Human in the Loop for Fuzz Testing: Literature Review and the Road Ahead

JIONGCHI YU*, Singapore Management University, Singapore
XIAOLIN WEN*, Nanyang Technological University, Singapore
SIZHE CHENG, Nanyang Technological University, Singapore
XIAOFEI XIE, Singapore Management University, Singapore
QIANG HU†, Tianjin University, China
YONG WANG†, Nanyang Technological University, Singapore

Fuzz testing is one of the most effective techniques for detecting bugs and vulnerabilities in software. However, as the basis of fuzz testing, automated heuristics often fail to uncover deep or complex vulnerabilities. As a result, the performance of fuzz testing remains limited. One promising way to address this limitation is to integrate human expert guidance into the fuzz testing paradigm. Even though some work has been proposed in this direction, there is still a lack of a systematic research roadmap for combining Human-in-the-Loop (HITL) and fuzz testing, hindering the potential for further enhancing fuzzing effectiveness. To bridge this gap, this paper outlines a forward-looking research roadmap for HITL for fuzz testing. Specifically, we highlight the promise of visualization techniques for interpretable fuzzing processes, as well as on-the-fly interventions that enable experts to guide fuzzing toward hard-to-reach program behaviors. Moreover, the rise of Large Language Models (LLMs) introduces new opportunities and challenges, raising questions about how humans can efficiently provide actionable knowledge, how expert meta-knowledge can be leveraged, and what roles humans should play in the intelligent fuzzing loop with LLMs. To address these questions, we survey existing work on HITL fuzz testing and propose a research agenda emphasizing future opportunities in (1) human monitoring, (2) human steering, and (3) human-LLM collaboration. We call for a paradigm shift toward interactive, human-guided fuzzing systems that integrate expert insight with AI-powered automation in the next-generation fuzzing ecosystem.

CCS Concepts: • **Software and its engineering → Software testing and debugging**; • **Human-centered computing → Information visualization**; **Human computer interaction (HCI)**.

Additional Key Words and Phrases: Fuzzing, Visualization, Human-Computer Interaction

1 Introduction

As software systems continue to grow in scale and complexity, ensuring their security and reliability has become a central challenge in software engineering. Among dynamic testing methods, fuzz testing, or fuzzing, has emerged

*Both authors contributed equally to this research.
†Corresponding authors

Authors' Contact Information: Jiongchi Yu, Singapore Management University, Singapore, Singapore, jiongchiyu@acm.org; Xiaolin Wen, xiaolin004@e.ntu.edu.sg, Nanyang Technological University, Singapore, Singapore; Sizhe Cheng, sizhe003@e.ntu.edu.sg, Nanyang Technological University, Singapore, Singapore; Xiaofei Xie, xfxie@smu.edu.sg, Singapore Management University, Singapore, Singapore; Qiang Hu, qianghu@tju.edu.cn, Tianjin University, Tianjin, China; Yong Wang, yong-wang@ntu.edu.sg, Nanyang Technological University, Singapore, Singapore.

ACM 1557-7392/2026/8-ART
https://doi.org/10.1145/3833416

as one of the most effective techniques. By automatically generating and executing large volumes of randomized or mutated inputs, fuzzers can uncover critical defects, including memory safety violations, boundary errors, and protocol inconsistencies. Since its introduction in the early 1990s [40], fuzzing has evolved from simple random testing into a rich ecosystem of tools and techniques, widely deployed in both industry [1, 4] and academia [7, 11]. Its ability to explore deep code paths and produce diverse test cases has made fuzzing indispensable in software security.

Despite these advances, fuzzing still faces non-negligible limitations. Automated fuzzers often struggle to overcome semantic roadblocks, construct robust drivers, and interpret complex input structures. The large volumes of runtime data they generate are difficult to interpret without domain expertise, and automated heuristics can easily plateau in coverage. These limitations make human expertise valuable, but they also make HITL fuzzing more challenging than human involvement in many other testing settings. Fuzzing is a high-throughput and feedback-driven search process: millions of inputs may be generated and executed, feedback signals are often low-level and noisy, and useful intervention points are distributed across preprocessing, seed management, scheduling, testcase synthesis, and execution. As a result, the human role is not limited to reviewing test results or manually designing a small number of test cases. Instead, testers must interpret evolving search states, identify semantic bottlenecks, and translate domain knowledge into actions that can influence the automated loop without sacrificing scalability or reproducibility.

These characteristics create a distinctive interaction problem for HITL fuzzing. Human input can be effective when it helps select meaningful seeds, specify targets, annotate hidden states, refine harnesses, inject constraints, or interpret coverage plateaus. However, such input is costly and difficult to time. Early intervention may be too speculative, while late intervention may require stopping, reconfiguring, and rerunning the campaign. Moreover, many fuzzing decisions are indirect: a human may observe a runtime bottleneck, but the actual intervention may need to occur through seed updates, scheduling hints, mutation constraints, or instrumentation changes. This cross-phase coupling makes HITL fuzzing different from one-shot manual inspection, post-hoc test result analysis, or conventional interactive debugging.

Existing surveys have begun to discuss these issues from different perspectives. Kadron *et al.* [29] discuss how expert guidance can augment fuzzing and symbolic execution. Yan *et al.* [61] categorize human contributions such as knowledge injection, seed management, and visualization. Bohme [5] raises fundamental questions about usability and human-in-the-loop communication in fuzzing. Huang *et al.* [25] review emerging LLM-for-fuzzing techniques while calling for stronger HITL perspectives. However, these surveys remain fragmented. They do not yet provide a systematic phase-oriented taxonomy of human involvement in fuzzing, nor do they fully consider the evolving interplay between human expertise and emerging LLM-assisted techniques.

Recent progress in large language models (LLMs) has opened new possibilities for fuzzing. LLMs can synthesize structured inputs, infer protocols from documentation, generate fuzz drivers, and act as lightweight solvers for path conditions. Early works demonstrate that LLMs can, in some scenarios, partially substitute human roles in fuzzing by providing knowledge, adaptivity, and generative capability at scale [14, 39, 49, 57]. Nonetheless, LLMs remain hampered by serious limitations [25, 63]. They may produce brittle, low-quality, or hallucinated artifacts, struggle with reasoning transparency, and suffer from context-length limits, inconsistency, or insufficient domain grounding. These defects can lead to invalid test inputs, false positives, or coverage gaps in fuzzing contexts. Recent surveys [24, 28] further identify common pitfalls in LLM-based fuzzing, such as invalid driver generation, limited input diversity, and model hallucination in reasoning. The emerging paradigm is therefore not human versus LLM, but human-LLM collaboration: future fuzzing workflows will likely blend human oversight, domain judgment, and validation with the generative and adaptive capabilities of LLMs, aiming to balance efficiency and reliability.

Building on insights from the human-computer interaction (HCI) community and motivated by recent progress in LLM-assisted fuzzing, this paper conducts a systematic survey that bridges fuzzing, HCI, and LLM-assisted

fuzzing. From an initial corpus of 1,308 papers, we identify 44 works that explicitly integrate human involvement into fuzzing. Our analysis categorizes these works into two broad roles: (1) human monitoring, where humans observe and interpret fuzzing states, and (2) human steering, where humans intervene to influence exploration. The review shows that existing HITL fuzzing systems improve visibility into fuzzing behavior, but they still provide limited support for acting on runtime evidence. Human steering is concentrated in offline phases such as preprocessing and seed management, while scheduling and testcase synthesis remain difficult to expose and control. Existing evaluations also focus mainly on coverage and bug discovery, with limited attention to human effort, intervention cost, or return on expert input.

These findings suggest that HITL fuzzing requires more than adding a user interface around an automated fuzzer. It requires interaction mechanisms that preserve fuzzing throughput while making human expertise timely, actionable, and measurable. In this context, LLMs offer a new opportunity to reduce manual burden in tasks such as seed generation, driver construction, constraint reasoning, and crash interpretation. However, LLM outputs may also be invalid, brittle, or insufficiently grounded in program behavior. The emerging paradigm is therefore human-LLM collaborative fuzzing, where LLMs provide scalable assistance, and humans retain responsibility for goals, validation, and domain judgment. Building on this review, we articulate the opportunities and risks of this paradigm and propose a roadmap toward a more sustainable, scalable, and effective fuzzing ecosystem. Our contributions are as follows:

- **Systematic Literature Review.** We collect and analyze 44 research papers that explore HITL in fuzzing, providing a structured taxonomy of approaches, their strengths, limitations, and application scenarios.
- **Interaction Design Space Analysis.** Drawing on fuzzing's five canonical phases, HCI principles, and models of LLM-assisted fuzzing, we map the design space of human involvement, discussing in the literature review the trade-offs they entail and what challenges remain unresolved.
- **Visionary HITL Fuzzing Roadmap.** We extend the discussion to the role of LLMs in fuzzing, highlighting how LLMs can augment or replace human tasks, and we outline research directions about how humans can be involved in future fuzz testing and LLM-assisted fuzzing.

2 Background

2.1 Fuzz Testing Workflow

Fuzz testing (fuzzing) is a dynamic testing technique that repeatedly generates and executes test inputs to expose crashes, hangs, and other abnormal behaviors. Since the early empirical study of random testing on UNIX utilities [40], fuzzing has evolved from simple random input generation into a broad family of feedback-guided, coverage-guided, directed, grammar-based, hybrid, and domain-specific testing techniques [6, 7, 11, 18]. Modern fuzzers commonly operate as an iterative feedback loop in which generated inputs are executed, execution feedback is collected, and subsequent generation strategies are adapted accordingly.

Following prior work [61], we conceptualize a contemporary fuzzing workflow as five interdependent phases, as illustrated in Figure 1. These phases provide the technical foundation for analyzing where and how human involvement can be introduced.

- **Preprocessing.** This phase prepares the target program and its execution environment before fuzzing starts. It usually includes instrumentation and harness construction. Coverage-guided fuzzers such as AFL++ rely on instrumentation to expose edge coverage as feedback [18]. Symbolic and taint-based techniques further use program analysis to expose constraints and data dependencies [9, 20]. This phase determines what signals the fuzzer can observe and which program components can be reached during testing.
- **Seed Management.** This phase curates the input corpus that bootstraps the fuzzing campaign. It includes seed selection and seed minimization. It also updates the corpus when newly generated inputs expose

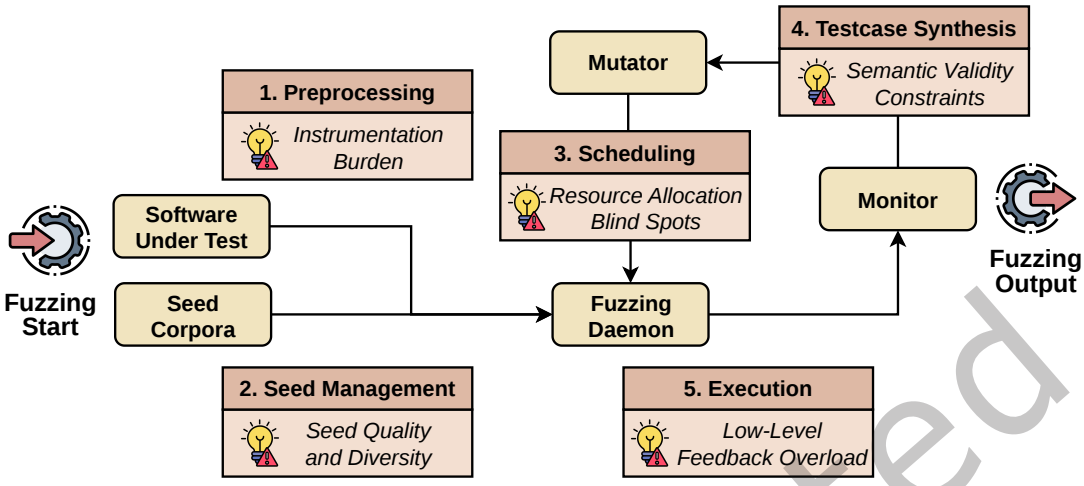

Fig. 1. A typical workflow of five key phases for fuzz testing.

valuable behavior. Seed quality strongly affects coverage growth because well-formed seeds can help the fuzzer pass early parsing checks and reach deeper logic [34, 45]. For example, a valid protocol message can expose state transitions that random bytes rarely trigger.

- **Scheduling.** This phase decides which seed should be selected next and how much mutation energy should be assigned to it. Effective scheduling balances exploration and exploitation under a limited testing budget. Coverage-based power schedules prioritize seeds that are more likely to exercise rare paths [7]. Directed fuzzers further bias seed selection toward specific targets, such as recently patched code and security-critical functions [6]. Scheduling therefore plays a central role in resource allocation.
- **Testcase Synthesis.** This phase generates new test inputs. Mutation-based fuzzers transform existing seeds. Generation-based fuzzers synthesize inputs from grammars, models, and specifications. Grammar-based fuzzers such as NAUTILUS generate syntactically valid inputs for structured languages [2]. Hybrid fuzzers such as Driller combine fuzzing with symbolic execution to satisfy hard path constraints [52]. The quality of test case synthesis directly affects whether the fuzzer can move beyond shallow coverage.
- **Execution.** This phase runs generated inputs against the target program and collects runtime feedback. The feedback may include coverage, crashes, hangs, sanitizer reports, and execution traces. AddressSanitizer exposes memory safety violations during execution [47]. Coverage-guided fuzzers then use execution feedback to update the seed corpus and guide later mutations [18]. Execution also produces the evidence that developers inspect during triage and debugging.

These components form a tightly coupled feedback loop that enables fuzzers to iteratively refine their input corpus, adapt exploration strategies, and progressively drive execution into deeper and less-tested regions of the program. This automated pipeline has made fuzzing one of the most effective techniques for vulnerability discovery.

2.2 Conceptual Foundations of HITL Fuzzing

Human-in-the-Loop fuzzing lies at the intersection of fuzz testing, which includes human-computer interaction, visual analytics, expert-in-the-loop software testing, and human-AI collaboration. Traditional fuzzing research

emphasizes automation. The fuzzer generates inputs, collects feedback, updates the corpus, and continues the search with limited human involvement. In practice, however, human expertise is often needed for harness construction, seed selection, target identification, campaign configuration, and crash triage [5, 42]. These tasks shape the search space and influence whether the fuzzer can overcome semantic roadblocks.

The need for human involvement comes from several fuzzing-specific challenges, specifically:

- **Difficulty in Handling Semantic Constraints.** When inputs must satisfy complex structures such as protocols, trees, state machines, or cross-message consistency rules, coverage-guided or mutation-based heuristics inevitably produce a large number of invalid inputs or fail to progress beyond superficial exploration.
- **Challenges in Traversing Long or Sparse Execution Paths.** Some vulnerabilities lie behind multiple intermediate states or require specific combinations of conditions. Simple mutation often cannot bypass such bottleneck checks in a single step, leaving deep paths unexplored.
- **Lack of Explainability and Strategy Transparency.** Automated fuzzers are rarely able to reveal why exploration stalls on a particular path or where the search is stuck, making it difficult for users to update.
- **Blind Spots in Resource Allocation.** Given finite computational budgets, fuzzers must trade off between broad exploration and deep investigation. Automated heuristics, however, often make suboptimal allocation decisions, wasting resources or missing critical opportunities.
- **Limited Support for Human Feedback.** Current fuzzing workflows provide little opportunity for testers to intervene in the strategy while fuzzing is in progress, constraining the potential benefits of expert guidance.

Beyond these limitations, existing research [5] has emphasized in their methodological reflection that fuzzing research should move toward incorporating HITL components. Whereas most contemporary frameworks treat human experts merely as initiators (e.g., setting up the environment or triaging potential bugs), future directions should enable human auditors to engage during the fuzzing process itself—for instance, identifying roadblocks, adjusting exploration paths, or reconfiguring mutation strategies on the fly.

Reflection of fuzzing workflows as illustrated in subsection 2.1 and existing limitations, inserting well-designed HCI touchpoints into different phases of the fuzzing loop represents a key opportunity to unlock greater potential. By introducing a HITL perspective, fuzzing can be enhanced in several important ways:

- **Bridging Semantic Gaps.** Human experts can leverage their understanding of protocol semantics, business logic, or vulnerability patterns to guide mutation toward more meaningful directions, thereby reducing the inefficiency of blind or invalid input generation.
- **Breaking Exploration Bottlenecks.** When automated exploration stalls at specific branches, checkpoints, or path boundaries, experts can intervene by adjusting strategies, introducing intermediate constraints, or manually crafting transition inputs that help the fuzzer progress further.
- **Enhancing Explainability and Controllability.** Through interactive visualization and control interfaces, analysts can gain insight into the internal state of fuzzing, enabling them to determine when and how to intervene, redirect, or fine-tune the process.
- **Co-governing Resource Allocation and Strategy.** Under resource constraints, human guidance can complement automated heuristics in making high-level scheduling decisions, such as prioritizing certain sub-paths for deeper exploration or deprecating unpromising seeds.

From an HCI perspective, we conceptualize HITL fuzzing as a two-layer interaction loop between automated fuzzing and human expertise, as further elaborated in section 4. These two layers correspond to two fundamental questions in interactive fuzzing: (1) What can humans observe and understand from the fuzzing process? (2) How can humans act on this understanding to influence the fuzzing process?

- **Human Monitoring (Passive Monitoring).** Testers act primarily as observers, gaining visibility into the fuzzing process via dashboards, coverage heatmaps, path visualizations, and similar artifacts. In this way, humans do not actively change fuzzing policies during execution. They analyze runtime state and results to identify anomalies, diagnose bottlenecks, and discover blind spots, which is useful for situational awareness, post-hoc analysis, and surfacing cues that indicate where intervention might be beneficial.
- **Human Steering (Proactive Steering).** Testers take an active role in directing the fuzzer by adjusting seed priorities, tuning mutation weights, injecting constraints, suppressing unproductive paths, or specifying intermediate mutation strategies. Interventions can range from coarse high-level guidance to fine-grained, runtime decisions at particular control points. This semi-automated, hybrid approach allows human expertise to influence exploration policies and search trajectories directly.

The two layers form a feedback loop that extends human involvement from peripheral configuration and final result inspection to the mid-loop stages of fuzzing. Automated fuzzing produces runtime signals, and human monitoring turns these signals into interpretable evidence. Based on this evidence, human steering translates expert judgment into updated fuzzing strategies, which further influence later executions. This loop provides the conceptual foundation for our survey and clarifies why HITL fuzzing is not merely a matter of visualizing results after fuzzing ends. A complete HITL fuzzing system should support both monitoring and steering, while connecting them through actionable feedback. Such a shift enables expert knowledge to be injected into exploration decisions promptly and improves the explainability, responsiveness, and adaptability of fuzzing workflows.

2.3 LLM in the Loop Fuzz Testing

Recent advances in LLMs have introduced a new form of human-AI collaboration for fuzz testing. Traditional fuzzers usually rely on manually designed heuristics, predefined grammars, and program feedback. LLMs provide a different capability. They can synthesize structured inputs, infer constraints from natural language specifications, generate fuzz drivers, and summarize testing feedback [14, 57, 64]. These capabilities overlap with tasks that often require human expertise in conventional fuzzing workflows. As a result, LLMs can assist humans in seed generation, driver construction, testcase synthesis, and strategy refinement.

Existing LLM-based fuzzing studies can be organized according to the fuzzing tasks they support. The first line of work focuses on seed and input generation. TitanFuzz treats LLMs as zero-shot fuzzers for deep-learning libraries by generating and mutating test programs [14]. Fuzz4All uses LLMs to generate and mutate inputs across different programming languages [57]. WhiteFox applies a multi-agent LLM framework to compiler fuzzing, where LLM agents summarize optimization triggers and synthesize test programs [62]. These studies show that LLMs can provide generative power when input structures are difficult to encode manually.

The second line of work uses LLMs as adaptive reasoning components. ChatAFL extracts protocol knowledge from RFCs and generates messages that help protocol fuzzers overcome coverage plateaus [39]. HyLLfuzz uses LLMs to translate path conditions into natural language and generate inputs that satisfy these conditions [39]. G2Fuzz synthesizes input generators for complex non-textual formats when conventional fuzzers stagnate [65]. CovRL-Fuzz further explores how coverage feedback can guide LLM-based mutation strategies during fuzzing [16]. These studies suggest that LLMs can support interpretation and adaptation in the fuzzing loop.

The third line of work focuses on fuzz driver generation and repair. Zhang *et al.* [64] evaluate the effectiveness of LLMs for generating fuzz drivers and improving them through iterative repair. CKGFuzzer [58] combines LLM agents with code knowledge graphs to plan API usage, generate drivers, repair drivers, synthesize seeds, and analyze crashes. These studies indicate that LLMs can reduce expert effort in preprocessing, especially when writing correct and effective drivers is costly.

Although these studies demonstrate the potential of LLMs in fuzzing, they also expose new challenges. LLMs may generate invalid inputs, brittle drivers, hallucinated constraints, and unreliable explanations. Their outputs can also be sensitive to prompts and context limits. Therefore, LLMs should not be viewed as a full replacement for human expertise. Instead, they are better understood as collaborators in the HITL fuzzing loop. They can strengthen human monitoring by summarizing runtime signals and explaining bottlenecks. They can also support human steering by proposing seeds, constraints, targets, and configuration changes. Human oversight remains necessary to validate LLM-generated artifacts, correct unreliable reasoning, and align fuzzing strategies with domain knowledge.

In this survey, we treat LLM-in-the-loop fuzzing as an emerging extension of HITL fuzzing. The key issue is not whether humans or LLMs should control fuzzing alone. Instead, the central question is how automated fuzzers, LLM-based assistants, and human experts can be coordinated into a reliable and interpretable fuzzing workflow.

3 Review Methodology

The scope of our survey focuses on studies that explicitly integrate human involvement into fuzzing workflows. A paper is considered within scope if it satisfies two conditions: (1) It must be directly related to fuzzing. (2) It must contain a concrete form of HITL contribution, such as visualization-assisted monitoring, interactive seed management, target specification, constraint injection, or domain knowledge integration. Papers that only mention fuzzing without human interaction are excluded. Papers where human input is limited to final result inspection are also excluded.

To capture the recent evolution of automation in software testing, we also include LLM-assisted fuzzing studies when LLMs play roles analogous to human experts. Examples include seed generation, fuzz driver construction, constraint reasoning, and guided mutation. This scope allows us to compare traditional HITL fuzzing with emerging human-LLM collaborative fuzzing. In total, our survey covers papers published between 2007 and 2025 across software engineering, security, and human-computer interaction venues.

Our review protocol follows established systematic literature review practices in software engineering [30, 50, 67, 69] and snowballing guidelines [27, 56]. The protocol consists of five main steps: database search, paper screening, snowballing, quality assessment, and data extraction. We use the five fuzzing phases introduced in subsection 2.1 and the visualization-oriented fuzzing taxonomy in prior work [33, 61] to guide our classification.

3.1 Database Search

We first conducted a keyword-based search across four digital libraries and scholarly search engines: Google Scholar, ACM Digital Library, IEEE Xplore, and DBLP. Google Scholar was used to provide broad coverage of published papers and preprints. ACM Digital Library and IEEE Xplore were used to cover major software engineering, security, and HCI venues. DBLP was used to cross-check bibliographic records and identify venue versions.

We focused on papers published over the past twenty years to capture both foundational and recent developments in HITL fuzzing. This time window covers early human-assisted fuzzing efforts, such as Flayer [15], as well as recent studies on interactive fuzzing, visualization-assisted fuzzing, and LLM-assisted fuzzing.

We constructed search strings by combining fuzzing terms with terms related to human monitoring and human steering. To cover LLM-assisted fuzzing, we also added AI-related terms that describe expert-like automation in fuzzing. The term groups are shown in Table 1. The final queries were constructed by combining one fuzzing term with at least one term from the other groups. We adapted the syntax of each query to the search interface of each digital library. Representative search strings are listed in Table 2.

The initial database search returned 1,308 records. After removing duplicates and overlapping records, 548 records remained. All authors participated in screening the titles and abstracts of these records and retained

Table 1. Search term groups used in the database search.

Term Group	Keywords
Fuzzing Terms	fuzzing, fuzz test, fuzz testing, fuzzer
Human Monitoring Terms	visualization, visual, dashboard, interface, monitoring, inspection
Human Steering Terms	interactive, guided, assisted, interaction, intervention, steering, annotation
LLM-assisted Fuzzing Terms	large language model, LLM, agent, prompt, driver generation, seed generation

Table 2. Representative search strings used in the review.

Search Strings
(”fuzzing” OR ”fuzz testing” OR ”fuzzer”) AND (”visualization” OR ”visual” OR ”dashboard” OR ”interface” OR ”monitoring”)
(”fuzzing” OR ”fuzz testing” OR ”fuzzer”) AND (”interactive” OR ”guided” OR ”assisted” OR ”interaction” OR ”intervention” OR ”steering”)
(”fuzzing” OR ”fuzz testing” OR ”fuzzer”) AND (”annotation” OR ”target specification” OR ”seed injection” OR ”constraint injection”)
(”fuzzing” OR ”fuzz testing” OR ”fuzzer”) AND (”large language model” OR ”LLM” OR ”agent” OR ”prompt” OR ”driver generation” OR ”seed generation”)

343 papers for full-text assessment. After full-text screening, 200 papers were retained from the keyword-based search.

3.2 Paper Selection Criteria

We applied predefined inclusion and exclusion criteria during both title and abstract screening and full-text screening. A paper was included if it satisfied all of the following criteria:

- ✔ **I1:** The paper is written in English.
- ✔ **I2:** The paper is directly related to fuzzing.
- ✔ **I3:** The paper includes concrete human involvement in the fuzzing process, such as visualization-assisted monitoring, seed injection, annotation, target specification, constraint injection, and interactive configuration.
- ✔ **I4:** For LLM-assisted fuzzing studies, the paper uses LLMs to perform expert-like fuzzing tasks, such as seed generation, driver construction, constraint reasoning, and guided mutation.
- ✔ **I5:** The paper provides enough technical detail to support classification and synthesis.

A paper was excluded if it met any of the following criteria:

- ✗ **E1:** The paper has fewer than four pages.
- ✗ **E2:** The paper is a book, keynote record, panel summary, technical report, tool-demo-only paper, editorial, or work from a venue not subject to full peer review.
- ✗ **E3:** The paper only mentions fuzzing without discussing a fuzzing workflow, technique, tool, or evaluation.
- ✗ **E4:** The paper only involves humans as final result reviewers after fuzzing ends.
- ✗ **E5:** The paper describes a fully automated fuzzing technique without any human-facing interaction, human-provided knowledge, or an LLM-based expert-like component.

✗ **E6:** The paper is a duplicate or an overlapping version of another included study.

When multiple versions of the same work were available, we retained the most complete peer-reviewed version. Journal versions were preferred when they substantially extended the same work. Full papers were preferred over short papers when both versions described the same system.

3.3 Snowballing Search

To reduce the risk of missing relevant papers, we conducted backward and forward snowballing after the database search [27, 56]. Snowballing was applied to all papers retained after full-text screening.

For backward snowballing, we inspected the reference lists of the selected papers. This step helped us identify earlier and foundational studies that were not captured by the keyword search. For forward snowballing, we searched papers that cited the selected papers using Google Scholar. This step helped us identify later studies that extended or built upon the selected work.

The snowballing process followed the same screening and quality criteria as the database search. Newly identified papers were first screened by title and abstract. Potentially relevant papers were then assessed through full-text reading. Duplicate papers and overlapping versions were removed before final inclusion. The process was iterative. Newly included papers were added to the seed set for the next snowballing round. We stopped when a complete iteration produced no additional eligible papers.

In total, we conducted three snowballing iterations. In the first iteration, we identified 52 additional records, removed 18 duplicates or overlapping records, and included 10 papers after full-text screening. In the second iteration, we identified 9 additional records and removed 6 duplicates or overlapping records. In the final iteration, no additional eligible papers were found, so we stopped the snowballing process. Across all iterations, and after duplicate removal and full-text screening, 13 papers were added to the final corpus through snowballing.

3.4 Quality Assessment

We conducted a quality assessment for all papers that passed full-text screening. Following common practice in systematic literature reviews in software engineering [30, 56], the goal of this step was to ensure that the selected papers provided sufficient methodological and technical information for reliable coding and comparison.

Each paper was assessed using a five-point Likert scale, where 1 indicates that the criterion is not satisfied, 3 indicates that the criterion is partially satisfied, and 5 indicates that the criterion is fully satisfied. The assessment criteria are shown in Table 3. The criteria examine whether a paper clearly describes its fuzzing context, the form of human involvement or LLM-based assistance, the interaction mechanism, the evaluation evidence, and the methodological details needed for cross-study comparison.

Two authors independently assessed each paper. We then measured inter-rater agreement using Cohen's kappa. The resulting kappa value is 0.78, indicating substantial agreement between the two assessors. Disagreements were resolved through discussion and an arbitrator. Papers with an average quality score below 2.0 were excluded from the final corpus, resulting in XX excluded low-quality papers.

3.5 Data Extraction and Coding

After the final set of papers was determined, we extracted bibliographic, methodological, and technical information from each study. The extracted information covered publication metadata, fuzzing targets, involved fuzzing phases, HITL roles, interaction mechanisms, intervention timing, automation level, evaluation methods, and evaluation metrics. We coded each paper according to the five fuzzing phases introduced in subsection 2.1, namely preprocessing, seed management, scheduling, testcase synthesis, and execution. We also coded each paper according to the two HITL roles: human monitoring and human steering. A paper could receive multiple phase labels or role labels when it contributed to multiple parts of the fuzzing workflow.

Table 3. Quality assessment criteria for selected studies.

Criterion	Description	Scale
QA1	The study clearly describes its fuzzing target, workflow phase, and technical context.	1-5
QA2	The study clearly describes the form of human involvement or LLM-based expert-like assistance.	1-5
QA3	The study provides enough information to classify the interaction mechanism and intervention timing.	1-5
QA4	The study reports evaluation evidence, such as coverage, bugs, usability, user effort, or case studies.	1-5
QA5	The study provides sufficient methodological detail to support comparison with other studies.	1-5

All authors participated in the manual coding process. In the first round, the authors independently read and categorized the candidate papers using the coding dimensions above. In the second round, we compared the labels and discussed inconsistent cases. We revisited 12 papers whose labels differed during the first round of coding. After the discussion, the authors finalized the classification results. These coded results were used to construct the taxonomy and synthesis in section 4.

3.6 Final Review Data Flow

After database search, screening, snowballing, quality assessment, and coding, the final corpus contains 44 papers. These include 35 technical research papers and 9 survey papers. The selected papers span software engineering, security, and human-computer interaction venues. The publication years range from 2007 to 2025.

Figure 2 reports the distribution of the final corpus. The left panel shows the temporal trend of the reviewed papers. The right panel shows how HITL fuzzing studies are distributed across the five fuzzing phases and the two human roles. The total counts may exceed the number of papers because one paper may contribute to multiple fuzzing phases and may support both human monitoring and human steering.

3.7 Research Questions

Based on the final corpus and coding schema, we conduct the literature synthesis to answer the following review questions:

- **RQ1:** How has human involvement been integrated into different phases of the fuzzing workflow?
- **RQ2:** What forms of human monitoring and human steering have been supported by existing HITL fuzzing studies?
- **RQ3:** What research gaps can be derived from the existing literature, and how do they motivate future directions for HITL fuzzing and human-LLM collaborative fuzzing?

4 Existing Research on HITL Fuzzing

To answer **RQ1** and **RQ2**, we organize existing HITL fuzzing studies using a *phase-role-tool-affordance* conceptual model. The first dimension is the fuzzing workflow phase in which human involvement is introduced. Following subsection 2.1, we consider five phases: preprocessing, seed management, scheduling, testcase synthesis, and execution. The second dimension is the human role. We distinguish between **Human Monitoring (HM)**, where

Table 4. Summary of HITL for fuzzing studies and the phases of the fuzzing workflow where humans are involved. The five phases considered are preprocessing, seed management, scheduling, testcase synthesis, and execution. Human participation is categorized into two modes: *Human Monitoring (HM)* and *Human Steering (HS)*. Citation counts are from Google Scholar as of June 2026.

	Year	Cit.	(1) Preproc.	(2) Seed Mng.	(3) Scheduling	(4) Testcase Synth.	(5) Execution
Drewry et al. [15]	2007	113	HS				HM
Machiry et al. [38]	2013	964		HS	HS		HM
Vainio et al. [53]	2014	4					HM
Opmanis et al. [43]	2016	14		HS			HM
Shoshitaishvili et al. [51]	2017	91		HS			HM
Cottam et al. [13]	2017	3		HS			HM
Zhou et al. [68]	2019	27		HS			HM
Aschermann et al. [2]	2019	411	HS	HS			HM
Aschermann et al. [3]	2020	220	HS		HS		HM
Hussain et al. [26]	2021	16				HM	
Fioraldi et al. [19]	2021	17			HS	HM	HM
Grishin et al. [23]	2022	3			HS		HM
Lu et al. [37]	2023	3		HM HS		HS	HM
Yan et al. [60]	2023	8	HS	HM HS			HM
Bundt et al. [8]	2023	8	HS	HS			HM
Coppa et al. [12]	2023	3	HM HS	HM HS			HM
Koffi et al. [31]	2023	1	HS	HS			HM
Fang et al. [17]	2024	16		HS			HM
Xu et al. [59]	2024	1		HM	HM HS		
Gridin et al. [22]	2024	2	HS			HS	HM
Chambers et al. [10]	2024	13	HS	HS	HS		HM
Li et al. [36]	2025	1	HS		HS		HM
Li et al. [35]	2025	4			HS	HM HS	HM
Wen et al. [54]	2025	3	HS	HM			HM
Koffi et al. [32]	2025	3	HS	HS			HM

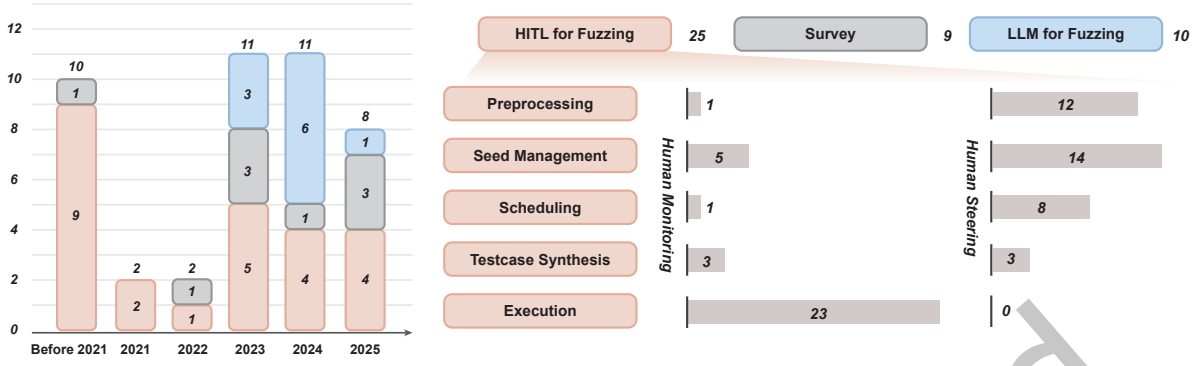

Fig. 2. Distribution of collected articles. The left panel shows the temporal trend of published works, grouped by year. The right panel details how articles on HITL for fuzzing are distributed across different fuzzing stages, further distinguishing between human monitoring and human steering roles. The totals may exceed the overall number of reviewed papers because some works contribute to multiple phases of human monitoring or human steering.

users observe and interpret fuzzing states, and **Human Steering (HS)**, where users actively influence fuzzing strategies.

This conceptual model provides the basis for our phase-level and tool-level synthesis. The five phases represent different intervention points in the fuzzing loop. Preprocessing and seed management occur mainly before or at the beginning of a campaign, scheduling and testcase synthesis shape the search process during exploration, and execution produces the runtime feedback that reveals whether the campaign is progressing. Human monitoring and human steering play different roles in these phases. Monitoring improves visibility and diagnosis, while steering translates human judgment into concrete fuzzing actions. Tool affordances further explain how these roles are realized in practice. Visualization and dashboards mainly support monitoring by exposing fuzzing states and bottlenecks. Annotations, seed construction, target specification, constraint injection, and parameter adjustment mainly support steering by turning human judgment into changes to the fuzzing process. LLM-assisted recommendations may support both roles, depending on whether they summarize evidence or propose concrete actions.

In particular, we examine existing HITL fuzzing papers across the five phases of the fuzzing workflow. Table 4 provides a paper-level mapping of where human monitoring and human steering are introduced, while further aggregating this mapping by year and phase to reveal temporal and phase-level trends.

4.1 Quantitative Synthesis of Existing Research

Two observations can be made from the aggregated distribution: (1) Human involvement in fuzzing is unevenly distributed across phases. Execution is the most frequently covered phase, which reflects the dominance of monitoring-oriented designs that visualize runtime feedback, crashes, traces, and coverage changes. Seed management and preprocessing also receive substantial attention because they provide practical entry points for human steering, such as seed injection, target selection, annotation, and configuration. (2) The middle of the fuzzing pipeline remains less explored. Scheduling and testcase synthesis receive fewer studies, even though they directly affect fuzzing input generation. This suggests that real-time strategy adjustment and generation-level intervention remain important open problems.

Table 6 also reports citation counts as a descriptive indicator of each study's visibility. At the phase level, highly cited works are not limited to execution-oriented monitoring, but also appear in preprocessing, seed

Table 5. Taxonomic dimensions for analyzing HITL fuzzing studies.

Dimension	Categories
Fuzzing Phase	Preprocessing, Seed management, Scheduling, Testcase Synthesis, Execution
Human Role	Human Monitoring, Human steering, Hybrid Intervention
Interaction Mechanism	Visualization, Dashboard, Annotation, Seed Construction, Target Specification, Constraint Injection, Parameter Adjustment, LLM-Assisted Recommendation
Intervention Timing	Pre-Execution Configuration, Runtime Intervention, Post-Execution Interpretation
Evaluation Evidence	Coverage Improvement, Bug Discovery, Execution Efficiency, Input Validity, Usability, Human Effort

Table 6. Year-wise distribution of HITL fuzzing studies across fuzzing phases. A paper is counted once for a phase if it introduces either human monitoring or human steering in that phase. Papers covering multiple phases are counted in multiple columns.

Year	#Papers	Preproc.	Seed Mng.	Scheduling	Testcase Synth.	Execution
2007-2010	1	1	0	0	0	1
2011-2015	2	0	1	1	0	2
2016-2020	6	2	5	1	0	6
2021-2025	16	9	10	6	5	14
Total	**25**	**12**	**16**	**8**	**5**	**23**

management, and scheduling, suggesting that influential HITL fuzzing studies often affect multiple parts of the fuzzing workflow. However, citation counts are influenced by publication age and community adoption, so we use them only as complementary bibliometric context rather than as a measure of study quality.

Based on the reviewed papers, we first summarize the high-level taxonomy of HITL fuzzing in Table 5. The taxonomy captures the main analytical dimensions observed across the corpus, including fuzzing phase, human role, interaction mechanism, intervention timing, automation level, and evaluation evidence. This table provides an overview of the design space. In the following subsections, we further refine this taxonomy through phase-level discussions and analyze how these dimensions appear in each fuzzing phase.

This taxonomy shows that current HITL fuzzing research occupies only part of the design space. Many systems support human monitoring through dashboards and visual analytics, but fewer systems connect monitoring results to direct steering actions. Existing steering is often pre-execution and discrete, such as adding seeds, annotating code, specifying targets, or changing parameters. Runtime and mixed-initiative steering remain comparatively rare.

4.2 Phase I: Preprocessing

Preprocessing prepares the program under test and its execution environment before the main fuzzing loop starts. It covers instrumentation, harness construction, driver modification, target selection, configuration, and domain modeling. Conceptually, this phase determines what the fuzzer can observe and which program behaviors it can reach. A weak preprocessing setup may hide useful feedback signals, produce invalid executions, or block access to meaningful program states. Therefore, preprocessing is a high-leverage phase for HITL fuzzing, where early expert input can reshape the later search space.

Existing preprocessing work is mainly steering-oriented. This is consistent with the practical burden reported by Qiao *et al.* [44], where effective fuzzing often requires practitioners to craft invariants, initialize states, and write harnesses before testing can proceed. Monitoring support is less common, but FuzzPlanner [12] shows how visualizing binaries, CVE mappings, and process interactions can help analysts inspect firmware targets and design fuzzing campaigns before execution.

Most steering work changes the interface between the target and the fuzzer. Flayer [15] allows testers to force branches or skip calls to bypass guard conditions. IJON [3] introduces annotations that expose hidden states and optimization objectives. Bundt *et al.* [8] use compartment analysis to suggest harness or environment modifications for reaching otherwise hidden regions. InFuzz [60] lets testers modify drivers and annotate code to expose hidden paths. These studies show that preprocessing steering often works by making hidden states observable or reachable.

Another group of studies uses human input to configure targets and campaigns. FuzzPlanner [12] allows users to select binaries, filter targets by vulnerability score, and configure firmware fuzzing experiments. VDFuzz [36] supports CFG inspection and target annotation before recompilation. Koffi *et al.* [32] rely on expert-specified target functions or code regions for directed fuzzing. HIFuzz [10] asks testers to define the UAV task space, including roles, devices, mission sequences, and environmental parameters. PrettiSmart [54] lets users configure smart-contract simulation boundaries, such as user numbers, owner addresses, and balance constraints.

A further form of preprocessing steering is domain-model or constraint injection. NAUTILUS [2] shows that grammar-based fuzzing benefits from input models enriched with identifiers or symbol lists. Gridin *et al.* [22] use expert-provided restrictions in cryptographic test-vector generation to bypass domain-specific barriers. These works indicate that preprocessing is especially important when input validity depends on semantic constraints rather than syntax alone.

Takeaway: Preprocessing is mainly a steering-oriented phase, where human annotations, harness changes, target choices, and domain constraints shape the search space before fuzzing starts. However, these interventions are mostly offline and static, leaving limited support for runtime adaptation.

4.3 Phase II: Seed Management

Seed management curates, prioritizes, and updates the input corpus that drives fuzzing exploration. It includes selecting initial seeds, removing redundant inputs, identifying valuable seed–path relations, and injecting new seeds when exploration stalls. Conceptually, this phase determines where fuzzing starts and how input diversity is maintained. It is a natural entry point for HITL fuzzing because seeds are concrete artifacts that humans can inspect, edit, and extend with domain knowledge.

Existing work supports seed monitoring by making the corpus and its effects more visible. FuzzInspector [37] highlights seed–path differences and low-coverage functions, helping users identify under-tested inputs. InFuzz [60] shows which seeds approach bottleneck locations, allowing testers to inspect the relation between inputs and hard-to-reach branches. FuzzPlanner [12] visualizes communication channels and seed sources in firmware fuzzing, supporting analysts in selecting promising inputs. ViScheduler [59] uses embedding-based visualizations to reveal seed clusters, anomalies, and coverage diversity. Qiao *et al.* [44] further show that seed and target-sequence preparation can strongly affect fuzzing effectiveness, while PrettiSmart [54] visualizes generated input sequences to expose bias, diversity, and coverage dynamics.

On the steering side, humans directly reshape the seed corpus. FuzzInspector [37] lets users define seed constraints to steer execution toward selected branches. InFuzz [60] and Koffi *et al.* [31] allow testers to add seeds that bypass blocking branches revealed by visualization. Cottam *et al.* [13] reuse real user input streams as seeds, preserving domain relevance for later mutation. FuzzPlanner [12] allows users to confirm and prioritize

seed inputs from emulation traces or proof-of-concept artifacts. VisFuzz [68] supports manual construction and injection of targeted seeds to overcome semantic bottlenecks. Koffi *et al.* [32] further use expert feedback to refine initial seed sets and target lists.

Other studies demonstrate seed steering in broader or domain-specific settings. HaCRS [51] allows human assistants to propose new inputs from program logs and execution traces. Bundt *et al.* [8] uses ranked compartments to guide targeted seed selection for unexplored regions. HIFuzz [10] lets experts select representative or boundary tests from one testing layer and promote them as seeds for the next layer. Dynodroid [38] allows users to inject meaningful events, such as credentials or GUI actions, into automated input generation. NAUTILUS [2] also benefits from user inspection and selection of interesting minimized seeds for further generation.

Takeaway: Seed management is the most mature phase for direct human steering because seeds are inspectable and easy to modify. Existing work supports seed visualization, injection, and refinement, but rarely measures the human effort or expertise required for effective seed intervention.

4.4 Phase III: Scheduling

Scheduling decides which seed or task should be selected next and how much fuzzing energy should be allocated to it. It controls the balance between exploration and exploitation under a limited testing budget. Conceptually, scheduling is the resource-allocation layer of fuzzing. It translates runtime feedback into search priorities. For HITL fuzzing, this phase is challenging because scheduling decisions are frequent, dynamic, and often too fine-grained for direct manual control. As a result, human involvement in scheduling usually appears as high-level strategy adjustment rather than low-level queue manipulation.

Existing work first supports scheduling monitoring by making scheduling behavior more visible. ViScheduler [59] visualizes seed clusters, anomalies, runtime features, and their evolution, helping analysts understand how scheduling decisions affect coverage progress. FuzzSplore [19] exposes coverage, mutation, and campaign-level metrics across fuzzers, allowing users to identify underperforming configurations and resource imbalance. These systems show that scheduling monitoring is useful when low-level queue behavior is aggregated into interpretable patterns.

On the steering side, existing systems let users adjust priorities, targets, or resource allocation. FuzzSplore [19] allows analysts to reallocate CPU time, drop underperforming fuzzers, and tune parameter weights during a campaign. DDGF [17] lets users flag interesting paths so that later scheduling can bias toward seeds exercising those paths. ViScheduler [59] allows experts to amplify energy for selected clusters or outliers. VDFuzz [36] lets testers add unexplored basic blocks or target points through visualization, which are then prioritized by the scheduler. GDFuzz [35] combines explanation with a dual-queue strategy, allowing users to adjust queue behavior based on feature importance.

Other studies expose scheduling control at different abstraction levels. IJON [3] allows users to shift search objectives through annotations and queue-related parameters. HIFuzz [10] uses human decisions as a gatekeeper between testing layers, where experts decide which test batches should move toward higher-fidelity execution. Grishin *et al.* [23] allow analysts to pause fuzzing, reorder the AFL queue, or switch target functions at runtime. Dynodroid [38] similarly lets users pause automatic generation, inject custom events, and affect which inputs are prioritized next.

Takeaway: Scheduling is a steering-oriented but underexplored phase. Existing work supports coarse human control over priorities, targets, and resources, but fine-grained scheduling remains difficult because decisions are frequent and hard to interpret in real time.

4.5 Phase IV: Testcase Synthesis

Testcase synthesis generates new inputs from existing seeds, grammars, models, or domain constraints. It is the phase where fuzzing turns current knowledge into new executions. Traditional mutation-based fuzzers rely on predefined operators such as byte flips, arithmetic mutations, and splicing [18]. These operators are efficient, but they are often opaque to users and may generate many invalid or shallow inputs when the target requires semantic structure. From a HITL perspective, testcase synthesis is difficult because generation decisions are frequent and low-level. Human involvement therefore needs to focus on making synthesis behavior interpretable and converting high-level intent into actionable generation guidance.

Existing monitoring work mainly exposes how inputs are generated and mutated. FMViz [26] visualizes AFL-generated inputs at the byte level, helping users inspect mutation patterns and understand how test cases evolve. FuzzSplore [19] presents testcase clusters and mutation trajectories, allowing analysts to observe whether the fuzzer explores diverse input regions or converges too narrowly. GDFuzz [35] further uses explainable feature maps to show which input bytes are important for reaching target paths. These studies make testcase synthesis more transparent by turning low-level mutation behavior into interpretable visual evidence.

Steering work in this phase is more limited, but it shows how humans can shape generation when semantic barriers are exposed. FuzzInspector [37] allows testers to define constraints derived from seed and CPU-state analysis, guiding new test cases toward unexplored states. GDFuzz [35] extends its monitoring interface into steering by allowing users to refine sample masks and selectively protect or expose bytes during mutation. Gridin *et al.* [22] demonstrate a domain-specific form of steering in cryptographic test-vector generation, where experts inject constraints so that generated inputs can bypass cryptographic barriers and exercise deeper logic.

Takeaway: Testcase synthesis remains underexplored in HITL fuzzing. Existing work improves the visibility of mutation behavior and supports limited constraint-based steering, but few systems translate high-level human intent into adaptive generation rules.

4.6 Phase V: Execution

Execution runs generated inputs on the program under test and produces runtime evidence, such as coverage changes, crashes, hangs, traces, path profiles, and domain-specific state updates. Conceptually, this phase exposes the progress and limitations of a fuzzing campaign. Since execution often produces large volumes of low-level signals, human involvement in this phase mainly focuses on making runtime behavior visible and interpretable.

Early work presents execution progress through dashboards and runtime summaries. Vainio [53] combines fuzzer status with host-level metrics to help users observe crashes, progress, and distributed campaign health. Opmanis *et al.* [43] visualizes large-scale testing results with matrix, timeline, and chart-based views, supporting regression inspection and root-cause analysis. These works show how execution logs can be transformed into operational views for human analysis. Further, VisFuzz [68] visualizes fuzzing progress and stagnation points. InFuzz [60] reports bottleneck branches and contextual data flow to explain why certain paths remain hard to reach. FuzzSplore [19] summarizes execution, coverage, and mutation dynamics across fuzzing campaigns. FuzzInspector [37] links execution behavior with seed-path differences and low-coverage functions, helping users locate under-tested regions.

Existing approaches also try to connect execution evidence with follow-up human intervention. Flayer [15] exposes taint flow and conditional jumps, allowing analysts to understand blocking conditions. HaCRS [51] connects execution traces with a human-assistance channel, through which users can propose new inputs. Grishin *et al.* [23] allow users to pause fuzzing and adjust targets or queue order based on runtime feedback. These works show that execution evidence can guide human action, although the actual intervention is usually realized through later changes to seeds, targets, or scheduling choices.

Domain-specific systems enrich execution-level interpretation with task-specific semantics. DDGF [17] presents path profiles for directed exploration. VDFuzz [36] uses heatmaps and function-hit information to expose unexplored regions. GDFuzz [35] provides explainable feedback about input features that contribute to reaching target paths. PrettiSmart [54] visualizes smart-contract behaviors such as function calls, state updates, and fund transfers. HIFuzz [10] interprets execution outcomes in UAV testing with respect to tasks, environments, and mission constraints.

Finding: Execution is the main phase where humans interpret fuzzing behavior through runtime evidence. Existing work provides rich views of progress, paths, crashes, and domain states, but direct runtime intervention remains limited.

4.7 Comparison with Existing Surveys

Existing surveys have explored the intersection of human expertise and fuzzing, each contributing valuable perspectives to the discourse. Specifically, Kadron *et al.* [29] synthesizes expert-in-the-loop levers across fuzzing and symbolic execution, arguing that small, well-placed expert inputs can significantly improve path depth and bug-finding efficiency. Zhang *et al.* [66] provides a categorical overview of HITL fuzzing, identifying key areas such as knowledge injection, interactive seed management, and visualization-assisted monitoring. Complementing these, empirical studies conducted by Nourry *et al.* [42] and Qiao *et al.* [44] uncover the practical usability barriers and manual burdens faced by developers, emphasizing needs in setup, configuration, and interoperability.

From a visualization standpoint, Kummita *et al.* [33] offer structured task taxonomies and advocate for frameworks that expose internal fuzzing metrics to support expert monitoring. Reflective research conducted by Böhme *et al.* [5] frames broader HITL questions concerning communication, usability, and dynamic human direction.

While these works collectively underscore the importance of human involvement and begin to map the design space, they remain fragmented in their treatment of the non-comprehensive fuzzing lifecycle (e.g., limited to seed management) or narrowed HITL approaches (e.g, only focus on visualization instead of interactive human steering). Furthermore, in the new era of LLM-assisted testing age, they do not consider the AI-collaborated paradigm of HITL for fuzzing.

5 The Road Ahead for HITL Fuzzing

Our review identifies three recurring gaps in current HITL fuzzing research: (1) Existing systems provide rich runtime visibility, especially during execution, but offer limited support for acting on such evidence during an ongoing campaign. (2) Scheduling and testcase synthesis remain comparatively underexplored, partly because their internal decisions are frequent, low-level, and difficult for humans to interpret. (3) Human involvement is often offline and costly, while existing evaluations rarely quantify the effort required for effective human intervention. These gaps motivate a staged roadmap for future HITL fuzzing research.

We organize the roadmap into three stages. Stage I focuses on making fuzzing behavior more observable and explainable. Stage II turns such evidence into runtime and strategic human intervention. Stage III extends the loop with LLM-based assistants and agents that reduce low-level human effort while keeping humans responsible for high-level goals and validation.

As illustrated in Figure 3, the three stages form a dependency chain for future HITL fuzzing research. Specifically, Stage I provides the signals and explanations needed for effective intervention. Stage II provides the runtime interfaces and action spaces through which human guidance can affect the fuzzing loop. Stage III reuses the same observability and steering interfaces as the substrate for LLM-assisted and agent-based fuzzing. In this view, the roadmap progresses from **visibility**, to **actionability**, and finally to **scalable human-AI collaboration**.

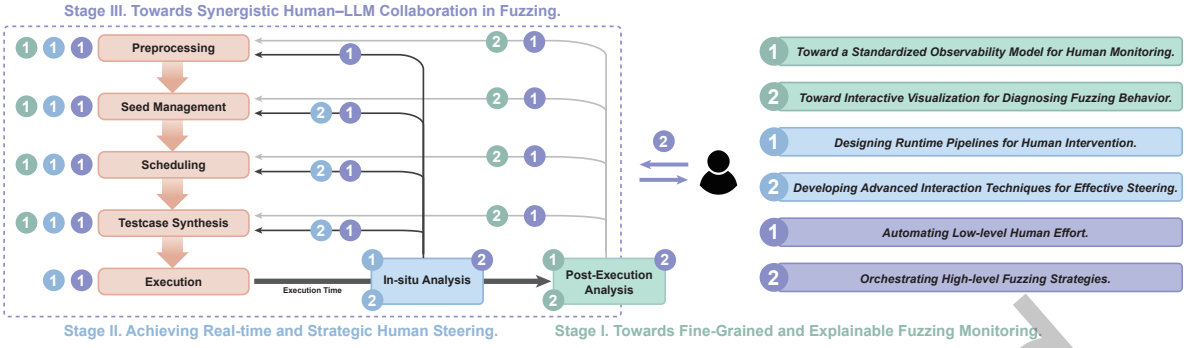

Fig. 3. The interaction design space for human involvement in fuzzing across three stages. Numbered circles mark six future research directions and their corresponding phases in the fuzzing pipeline. Gray arrows denote traditional offline feedback after execution, while black arrows denote in-situ runtime intervention. Stage II highlights the transition from offline post-execution analysis to real-time supervision and steering.

5.1 Stage I: Towards Fine-Grained and Explainable Fuzzing Monitoring

Stage I addresses the evidence gap revealed by our review. Existing systems provide useful execution-level views, but they often only expose final outcomes, such as coverage growth, crashes, and bug reports. These signals indicate whether a campaign has progressed, but they often do not explain why it has stalled. Intermediate decisions in scheduling and testcase synthesis remain especially difficult to inspect. This limits the ability of testers to diagnose inefficiencies in the middle of the fuzzing loop. We identify two concrete research problems.

Research Problem 1: Cross-Phase Fuzzing Observability. A promising direction is to design an observability model that records how information flows across fuzzing phases. Such a model should not only collect execution results, but also capture seed provenance, scheduling decisions, mutation lineage, operator effectiveness, constraint use, and crash-triggering evidence. This would allow testers to answer questions such as: Which seeds are repeatedly selected but produce little new coverage? Which mutation operators generate valid inputs for a specific target? Which preprocessing constraints later affect path reachability? Which execution traces explain why a branch remains unreachable?

A cross-phase observability model would also support systematic comparison across fuzzers and campaigns. By linking seed provenance to scheduling priorities, mutation operators to coverage growth, and execution outcomes to bug discovery, researchers can analyze the causal chain behind fuzzing effectiveness rather than only reporting final coverage. This direction can be evaluated by measuring diagnosis accuracy, time-to-bottleneck identification, cross-fuzzer comparability, and the ability to reproduce campaign-level explanations.

Research Problem 2: Diagnostic visual analytics for fuzzing. A second direction is to design visualization systems that support comparison, temporal analysis, and cross-phase reasoning. Prior work has shown the value of dashboards, coverage views, and behavior summaries for understanding fuzzing progress [19, 33, 68]. However, future interfaces should go beyond aggregate views. They should help testers inspect seed diversity, scheduling bias, mutation history, path stagnation, and domain-specific state changes.

For example, a diagnostic interface could reveal whether a scheduler repeatedly favors low-cost seeds while neglecting rare but high-value inputs. It could show whether a mutation operator produces structurally valid inputs only for certain seed clusters. It could also expose whether a plateau is caused by input validity constraints, poor seed diversity, or insufficient target prioritization. Interactive visual analytics and what-if interfaces [21, 55] can help humans move from passive inspection to evidence-based diagnosis.

Summary of Stage I: Stage I makes fuzzing behavior more visible and explainable. Its concrete outcome should be an evidence layer that supports diagnosis across phases, rather than another dashboard that only reports final results. This evidence layer prepares the foundation for runtime intervention in Stage II.

5.2 Stage II: Achieving Real-time and Strategic Human Steering

From Table 4, it is evident that existing efforts on Human Steering (HS) remain highly uneven across the fuzzing pipeline. Stage II addresses the actionability gap. Our review shows that many human interventions occur before fuzzing starts, such as selecting seeds, specifying targets, configuring harnesses, or injecting constraints. By contrast, runtime intervention remains limited. This creates an "observe-but-not-act" gap: testers may identify a bottleneck during execution, but they often need to stop, reconfigure, and rerun the campaign before their insight can affect the search. Stage II aims to close this gap by making human guidance timely, controlled, and measurable.

Research Problem 3: Runtime Hooks for In-Situ Intervention. Future fuzzers should expose runtime hooks that allow testers to change selected fuzzing decisions while execution continues. These hooks could support actions such as increasing the priority of a seed cluster, changing the energy assigned to a target, enabling a mutation operator for a specific input structure, injecting a semantic constraint, or temporarily focusing exploration on a security-critical module. Such hooks would turn human insight into immediate changes in the fuzzing loop.

The key challenge is to accept human guidance without breaking throughput, reproducibility, or search stability. A promising design is a modular fuzzing loop in which scheduling, mutation, and triage components are parameterized and can be updated at runtime. Human input can then be represented as bounded hints or constraints, rather than arbitrary manual edits. This direction can be evaluated by comparing in-situ intervention with traditional stop-reconfigure-rerun workflows, using metrics such as time-to-coverage, bug discovery latency, intervention overhead, and reproducibility of guided campaigns.

Research Problem 4: Intent-level steering interfaces. Runtime hooks are useful only if humans can express intent efficiently. Current practice often requires low-level parameter tuning or direct modification of the fuzzer, which limits usability. Future systems should provide intent-level abstractions that translate human goals into concrete fuzzing actions. For example, a tester may want to focus on parser error-handling paths, void redundant GUI events, increase exploration of rare state transitions, or prioritize inputs that reach authentication logic. The system should convert such intent into updates to seeds, schedules, targets, or mutation policies.

This problem is grounded in the finding that scheduling and testcase synthesis are hard for humans to control directly. Instead of asking users to manipulate queues, energy values, or mutation operators, future systems should provide higher-level interaction techniques inspired by HCI and visual analytics, such as parameterized workflows, interactive dashboards, what-if analysis, and natural-language interfaces [41, 46, 48]. This direction can be evaluated by measuring steering precision, user effort, cognitive load, and whether translated actions improve exploration compared with manual parameter tuning.

Summary of Stage II: Stage II turns diagnostic evidence into actionable control. Its concrete outcome should be a fuzzing loop where human insight can affect an ongoing campaign through controlled, measurable, and reproducible interventions. It also provides the action space that LLM-based agents can use in Stage III.

5.3 Stage III: Towards Synergistic Human-LLM Collaboration in Fuzzing

Stage III addresses the scalability and effort gap. LLM-assisted fuzzing introduces a new form of human–AI collaboration. Recent studies show that LLMs can generate seeds, synthesize structured inputs, construct fuzz drivers, repair harnesses, reason about constraints, and summarize testing feedback [14, 39, 57, 58, 64]. These

capabilities overlap with tasks that previously required human expertise. However, LLMs also introduce reliability risks, including invalid inputs, brittle drivers, hallucinated constraints, and unstable recommendations. Therefore, the research challenge is not to replace humans, but to design trustworthy collaboration between humans, LLMs, and fuzzers.

Research Problem 5: Verifiable Automation of Low-Level Human Effort. Future work can use LLMs to reduce repetitive human effort in fuzzing. Examples include seed deduplication, seed generation, driver construction, crash clustering, constraint explanation, and parameter recommendation. These tasks are costly for humans but often structured enough for LLM assistance. However, LLM-generated artifacts should not be directly trusted. A generated seed should be checked against execution behavior. A generated driver should be validated by compilation, runtime execution, and coverage. A suggested constraint should be checked against program behavior or domain rules. A crash summary should be verified against traces and sanitizer reports.

This direction therefore calls for verifiable LLM-assisted fuzzing workflows. LLM outputs should be integrated with execution feedback, static analysis, dynamic analysis, and human validation. The evaluation should not only report coverage or bugs, but also measure invalid-output rate, correction cost, expert time saved, and the number of accepted recommendations. Such metrics connect LLM society to the broader HITL evaluation gap identified in our review.

Research Problem 6: Human-supervised LLM orchestration of fuzzing strategies. Beyond local assistance, LLMs may support high-level fuzzing orchestration. For example, an LLM-based assistant could summarize runtime evidence, compare alternative strategies, recommend when to switch between mutation-based and grammar-based generation, or coordinate specialized agents for exploration, analysis, and triage. In such workflows, humans would provide goals, constraints, and sanity checks, while LLM agents perform detailed planning and adaptation.

This direction depends on the earlier stages. The observability model from Stage I provides the evidence that LLM agents need to reason about campaign behavior. The runtime hooks from Stage II provide the action space through which LLM agents can propose or execute strategy updates. A human-supervised orchestration system could therefore include different roles, such as an explorer agent for generating input variants, an analyst agent for interpreting crashes and traces, and a strategist agent for recommending scheduling or target changes. The main challenge is to maintain transparency and accountability. Humans should be able to inspect why an agent recommends a strategy, what evidence supports it, and what risks it may introduce. This direction can be evaluated with cost-effectiveness metrics such as bugs per expert-hour, coverage gain per accepted recommendation, and failure rate of autonomous suggestions.

Summary of Stage III: Stage III extends HITL fuzzing toward human–LLM collaboration. LLMs can reduce low-level effort and assist strategy planning, but humans remain necessary for supervision, validation, and domain judgment. The key research challenge is to make such collaboration reliable, explainable, and cost-effective.

5.4 Concluding Remarks on the Road Ahead.

Together, the three stages outlined above sketch a progressive roadmap for advancing human-in-the-loop fuzzing. Stage I emphasizes richer and more explainable monitoring, calling for unified observability models and interactive visual analytics that move beyond coarse, outcome-oriented metrics. Stage II extends these advances into runtime, integrating human expertise directly into the execution loop through pipelines for in-situ intervention and advanced interaction techniques that make steering both more usable and more effective. Stage III looks further ahead, envisioning synergy between humans and LLMs, where intelligent models automate low-level tasks, orchestrate high-level strategies, and emulate human expertise at scale, enabling fuzzing workflows that are adaptive, domain-aware, and intelligence-driven.

This staged perspective highlights a gradual evolution: from post-execution analysis, to interactive runtime steering, and finally to semi-autonomous collaboration with intelligent systems. By systematically strengthening monitoring, steering, and orchestration, future research can bridge the gap between automated fuzzing and human expertise, paving the way for fuzz testing that is not only more efficient and adaptive but also more explainable, trustworthy, and aligned with real-world software testing needs.

6 Conclusion

This paper presents a systematic review and research roadmap for human-in-the-loop fuzz testing. We characterize human involvement across five fuzzing phases and distinguish between human monitoring and human steering. Our findings show that existing work provides substantial support for interpreting execution behavior, but offers limited runtime intervention, while scheduling and testcase synthesis remain difficult to inspect and control. We therefore propose a progressive roadmap from fine-grained observability to real-time human steering, and ultimately to trustworthy human-LLM collaboration. By making expert knowledge more visible, actionable, and measurable, future HITL fuzzing systems can better combine automated exploration with human judgment to support more adaptive, explainable, and reliable vulnerability discovery.

Acknowledgment

This research is partially supported by the Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN), the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (NTU Tier 1, RG104/25), and the Tianjin University-Longyan University Independent Research Fund (Grant No. 2026XSU-0035). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Cyber Security Agency of Singapore, or the Ministry of Education, Singapore.

References

[1] AFLplusplus. 2025. AFLplusplus. https://github.com/AFLplusplus/AFLplusplus/.
[2] Cornelius Aschermann, Tommaso Frassetto, Thorsten Holz, Patrick Jauernig, Ahmad-Reza Sadeghi, and Daniel Teuchert. 2019. NAUTILUS: Fishing for deep bugs with grammars.. In *NDSS*, Vol. 19. 337.
[3] Cornelius Aschermann, Sergej Schumilo, Ali Abbasi, and Thorsten Holz. 2020. Ijon: Exploring deep state spaces via fuzzing. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1597–1612.
[4] Sadegh Bamohabbat Chafjiri, Phil Legg, Michail-Antisthenis Tsompanas, and Jun Hong. 2023. Honggfuzz+: Fuzzing by Adaptation of Cryptographic Mutation. *Phil and Tsompanas, Michail-Antisthenis and Hong, Jun, Honggfuzz+: Fuzzing by Adaptation of Cryptographic Mutation* (2023).
[5] Marcel Böhme, Cristian Cadar, and Abhik Roychoudhury. 2020. Fuzzing: Challenges and reflections. *IEEE Software* 38, 3 (2020), 79–86.
[6] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. 2017. Directed greybox fuzzing. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2329–2344.
[7] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. 2016. Coverage-based greybox fuzzing as markov chain. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1032–1043.
[8] Joshua Bundt, Andrew Fasano, Brendan Dolan-Gavitt, William Robertson, and Tim Leek. 2023. Homo in Machina: Improving Fuzz Testing Coverage via Compartment Analysis. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 117–128.
[9] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. 2008. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs.. In *OSDI*, Vol. 8. 209–224.
[10] Theodore Chambers, Michael Vierhauser, Ankit Agrawal, Michael Murphy, Jason Matthew Brauer, Salil Purandare, Myra B Cohen, and Jane Cleland-Huang. 2024. Hifuzz: Human interaction fuzzing for small unmanned aerial vehicles. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–14.
[11] Peng Chen and Hao Chen. 2018. Angora: Efficient fuzzing by principled search. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 711–725.

[12] Emilio Coppa, Alessio Izzillo, Riccardo Lazzeretti, and Simone Lenti. 2023. FuzzPlanner: Visually Assisting the Design of Firmware Fuzzing Campaigns. In *2023 IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE, 1–11.
[13] Joseph A Cottam, Leslie Blaha, Dimitri Zarzhitsky, Mathew Thomas, and Elliott Skomski. 2017. Crossing the streams: Fuzz testing with user input. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 4362–4371.
[14] Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. 2023. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models. In *Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis*. 423–435.
[15] Will Drewry and Tavis Ormandy. 2007. Flayer: Exposing Application Internals. *WOOT* 7 (2007), 1–9.
[16] Jueon Eom, Seyeon Jeong, and Taekyoung Kwon. 2024. Fuzzing javascript interpreters with coverage-guided reinforcement learning for llm-based mutation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1656–1668.
[17] Haoran Fang, Kaikai Zhang, Donghui Yu, and Yuanyuan Zhang. 2024. DDGF: Dynamic Directed Greybox Fuzzing with Path Profiling. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 832–843.
[18] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. 2020. {AFL++}: Combining incremental steps of fuzzing research. In *14th USENIX workshop on offensive technologies (WOOT 20)*.
[19] Andrea Fioraldi and Luigi Paolo Pileggi. 2021. Fuzzsplore: Visualizing feedback-driven fuzzing techniques. *arXiv preprint arXiv:2102.02527* (2021).
[20] Vijay Ganesh, Tim Leek, and Martin Rinard. 2009. Taint-based directed whitebox fuzzing. In *2009 IEEE 31st international conference on software engineering*. IEEE, 474–484.
[21] Michael Gleicher. 2017. Considerations for visualizing comparison. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 413–423.
[22] Iaroslav Gridin and Antonis Michalas. 2024. Point Intervention: Improving ACVP Test Vector Generation Through Human Assisted Fuzzing. In *International Conference on Information and Communications Security*. Springer, 43–62.
[23] Maxim Grishin et al. 2022. Human-Controlled Fuzzing With AFL. (2022).
[24] Linghan Huang, Peizhou Zhao, Huaming Chen, and Lei Ma. 2024. Large language models based fuzzing techniques: A survey. *arXiv e-prints* (2024), arXiv–2402.
[25] Linghan Huang, Peizhou Zhao, Lei Ma, and Huaming Chen. 2025. On the Challenges of Fuzzing Techniques via Large Language Models. In *2025 IEEE International Conference on Software Services Engineering (SSE)*. IEEE, 162–171.
[26] Aftab Hussain and Mohammad Amin Alipour. 2021. Fmviz: Visualizing tests generated by AFL at the byte-level. *arXiv preprint arXiv:2112.13207* (2021).
[27] Samireh Jalali and Claes Wohlin. 2012. Systematic literature studies: database searches vs. backward snowballing. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*. 29–38.
[28] Yu Jiang, Jie Liang, Fuchen Ma, Yuanliang Chen, Chijin Zhou, Yuheng Shen, Zhiyong Wu, Jingzhou Fu, Mingzhe Wang, Shanshan Li, et al. 2024. When fuzzing meets llms: Challenges and opportunities. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 492–496.
[29] Ismet Burak Kadron, Yannic Noller, Rohan Padhye, Tevfik Bultan, Corina S Păsăreanu, and Koushik Sen. 2023. Fuzzing, Symbolic Execution, and Expert Guidance for Better Testing. *IEEE Software* 41, 1 (2023), 98–104.
[30] Staffs Keele et al. 2007. *Guidelines for performing systematic literature reviews in software engineering*. Technical Report. Technical report, ver. 2.3 ebse technical report. ebse.
[31] Koffi Anderson Koffi. 2023. *Efficient Seed Generation for Expert-Based Directed Fuzzing*. Master's thesis. University of Idaho.
[32] Koffi Anderson Koffi, Vyron Kampourakis, Constantinos Kolias, Jia Song, and Robert C Ivans. 2025. Speeding-up fuzzing through directional seeds. *International journal of information security* 24, 2 (2025), 77.
[33] Sriteja Kummita, Miao Miao, Eric Bodden, and Shiyi Wei. 2024. Visualization Task Taxonomy to Understand the Fuzzing Internals (Registered Report). In *Proceedings of the 3rd ACM International Fuzzing Workshop*. 13–22.
[34] Caroline Lemieux and Koushik Sen. 2018. Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*. 475–485.
[35] Junru Li, Zhongxu Yin, Guoxiao Zong, and Haiya Sang. 2025. GDFuzz: An efficient directed fuzzing method based on XAI. *Journal of Systems and Software* (2025), 112568.
[36] Xudong Li, Pengfei Wang, and Baosheng Wang. 2025. VDFUZZ: Visual Directed Fuzzer for Understanding and Intervening Fuzzing. In *2025 IEEE 6th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*. IEEE, 1078–1090.
[37] Han-Lin Lu, Ren-Jie Zhuang, and Shih-Kun Huang. 2023. Fuzz Testing Process Visualization. *Journal of Information Science & Engineering* 39, 5 (2023).
[38] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. 2013. Dynodroid: An input generation system for android apps. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. 224–234.
[39] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large language model guided protocol fuzzing. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*, Vol. 2024.

[40] Barton P Miller, Lars Fredriksen, and Bryan So. 1990. An empirical study of the reliability of UNIX utilities. *Commun. ACM* 33, 12 (1990), 32–44.
[41] Xin Mou, Hasan M Jamil, and Xiaogang Ma. 2017. Visflow: A visual database integration and workflow querying system. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 1421–1422.
[42] Olivier Nourry, Yutaro Kashiwa, Bin Lin, Gabriele Bavota, Michele Lanza, and Yasutaka Kamei. 2023. The human side of fuzzing: Challenges faced by developers during fuzzing activities. *ACM Transactions on Software Engineering and Methodology* 33, 1 (2023), 1–26.
[43] Rudolfs Opmanis, Paulis Kikusts, and Martins Opmanis. 2016. Visualization of large-scale application testing results. *Baltic Journal of Modern Computing* 4, 1 (2016), 34.
[44] Guanming Qiao and Partha Protim Paul. 2025. Human Side of Smart Contract Fuzzing: An Empirical Study. *arXiv preprint arXiv:2506.07389* (2025).
[45] Alexandre Rebert, Sang Kil Cha, Thanassis Avgerinos, Jonathan Foote, David Warren, Gustavo Grieco, and David Brumley. 2014. Optimizing seed selection for fuzzing. In *23rd USENIX Security Symposium (USENIX Security 14)*. 861–875.
[46] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.
[47] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitriy Vyukov. 2012. {AddressSanitizer}: A fast address sanity checker. In *2012 USENIX annual technical conference (USENIX ATC 12)*. 309–318.
[48] Vidya Setlur, Sarah E Battersby, Melanie Tory, Rich Gossweiler, and Angel X Chang. 2016. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th annual symposium on user interface software and technology*. 365–377.
[49] Cheng Shi, Jiongchi Yu, Ziming Zhao, Jiongyi Chen, and Fan Zhang. 2025. CGIFuzz: Enabling Gray-Box Fuzzing for Web CGI of IoT Devices. *IEEE Transactions on Information Forensics and Security* (2025).
[50] Jieke Shi, Zhou Yang, and David Lo. 2024. Efficient and green large language models for software engineering: Vision and the road ahead. *ACM Transactions on Software Engineering and Methodology* (2024).
[51] Yan Shoshitaishvili, Michael Weissbacher, Lukas Dresel, Christopher Salls, Ruoyu Wang, Christopher Kruegel, and Giovanni Vigna. 2017. Rise of the hacrs: Augmenting autonomous cyber reasoning systems with human assistance. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 347–362.
[52] Nick Stephens, John Grosen, Christopher Salls, Andrew Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2016. Driller: Augmenting fuzzing through selective symbolic execution.. In *NDSS*, Vol. 16. 1–16.
[53] Jarmo Vainio. 2014. The use of data visualization in fuzz test monitoring. (2014).
[54] Xiaolin Wen, Tai D Nguyen, Lun Zhang, Jun Sun, and Yong Wang. 2025. PrettiSmart: Visual Interpretation of Smart Contracts via Simulation. *IEEE Transactions on Visualization and Computer Graphics* (2025).
[55] James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. 2019. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 56–65.
[56] Claes Wohlin. 2014. 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*. 1–10.
[57] Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. 2024. Fuzz4all: Universal fuzzing with large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.
[58] Hanxiang Xu, Wei Ma, Ting Zhou, Yanjie Zhao, Kai Chen, Qiang Hu, Yang Liu, and Haoyu Wang. 2025. CKGFuzzer: LLM-Based Fuzz Driver Generation Enhanced By Code Knowledge Graph. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 243–254.
[59] Hang Xu and Jianshan Peng. 2024. ViScheduler: Visualized Seed Scheduling Based on Runtime Features. In *2024 10th International Conference on Computer and Communications (ICCC)*. IEEE, 815–820.
[60] Qian Yan, Huayang Cao, Shuaibing Lu, and Minhuan Huang. 2023. InFuzz: An Interactive Tool for Enhancing Efficiency in Fuzzing through Visual Bottleneck Analysis (Registered Report). In *Proceedings of the 2nd International Fuzzing Workshop*. 56–61.
[61] Qian Yan, Minhuan Huang, and Huayang Cao. 2022. A survey of human-machine collaboration in fuzzing. In *2022 7th IEEE International Conference on Data Science in Cyberspace (DSC)*. IEEE, 375–382.
[62] Chenyuan Yang, Yinlin Deng, Runyu Lu, Jiayi Yao, Jiawei Liu, Reyhaneh Jabbarvand, and Lingming Zhang. 2024. Whitefox: White-box compiler fuzzing empowered by large language models. *Proceedings of the ACM on Programming Languages* 8, OOPSLA2 (2024), 709–735.
[63] Yupeng Yang, Shenglong Yao, Jizhou Chen, and Wenke Lee. 2025. Hybrid Language Processor Fuzzing via {LLM-Based} Constraint Solving. In *34th USENIX Security Symposium (USENIX Security 25)*. 6299–6318.
[64] Cen Zhang, Yaowen Zheng, Mingqiang Bai, Yeting Li, Wei Ma, Xiaofei Xie, Yuekang Li, Limin Sun, and Yang Liu. 2024. How effective are they? exploring large language model based fuzz driver generation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1223–1235.
[65] Kunpeng Zhang, Zongjie Li, Daoyuan Wu, Shuai Wang, and Xin Xia. 2025. Low-Cost and Comprehensive Non-textual Input Fuzzing with LLM-Synthesized Input Generators. *arXiv preprint arXiv:2501.19282* (2025).

[66] Xiaohan Zhang, Cen Zhang, Xinghua Li, Zhengjie Du, Bing Mao, Yuekang Li, Yaowen Zheng, Yeting Li, Li Pan, Yang Liu, et al. 2024. A survey of protocol fuzzing. *Comput. Surveys* 57, 2 (2024), 1–36.
[67] Yanjie Zhao, Xinyi Hou, Shenao Wang, and Haoyu Wang. 2024. Llm app store analysis: A vision and roadmap. *ACM Transactions on Software Engineering and Methodology* (2024).
[68] Chijin Zhou, Mingzhe Wang, Jie Liang, Zhe Liu, Chengnian Sun, and Yu Jiang. 2019. Visfuzz: Understanding and intervening fuzzing with interactive visualization. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1078–1081.
[69] Xin Zhou, Sicong Cao, Xiaobing Sun, and David Lo. 2024. Large language model for vulnerability detection and repair: Literature review and the road ahead. *ACM Transactions on Software Engineering and Methodology* (2024).