding five years of work on LLM-based, tool-using, autonomous, and multi-agent workflows. The GitHub stream captured public engineering failures involving tools, APIs, execution environments, user inputs, and infrastructure. The replication package contains the cleaned data, coding prompts, and analysis materials.

1) *Literature Retrieval and Screening:* The literature stream also referred to snowballing practices in software engineering [39], [40]. Table I gives the auditable retrieval protocol. We used broad retrieval terms to avoid missing studies that describe agent failures using different terminology; precision was enforced during screening.

2) *GitHub Retrieval and Cleaning:* GitHub search hits were retrieved using API queries and traced to public issues and pull requests, including their discussions, commits, logs, and fix descriptions. Retrieval combined the system-context terms summarized in Figure 1 with failure-symptom terms such as *failure*, *bug*, *error*, *loop*, *timeout*, *wrong tool*, *invalid output*, and *crash*. The *agentic AI*, *agentic AI fault*, and *agentic system* query families covered ten years ending in May 2025; *agentic workflow* was retrieved through August 2025. To reduce missed records caused by large result sets and API pagination limits, queries were time-sliced into three-month

windows from mid-2023 onward and six-month windows before that.

Cleaning used fuzzy keyword filtering, regular expressions, and manual evidence inspection. We removed exact duplicates, inaccessible records, traditional software defects without agentic workflow behavior, unrelated records, news-like items without primary GitHub evidence, automatically generated comments, and records lacking enough context for episode extraction. The retrieval and reduction funnel was: approximately 7,000 search hits (5,313 distinct issues/PRs), 1,109 after fuzzy matching (partial ratio ≥ 80), and 995 after cleaning. Of these 995 candidate artifacts, 467 contained at least one qualifying failure after step-level inspection. The 467 artifacts comprised 211 issues, 207 pull requests, and 49 items subsequently deleted or moved. These 467 artifacts yielded 3,654 candidate single-fault steps and 1,196 qualifying failure episodes for annotation, originating from 229 repositories. The median repository contributed three episodes, and the largest repository accounted for 8.9% of episodes.

B. Episode Extraction and Taxonomy Construction

A failure episode is a single-fault evidence fragment describing one failure occurrence together with its symptom, context, and, when available, trigger, responsible component, or fix. We use episodes rather than whole papers, issues, or pull requests because one source can contain multiple failures, and several comments can repeatedly describe the same failure. Before any taxonomy labels were assigned, an LLM step-splitting prompt without access to the taxonomy separated each candidate issue/PR into single-fault steps. Episode boundaries were then manually inspected. Distinct fault events were kept separate, while repeated descriptions of the same event were merged. Of the 467 issues/PRs yielding qualifying failures, 301 generated multiple episodes; these within-artifact dependencies motivate clustered inference.

We constructed the taxonomy through a traceable inductive-refinement process rather than by adopting a fixed error list. Literature records were first open-coded to identify recurring failure mechanisms. Candidate labels were consolidated using explicit merge, split, and retention rules: synonymous labels were merged when they referred to the same mechanism; surface-similar failures were split when they differed in cause, diagnostic signal, or remediation strategy; and categories were retained only when they provided diagnostic or engineering value beyond existing labels. This produced an initial literature-grounded codebook containing failure names, definitions, boundary notes, and examples.

The initial codebook was then applied to GitHub episodes as an external stress test against engineering evidence. Episodes that matched an existing definition were coded directly, whereas unmatched episodes were collected as candidate refinements. New types were added only when multiple GitHub records showed a recurring mechanism that could not be reduced to an existing category. After each addition, rename, split, or merge, previously coded records were revisited so that the final dataset used a consistent version of the codebook.

Evidence, Coding, and Validation Pipeline

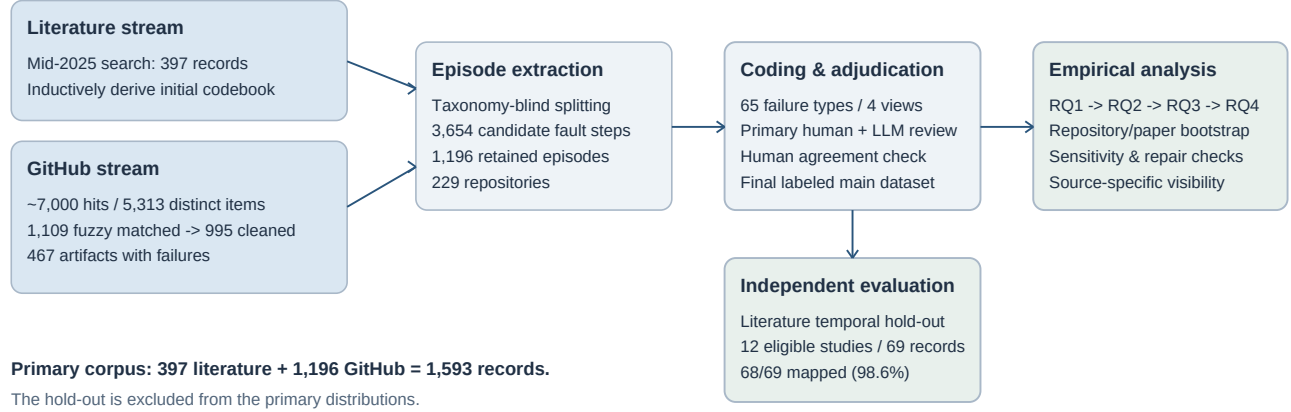


Fig. 1. Overview of the methodology. Literature evidence is used to derive the initial taxonomy, GitHub evidence is used to extend and stress-test it, and reliability controls are used to adjudicate the final coded dataset.

TABLE I
LITERATURE RETRIEVAL AND SCREENING PROTOCOL.

Protocol item	Specification
Databases and expansion	IEEE Xplore, ACM Digital Library, and arXiv; backward and forward snowballing from relevant studies.
Search string	(<i>agent OR agentic OR LLM agent OR autonomous agent OR tool-use agent OR multi-agent OR workflow orchestration OR reflection</i>) AND (<i>failure OR fault OR error OR bug OR breakdown OR invalid output</i>).
Inclusion criteria	I1: studies or evaluates an agentic, tool-using, or multi-agent workflow; I2: reports failure phenomena, causes, symptoms, error patterns, or evaluation breakdowns; I3: provides evidence concrete enough to extract at least one failure episode.
Exclusion criteria	E1: traditional software defects without agentic workflow behavior; E2: pure model risks without workflow or tool-use context; E3: method-only papers with no identifiable failure evidence; E4: duplicate or superseded versions; E5: inaccessible full text.
Screening procedure	Title/abstract screening followed by full-text screening. Ambiguous records were retained until full-text inspection to avoid premature exclusion.

The taxonomy was frozen only after this back-coding pass no longer revealed recurring unmatched mechanisms.

The final taxonomy should therefore be read as the stable endpoint of this systematic induction and stress-testing process. The displayed codebook contains 65 operational types within 16 families and retains the AF, NF, and MF source-attribution branches. Counts in the full classification tree are expressed as literature + GitHub and reflect the latest type-level assignments; top-level literature attribution counts include auxiliary labels and should not be treated as a fully human-adjudicated distribution. It has three linked layers. Each GitHub-derived episode receives: (1) a source attribution label, distinguishing Agentic, Non-Agentic, and Mixed failures; (2) the most specific failure type supported by the evidence; and (3) a workflow-stage and agent-capability localization. The source label captures responsibility, the failure type captures the concrete mechanism, and the stage-capability pair captures the intermediate process state in which the failure is introduced or most directly manifested. Section IV defines these labels and their boundary rules.

C. Annotation and Reliability Controls

The annotation protocol was designed to avoid forcing ambiguous evidence into overly precise labels. Annotators first determined whether the evidence described a genuine failure and then coded source attribution, failure type, workflow stage, and capability. The primary human annotator held B.Sc. and M.Sc. degrees in computer science and had five years of professional software-engineering experience. DeepSeek-V3.1 provided an auxiliary full-data review through the API model `deepseek-chat`, with temperature zero and a three-run majority vote (accessed September 18, 2025), using structured prompts included in the replication package. Keyword matching, dictionary matching, regular-expression checks, and structured LLM suggestions were used only to propose candidate labels; final statistics use adjudicated annotations. Disagreements were reviewed against the original evidence and boundary rules. Because one reviewer in the full double-coding process was an LLM-based auxiliary reviewer, we report these values as *human-LLM agreement* rather than as human-human inter-rater reliability. Cohen’s κ was computed before adjudication, with interpretation guided by established agreement scales [41].

TABLE II

INDEPENDENT HUMAN–HUMAN ANNOTATION AGREEMENT (COHEN’S κ).
THE 50 ADDITIONAL NON-FAILURE STEPS ARE NOT INCLUDED IN THESE
FAILURE-LABEL ROWS.

Dataset	n	Source	Stage	Cap.	Type
GitHub	150	0.71	0.69	0.67	0.65
Literature	80	0.79	0.61	0.63	0.72

TABLE III

ANNOTATION AND RELIABILITY CONTROLS.

Control	Implementation
Source attribution	Classify each episode as Agentic, Non-Agentic, or Mixed according to whether the primary responsibility lies in agent behavior, external conditions, or their interaction.
Failure-type coding	Select the most specific dictionary type supported by evidence; use general labels only when finer attribution is unsupported.
Matrix localization	Map each episode to one workflow stage and one agent capability to capture its intermediate process state.
Tool support	Use fuzzy keyword matching, dictionary matching, regular expressions, and structured LLM suggestions only as candidate labels; final statistics use adjudicated labels.
Reliability check	Report full-dataset human–LLM agreement as auxiliary evidence and independent human–human checks on 150 GitHub failures and 80 literature records (with 50 additional non-failure steps reviewed).

We also conducted pilot calibration before final coding. Initial agreement was moderate ($\kappa \approx 0.3$ – 0.4), largely because of ambiguous boundary cases and inconsistent interpretations of fine-grained failure types and stage–capability localization. We refined the codebook, evidence-extraction templates, and boundary rules, then froze the protocol before full double coding and adjudication. For the literature stream, human–LLM agreement was $\kappa = 0.7211$ for `Failure_name`. For the GitHub stream, human–LLM agreement was $\kappa = 0.8078$ for the corresponding failure-name labels.

To provide a human-only reliability check across all label dimensions, a second annotator with 14 years of computer-science experience was independently calibrated on a disjoint set and then annotated, blind to previous labels and LLM outputs, 150 GitHub failure episodes, 50 non-failure steps, and 80 literature records. Cohen’s κ was computed against the original annotations: for GitHub, source attribution 0.71, stage 0.69, capability 0.67, and failure type 0.65; for literature, source 0.79, stage 0.61, capability 0.63, and type 0.72. The literature source agreement is a sampled reliability check and does not convert the 397 literature attribution labels into a fully human-adjudicated distribution. The 50 non-failure steps were also reviewed; no separate classification metric is reported for them. Human–LLM agreement remains auxiliary evidence. Table III summarizes the main reliability controls.

D. Temporal Hold-Out and Taxonomy Coverage

To evaluate taxonomy coverage beyond the initial mid-2025 retrieval window, we re-ran the original literature search for July 2025–September 24, 2026 under unchanged eligibility criteria. Following deduplication and title/abstract screening, 13 accessible studies entered full-text screening and 12 met the inclusion criteria. The taxonomy was frozen *before* coding these studies; hence this update serves as a temporal hold-out rather than further development data. The 12 studies provided 69 failure records. These records are *not* added to the primary corpus of 397 literature and 1,196 GitHub episodes used for the source-comparison plots and tables.

E. Analysis

After adjudication, we tested source contrasts using chi-square tests for overall distributions and two-sided category-versus-rest Fisher exact tests for major categories/cells, with Benjamini–Hochberg correction and Newcombe 95% confidence intervals for percentage-point gaps. Because failure episodes cluster within repositories and papers, we additionally computed 95% cluster-bootstrap intervals using the repository and paper, respectively, as resampling units. We examined stability under repository weighting, per-repository caps, exclusion of the top ten repositories, leave-one-repository-out analyses, and collapsing repeated mentions. These checks assess stability of the observed source contrast, not its causal explanation. Then we analyzed the dataset according to the four research questions. RQ1 compares failure-type coverage between literature and GitHub by reporting within-source percentages and percentage-point gaps. RQ2 analyzes the workflow-stage dimension alone, corresponding to the horizontal process axis of the taxonomy. RQ3 analyzes the agent-capability dimension alone, corresponding to the vertical functional axis. RQ4 integrates the preceding results across both evidence sources; the stage–capability matrix is used as a synthesis view of intermediate process states rather than as a separate taxonomy dimension. These analyses are reported in Section V; broader implications are discussed in Section VI.

IV. TAXONOMY OF AGENTIC WORKFLOW FAILURES

This section defines the taxonomy used in the empirical analysis. It targets agentic workflows and localizes each failure along four views: source attribution, failure type, workflow stage, and agent capability. The central structure is a two-axis stage–capability matrix that localizes intermediate process states inside agent operation.

The taxonomy should be interpreted as the frozen result of the inductive–refinement process described in Section III-B: literature evidence provided the initial open-coded mechanisms, GitHub evidence stress-tested and extended the codebook, and all labels were back-coded after merge, split, rename, or addition decisions.

Design rationale. Similar symptoms can have different causes. A failed tool call may result from an incorrect agent-generated argument, an unavailable external API, or a malformed tool

response combined with missing agent-side validation. Treating all of these as a single “tool failure” would obscure both diagnosis and remediation. We therefore separate four coding decisions: *where* the failure occurs in the workflow, *which* capability is implicated, *what* mechanism occurred, and *who* or *what* is primarily responsible.

A. Two-Axis Stage–Capability Localization

Figure 2 displays the complete 65-type dictionary under 16 failure families across the Agentic (AF), Non-Agentic (NF), and Mixed (MF) branches, together with an explicit stage–capability matrix. Each type is labeled with a stable identifier (e.g., AF05 or NF03) and its paired literature–GitHub counts; full operational definitions and decision boundaries are in the replication codebook. The stage axis follows the left-to-right lifecycle of an agentic workflow: S0 captures inherited model and training conditions; S1 captures goal interpretation and scoping; S2 captures strategy and task decomposition; S3 captures iterative execution and refinement; and S4 captures final synthesis and response generation. The capability axis captures the functional capacity implicated by the evidence: A: Perception, B: Learning, C: Reasoning, D: Actuation, E: Self-Regulation, and F: Agent/Human Interaction.

The two axes are intentionally orthogonal. A reasoning failure, for example, may appear during goal interpretation, planning, runtime adaptation, or final synthesis. An actuation failure may appear when choosing a tool route, executing a concrete API call, running code, or formatting the final deliverable. The matrix cells therefore should not be read as separate failure types. They are localization cues for identifying the process state responsible for an observed failure. This is the main distinction between our taxonomy and manifestation-oriented classifications: the same final symptom can be traced to different intermediate workflow states.

B. Source Attribution and Failure Mechanisms

Figure 2 organizes the taxonomy into seven Agentic families, five Non-Agentic families, and four Mixed families, with 34 AF, 22 NF, and nine MF operational codes (including general codes). The chart totals 889 Agentic (271 literature + 618 GitHub), 668 Non-Agentic (100+568), and 36 Mixed (26+10) records. The paired counts represent type-coded evidence, not population incidence: the literature component of top-level source attribution relies partly on auxiliary LLM annotations and is not used to infer a cross-source attribution distribution. Source attribution identifies whether responsibility lies primarily in autonomous agent behavior, external conditions, or their interaction. The concrete mechanisms include AF05 Tool Misuse (11+178), NF03 External API/Tool Failure (11+315), AF31 Perception Interpretation (34+12), and NF14 Input Formatting (0+65). Family-level $S \times$ capability labels indicate predominant localization among the family’s records, not a restriction to that single matrix cell.

The empirical RQs follow the same taxonomy structure. RQ1 analyzes failure-type coverage and source-normalized gaps. RQ2 analyzes the horizontal workflow-stage dimension.

RQ3 analyzes the vertical agent-capability dimension. RQ4 integrates the two evidence sources across these dimensions, using the stage–capability matrix as a synthesis view rather than as an additional taxonomy dimension.

Running example (identical symptom, different source).

Consider a failed tool invocation during S3: Iterative Execution. If the agent invents an incorrect tool argument, the primary code is Agentic / Tool Misuse / $S3 \times D$. If the request is valid but an external API is unavailable, it is Non-Agentic / External API/Tool Failure / $S3 \times D$. If malformed external feedback and omitted agent-side validation jointly contribute, a Mixed source label may be warranted. The symptom is similar in all three cases, but the plausible intervention differs across the agent, interface, and boundary. This example is illustrative, not an extra observed episode.

C. Boundary Rules

Each episode receives one *primary* stage and one *primary* capability, rather than an assertion that no other stage or capability contributed. If the introduction point of the responsible defect is evidenced, that point takes precedence; otherwise, annotators use the first observable manifestation. Each episode is coded with the most specific label supported by evidence. General labels are used only when top-level attribution is clear but finer evidence is absent. Mixed labels are reserved for cases where agent behavior and external, human, or system conditions jointly contribute, or where the evidence does not support a single dominant source. Table IV summarizes the rules used to keep coding consistent.

Together, these rules make the taxonomy process-oriented rather than outcome-oriented. Failure type names the mechanism, source attribution identifies responsibility, and the horizontal–vertical matrix localizes the intermediate workflow state. This separation prevents final symptoms from being overgeneralized while keeping the coding scheme compact enough for empirical comparison.

V. EMPIRICAL STUDY RESULTS

The primary (non-hold-out) dataset contains 1,593 classified failure records: 397 from literature and 1,196 from GitHub. Seven GitHub records lacked sufficient evidence for a complete stage–capability pair and are excluded only from matrix-level analyses, yielding 1,586 valid matrix records. Percentages are normalized within each source unless otherwise stated. The revised taxonomy tree assigns 618 GitHub records (51.7%) to Agentic Failures, 568 (47.5%) to Non-Agentic Failures, and ten (0.8%) to Mixed Failures. Its paired type-count summaries also show 271, 100, and 26 literature records, respectively; these literature-level source-attribution groupings involve auxiliary LLM annotations and have not been fully human-adjudicated, so we do not treat them as a directly comparable source-attribution distribution.

A. RQ1: Failure-Type Coverage and Source Gaps

Operationalization. We compare literature and GitHub records by failure type using source-normalized percentages.

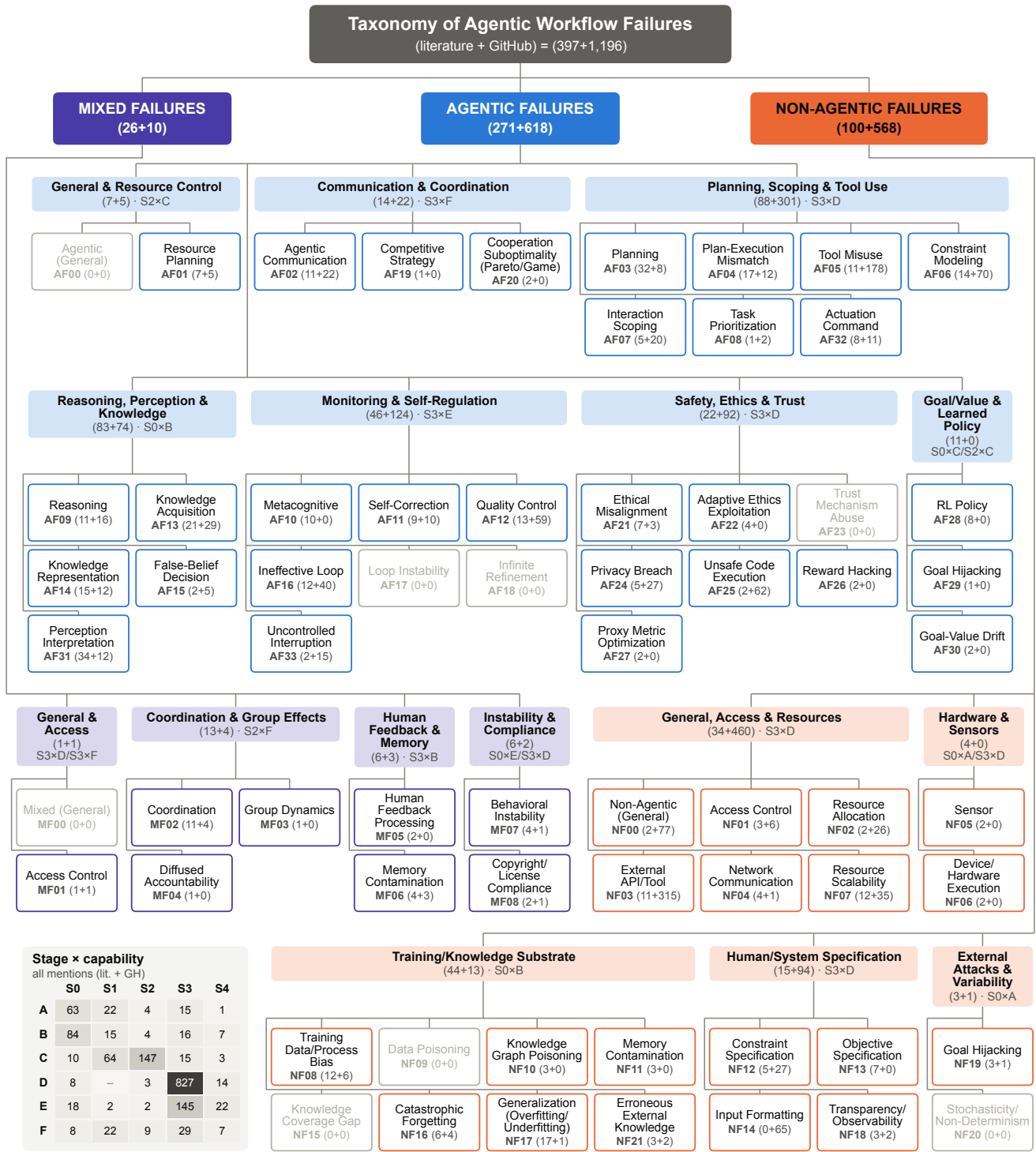


Fig. 2. Full taxonomy of agentic-workflow failures with three source-attribution categories, 16 failure families, 65 operational failure types, and a stage-capability matrix. Each paired count is (literature + GitHub), totaling 397 + 1,196 failure records. The top-level counts are Agentic (271+618), Non-Agentic (100+568), and Mixed (26+10); literature attribution in these displays includes auxiliary labels and is not a fully human-adjudicated source distribution. Family stage-capability tags describe dominant localization, not an additional type label. Matrix cells show primary stage (S0–S4) by capability (A–F) for 1,586 valid records (397 literature; 1,189 GitHub); the seven GitHub records without complete matrix labels remain in type-level counts. A dash denotes zero; the largest cell is S3 × D (827).

The percentage-point gap is defined as GitHub percentage minus literature percentage, where positive values indicate GitHub-dominant types and negative values indicate literature-dominant types.

TABLE IV
CONDENSED BOUNDARY RULES FOR APPLYING THE TAXONOMY.

Decision point	Rule	Example consequence
Earliest responsible stage	Prefer the evidenced stage where the defect was introduced; otherwise use its first observable manifestation, not merely the final user-visible symptom.	Goal misunderstanding leading to wrong output is S1; wrong final synthesis after correct work is S4.
Stage boundary	Separate task understanding (S1), planning-time choices (S2), runtime action/feedback/recovery (S3), and final response construction (S4).	Bad tool choice in a plan is S2; correct tool with malformed runtime arguments is S3.
Capability boundary	Separate decisions and inferences from concrete actions, and separate knowledge/memory from monitoring and recovery.	Wrong algorithm choice is C; malformed shell command is D; context loss is B; repeated failed retry is E.
Source attribution	Use Agentic for avoidable agent behavior, Non-Agentic for external conditions outside direct control, and Mixed only when both sides materially contribute or evidence is underdetermined.	API outage is Non-Agentic; wrong arguments are Agentic; malformed tool response plus no validation is Mixed.
Specificity	Use the most specific supported failure type; use AF00, NF00, or MF00 only when the source is clear but evidence does not support a finer subtype.	Revisit general labels when logs, comments, or fixes reveal a concrete mechanism.

TABLE V
LARGEST SOURCE DIFFERENCES BY FAILURE TYPE. PERCENTAGES ARE NORMALIZED WITHIN EACH SOURCE; GAPS ARE GITHUB PERCENTAGE MINUS LITERATURE PERCENTAGE.

GitHub-dominant types				Literature-dominant types			
Failure type	Lit.%	GH%	Gap	Failure type	Lit.%	GH%	Gap
External API/Tool Failure (NF03)	2.8	26.3	+23.6	Perception Interpretation Failure (AF31)	8.6	1.0	-7.6
Tool Misuse Failure (AF05)	2.8	14.9	+12.1	Planning Failure (AF03)	8.1	0.7	-7.4
Non-Agentic Failure (General, NF00)	0.5	6.4	+5.9	Generalization Failure (NF17)	4.3	0.1	-4.2
Input Formatting Failure (NF14)	0.0	5.4	+5.4	Plan-Execution Mismatch (AF04)	4.3	1.0	-3.3
Unsafe Code Execution (AF25)	0.5	5.2	+4.7	Knowledge Acquisition Failure (AF13)	5.3	2.4	-2.9
Constraint Modeling Failure (AF06)	3.5	5.9	+2.3	Knowledge Representation (AF14)	3.8	1.0	-2.8

Results. Table V shows source-specific concentration at the failure-type level using the names, codes, and paired counts in Figure 2. External API/Tool Failure (NF03) is the largest GitHub-dominant type: 315 GitHub records (26.3%) versus 11 literature records (2.8%), a +23.6 percentage-point gap. Tool Misuse Failure (AF05) and Non-Agentic Failure (General, NF00) have +12.1 and +5.9 percentage-point gaps. The three largest combined types are NF03 (326 records, 20.5%), AF05 (189 records, 11.9%), and Constraint Modeling Failure (AF06; 84 records, 5.3%); together they account for 599 of 1,593 classified records (37.6%).

Finding 1. GitHub-derived evidence has a pronounced concentration in tool/API, input-formatting, code-execution, and other operational failure types.

The three most frequent GitHub types—NF03, AF05, and NF00—account for 570 of 1,196 episodes (47.7%), whereas the same types account for 24 of 397 literature records (6.0%). Input Formatting Failure (NF14; 0 literature + 65 GitHub) and Unsafe Code Execution Failure (AF25; 2+62) reinforce this source-specific visibility of operational failure mechanisms.

Finding 2. Literature-derived evidence places proportionally greater emphasis on perception interpretation, planning, generalization, plan-execution mismatch, knowledge acquisition, and knowledge representation.

The largest negative percentage-point gap belongs to Perception Interpretation Failure (AF31), with 34 literature records (8.6%) and 12 GitHub episodes (1.0%), a -7.6 pp gap. Planning Failure (AF03) follows with 32 literature records (8.1%) and eight GitHub episodes (0.7%), a -7.4 pp gap. Gen-

eralization Failure (NF17; 17+1), Plan-Execution Mismatch Failure (AF04; 17+12), Knowledge Acquisition Failure (AF13; 21+29), and Knowledge Representation Failure (AF14; 15+12) also have literature-dominant normalized shares. Together, these six types represent 136 of 397 literature records (34.3%). Although the literature share is higher, AF13 is also observed in 29 GitHub episodes (2.4% of that source).

These source-specific distributions motivate the separate comparisons of primary workflow stage and agent capability. They describe the observable evidence in the two sampled collections, not a causal difference in true deployment prevalence.

B. RQ2: Workflow-Stage Localization

Operationalization. We compare the two sources across workflow stages S0–S4 using source-normalized percentages. The stage gap is computed as the GitHub percentage minus the literature percentage and is reported in Figure 3(a).

Results. Figure 3(a) shows that S3: Iterative Execution and Refinement has the largest positive source gap. S3 contains 925 GitHub records (77.3%) and 122 literature records (30.7%), yielding a +46.6 percentage-point GitHub gap. In the combined dataset, S3 contains 1,047 records (65.7%), making it the dominant workflow stage overall.

Finding 3. S3: Iterative Execution and Refinement is the dominant GitHub stage and the largest combined-stage hotspot, while literature-derived evidence is more evenly distributed across S0, S2, and S3.

The dominance of S3 indicates that many observed failures are localized after the workflow has moved beyond problem interpretation and planning. However, the difference

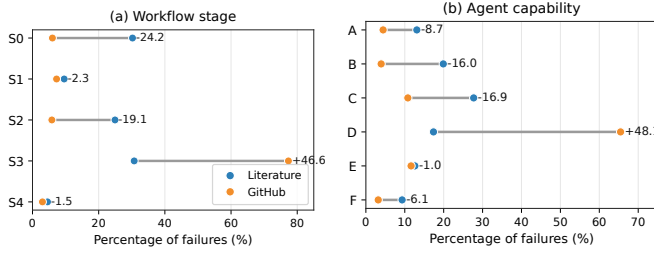


Fig. 3. Comparison of literature-derived and GitHub-derived failures across workflow stages and agent capabilities. Labels indicate the GitHub-minus-literature percentage-point gap. The main marginal gaps are statistically reliable: the stage distribution differs by source, $\chi^2(4) = 348.92$, $p < 0.001$, $V = 0.469$, with S3 showing a +46.6 pp gap, 95% CI [+39.4,+53.2], adjusted Fisher $p < 0.001$; the capability distribution also differs, $\chi^2(5) = 334.19$, $p < 0.001$, $V = 0.458$, with D: Actuation showing a +48.1 pp gap, 95% CI [+41.3,+54.1], adjusted Fisher $p < 0.001$.

between 77.3% in GitHub and 30.7% in literature shows that practical failure evidence is especially concentrated in runtime execution, iterative refinement, feedback handling, and environment-facing action. This pattern is consistent with the high frequency of tool, API, input-format, code-execution, and loop-related failure types reported in RQ1.

Across the four GitHub retrieval query families, S3 accounts for 51–89% of records (versus 30.7% for the original literature corpus), indicating that the observed concentration is not confined to one query family.

Finding 4. Literature-derived evidence retains stronger coverage of earlier workflow stages, especially S2: Strategy and Task Decomposition and S0: Foundational Model and Training.

The largest negative stage gap is S0, with 120 literature records (30.2%) and 72 GitHub records (6.0%), yielding a -24.2 percentage-point gap. S2 also has a negative gap: 99 literature records (24.9%) versus 70 GitHub records (5.9%), yielding a -19.1 percentage-point gap. S1 and S4 are smaller in both sources; S1 remains relatively close across the two sources, while S4 remains proportionally higher in the literature data than in GitHub.

This distribution shows that the literature data provides broader coverage of pre-execution failures, including model-level behavior, goal interpretation, strategy formation, and task decomposition, whereas these stages form a much smaller share of the practice-oriented GitHub corpus.

C. RQ3: Agent-Capability Localization

Operationalization. We compare literature and GitHub records across six agent capabilities: A: Perception, B: Learning, C: Reasoning, D: Actuation, E: Self-Regulation, and F: Agent/Human Interaction. Percentages are normalized within each source, and source gaps are reported in Figure 3(b).

Results. Figure 3(b) shows that D: Actuation is the largest GitHub-dominant capability. It contains 783 GitHub records (65.5%) and 69 literature records (17.4%), producing a +48.1 percentage-point gap. In the combined dataset, D: Actuation contains 852 records (53.5%), making it the dominant capability overall.

Finding 5. GitHub-derived failures are dominated by D: Actuation, indicating that practice-oriented failures are strongly concentrated in the capability responsible for executing actions.

This result refines the stage-level finding. GitHub failures are not only located in S3; they are specifically concentrated in the capability that performs environment-facing operations. These operations include tool calls, code execution, API invocation, file manipulation, command execution, and integration with external systems. This explains why GitHub-dominant failure types in RQ1 are operational rather than purely cognitive.

The actuation contrast also remains positive under repository-clustered uncertainty estimates, as reported in Section V-E.

Finding 6. Literature-derived failures emphasize C: Reasoning and B: Learning, while E: Self-Regulation remains comparatively similar across the two sources.

C: Reasoning contains 110 literature records (27.7%) and 129 GitHub records (10.8%), producing a -16.9 percentage-point gap. B: Learning contains 79 literature records (19.9%) and 49 GitHub records (4.1%). In contrast, E: Self-Regulation accounts for 50 literature records (12.6%) and 139 GitHub records (11.6%), producing only a -1.0 percentage-point gap. Thus, monitoring, correction, and iterative control failures are more balanced across the two evidence sources than reasoning- or learning-related failures.

D. RQ4: Integrated Cross-Source Findings

Operationalization. RQ4 integrates the preceding three analyses rather than introducing a fourth independent taxonomy dimension. We combine the literature and GitHub evidence across failure type, workflow stage, and agent capability to identify the main conclusions supported by both marginal distributions and their intersections. The stage-capability matrix in Figure 4 is used as a synthesis view because it shows how the workflow-stage and agent-capability dimensions intersect in each source and in the combined dataset.

Results. Integrating the three preceding dimensions yields a consistent cross-source conclusion: the failure surface visible in public GitHub evidence is concentrated in runtime execution and actuation, while the literature provides broader coverage of planning-, reasoning-, perception-, and learning-oriented mechanisms. Figure 4 shows the literature, GitHub, and combined stage-capability distributions side by side. The largest matrix hotspot contains 827 combined records, corresponding to 52.1% of the 1,586 records with valid matrix labels. It contains 771 GitHub records and 56 literature records. The next largest cells are S2 × C: Reasoning (147 records, 9.3%) and S3 × E: Self-Regulation (145 records, 9.1%).

Finding 7. After integrating the two sources, the dominant empirical conclusion is that S3 × D: Actuation is the main practical bottleneck, and it is mainly driven by GitHub-derived evidence.

The integrated result shows that the main practice-oriented cluster is more specific than either “S3 failures” or “actuation failures” alone. It is the combination of iterative execution

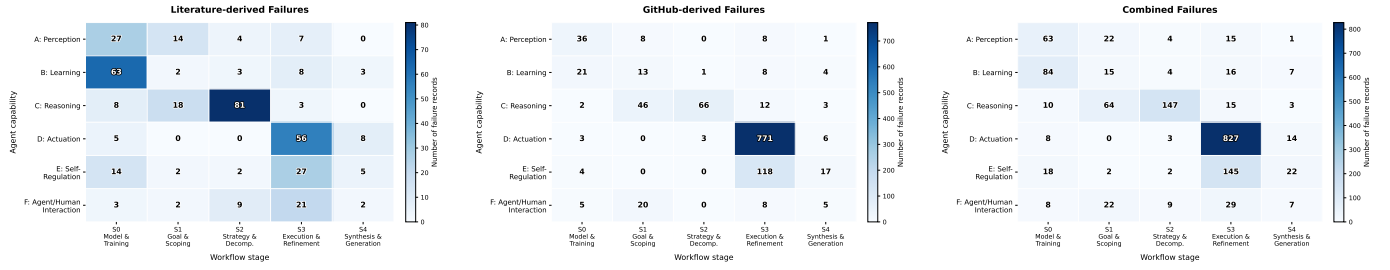


Fig. 4. Stage-capability heatmaps for literature (left), GitHub (center), and the combined dataset (right). Each cell shows the number of failures localized to a workflow stage and capability pair.

and actuation that dominates the GitHub evidence. $S3 \times D$ accounts for 771 of 1,189 valid GitHub matrix records (64.8%), compared with 56 of 397 literature records (14.1%). This cell alone is larger than any other stage-capability cell in either source.

This intersection explains why the stage and capability results reinforce each other rather than representing two unrelated patterns.

Finding 8. The integrated comparison reveals source-specific visibility: literature-derived records emphasize reasoning- and learning-oriented cells, while public GitHub records emphasize execution-time actuation and self-regulation cells.

The literature heatmap shows stronger secondary concentrations in $S2 \times C$ and $S0 \times B$, whereas GitHub is dominated by $S3 \times D$ and also contributes heavily to $S3 \times E$.

The top three integrated cells— $S3 \times D$, $S2 \times C$, and $S3 \times E$ —account for 1,119 combined records, or 70.6% of all valid matrix records. However, they represent different mechanisms and sources: $S3 \times D$ is strongly GitHub-driven, $S3 \times E$ is also GitHub-heavy, and $S2 \times C$ has a larger literature contribution. RQ4 therefore distills the preceding analyses into two research conclusions: the collected GitHub evidence is dominated by execution-time actuation, while the reviewed literature offers stronger coverage of reasoning, learning, and planning mechanisms.

E. Robustness and Additional Diagnostic Checks

Clustered source contrasts. Resampling papers and repositories as clusters yielded the following GitHub-minus-literature differences with 95% cluster-bootstrap confidence intervals: $S3$, +46.6 percentage points [39.1, 53.6]; D : Actuation, +48.1 [40.7, 55.9]; and $S3 \times D$, +50.7 [43.3, 58.7]. These intervals account for clustering and differ from the record-level Newcombe intervals shown in Figure 3. The $S3$ difference remained within +43 to +47 points under repository weighting, per-repository caps, exclusion of the ten largest repositories, leave-one-repository-out analysis, and collapsing repeated mentions. When the literature was restricted to 22 empirical LLM-agent studies, differences remained for $S3$ (+47.2 points; 95% cluster-bootstrap CI [37.3, 56.5]) and $S3 \times D$ (+52.1 points; [42.5, 61.7]). This establishes a persistent *observed source contrast* under these tests but does not identify whether

reporting practices, retrieval, observability, or underlying incidence generate it.

Diagnoses within the main hotspot. Among the 771 GitHub episodes in $S3 \times D$, 61% were non-agentic and 39% agentic; the reported breakdown includes 41% external API/tool failures and 20% tool misuse. Thus, even within one stage-capability cell, a similar “failed tool call” symptom may point to different responsibility boundaries. In an exploratory inspection of 50 episodes, 17 cases had a visible fix. All ten non-agentic cases with visible fixes were addressed outside the agent, whereas four of seven agentic cases were fixed in the agent itself; the remaining three used configuration, framework, or documentation changes. These descriptive results support differentiated triage *in the observed sample*; they do not demonstrate intervention efficacy or generalizable repair success rates.

Temporal hold-out. The additional 12 eligible studies from July 2025–September 24, 2026 yielded 69 failure records, of which 68 (98.6%) could be assigned to existing operational types without expanding the frozen taxonomy. The one unmatched record concerns “context collapse” and is a candidate for a new lower-level type; no new top-level family, stage, or capability was required. This is a coverage check on later literature, not a recomputation of the 1,593-record primary dataset (Section III-D).

VI. DISCUSSION

The data support three conclusions about *source-specific visibility*. First, public GitHub artifacts display strong concentrations in execution and actuation, whereas the sampled literature makes other failure locations more visible; these contrasts are not estimates of differences in underlying real-world incident rates. Second, source attribution adds diagnostic information beyond the surface symptom, as shown by the mix of agentic and non-agentic failures inside $S3 \times D$. Third, robustness checks and the temporal hold-out strengthen confidence in the observed pattern and taxonomy coverage, while leaving the causes of the source difference unresolved.

A. Implications for Academic Research

Visibility rather than proven prevalence. *Finding 2*, *Finding 3*, *Finding 4*, and *Finding 8* describe where the two public evidence collections differ. Sampling frames, developer

reporting incentives, indexing and retrieval, and the observability of intermediate agent decisions differ between research papers and GitHub issues/PRs. Studies of software-repository mining have likewise identified limits of representativeness and reporting [42], [43]. Clustered and subset analyses reduce some compositional concerns but cannot distinguish an unreported failure from an absent failure. For this reason, we avoid equating the observed source contrast with a causal research–practice gap.

Complementarity with trace-oriented taxonomies. Our 65-type to 14-mode mapping and the 50-item inspection suggest additional expressibility for tool/API, runtime, and external-environment failures, while MAST remains more detailed for inter-agent coordination (Section II). Broader scope is not itself proof of higher diagnostic accuracy; future studies should compare coder reliability, utility, and repair decisions on common labeled cases.

Evaluation beyond final-task success. *Finding 3*, *Finding 5*, and *Finding 7* motivate reporting tool-call validity, execution results, feedback use, retry histories, and recoverability, alongside final answers. The added temporal hold-out shows that most later-literature records fit the frozen codebook, but a context-collapse case suggests that the lower-level failure dictionary should remain open to refinement.

B. Implications for Industrial Deployment

Boundary-aware engineering triage. The dominant S3 × D cell includes both non-agentic (61%) and agentic (39%) episodes. These can share a failed-tool-call symptom while calling for different investigations: API contracts, credentials, or dependency health versus tool choice, argument construction, or agent-side validation. The limited visible-fix sample is compatible with this distinction (all 10 non-agentic cases fixed outside the agent; 4 of 7 agentic cases fixed in the agent), but is exploratory and should not be interpreted as a prospective evaluation of a repair method.

What the taxonomy can and cannot yet establish. Source-aware coding can organize investigations and make responsibility boundaries explicit. It does not itself supply a root-cause graph, a tested fault-injection procedure, or a deployed monitoring system. Building and evaluating those interventions, and testing whether the labels improve engineering outcomes relative to symptom-only triage, remain future work.

Overall, the evidence favors a systems-oriented view of agentic reliability, tempered by the reporting and observability limitations of the two public evidence sources.

VII. THREATS TO VALIDITY

Construct validity. The analysis describes failure *episodes*, not unique projects or independent incidents. A taxonomy-blind LLM split candidate issues/PRs into single-fault steps before coding, and the resulting boundaries were manually inspected, but step splitting remains interpretive. Failures may span several stages or capabilities; the labels denote a primary localization, not exclusive involvement. We prefer the evidenced introduction stage and otherwise the first observable

manifestation. We do not estimate the prevalence of multi-stage or multi-capability cases; a stratified sensitivity analysis remains for future work. The fixed codebook also cannot guarantee that every previously unseen lower-level mechanism fits a current type.

Internal validity and annotation reliability. Source attribution, failure types, and stage/capability localizations depend on evidence quality and coding judgments. Full-data human–LLM agreement is auxiliary, while the independent human checks cover samples of 150 GitHub failures and 80 literature records. The extra 50 non-failure steps were inspected without a separately reported confusion measure. The literature source-label reliability sample does not make all 397 literature source labels human-adjudicated. Despite explicit boundary rules and adjudication, errors or missed contextual evidence remain possible.

External validity and source visibility. The GitHub data represent 1,196 episodes from 467 candidate issues/PRs across 229 repositories rather than a random sample of deployed agents. Public issues expose execution and integration problems more readily than some internal decisions; papers have different selection, reporting, and abstraction conventions. Repository-clustered confidence intervals, repository exclusions, weighting, and the restricted 22-study literature comparison show that the measured contrasts persist across several checks, but cannot distinguish sampling/observability biases from differences in underlying failure incidence. The 69-record temporal hold-out supports codebook coverage of later studies without establishing saturation in future settings. Neither the source shares nor the type frequencies should be read as population prevalence estimates.

VIII. CONCLUSION

We developed a source-aware agentic-workflow failure taxonomy and applied it to 397 literature-derived records and 1,196 GitHub-derived failure episodes. Its distinct coding views capture responsibility attribution, failure type, workflow stage, and agent capability, with the stage–capability matrix highlighting intermediate process locations.

Public GitHub evidence is concentrated in S3: Iterative Execution and D: Actuation, particularly their intersection, whereas the sampled literature provides greater visibility into planning, learning, reasoning, and other regions. These source contrasts remain under cluster-bootstrap and repository sensitivity analyses, including a restriction to empirical LLM-agent studies, but do not reveal their causal origin. Independent human-label checks strengthen assessment of coding reliability. In a separate temporal hold-out, 68 of 69 later-literature failure records matched existing types.

The exploratory analysis of visible fixes illustrates how source attribution distinguishes agent-side from external-system intervention loci within a shared failure hotspot. Future research should evaluate whether this additional localization improves triage outcomes and develop and test root-cause analysis, failure injection, and runtime monitoring techniques.

We therefore view agentic-workflow reliability as a source-aware systems problem rather than solely a model-output problem.

DATA AVAILABILITY

The replication package for this study contains the 397 literature-derived and 1,196 GitHub-derived failure records, the taxonomy codebook, annotation labels, coding prompts, and supporting analysis scripts. These materials support inspection of the coding protocol and reproduction of the empirical analyses reported in this paper. The currently available replication package is hosted at <https://anonymous.4open.science/r/literature-github-AWfailure-taxonomy>.

REFERENCES

- [1] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, “On the opportunities and risks of foundation models,” 2021. [Online]. Available: <https://arxiv.org/abs/2108.07258>
- [2] L. Weidinger, J. Uesato, M. Rauh, C. Griffin, P. Huang, J. Mellor, A. Prasad, I. Gabriel, W. Isaac, G. Irving *et al.*, “Taxonomy of risks posed by language models,” in *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, 2022, pp. 214–229.
- [3] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, “On the dangers of stochastic parrots: Can language models be too big?” in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021, pp. 610–623.
- [4] E. Tabassi, “Artificial Intelligence Risk Management Framework (AI RMF 1.0),” National Institute of Standards and Technology, Tech. Rep. NIST AI 100-1, 2023.
- [5] Y. Liu, Y. Yao, J. Ton, X. Zhang, R. Guo, H. Cheng, Y. Klovchov, M. F. Taufiq, and H. Li, “Trustworthy LLMs: A survey and guideline for evaluating large language models’ alignment,” NeurIPS 2023 Workshop on Socially Responsible Language Modelling Research (SoLaR), 2023. [Online]. Available: <https://openreview.net/forum?id=oss9uaPFfB>
- [6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing reasoning and acting in language models,” in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=WE_vluYUL-X
- [7] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, “Toolformer: Language models can teach themselves to use tools,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023. [Online]. Available: <https://openreview.net/forum?id=Yacmpz84TH>
- [8] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian *et al.*, “ToolLLM: Facilitating large language models to master 16000+ real-world APIs,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=dHng2O0Jjr>
- [9] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, “AutoGen: Enabling next-gen LLM applications via multi-agent conversation,” 2023. [Online]. Available: <https://arxiv.org/abs/2308.08155>
- [10] G. Li, H. A. A. K. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, “CAMEL: Communicative agents for “mind” exploration of large language model society,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/a3621ee907def47c1b952ade25c67698-Abstract-Conference.html
- [11] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, “MetaGPT: Meta programming for a multi-agent collaborative framework,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=VtmBAGCN7o>
- [12] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, “ChatDev: Communicative agents for software development,” in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Bangkok, Thailand: Association for Computational Linguistics, 8 2024, pp. 15 174–15 186. [Online]. Available: <https://aclanthology.org/2024.acl-long.810/>
- [13] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang *et al.*, “AgentBench: Evaluating LLMs as agents,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=zAdUB0aCTQ>
- [14] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, and G. Neubig, “WebArena: A realistic web environment for building autonomous agents,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=oKn9c6ytLx>
- [15] T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T. J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, Y. Xu, S. Zhou, S. Savarese, C. Xiong, V. Zhong, and T. Yu, “OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments,” in *Advances in Neural Information Processing Systems*, 2024. [Online]. Available: <https://openreview.net/forum?id=tN61DTtr4Ed>
- [16] H. Trivedi, T. Khot, M. Hartmann, R. Manku, V. Dong, E. Li, S. Gupta, A. Sabharwal, and N. Balasubramanian, “AppWorld: A controllable world of apps and people for benchmarking interactive coding agents,” in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 16 022–16 076. [Online]. Available: <https://aclanthology.org/2024.acl-long.850/>
- [17] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, “ τ -bench: A benchmark for tool-agent-user interaction in real-world domains,” 2024. [Online]. Available: <https://arxiv.org/abs/2406.12045>
- [18] J. Lu, T. Holleis, Y. Zhang, B. Aumayer, F. Nan, H. Bai, S. Ma, S. Ma, M. Li, G. Yin, Z. Wang, and R. Pang, “ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities,” in *Findings of the Association for Computational Linguistics: NAACL 2025*. Albuquerque, New Mexico: Association for Computational Linguistics, Apr. 2025, pp. 1160–1183. [Online]. Available: <https://aclanthology.org/2025.findings-naacl.65/>
- [19] C. Ma, J. Zhang, Z. Zhu, C. Yang, Y. Yang, Y. Jin, Z. Lan, L. Kong, and J. He, “AgentBoard: An analytical evaluation board of multi-turn LLM agents,” in *Advances in Neural Information Processing Systems*, vol. 37, 2024, pp. 74 325–74 362. [Online]. Available: <https://openreview.net/forum?id=4S8agvKjle>
- [20] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M. Wong, “Orthogonal defect classification—a concept for in-process measurements,” *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.
- [21] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.
- [22] IEEE, “IEEE Std 1044-2009: IEEE standard classification for software anomalies,” IEEE, Tech. Rep. IEEE Std 1044-2009, 2009, accessed: May 26, 2026. [Online]. Available: <https://standards.ieee.org/standard/1044-2009.html>
- [23] B. C. Das, M. H. Amini, and Y. Wu, “Security and privacy challenges of large language models: A survey,” 2024. [Online]. Available: <https://arxiv.org/abs/2402.00888>
- [24] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Cheng, Q. Zhang, W. Qin, Y. Zheng, X. Qiu, X. Huang, and T. Gui, “The rise and potential of large language model based agents: A survey,” 2023. [Online]. Available: <https://arxiv.org/abs/2309.07864>
- [25] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, “Large language model based multi-agents: A survey of progress and challenges,” 2024. [Online]. Available: <https://arxiv.org/abs/2402.01680>
- [26] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. T. Iqbal, P. N. Bennett, K. Inkpen *et al.*, “Guidelines for human-AI interaction,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–13.

- [27] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. A. Zaharia, J. E. Gonzalez, and I. Stoica, "Why do multi-agent LLM systems fail?" in *Advances in Neural Information Processing Systems*, vol. 38, 2025, datasets and Benchmarks Track. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2025/hash/b1041e52d3be19f0a9bc491657488e4a-Abstract-Datasets_and_Benchmarks_Track.html
- [28] S. Kapoor, B. Stroebel, Z. S. Siegel, N. Nadgir, and A. Narayanan, "AI agents that matter," 2024. [Online]. Available: <https://arxiv.org/abs/2407.01502>
- [29] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "SWE-bench: Can language models resolve real-world GitHub issues?" in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=VTF8yNQm66>
- [30] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-computer interfaces enable automated software engineering," in *Advances in Neural Information Processing Systems*, vol. 37, 2024. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf
- [31] X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, H. H. Tran, F. Li, R. Ma, M. Zheng, B. Qian, Y. Shao, N. Muennighoff, Y. Zhang, B. Hui, J. Lin, R. Brennan, H. Peng, H. Ji, and G. Neubig, "OpenHands: An open platform for AI software developers as generalist agents," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: <https://openreview.net/forum?id=OJd3ayDDoF>
- [32] C. S. Xia, Y. Deng, S. Dunn, and L. Zhang, "Agentless: Demystifying LLM-based software engineering agents," 2024. [Online]. Available: <https://arxiv.org/abs/2407.01489>
- [33] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of ChatGPT," in *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*, 2023, pp. 23–30.
- [34] T. Xiao, C. Treude, H. Hata, and K. Matsumoto, "DevGPT: Studying developer-ChatGPT conversations," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, 2024, pp. 227–230.
- [35] H. Hao, K. A. Hasan, H. Qin, M. Macedo, Y. Tian, S. H. H. Ding, and A. E. Hassan, "An empirical study on developers' shared conversations with ChatGPT in GitHub pull requests and issues," *Empirical Software Engineering*, vol. 29, p. 150, 2024. [Online]. Available: <https://link.springer.com/article/10.1007/s10664-024-10540-x>
- [36] A. W. Rahardja, J. Liu, W. Chen, Z. Chen, and Y. Lou, "Can agents fix agent issues?" in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. [Online]. Available: <https://chenzhenpeng18.github.io/papers/NeurIPS25.pdf>
- [37] R. Ehsani, S. Pathak, E. Parra, S. Haiduc, and P. Chatterjee, "What characteristics make ChatGPT effective for software issue resolution? an empirical study of task, project, and conversational signals in GitHub issues," *Empirical Software Engineering*, vol. 31, p. 22, 2026. [Online]. Available: <https://link.springer.com/article/10.1007/s10664-025-10745-8>
- [38] S. Zhang, M. Yin, J. Zhang, J. Liu, Z. Han, J. Zhang, B. Li, C. Wang, H. Wang, Y. Chen, and Q. Wu, "Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems," in *Proceedings of the 42nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 267. PMLR, 2025, pp. 76 583–76 599. [Online]. Available: <https://proceedings.mlr.press/v267/zhang25cq.html>
- [39] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University and Durham University Joint Report, Tech. Rep. EBSE 2007-001, 2007.
- [40] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–10.
- [41] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.
- [42] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 92–101.
- [43] C. Bird et al., "Fair and balanced? bias in bug-fix datasets," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on Foundations of Software Engineering*, 2009, pp. 121–130.