A Survey of Autonomous Agents and AI-Assisted Debugging: Integrating Large Language Models and Human-Computer Interaction

Abstract

The integration of autonomous agents and AI-assisted debugging represents a transformative shift in software engineering, leveraging advanced AI technologies and human-computer interaction (HCI) paradigms to enhance efficiency and effectiveness. Central to this evolution are Large Language Models (LLMs), which contribute to tasks such as code comprehension, generation, and automatic program repair. However, their limitations necessitate platforms that enhance accessibility and customization. Incorporating Human-In-The-Loop (HITL) methodologies is crucial for refining AI systems, aligning them with human expectations. This survey explores the role of LLMs in debugging, emphasizing their capabilities in code analysis, program synthesis, and automatic program repair. It highlights the integration of human feedback, which enhances precision and reliability, and discusses the importance of human-centered AI in fostering trust and user engagement. The survey also addresses challenges such as the opacity of LLMs, dependency on high-quality test cases, and the need for robust methodologies to handle incomplete data. Future directions include refining LLMs for domain-specific applications, expanding frameworks to support additional programming languages, and enhancing interactivity in AI systems. Ethical considerations and user trust are paramount, requiring transparent, user-friendly AI systems aligned with societal norms. By addressing these challenges and opportunities, AI-assisted debugging can significantly improve software development processes, fostering more efficient, effective, and human-aligned solutions.

1 Introduction

1.1 Overview of Autonomous Agents and AI-Assisted Debugging

The integration of autonomous agents and AI-assisted debugging represents a significant advancement in software engineering, utilizing sophisticated AI technologies within human-computer interaction (HCI) frameworks. Autonomous agents are designed to operate independently, emulating human cognitive processes, which enhances their ability to identify and rectify software anomalies in complex environments [1]. These agents adapt to evolving conditions, playing a crucial role in modern software systems.

A key component of this evolution is the use of Large Language Models (LLMs), which substantially contribute to code comprehension, generation, and automatic program repair. LLMs excel in code analysis and program synthesis, addressing challenges in automated software engineering by clarifying unexpected program behaviors [2]. However, their limitations in executing complex tasks within interactive environments highlight the need for platforms that enhance accessibility and customization [3]. Integrating LLMs with computational experiments enhances the modeling of complex social systems through anthropomorphic agents, showcasing their versatility [4].

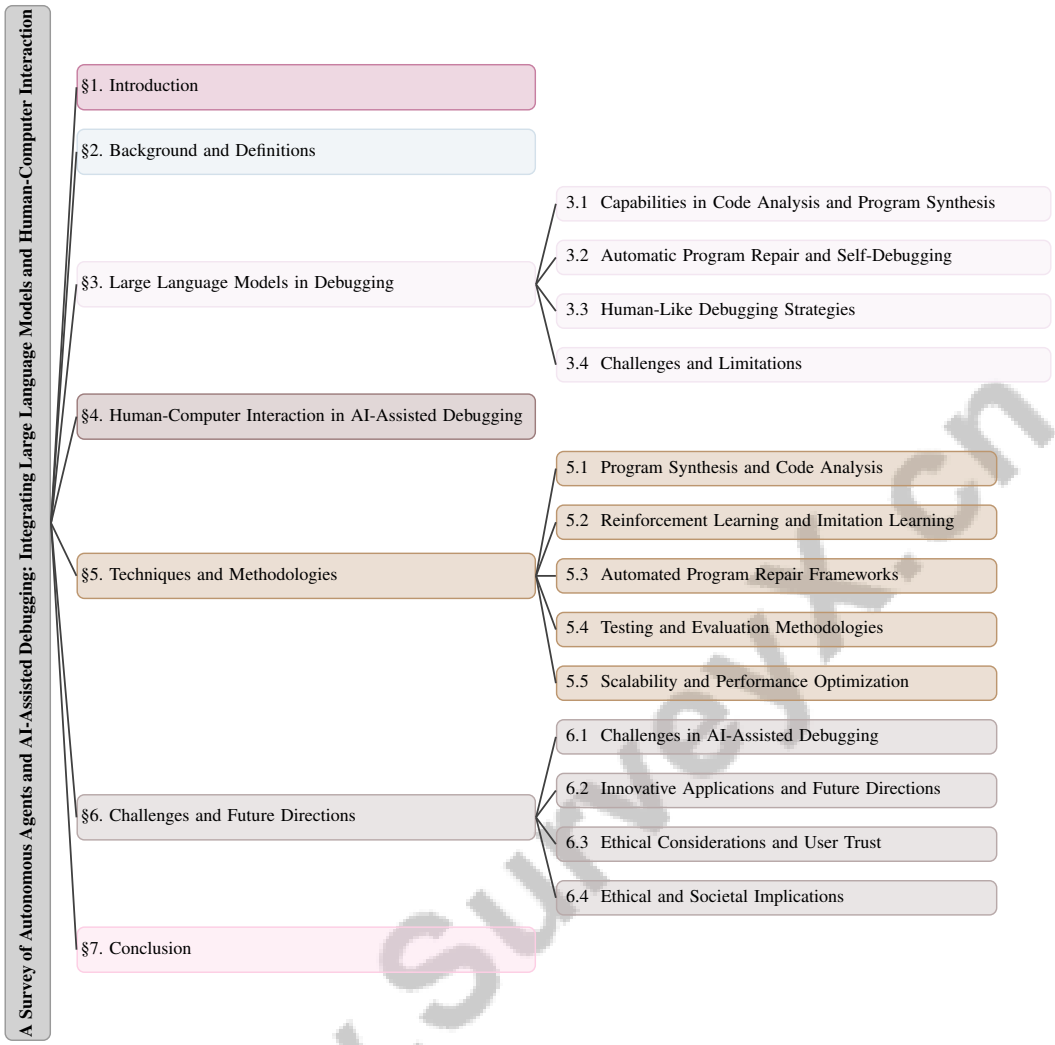


Figure 1: chapter structure

Incorporating human feedback through Human-In-The-Loop (HITL) methodologies is vital for improving the precision and reliability of AI systems. HITL approaches leverage human expertise to refine machine learning models, ensuring alignment with human expectations [5]. Given the subjective nature of bug identification, human judgment is essential in debugging, while transparency in AI systems facilitates effective human involvement [6]. This approach enhances understanding of machine learning model decisions through the visualization of local decision boundaries [7].

The dynamic field of autonomous agents and AI-assisted debugging aims to synergize AI technologies with human insights, fostering more efficient software development processes. This collaboration between AI and HCI communities promotes a human-centered AI (HCAI) approach, addressing challenges from a technology-centric development paradigm [8]. Furthermore, LLMs can generate personalized, context-aware feedback for logical errors in programming assignments, illustrating their potential to enhance software development practices [9].

The rapid evolution of foundation models like GPT-4 and DALL-E has sparked a technological revolution, enabling natural language prompts to serve as executable code [10]. This progress is supported by frameworks that facilitate the visualization, understanding, and analysis of program behavior, ensuring AI tools remain effective and user-friendly [11]. As the field advances, understanding user satisfaction with LLMs—by exploring user intents, experiences, and concerns—remains crucial, addressing a research gap focusing predominantly on LLM capabilities rather than user-centric evaluations. Recent advancements also include algorithms enabling LLMs to debug their code using self-reflection and search, essential for improving code quality [12].

AI-generated, for reference only.

1.2 Significance in Modern Software Development

The integration of autonomous agents, such as RepairAgent and CodeAgent, alongside AI-assisted debugging technologies signifies a substantial evolution in modern software development. These advancements streamline the debugging process through autonomous planning and execution of bug fixes while enhancing code reviews via multi-agent collaboration. RepairAgent employs a large language model to autonomously gather information, validate fixes, and invoke appropriate tools, successfully addressing numerous bugs, including those previously deemed unsolvable. Similarly, CodeAgent automates code reviews through multiple agents that detect inconsistencies, identify vulnerabilities, and ensure adherence to coding standards, thus improving efficiency and effectiveness in software development practices [13, 14]. Automated Program Repair (APR) techniques, exemplified by tools like Repairnator, demonstrate human-competitive capabilities by generating patches for build failures, addressing inefficiencies in traditional bug-fixing practices. This automation reduces the need for extensive manual oversight, thereby lowering costs and accelerating the development cycle.

Large language models (LLMs) play a pivotal role in advancing debugging processes by enhancing situational awareness and contextual inference, critical for optimizing tasks and minimizing human intervention. Nonetheless, challenges remain, particularly in ensuring the accuracy of control flow graph (CFG) generation, which is essential for effectively deploying autonomous agents and AI-assisted debugging systems [15].

Human feedback integration is crucial for advancing autonomous systems. Frameworks like AI-in-the-Loop (AIITL) have demonstrated the ability to reduce human effort while maintaining or improving classification accuracy compared to traditional Human-In-The-Loop (HITL) systems [16]. This integration is vital for enhancing AI performance in complex environments where human judgment is indispensable. Additionally, incorporating reviewer insights in code review processes can improve the quality of fix suggestions in automatic program repair techniques [17].

The development of human-centered AI systems ensures transparency, fairness, and accountability in algorithmic decision-making. Initiatives like Prompt Sapper enhance the development of AI-native services, increasing accessibility for non-technical users and improving user experience [18]. Furthermore, RING, a multilingual repair engine powered by a large language model, aims to overcome the limitations of existing automated repair techniques [19].

Technologies such as print debugging methods guided by LLMs, which insert print statements to output variable values during execution, improve data flow tracing and error identification, thereby enhancing debugging efficiency [20]. These advancements bolster software development teams' capabilities and foster user trust and effective communication between agents and users, providing deeper insights into event circumstances. Key challenges include potential biases in model outputs, susceptibility to adversarial attacks, and risks associated with autonomous AI systems. Understanding human explanations in eXplainable AI is essential for building trust and interpretability, addressing discrepancies between academic research and developers' actual needs. The significance of new modeling techniques, such as Adaptive Hybrid Modeling, further enhances predictive accuracy and generalization in machine learning applications [2]. Prioritizing user needs and preferences in automation technologies is critical for their successful design and development [6], ensuring that advancements in AI and autonomous systems align with human expectations.

1.3 Structure of the Survey

This survey is structured to provide a comprehensive exploration of the integration of autonomous agents and AI-assisted debugging, focusing on large language models and human-computer interaction. The paper is organized into several key sections, each addressing distinct aspects of the topic.

The introduction highlights the significance of autonomous agents and AI-assisted debugging in modern software development, emphasizing the roles of large language models and human-computer interaction. The background and definitions section offers an overview of core concepts, including definitions of key terms such as 'autonomous agent', 'large language model', 'human in the loop', and 'AI-assisted debugging', alongside their historical development.

The third section examines the role of large language models in debugging, discussing their capabilities in code analysis and program synthesis, applications in automatic program repair and

self-debugging, and their potential to mimic human-like debugging strategies while addressing challenges and limitations faced by these models.

The fourth section focuses on human-computer interaction in AI-assisted debugging, emphasizing the 'human in the loop' approach. It discusses the importance of human feedback and oversight, strategies for designing effective human-AI interactions, and the role of human-centered AI in enhancing user engagement.

The fifth section details various techniques and methodologies employed in AI-assisted debugging, including program synthesis, code analysis, reinforcement learning, imitation learning, automated program repair frameworks, testing and evaluation methodologies, and approaches for optimizing scalability and performance.

The penultimate section addresses challenges and future directions in AI-assisted debugging and autonomous agents, identifying current challenges, exploring innovative applications and potential future directions, and examining ethical considerations and societal implications.

In conclusion, the discussion highlights the significant potential of integrating large language models (LLMs) with human-computer interaction (HCI) to improve debugging processes. This integration leverages explanation-based human debugging techniques, utilizing human feedback to identify and rectify issues in machine learning models, particularly in natural language processing (NLP). The development of frameworks like the Large Language Model Debugger (LDB) enables LLMs to analyze and refine generated code by incorporating runtime execution information, mimicking human debugging practices. This approach enhances debugging accuracy and fosters a collaborative relationship between humans and AI, leading to more effective and user-centered NLP systems. Future research may focus on optimizing these interactions to further enhance user satisfaction and the overall debugging experience [21, 22, 23, 24].The following sections are organized as shown in Figure 1.

2 Background and Definitions

2.1 Core Concepts and Definitions

Understanding foundational concepts is critical in the rapidly evolving field of AI-assisted debugging and autonomous systems. This section defines key terms central to this survey.

An 'autonomous agent' operates independently, interacting with other agents to manage complexities within intelligent systems. These agents are vital in AI-assisted debugging, autonomously enhancing capabilities through resource integration and tasks like automated code reviews and collaborative decision-making. 'Shared autonomy' optimizes the balance between human and agent interventions, crucial for user satisfaction and system performance [25]. In debugging, autonomous agents collaborate with human developers, using frameworks like RCAgent to support decision-making.

'Large Language Models' (LLMs) are sophisticated AI systems trained on extensive datasets to comprehend and generate human language. They significantly aid debugging by facilitating code analysis and program synthesis, providing insights into programming concepts, and supporting students and educators through multi-level feedback mechanisms such as 'feedback-ladders' [26]. Their role is underscored by adaptive reasoning and multi-turn interactions [27]. Ensuring transparency in LLMs is crucial, especially in applications with substantial societal impacts [28].

The 'human in the loop' (HITL) paradigm integrates human intelligence at various machine learning stages, aligning AI systems with human expectations through feedback and oversight, crucial for model refinement and accuracy enhancement. HITL systems leverage human insights, essential for tackling complex problem-solving scenarios and ensuring robust AI performance [29].

'Automatic Program Repair' (APR) generates patches to rectify bugs, enhancing software maintenance reliability and efficiency. APR systems use AI to identify and rectify bugs with minimal human intervention, streamlining debugging. Challenges include managing complex multi-location bugs and the 'patch overfitting problem', where patches only succeed for specific test cases [30]. Evaluation benchmarks like SWE-Bench assess APR strategies based on issue descriptions and failing test suites.

AI-assisted debugging employs artificial intelligence to enhance the debugging process through code analysis, program synthesis, and machine learning, identifying, diagnosing, and rectifying software

anomalies more effectively than traditional methods. Techniques like 'reinforcement learning' are integral to developing adaptive AI systems that learn from human behavior and environmental interactions, enhancing problem-solving capabilities. Categorizing bugs based on root causes and impacts, such as semantic, memory, and concurrency issues, is critical for targeted debugging efforts [31].

A comprehensive understanding of these concepts in autonomous agents and AI-assisted debugging is essential for recognizing their transformative effects on contemporary software development, emphasizing human-like robustness and adaptability. This shift is exemplified by advancements in automatic program repair, prioritizing criteria like understandability, correctness, and completeness, along with human-centered automation addressing user needs. The rise of LLMs as AI pair programmers underscores the importance of effective debugging practices and transparency in AI applications, ensuring these technologies enhance productivity while remaining user-friendly [32, 33, 28, 34, 6]. Integrating 'Human-Computer Interaction' (HCI) principles is also crucial, addressing cognitive load challenges and maintaining trust and user experience in AI systems. Key terms such as 'Partial Code Reuse Chain (PCR-Chain)', 'non-fully qualified names (non-FQNs)', 'last-mile syntax errors', 'AI tools', 'ChatGPT', and 'human software engineers' further enrich the discourse on AI-assisted debugging.

2.2 Historical Development of Technologies

The historical development of AI-assisted debugging and autonomous systems has been marked by milestones reshaping modern software engineering practices. The inception of Automatic Program Repair (APR) signifies a pivotal shift, initially characterized by test-driven approaches relying on failing test cases to identify bugs, thus limiting applicability [35]. Progress in the field includes benchmarks like Scratch, leveraging LLMs to improve comprehension, debugging, and idea generation in game code [36].

The evolution of APR has been critically analyzed, focusing on automatic software repair techniques and evaluation methodologies [33]. This advancement is bolstered by real-world data from sources like GitHub issues and open-source Python projects, providing a robust foundation for conversational testing frameworks [37]. Challenges persist, notably the language-specific nature of existing automated repair techniques, complicating adaptation to new programming languages [19].

Human involvement in AI systems has been crucial, with the HITL paradigm enhancing data quality and model transparency, underscoring the necessity of human insights for improving AI reliability and accuracy [31]. Challenges remain, such as high sample complexity in reinforcement learning and potential catastrophic failures during training, necessitating continuous refinement of HITL methodologies [29]. The categorization of HITL research in natural language processing (NLP) provides valuable historical context, focusing on tasks, goals, human interactions, and feedback learning methods [21].

The historical trajectory of AI integration traditionally follows stages of monitoring, analysis, planning, execution, and feedback, reflecting the growing sophistication of AI systems in debugging processes [38]. Traditional reinforcement learning methods face limitations, particularly in managing multimodal inputs and optimizing control policies in complex environments, driving the development of more advanced technologies [25]. The introduction of cognitive modules and reusable architectures, as seen in the adaptation of Melting Pot scenarios for LLM-augmented autonomous agents, represents a significant advancement toward fostering better cooperation and robustness in AI systems [39].

The historical evolution of AI-assisted debugging and autonomous systems reflects a continuous trajectory toward more sophisticated, user-centric, and ethically aware frameworks. This progression is steering the field toward greater efficiency and effectiveness in software development and debugging processes, aligning technological advancements with human values and expectations [40]. The survey of agent structures and the advantages of LLM-based agents highlights the historical development of these technologies, emphasizing their role in enhancing predictive modeling techniques by overcoming the limitations of linear assumptions and effectively utilizing non-linear relationships.

In recent years, the integration of Large Language Models (LLMs) into software development processes has garnered significant attention due to their potential to enhance debugging efficiency. Figure 2 illustrates the hierarchical organization of concepts related to the role of LLMs in debugging,

detailing their capabilities in code analysis and program synthesis, automatic program repair and self-debugging techniques, as well as human-like debugging strategies. This figure not only highlights the advancements and tools associated with LLMs but also addresses the challenges and limitations inherent in their application. Furthermore, it outlines future directions for research and development in this promising area, providing a comprehensive overview of how LLMs can be leveraged to improve software quality and reliability.

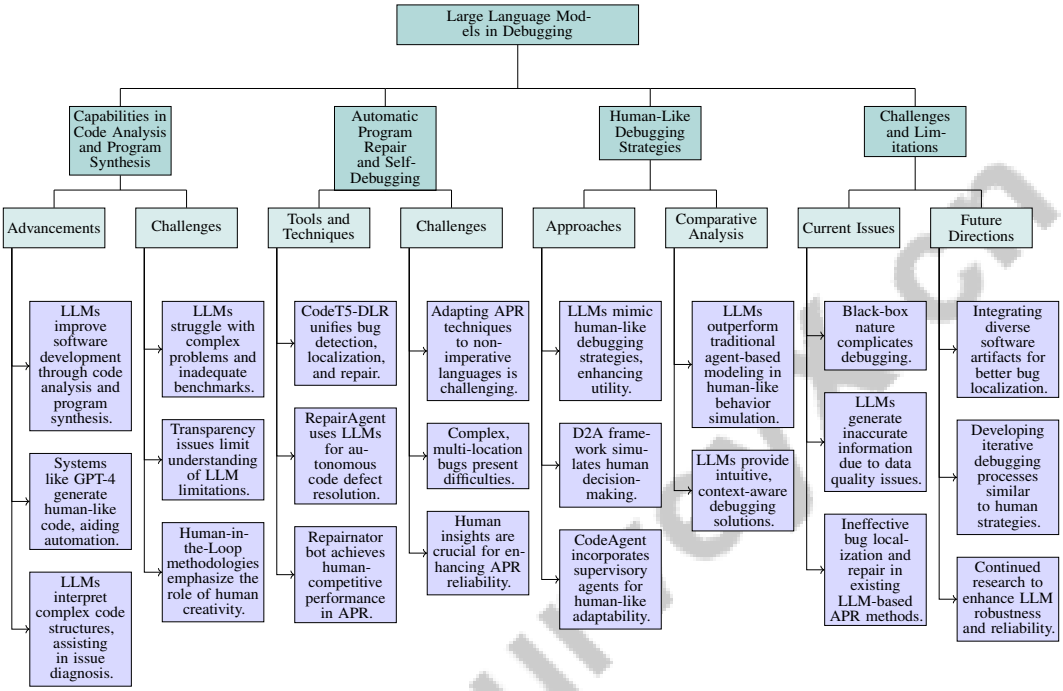


Figure 2: This figure illustrates the hierarchical organization of concepts related to the role of Large Language Models (LLMs) in debugging, detailing their capabilities in code analysis and program synthesis, automatic program repair and self-debugging techniques, human-like debugging strategies, and the associated challenges and limitations. The figure highlights the advancements, tools, techniques, and future directions in leveraging LLMs for software development.

3 Large Language Models in Debugging

3.1 Capabilities in Code Analysis and Program Synthesis

Large language models (LLMs) are integral to advancing code analysis and program synthesis, significantly improving software development. Systems like GPT-4, trained on vast datasets, generate human-like code, simplifying programming tasks with automation and intelligent suggestions [11]. LLMs excel in interpreting complex code structures, aiding developers in precise issue diagnosis and resolution. Their integration with computational experiments enhances understanding of software systems [4], reducing cognitive load and optimizing workflows.

In program synthesis, LLMs generate coherent code snippets addressing complex challenges. Techniques like PCR-Chain enhance code compilation efficiency without extensive retraining [11]. Frameworks such as CodeAct showcase LLM agents executing actions via Python, adapting outputs based on real-time observations.

Despite progress, challenges remain, particularly in addressing complex problems where LLMs often underperform. Current benchmarks inadequately assess reasoning and adaptability, while transparency issues hinder understanding of limitations [41, 28]. Human-in-the-Loop (HITL) methodologies highlight the importance of human creativity and ethical judgment in refining LLM outputs to meet user expectations.

LLMs optimize software development, enhancing productivity and enabling more intelligent engineering solutions. They automate code generation from natural language prompts, reducing manual coding errors. LLMs excel in automated program repair, utilizing diverse software artifacts for bug localization and patch generation, outperforming traditional methods. Recent advancements, like the LLM programming workflow (LPW), have improved code generation accuracy through structured refinement, achieving state-of-the-art performance across coding benchmarks [42, 43, 44]. Addressing challenges while incorporating human insights will further advance LLMs, paving the way for robust programming methodologies.

3.2 Automatic Program Repair and Self-Debugging

LLMs serve as transformative tools in Automatic Program Repair (APR) and self-debugging, enhancing the software development lifecycle. Frameworks like CodeT5-DLR unify bug detection, localization, and repair, improving debugging efficacy [45]. RepairAgent leverages LLMs for iterative planning, execution, and validation, autonomously addressing code defects [14].

The Repairnator bot exemplifies LLM application in APR, autonomously monitoring CI builds, reproducing test failures, and synthesizing patches, achieving human-competitive performance. This underscores LLMs' potential to generate patches with minimal human intervention, streamlining debugging. The CFG-Chain method enhances APR by generating Control Flow Graphs through a structured Chain of Thought approach [15].

In self-debugging, methods like SELF-DEBUGGING teach LLMs to debug their generated code through few-shot prompting, enabling error identification and rectification based on execution results [46]. Systematic print debugging techniques leverage real-time feedback and test cases to improve outcomes [20].

Challenges persist, particularly in adapting APR techniques to non-imperative languages and addressing complex, multi-location bugs. High-Quality Automated Program Repair (HQAPR) integrates constraints from natural-language artifacts to enhance patch quality [30]. The RING framework employs a three-phase approach—fault localization, code transformation, and candidate ranking—to repair programs effectively [19].

Integrating human insights into APR is crucial; methods like LEARN2FIX generate alternative test inputs and query users for labeling, enhancing APR reliability [47]. The BESTER algorithm allows LLMs to debug through self-reflections, improving automatic repair capabilities [12]. The DEVLoRe approach integrates issue content, error stack traces, and debugging information, enhancing bug localization and repair [43].

LLMs in APR and self-debugging significantly advance software engineering, effectively localizing and repairing bugs using diverse artifacts, and improving self-debugging through iterative refinement. These models revolutionize debugging processes, especially when combined with human expertise and innovative methodologies [48, 49, 43, 50]. Continued development of LLM-based approaches holds substantial potential for further advancements.

3.3 Human-Like Debugging Strategies

LLMs increasingly mimic human-like debugging strategies, enhancing software development utility. This human-like robustness (HLR) aligns AI operations with human cognitive processes, akin to Turing Test evaluations [51]. By emulating human behaviors, LLMs provide natural language explanations for debugging actions, enhancing interpretability and user trust [52].

Integrating LLMs into debugging frameworks simulates coherent, contextually relevant human-like behaviors. The D2A framework illustrates agents evaluating states and selecting activities based on intrinsic motivations, simulating human decision-making [53]. Systems like HypoCompass enable students to act as Teaching Assistants, helping LLMs debug code and improve strategies through human interaction [32].

Innovations like CodeAgent incorporate supervisory agents to maintain focus on review objectives, ensuring human-like adaptability during debugging [13]. Agent S showcases enhanced planning capabilities and adaptability to dynamic interfaces, mimicking human problem-solving strategies [54].

These approaches underscore LLMs' potential to autonomously generate human-like explanations for actions in real-time, enhancing AI interpretability and effectiveness [1].

Comparing traditional agent-based modeling (ABM) with LLM-based agents highlights LLMs' superior capabilities in simulating human-like behaviors essential for effective debugging [4]. By leveraging these strategies, LLMs provide more intuitive, context-aware debugging solutions, improving software development processes and aligning AI systems with human expectations.

3.4 Challenges and Limitations

Despite significant advancements, LLMs face challenges and limitations hindering their full potential in software development. Figure 3 illustrates these challenges, including the black-box nature of LLMs, which complicates debugging, as existing explanation methods offer limited insights into model behavior [55]. This opacity is exacerbated by LLMs' potential to generate inaccurate information, linked to data quality and reliance on external validation [56].

LLMs' effectiveness diminishes in hard-level problems, where complex issues and advanced algorithms present hurdles [20]. Additionally, reliance on LLMs for bug repairs in Python assignments may not generalize to other languages, complicating implementation [9].

Existing LLM-based APR methods struggle with ineffective bug localization and repair due to a failure to integrate multiple software artifacts [43]. Furthermore, LLM reasoning limitations and substantial token consumption for generating and verifying plans constrain planning-driven programming approaches [42].

Current action formats' inability to dynamically compose multiple tools or adjust actions based on environmental observations restricts agents' effectiveness in real-world applications [10]. While LLMs like ChatGPT outperform novice programmers in solving easy and medium-level problems, they lag behind experienced programmers in handling complex scenarios [11].

Moreover, existing methods often overlook iterative debugging processes employed by human programmers, resulting in ineffective bug identification and correction [12]. This limitation highlights the necessity of incorporating human-like iterative strategies to enhance LLM efficacy in debugging tasks.

Addressing these challenges is essential for advancing LLM robustness, adaptability, and reliability in debugging applications. Continued research is vital to overcome limitations and fully harness LLM capabilities in software engineering. Recent advancements, like integrating diverse software artifacts for bug localization and program repair, demonstrate LLMs' potential to improve APR through effective patch generation. Tools like GeneUS automate requirements engineering by transforming documents into user stories, streamlining workflows. Innovative workflows, such as the LLM Programming Workflow (LPW), show promise in refining code generation through structured phases, leading to improved accuracy [43, 28, 7, 42, 44]. These developments underscore the necessity for ongoing exploration and refinement of LLM applications to optimize their role in software development, ultimately resulting in more efficient and effective engineering practices.

4 Human-Computer Interaction in AI-Assisted Debugging

4.1 Human in the Loop

Incorporating the 'human in the loop' (HITL) approach is pivotal in AI-assisted debugging, enhancing accuracy and reliability by integrating human expertise and feedback. This methodology addresses AI complexities and uncertainties, ensuring alignment with human expectations and enhancing performance [55, 57]. HITL frameworks refine AI outputs through human judgment, adapting them to real-world scenarios [58].

As illustrated in Figure 4, the hierarchical categorization of HITL in AI systems highlights various frameworks and systems, along with their associated benefits and challenges, as well as applications and tools. This figure emphasizes the integration of human expertise to enhance AI model precision, reduce biases, and improve user engagement. HITL systems like FIND improve model precision, reduce biases, and enhance interpretability through structured human interaction [6]. This interaction is crucial for maintaining transparency, fostering user trust [59]. By integrating human insights, HITL

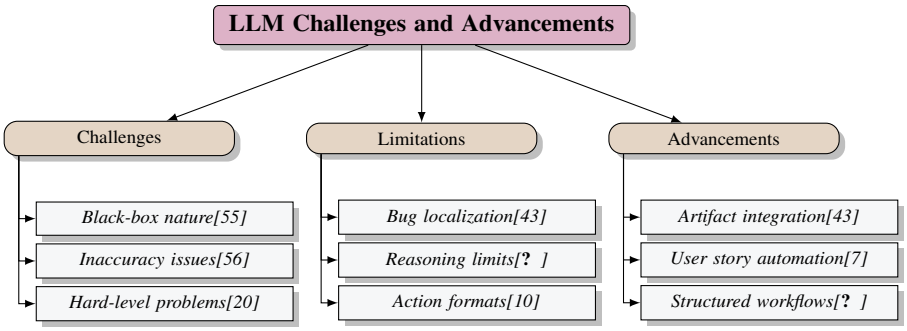


Figure 3: This figure illustrates the challenges, limitations, and recent advancements in the application of Large Language Models (LLMs) in software development, highlighting key issues such as the black-box nature of LLMs, inaccuracy in hard-level problems, and limitations in bug localization and reasoning. It also presents advancements like integrating diverse software artifacts and structured workflows to improve LLM efficacy.

systems facilitate effective communication between users and AI, enhancing trust and engagement [59].

Decentralized approaches, such as Dialogue Learning with Human-in-the-Loop (DL-HL), emphasize human supervision in guiding agent actions via peer-to-peer communication [59]. Tools like Prompt Sapper empower non-technical users to create sophisticated AI services, highlighting HITL methodologies' accessibility benefits [18].

Planning-driven programming approaches exemplify HITL integration by co-designing agent systems and user interfaces, enhancing efficiency and engagement [42]. This ensures AI systems accurately interpret user inputs, reducing error propagation and enhancing reliability.

HITL is indispensable for ensuring AI-assisted debugging systems' accuracy and reliability. By fostering collaborative environments where human expertise complements AI processes, HITL enhances decision-making capabilities, leading to effective and trustworthy solutions. It emphasizes human oversight in navigating AI complexities, advocating for a bidirectional framework aligning AI systems with human goals and ethical values while promoting societal adaptation to AI advancements. Moreover, it highlights the need for transparency, especially in the context of large language models (LLMs), to address diverse stakeholder needs in the evolving AI landscape [28, 60].

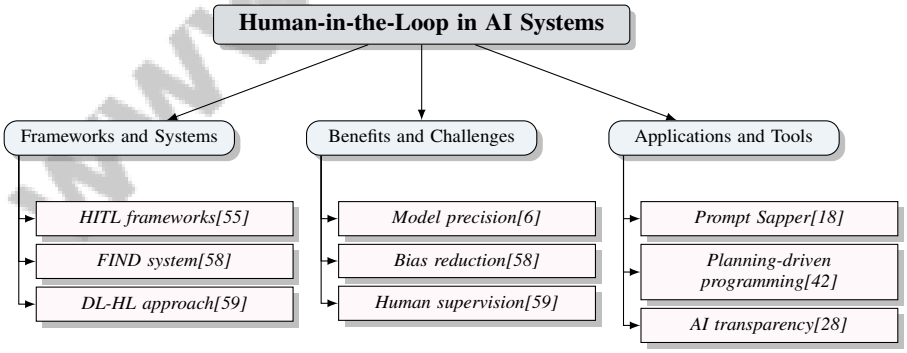


Figure 4: This figure illustrates the hierarchical categorization of Human-in-the-Loop (HITL) in AI systems, highlighting frameworks and systems, benefits and challenges, and applications and tools. It emphasizes the integration of human expertise to enhance AI model precision, reduce biases, and improve user engagement.

4.2 Human Feedback and Oversight in Debugging

Human feedback and oversight are crucial for enhancing AI performance in debugging, particularly in AI-assisted frameworks where output complexity poses challenges. Incorporating human feed-

AI-generated, for reference only.

back refines AI models, ensuring outputs align with human expectations and addressing capability uncertainties [61].

The Human-In-The-Loop (HITL) paradigm combines human intuition with AI's computational power, facilitating diverse scenario generation and improving debugging effectiveness [62]. Real-time human feedback enhances conversational agents' learning processes, allowing practical applications in real-world scenarios [59].

Enhancing transparency in AI systems is crucial for human oversight. Methods like option-centric rationale display communicate intelligent assistants' decision-making processes to users, improving understanding and trust [63]. Such transparency fosters user trust and ensures AI systems are reliable partners in debugging.

Frameworks like Cogment enable seamless collaboration between humans and AI agents by unifying training approaches, allowing effective integration of human feedback [64]. E-COACH demonstrates effective feedback utilization, leading to improved policy performance compared to traditional methods [57].

Despite advancements, challenges persist in maintaining user engagement and trust. The ad hoc nature of existing assurance techniques often lacks formal ties to established trust models, hindering reliable trust mechanisms [65]. If not managed properly, user feedback may inadvertently lower trust in the system, emphasizing the need for careful feedback mechanism design [66].

Designing for trust and avoiding over-reliance on AI systems is crucial, with studies highlighting the need for balanced human-AI interaction strategies that prioritize user understanding and decision-making. By leveraging human oversight to enhance data quality, scalability, and adaptability, AI systems can be more effectively integrated into debugging processes, ensuring responsiveness to user needs and delivering reliable outcomes [67].

Human feedback and oversight are essential in AI-assisted debugging, providing checks and balances that enhance AI systems' effectiveness. This involvement ensures AI-generated solutions address coding errors while aligning with broader human values. Studies show that human-centered approaches, such as explanation-based human debugging, enable users to effectively identify issues in natural language processing models. Frameworks like FIND empower users to debug deep learning classifiers trained on imperfect datasets, underscoring the importance of human input in mitigating biases and improving model performance. Integrating human perspectives fosters intuitive and responsive interactions between AI systems and users, leading to reliable and ethically aligned solutions [6, 21, 34, 58]. As AI technologies evolve, human oversight remains critical for guiding development and ensuring successful integration into software engineering practices.

4.3 Designing Effective Human-AI Interactions

Designing effective human-AI interactions is crucial for maximizing AI technologies' benefits while minimizing risks. Theoretical perspectives underscore user awareness and potential risks associated with AI and augmented reality (AR) integration, highlighting the need for informed user engagement in the design process [68].

Transparency-enhancing methods, like option-centric rationale displays, present all possible actions an autonomous agent could take along with the rationale for recommendations, enhancing user understanding and trust [63]. By clarifying AI decision-making processes, users can better comprehend AI actions, leading to improved trust and collaboration.

Conceptual models like DECAI, based on control systems theory, provide a framework for analyzing interaction dynamics between users and AI through their interfaces [69]. This model offers insights into designing interfaces that promote seamless and intuitive interactions, enhancing user satisfaction and system effectiveness.

Designing effective human-AI interactions necessitates a comprehensive approach emphasizing transparency, user awareness, and theoretical modeling. This strategy ensures AI systems operate effectively while aligning with human values and ethical principles. Transparency helps stakeholders understand AI operations across contexts, addressing unique challenges posed by advanced AI technologies like large language models (LLMs). Fostering user awareness encourages active engagement and agency, allowing users to co-design AI mechanisms rather than merely contesting

decisions. A bidirectional alignment framework acknowledges the dynamic nature of human-AI interactions, promoting ongoing collaboration to ensure AI systems evolve in ways reflecting human goals and societal values. Integrating these elements is crucial for developing responsible, user-centered AI solutions that enhance effectiveness and alignment with human needs [68, 28, 70, 6, 60]. As AI technologies continue to evolve, these strategies will be essential for fostering productive and trustworthy human-AI collaborations.

4.4 Human-Centered AI and User Engagement

Integrating Human-Centered AI (HCAI) into AI-assisted debugging systems is pivotal for enhancing user engagement and improving interaction outcomes. HCAI prioritizes human needs, values, and experiences in AI system design, fostering intuitive and satisfying user interactions. This approach is crucial in AI-assisted debugging, where user engagement and trust are vital for successful tool adoption and utilization [6].

A core principle of HCAI is aligning AI functionalities with human cognitive processes, ensuring users can easily comprehend and interact with AI systems. Incorporating user feedback into design and operation allows for continuous refinement and adaptation to user preferences [53]. Leveraging user insights enables HCAI systems to provide personalized and contextually relevant interactions, enhancing satisfaction and engagement.

HCAI frameworks also emphasize transparency and interpretability, critical for building user trust. Transparent AI systems that communicate decision-making processes enable users to understand and trust AI actions, leading to effective human-AI collaboration [63]. This transparency is particularly important in debugging scenarios, where users must trust AI recommendations to resolve software issues effectively.

Designing HCAI systems involves creating user-friendly interfaces that facilitate seamless interactions. By employing design principles prioritizing user experience, HCAI systems reduce cognitive load and enhance engagement, making AI-assisted debugging tools accessible and effective for a broader range of users [68]. These interfaces support users in understanding and interacting with AI systems, improving the overall quality and effectiveness of the debugging process.

The role of HCAI in engaging users and improving interaction outcomes is indispensable in developing AI-assisted debugging systems. By emphasizing human values and experiences, HCAI frameworks enhance AI systems' technical capabilities while ensuring alignment with user needs and expectations. This approach fosters effective and satisfying human-AI collaborations by promoting usability, explainability, and ethical considerations in AI development. Furthermore, integrating user experience design with AI technology addresses the limitations of traditional technology-centered methods, which have led to numerous AI-related incidents. HCAI frameworks advocate for a bidirectional alignment process that recognizes the dynamic interplay between human and AI objectives, ultimately supporting individuals and society in adapting to the evolving landscape of AI advancements [60, 71].

5 Techniques and Methodologies

A comprehensive understanding of the various techniques and methodologies in AI-assisted debugging is crucial for enhancing software development practices. Key components include program synthesis and code analysis, which streamline debugging while improving code accuracy and reliability. Table 1 presents a detailed summary of the diverse techniques and methodologies integral to AI-assisted debugging, illustrating their roles and applications across various domains within software development. Table 3 offers a detailed comparison of various AI-assisted debugging methodologies, emphasizing their distinct features and contributions to software development. This subsection explores these methodologies, focusing on their roles in automating code generation, refinement, and validation.

5.1 Program Synthesis and Code Analysis

Program synthesis and code analysis are vital for AI-assisted debugging, utilizing advanced techniques to enhance software development efficiency. These methods enable automated code generation,

Category	Feature	Method
Program Synthesis and Code Analysis	AI-Driven Techniques	PCR-Chain[3], CA[10]
Reinforcement Learning and Imitation Learning	Human-Centric Learning	ReHAC[72], PP[73], HuGE[74], IDA[25]
Automated Program Repair Frameworks	Language-Based Enhancements Human-Integration Techniques	RTT[48] MAPE-K-HMT[38]
Testing and Evaluation Methodologies	Comprehensive Assessment	LLM-TCG[75], Sim-ATAV[8]
Scalability and Performance Optimization	Parallel Processing Dynamic Adaptation	HILML[76] AHM[2]

Table 1: This table provides a comprehensive overview of various methodologies and techniques employed in AI-assisted debugging, categorized into five key areas: program synthesis and code analysis, reinforcement and imitation learning, automated program repair frameworks, testing and evaluation methodologies, and scalability and performance optimization. Each category is associated with specific features and methods, highlighting the diversity and sophistication of approaches used to enhance software development practices. The table also references significant studies and frameworks, underscoring the transformative potential of these advanced AI systems in improving debugging efficiency and accuracy.

refinement, and validation, reducing the need for extensive human intervention. For instance, Round-Trip Translation with Large Language Models (LLMs) has proven effective in correcting grammatical errors and can be applied to Automatic Program Repair (APR), surpassing traditional debugging methods. Research indicates that LLMs can effectively repair many bugs overlooked by conventional models, thus improving software debugging reliability. Integrating insights from code reviews into the repair process has also been shown to enhance bug fix accuracy, pointing to promising future research directions in automated software repair [48, 33, 77].

A notable methodology is the PCR-Chain, which employs a series of AI units to sequentially address coding issues, enhancing code compilability through structured interactions with LLMs [3]. This systematic breakdown of complex coding problems ensures precision in addressing each component, thereby improving overall code quality.

Additionally, CodeAct exemplifies advancements in handling complex tasks via dynamic code execution, utilizing existing software libraries and automated feedback mechanisms for self-debugging, which allows real-time code quality improvements [10]. The iterative repair of code specifications using LLMs highlights significant advancements in program synthesis, with studies showing that LLMs, particularly GPT-4 variants, excel in fixing bugs in declarative languages like Alloy, outperforming traditional methods while introducing minimal overhead. Innovative approaches such as Round-Trip Translation have also shown potential in restoring common coding patterns, effectively addressing unique bugs that conventional models may miss. Furthermore, multilingual repair engines like RING demonstrate LLMs' capability to facilitate cross-language code repair, enhancing programmer productivity [19, 48, 49]. Techniques like few-shot learning enable LLMs to perform code transformations with minimal examples, underscoring the importance of continuous feedback in optimizing program synthesis.

Methodologies such as Adaptive Hybrid Modeling (AHM) enhance predictive modeling by dynamically integrating linear and non-linear techniques based on real-time data analysis. This is essential for navigating modern software system challenges and improving program synthesis precision. By employing techniques like round-trip translation with LLMs and unified Detect-Localize-Repair frameworks, AI-assisted debugging adapts to evolving software development requirements, improving bug detection and repair automation [45, 48].

The methodologies surrounding program synthesis and code analysis in AI-assisted debugging reflect the transformative potential of advanced AI systems. Integrating frameworks like FIND, which allows human intervention to disable irrelevant features in deep learning classifiers, along with insights from programmers' debugging habits, enhances debugging efficiency and accuracy. This approach addresses challenges associated with imperfect training datasets and highlights the significance of human feedback in improving model performance. Utilizing natural language instructions from code reviews can further enhance automatic program repair techniques, leading to robust software development practices [58, 33, 21, 78, 77].

5.2 Reinforcement Learning and Imitation Learning

Reinforcement Learning (RL) and Imitation Learning (IL) are essential methodologies for enhancing AI-assisted debugging, providing structured frameworks that improve decision-making and facilitate human expertise integration. Research shows that incorporating human feedback significantly boosts intelligent systems' learning capabilities, with IL enabling agents to mimic expert behaviors, such as eye movements during code reading, to localize issues and generate comments. RL can be optimized through agent-agnostic human-in-the-loop approaches, allowing agents to learn from expert advice without being restricted by specific representations. This combination of RL and IL addresses real-world complexities and promotes the development of robust, adaptable AI models [79, 80, 81, 73].

In RL, state-of-the-art models like Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) refine debugging techniques by adapting to software environment complexities [79]. These models leverage environmental feedback to optimize debugging action selection, enhancing efficiency and accuracy. The ReHAC framework integrates RL by determining optimal moments for human intervention, contrasting with static heuristic methods that lack adaptability [72].

IL techniques such as Behavior Cloning (BC) and Generative Adversarial Imitation Learning (GAIL) replicate expert debugging strategies. By learning from demonstrations, IL models emulate human-like decision-making, enhancing AI systems' robustness in debugging tasks [79]. The Helix system exemplifies IL's potential by optimizing intermediate result reuse, minimizing execution time, and streamlining iterative debugging workflows [76].

The integration of human feedback is pivotal in both RL and IL, as demonstrated by Human Guided Exploration (HuGE), which utilizes asynchronous human input to guide RL task exploration [74]. This ensures AI systems align with human expectations and adapt to diverse debugging scenarios. Additionally, 'Protocol Programs' in RL emphasize structured human-agent interactions, facilitating effective learning and decision-making [73].

Shared autonomy frameworks dynamically balance human and AI actions based on expected returns, optimizing goal-oriented debugging strategies [25]. This dynamic decision-making enhances AI-assisted debugging systems' adaptability and performance, ensuring responsiveness to real-world challenges.

The integration of RL and IL in debugging processes underscores their significant role in developing intelligent, adaptive, and human-aligned AI systems. These methodologies facilitate efficient debugging practices and promote transparency and critical engagement with AI outputs, exemplified by frameworks like HypoCompass, which empower users to evaluate and debug code generated by LLMs. This trend fosters collaborative human-AI interactions that prioritize user understanding and ethical AI development considerations [28, 18, 16, 32]. By leveraging both autonomous learning and human expertise, RL and IL advance AI-assisted debugging capabilities, paving the way for more effective software development practices.

5.3 Automated Program Repair Frameworks

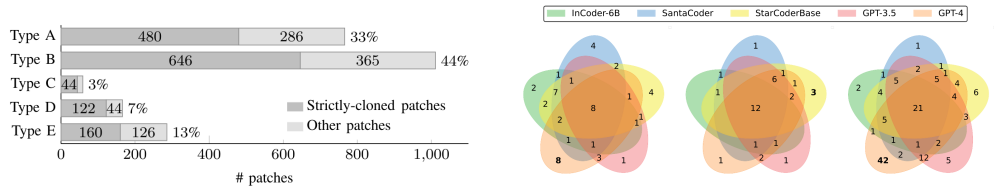
Automated Program Repair (APR) frameworks are crucial in enhancing AI-assisted debugging by systematically identifying and rectifying software bugs. These frameworks utilize developer-written tests to localize defects and generate candidate patches, though they often face challenges like patch overfitting, where generated patches may not be acceptable due to limited test cases. Recent advancements address these limitations by integrating constraints from natural-language software artifacts, such as bug reports and requirements specifications, to improve patch quality. Innovative approaches like PracAPR propose interactive repair systems that function within Integrated Development Environments (IDEs) without extensive test suites, streamlining the debugging process. By leveraging diverse software artifacts, including issue content and debug information, methodologies like DEVLoRe have demonstrated significant improvements in fault localization and patch generation, outperforming existing APR techniques [43, 35, 82, 30, 9]. These frameworks employ sophisticated algorithms to generate patches that address software defects, enhancing reliability and efficiency in software maintenance.

A significant study indicates that 68% of multi-hunk patches contain change clones, suggesting automated tools can effectively generate multi-location repairs [83]. This finding emphasizes the

need for robust APR frameworks capable of handling complex repair scenarios involving multiple code locations, thereby reducing the manual effort required in traditional debugging practices.

The MAPE-K-HMT framework exemplifies human-machine teaming integration in APR processes, extending the traditional MAPE-K loop to incorporate runtime models that facilitate human-machine interactions [38]. This approach enhances APR systems' adaptability and responsiveness, allowing dynamic adjustments to evolving software environments and user requirements. By incorporating human insights into the repair process, MAPE-K-HMT ensures automated repairs align with human expectations, improving overall software quality.

APR frameworks automate the debugging process, providing efficient and reliable solutions for software maintenance. By integrating sophisticated algorithms with human feedback mechanisms, these frameworks significantly enhance AI-assisted debugging systems' functionality. This approach facilitates effective, user-centered software development, as evidenced by studies demonstrating how such systems provide targeted guidance, help identify and resolve coding errors efficiently, and adapt to real-world dataset nuances. For instance, AI-generated hints can accelerate student learning in programming by addressing multiple issues simultaneously, while frameworks like FIND allow human intervention to refine text classifiers trained on imperfect datasets, ultimately leading to more robust software solutions [21, 58, 34, 33].



(a) The image represents a bar chart showing the distribution of different types of patches in a dataset.[83]

(b) Venn Diagrams of Model Coverage for Different Generative Models[48]

Figure 5: Examples of Automated Program Repair Frameworks

As shown in Figure 5, various techniques and methodologies have been developed to enhance the effectiveness and efficiency of automated program repair. The first image is a bar chart detailing the distribution of different types of patches in a dataset, categorized into five types: Type A, Type B, Type C, Type D, and Type E. This chart highlights the number of patches ranging from 0 to 1,000, distinguishing between 'Strictly' and other patch types, providing insights into the prevalence and diversity of applied patches. Complementing this is a Venn diagram illustrating model coverage for generative models, including InCoder-6B, SantaCoder, StarCoderBase, GPT-3.5, and GPT-4, showcasing the overlap and exclusivity in their task coverage. Together, these visualizations underscore the complexity and sophistication of current automated program repair frameworks, highlighting the diversity of patch types and the varying coverage of generative models in addressing software bugs [83, 48].

5.4 Testing and Evaluation Methodologies

Testing and evaluation methodologies are crucial for assessing the effectiveness and reliability of AI-assisted debugging systems. These methodologies establish a comprehensive framework for systematically evaluating debugging tools' efficacy, ensuring they effectively detect and localize bugs while facilitating program repair, thus aligning with established software development quality standards. Advanced techniques, such as the Detect-Localize-Repair framework based on the CodeT5 model and large language models for automatic test case generation from bug reports, enhance the debugging process and improve overall software reliability. Innovative tools like the echo-debugger and the Convergence Divergence Mapping algorithm further refine the debugging process by allowing developers to compare code executions [45, 87, 75].

Key aspects of evaluating AI-assisted debugging systems involve metrics like Exact-Code-Match (ECM) and Code BLEU, which measure the accuracy of generated code snippets in repairing defects [84]. These metrics quantify the precision of automated program repair processes, enabling developers to assess the quality of generated patches and their alignment with intended functionality.

Benchmark	Size	Domain	Task Format	Metric
CR-LLM[84]	31,163	Code Review	Code Repair	Exact-Code-Match, Code BLEU
HCAI-IL-RL[79]	1,000,000	Imitation Learning	Action Selection	Cumulative Reward, Episode Length
CCB[83]	3,049	Software Engineering	Patch Analysis	Change Clone Frequency, Strictly-Cloned Patch Percentage
IDAT[85]	9,000	Interactive Task-Solving	Collaborative Building	F1, MRR
KCBM[34]	872	Programming Education	Programming Assignment Evaluation	Overlap with Top-3 Missing KCs, Progress Reduction Rate
LLM-Bench[41]	23	Legal Reasoning	Multiple Choice Questions (mcqs)	Accuracy, F1-score
LLM-CodeGen[44]	104	Code Generation	Function Generation	Accuracy, Time Efficiency
SAD[86]	13,198	Artificial Intelligence	Question Answering	Accuracy, MMLU

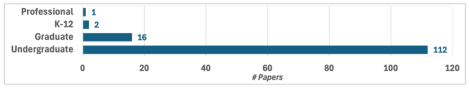
Table 2: This table presents a comprehensive overview of various benchmarks utilized in evaluating AI-assisted debugging and software engineering systems. Each benchmark is characterized by its size, domain, and task format, along with specific metrics used to assess performance. The benchmarks encompass diverse areas such as code review, programming education, and legal reasoning, highlighting the multifaceted approaches to testing and evaluation in different contexts.

Performance assessment using cross-validation metrics, such as mean squared error and R-squared values, is instrumental in evaluating AI models' generalization ability on unseen data [2]. These metrics provide insights into AI-assisted debugging systems' robustness, ensuring they effectively handle diverse programming scenarios and maintain high performance across contexts.

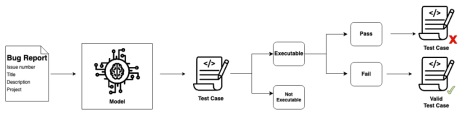
Generating test cases based on covering arrays is another critical evaluation method, assessing satisfaction of Signal Temporal Logic (STL) requirements [8]. This approach ensures comprehensive coverage of potential execution paths, identifying edge cases and verifying that the system adheres to specified temporal constraints. Employing rigorous testing methodologies enhances the reliability and predictability of AI-assisted debugging systems, ensuring correct functionality under various conditions.

Integrating these testing and evaluation methodologies is vital for continuously improving AI-assisted debugging systems. By employing various accuracy metrics, robust cross-validation techniques, and advanced test case generation informed by natural language bug reports, developers can significantly enhance debugging solutions' reliability and effectiveness. This approach streamlines software development processes, improves automated program repair quality, and addresses critical challenges such as semantic bugs and existing test suites' limitations [75, 78, 88, 45, 77].

Table 2 provides a detailed overview of representative benchmarks employed in the assessment of AI-assisted debugging systems, illustrating the diverse domains and metrics that underpin effective evaluation methodologies.



(a) Number of Papers by Academic Level[89]



(b) A flowchart illustrating the process of creating and validating test cases for a bug report[75]

Figure 6: Examples of Testing and Evaluation Methodologies

As shown in Figure 6, effective testing and evaluation methodologies are crucial for ensuring quality and reliability in software development and academic research. The first example, "Number of Papers by Academic Level," visually represents the distribution of academic contributions across various levels, quantifying the number of papers using a bar chart. This chart provides insights into research output and engagement at different academic stages. The second example, "A flowchart illustrating the process of creating and validating test cases for a bug report," showcases a systematic approach to software testing, delineating the steps from receiving a bug report to generating test cases through a model. Together, these examples underscore the diverse techniques and methodologies employed in

testing and evaluation, from academic research metrics to practical software development processes [89, 75].

5.5 Scalability and Performance Optimization

The scalability and performance optimization of AI-assisted debugging systems are essential for effectively managing large-scale software projects and intricate debugging tasks. Innovations like the Large Language Model Debugger (LDB) enhance debugging by segmenting programs into basic blocks and utilizing runtime execution information for efficient error identification. Additionally, leveraging print debugging techniques has shown promise in improving LLM performance on complex programming problems, achieving notable accuracy improvements in real-world coding challenges [20, 24]. These systems must efficiently process vast amounts of code and data while maintaining high performance and accuracy, necessitating advanced techniques and methodologies.

One approach to enhancing scalability is utilizing distributed computing frameworks, enabling parallel processing of debugging tasks across multiple nodes. This significantly reduces the time required to analyze and repair large codebases, improving overall debugging efficiency [2]. By leveraging distributed systems' computational power, AI-assisted debugging tools can scale to accommodate extensive software project demands.

Performance optimization is further achieved through adaptive learning algorithms that dynamically adjust to software environments' complexity and variability. Techniques like reinforcement learning and imitation learning allow AI systems to refine debugging strategies based on real-time feedback and historical data, enhancing their ability to navigate and resolve complex code issues [79]. These adaptive methodologies ensure AI-assisted debugging systems remain responsive and effective against evolving software challenges.

Moreover, implementing caching mechanisms and efficient data management strategies is crucial for optimizing AI-assisted debugging systems' performance. By minimizing redundant computations and effectively managing memory resources, these systems can reduce latency and improve debugging operation speed [76]. This optimization is essential for maintaining high throughput and ensuring AI tools can handle real-time debugging demands.

Scalability and performance optimization are crucial for AI-assisted debugging systems' effective implementation in large-scale software development environments, as they must efficiently handle complex logic flows and data operations, leverage runtime execution information, and integrate techniques like print debugging and unified frameworks for bug detection, localization, and repair to enhance debugging accuracy and productivity [45, 20, 87, 24]. By employing distributed computing, adaptive learning algorithms, and efficient data management techniques, these systems can deliver robust and efficient debugging solutions, enhancing software engineering processes' productivity and reliability.

Feature	Program Synthesis and Code Analysis	Reinforcement Learning and Imitation Learning	Automated Program Repair Frameworks
Automation Level	High	Moderate	High
Human Interaction	Minimal	Significant	Moderate
Performance Optimization	Advanced Techniques	Adaptive Learning	Sophisticated Algorithms

Table 3: The table provides a comparative analysis of three key methodologies in AI-assisted debugging: Program Synthesis and Code Analysis, Reinforcement Learning and Imitation Learning, and Automated Program Repair Frameworks. It highlights differences in automation levels, the extent of human interaction required, and the sophistication of performance optimization techniques employed by each approach. This comparison underscores the diverse strategies utilized to enhance software development practices through AI integration.

6 Challenges and Future Directions

6.1 Challenges in AI-Assisted Debugging

AI-assisted debugging systems encounter several challenges that hinder their effectiveness in software development. A significant issue is the lack of interactivity, preventing users from challenging or seeking alternative explanations, thereby necessitating more adaptable and interactive debugging tools [1]. This points to the need for systems that incorporate user feedback to enhance debugging

efficacy. Another challenge is the dependency on high-quality test cases, which are often scarce, particularly for complex user scenarios, limiting models like LDB that require extensive feedback [42]. Thus, it is crucial to develop methods that perform well even with incomplete data.

The opacity of LLM architectures poses an additional barrier, as proprietary designs restrict transparency, complicating effective human-AI collaboration [4]. This lack of transparency can impede AI integration in collaborative settings where human oversight is vital. In reinforcement learning, optimizing reward functions depends heavily on high-quality human feedback, which can introduce biases if the feedback is flawed [12]. Moreover, navigating known and unknown risks, such as biases and complacency, complicates robust AI system development [11]. The reliance on public LLMs can affect result generalizability, as human bias in code evaluation introduces variability [10].

APR techniques face challenges due to their reliance on bug reports and relevant test cases, which may not always be available in real-world scenarios [3]. Additionally, the insufficient incorporation of developer discussions or design rationales in benchmarks limits APR effectiveness [42]. Current automated repair systems often address historical rather than real-time issues, limiting their applicability in dynamic environments [10].

Existing methods' tendency to overfit training data leads to poor performance on unseen data due to simplistic assumptions [4]. This is further exacerbated by the need for substantial labeled training data for high accuracy, which is not always available [3]. Moreover, these methods often fail to leverage diverse software artifacts, resulting in suboptimal bug localization and repair performance [12].

Instability during online learning, particularly with sparse feedback, highlights ongoing challenges in AI-assisted debugging [11]. This instability can undermine AI systems' reliability and effectiveness, necessitating robust and adaptable learning algorithms. Addressing these challenges is vital for advancing AI-assisted debugging, ensuring these systems are technically proficient and aligned with human values and expectations. By overcoming these hurdles, AI systems can significantly enhance software development processes, offering timely assistance through programming hints, user story automation, and strategic testing guidance, ultimately improving code quality and user satisfaction [7, 90, 34, 11].

6.2 Innovative Applications and Future Directions

The future of AI-assisted debugging and autonomous agents is set to explore innovative applications aimed at enhancing accuracy, efficiency, and human-AI collaboration. A key area of focus is refining LLMs for domain-specific applications, customizing these models to address unique programming challenges while improving explainability [4]. This refinement will also involve integrating alternative solution representations to enhance code generation accuracy and optimize LLM reasoning capabilities [42].

Expanding frameworks like PCR-Chain to support additional programming languages and bolster robustness against emerging LLM architectures represents another promising direction [3]. Similarly, future research will adapt frameworks such as CodeAct to undertake more complex tasks and enhance robustness through further instruction tuning and dataset expansion [10].

Incorporating interactivity in rationale generation offers a significant research opportunity, particularly in continuous-action environments, facilitating dynamic user engagement [1]. Additionally, integrating diverse software artifacts into existing frameworks will broaden applicability across programming languages and improve debugging processes [43].

Future research should emphasize developing explainable AI systems that ensure timely human intervention and enhance safety protocols, especially in contexts like autonomous vehicles [5]. Fostering human-centered design will involve exploring novel methods to improve user interaction and trust, promoting effective human-AI collaboration.

Advancements in scene rendering tools and machine learning components, exemplified by frameworks like Sim-ATAV, aim to enhance realism and effectiveness in autonomous system testing [8]. Techniques like BESTER can be applied to more complex software engineering tasks, further enhancing performance and adaptability in real-world scenarios [12].

Finally, future research should investigate a broader range of software engineering tasks and AI approaches, assessing both quantitative and qualitative aspects of task execution to provide a comprehensive understanding of AI capabilities in software development [11]. By pursuing these innovative applications and research directions, the field of AI-assisted debugging can evolve, offering more precise, efficient, and collaborative solutions to modern software development challenges.

6.3 Ethical Considerations and User Trust

Integrating AI systems into debugging processes necessitates a thorough examination of ethical considerations and the establishment of user trust, both crucial for responsible deployment. Understanding the sociotechnical implications of AI, as evidenced by historical contexts of AI Safety, Fair Machine Learning, and Human-in-the-Loop Autonomy, reveals varying perceptions of ethical risks associated with technical systems. Fostering user trust can be achieved through algorithmic assurances that clarify AI behavior, enhancing user comprehension and confidence. As the software development landscape evolves with automation tools like GPT-4 for generating user stories and improving coding practices, prioritizing transparency and user-centered design is imperative to address diverse stakeholder needs interacting with AI technologies [91, 28, 34, 65, 7].

As AI-assisted debugging systems become more prevalent, prioritizing human values and aligning AI operations with ethical standards and societal expectations is essential. This alignment is critical in addressing the epistemic power that shapes the definition of software bugs, underscoring the necessity for diverse datasets that capture a wide range of perspectives and experiences.

Collaboration across disciplines is vital to address safety concerns, particularly in immersive technologies where human-AI interaction boundaries blur. This interdisciplinary approach is essential for tackling the complex ethical challenges posed by AI systems, particularly the risk of manipulating human behavior under the guise of achieving a perceived greater good. Misalignments in intent and values between humans and AI can arise, especially when there is an asymmetry in knowledge or computational power. The concept of "Bidirectional Human-AI Alignment" emphasizes ongoing mutual adjustments between humans and AI to ensure ethical collaboration. As AI technologies, such as large language models, become more integrated into various domains, including education, developing frameworks that promote transparency and understanding is crucial, allowing stakeholders to navigate ethical landscapes effectively while maintaining human oversight and critical assessment of AI outputs [92, 16, 91, 28, 60].

Developing AI systems must consider the dynamic interplay between human values and AI capabilities, recognizing the importance of interactive methods for specifying the values guiding AI behavior. To foster user trust, AI systems must incorporate assurance mechanisms that enhance transparency and reliability. "To foster user confidence and trust in AI-assisted debugging tools, it is essential to design mechanisms that clearly communicate the decision-making processes of these AI systems. This includes enhancing transparency through user-centered approaches that not only explain how AI operates but also engage users in adapting and co-designing the AI's internal mechanics. By ensuring that stakeholders understand the rationale behind AI-generated hints and recommendations, we can improve user interaction and support more effective problem-solving in programming tasks, ultimately leading to a more collaborative and trustworthy AI experience" [28, 34, 70]. Addressing challenges related to trust, privacy, and ethical considerations is crucial for the successful integration of large generative models into operating systems and other critical infrastructures. A multifaceted approach to large language model safety, integrating technical, ethical, and governance strategies, is necessary.

Furthermore, understanding the influences of robot errors on teaching dynamics is crucial for improving accuracy and efficiency in human-robot interactions. This understanding is vital for ensuring that AI systems do not inadvertently perpetuate biases or ethical lapses. The development of automated code generation tools like BESTER poses risks of misuse, such as generating malicious code [12]. The survey highlights the need for regulatory frameworks to support transparency in AI technologies and calls for interdisciplinary collaboration to address these challenges.

The ethical implications of AI-assisted debugging systems are critical, as they directly influence user trust—a vital component for effectively adopting and utilizing these technologies. Building user trust involves ensuring that AI systems provide reliable, understandable, and contextually relevant guidance, which enhances the debugging process and fosters a positive human-AI relationship.

This trust is further reinforced through algorithmic assurances that clarify the reasoning behind AI actions, ultimately leading to improved user confidence in the system's capabilities [34, 65, 91]. By prioritizing human values, fostering interdisciplinary collaboration, and implementing robust assurance mechanisms, AI systems can achieve greater alignment with societal norms and expectations, ultimately enhancing their acceptance and effectiveness in software development processes.

6.4 Ethical and Societal Implications

The deployment of AI-assisted debugging technologies introduces a myriad of ethical and societal implications that warrant careful consideration and proactive management. As these systems become increasingly integrated into critical environments, such as telecommunications and military applications, the need for robust ethical guidelines and assurance frameworks becomes paramount. Formulating comprehensive guidelines is essential to ensure that emerging technologies, particularly automated machine learning systems, are utilized in ways that uphold user autonomy and align with societal norms. This is especially critical in scenarios where the risk of misuse or unintended consequences is high, as these guidelines can help navigate the complexities of human-computer interaction, mitigate biases, and foster trust in AI systems. By prioritizing user perspectives and addressing ethical considerations, such guidelines can facilitate the responsible integration of advanced technologies into various domains, ultimately promoting a safer and more equitable technological landscape [6, 93, 94, 91].

A key ethical consideration in deploying AI-assisted debugging systems is balancing automation and human oversight. While these technologies offer significant efficiencies, they must be designed to benefit users without infringing on their autonomy or decision-making capabilities [94]. This requires implementing ethical guidelines that govern the design and operation of AI systems, ensuring they are transparent, explainable, and aligned with user values.

The societal implications of AI-assisted debugging technologies extend to issues of trust and transparency. As these systems are increasingly deployed in complex engineering problems, developing assurance frameworks that account for user variability and contextual factors is essential [65]. Such frameworks should provide clear insights into the decision-making processes of AI systems, fostering user trust and confidence in their reliability and effectiveness.

The broader deployment of AI in telecommunications and other critical sectors raises questions about the potential societal impacts of these technologies. Future research should explore the implications of AI systems on various stakeholders, including the potential for job displacement and the need for reskilling in the workforce [56]. By addressing these challenges, researchers and practitioners can ensure that AI-assisted debugging technologies contribute positively to society, enhancing productivity and innovation while safeguarding human values and ethical standards.

The deployment of AI-assisted debugging technologies presents significant ethical and societal implications, underscoring the necessity for a multifaceted approach that harmonizes technical, ethical, and societal dimensions. This approach should draw from insights across various AI research subfields, such as AI Safety, Fair Machine Learning, and Human-in-the-Loop Autonomy, which have historically grappled with integrating technical systems into normative social contexts. Furthermore, prioritizing transparency and human-centered design in these technologies is essential to ensure they align with diverse stakeholder needs and promote responsible AI use. By addressing the complexities of human-AI collaboration and the challenges of automation, we can better navigate the ethical landscape and foster innovations that benefit society at large [92, 91, 28, 21, 6]. By fostering interdisciplinary collaboration and developing robust assurance frameworks, stakeholders can navigate the complexities of AI deployment, ensuring that these technologies are used responsibly and effectively in diverse contexts.

7 Conclusion

The convergence of large language models (LLMs) and human-computer interaction (HCI) principles brings transformative improvements to AI-assisted debugging. Models like CodeT5-DLR have demonstrated enhanced capabilities in bug detection, localization, and program repair, significantly boosting the efficiency and precision of debugging tasks. Autonomous systems such as GITAGENT

showcase the potential of leveraging platforms like GitHub to autonomously address user queries, streamlining software development processes. The collaboration between AI and HCI communities is pivotal in crafting human-centered AI systems that are not only effective and ethical but also user-friendly. This partnership enhances communication between users and AI, leading to better interaction designs and heightened user satisfaction. Incorporating human cognitive capabilities into machine learning processes is essential to reinforce the reliability and trustworthiness of intelligent systems. The automation potential of LLMs in graphical user interfaces (GUIs) highlights the need for continuous research to bridge existing gaps and expand AI capabilities across diverse applications. Developing aligned and secure models is crucial for addressing safety challenges in generative AI-LLMs, ensuring their effectiveness and safe deployment. This survey underscores the promise of Human-In-The-Loop (HITL) systems in refining model accuracy, interpretability, and user trust. The success of tools like HypoCompass in enhancing educational outcomes exemplifies the practical benefits of AI integration. The importance of local explanations for debugging, the necessity of iterative feedback loops, and the substantial impact of human feedback on model performance are emphasized, despite the inherent challenges.

References

[1] Upol Ehsan, Pradyumna Tambwekar, Larry Chan, Brent Harrison, and Mark Riedl. Automated rationale generation: A technique for explainable ai and its effects on human perceptions, 2019.

[2] Vinicius G. Goecks, Gregory M. Gremillion, Vernon J. Lawhern, John Valasek, and Nicholas R. Waytowich. Efficiently combining human demonstrations and interventions for safe training of autonomous systems in real-time, 2018.

[3] Qing Huang, Jiahui Zhu, Zhenchang Xing, Huan Jin, Changjing Wang, and Xiwei Xu. A chain of ai-based solutions for resolving fqns and fixing syntax errors in partial code, 2023.

[4] Qun Ma, Xiao Xue, Deyu Zhou, Xiangning Yu, Donghua Liu, Xuwen Zhang, Zihan Zhao, Yifan Shen, Peilin Ji, Juanjuan Li, Gang Wang, and Wanpeng Ma. Computational experiments meet large language model based agents: A survey and perspective, 2024.

[5] Yousef Emami, Luis Almeida, Kai Li, Wei Ni, and Zhu Han. Human-in-the-loop machine learning for safe and ethical autonomous vehicles: Principles, challenges, and opportunities, 2024.

[6] Carlos Toxtli. Human-centered automation, 2024.

[7] Tajmilur Rahman and Yuecai Zhu. Automated user story generation with test case specification using large language model, 2024.

[8] Cumhur Erkan Tuncali, Georgios Fainekos, Danil Prokhorov, Hisahiro Ito, and James Kapinski. Requirements-driven test generation for autonomous vehicles with machine learning components, 2019.

[9] Jialu Zhang, José Cambronero, Sumit Gulwani, Vu Le, Ruzica Piskac, Gustavo Soares, and Gust Verbruggen. Repairing bugs in python assignments using large language models, 2022.

[10] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents, 2024.

[11] Nathalia Nascimento, Paulo Alencar, and Donald Cowan. Comparing software developers with chatgpt: An empirical investigation, 2023.

[12] Jialin Song, Jonathan Raiman, and Bryan Catanzaro. Effective large language model debugging with best-first tree search, 2024.

[13] Xunzhu Tang, Kisub Kim, Yewei Song, Cedric Lothritz, Bei Li, Saad Ezzini, Haoye Tian, Jacques Klein, and Tegawende F. Bissyande. Codeagent: Autonomous communicative agents for code review, 2024.

[14] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. Repairagent: An autonomous, llm-based agent for program repair, 2024.

[15] Qing Huang, Zhou Zou, Zhenchang Xing, Zhenkang Zuo, Xiwei Xu, and Qinghua Lu. Ai chain on large language model for unsupervised control flow graph generation for statically-typed partial code, 2023.

[16] Ethan Mollick and Lilach Mollick. Assigning ai: Seven approaches for students, with prompts, 2023.

[17] Rahul Gupta, Aditya Kanade, and Shirish Shevade. Deep learning for bug-localization in student programs, 2019.

[18] Zhenchang Xing, Qing Huang, Yu Cheng, Liming Zhu, Qinghua Lu, and Xiwei Xu. Prompt sapper: Llm-empowered software engineering infrastructure for ai-native services, 2023.

[19] Repair is nearly generation: Multilingual program repair with llms.

[20] Xueyu Hu, Kun Kuang, Jiankai Sun, Hongxia Yang, and Fei Wu. Leveraging print debugging to improve code generation in large language models, 2024.

[21] Piyawat Lertvittayakumjorn and Francesca Toni. Explanation-based human debugging of nlp models: A survey, 2021.

[22] Jiayin Wang, Weizhi Ma, Peijie Sun, Min Zhang, and Jian-Yun Nie. Understanding user experience in large language model interactions, 2024.

[23] Zijie J. Wang, Dongjin Choi, Shenyu Xu, and Diyi Yang. Putting humans in the natural language processing loop: A survey, 2021.

[24] Li Zhong, Zilong Wang, and Jingbo Shang. Debug like a human: A large language model debugger via verifying runtime execution step-by-step, 2024.

[25] Brandon J. McMahan, Zhenghao Peng, Bolei Zhou, and Jonathan C. Kao. Shared autonomy with ida: Interventional diffusion assistance, 2024.

[26] Hasnain Heickal and Andrew Lan. Generating feedback-ladders for logical errors in programming using large language models, 2024.

[27] Priyanka Kargupta, Ishika Agarwal, Dilek Hakkani-Tur, and Jiawei Han. Instruct, not assist: Llm-based multi-turn planning and hierarchical questioning for socratic code debugging, 2024.

[28] Q. Vera Liao and Jennifer Wortman Vaughan. Ai transparency in the age of llms: A human-centered research roadmap, 2023.

[29] Jindan Huang, Isaac Sheidlower, Reuben M. Aronson, and Elaine Schaertl Short. On the effect of robot errors on human teaching dynamics, 2024.

[30] Manish Motwani. High-quality automated program repair, 2021.

[31] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models, 2024.

[32] Qianou Ma, Hua Shen, Kenneth Koedinger, and Tongshuang Wu. How to teach programming in the ai era? using llms as a teachable agent for debugging, 2024.

[33] Martin Monperrus. A critical review of "automatic patch generation learned from human-written patches": Essay on the problem statement and the evaluation of automatic software repair, 2014.

[34] Laryn Qi, J. D. Zamfirescu-Pereira, Taehan Kim, Björn Hartmann, John DeNero, and Narges Norouzi. A knowledge-component-based methodology for evaluating ai assistants, 2024.

[35] Jiuang Zhao, Donghao Yang, Li Zhang, Xiaoli Lian, Zitian Yang, and Fang Liu. Enhancing automated program repair with solution design, 2024.

[36] Stefania Druga and Nancy Otero. Scratch copilot evaluation: Assessing ai-assisted creative coding for families, 2023.

[37] Anton Cheshkov, Pavel Zadorozhny, Rodion Levichev, Evgeny Maslov, and Ronaldo Franco Jaldin. Exploring the potential of conversational test suite based program repair on swe-bench, 2024.

[38] Jane Cleland-Huang, Ankit Agrawal, Michael Vierhauser, Michael Murphy, and Mike Prieto. Extending mape-k to support human-machine teaming, 2022.

[39] Manuel Mosquera, Juan Sebastian Pinzon, Manuel Rios, Yesid Fonseca, Luis Felipe Giraldo, Nicanor Quijano, and Ruben Manrique. Can llm-augmented autonomous agents cooperate?, an evaluation of their cooperative capabilities through melting pot, 2024.

[40] Dan Shi, Tianhao Shen, Yufei Huang, Zhigen Li, Yongqi Leng, Renren Jin, Chuang Liu, Xinwei Wu, Zishan Guo, Linhao Yu, Ling Shi, Bojian Jiang, and Deyi Xiong. Large language model safety: A holistic survey, 2024.

[41] Timothy R. McIntosh, Teo Susnjak, Nalin Arachchilage, Tong Liu, Paul Watters, and Malka N. Halgamuge. Inadequacies of large language model benchmarks in the era of generative artificial intelligence, 2024.

[42] Chao Lei, Yanchuan Chang, Nir Lipovetzky, and Krista A. Ehinger. Planning-driven programming: A large language model programming workflow, 2025.

[43] Qiong Feng, Xiaotian Ma, Jiayi Sheng, Ziyuan Feng, Wei Song, and Peng Liang. Integrating various software artifacts for better llm-based bug localization and program repair, 2024.

[44] Lincoln Murr, Morgan Grainger, and David Gao. Testing llms on code generation with varying levels of prompt specificity, 2023.

[45] Nghi D. Q. Bui, Yue Wang, and Steven Hoi. Detect-localize-repair: A unified framework for learning to debug with codet5, 2022.

[46] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug, 2023.

[47] Marcel Böhme, Charaka Geethal, and Van-Thuan Pham. Human-in-the-loop automatic program repair, 2019.

[48] Fernando Vallecillos Ruiz, Anastasiia Grishina, Max Hort, and Leon Moonen. A novel approach for automatic program repair using round-trip translation with large language models, 2024.

[49] Mohannad Alhanahnah, Md Rashedul Hasan, and Hamid Bagheri. An empirical evaluation of pre-trained large language models for repairing declarative formal specifications, 2024.

[50] Nan Jiang, Xiaopeng Li, Shiqi Wang, Qiang Zhou, Soneya Binta Hossain, Baishakhi Ray, Varun Kumar, Xiaofei Ma, and Anoop Deoras. Training llms to better self-debug and explain code, 2024.

[51] Bao-Gang Hu and Wei-Ming Dong. A design of human-like robust ai machines in object identification, 2021.

[52] Laura Fernández-Becerra, Miguel Ángel González-Santamarta, Ángel Manuel Guerrero-Higueras, Francisco Javier Rodríguez-Lera, and Vicente Matellán Olivera. Enhancing trust in autonomous agents: An architecture for accountability and explainability through blockchain and large language models, 2024.

[53] Yiding Wang, Yuxuan Chen, Fangwei Zhong, Long Ma, and Yizhou Wang. Simulating human-like daily activities with desire-driven autonomy, 2024.

[54] Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human, 2024.

[55] Xianlong Zeng, Fanghao Song, Zhongen Li, Krerkkiat Chusap, and Chang Liu. Human-in-the-loop model explanation via verbatim boundary identification in generated neighborhoods, 2021.

[56] Yanhu Wang, Muhammad Muzammil Afzal, Zhengyang Li, Jie Zhou, Chenyuan Feng, Shuaishuai Guo, and Tony Q. S. Quek. Large language models for base station siting: Intelligent deployment based on prompt or agent, 2024.

[57] Ishaan Shah, David Halpern, Kavosh Asadi, and Michael L. Littman. Convergence of a human-in-the-loop policy-gradient algorithm with eligibility trace under reward, policy, and advantage feedback, 2021.

[58] Piyawat Lertvittayakumjorn, Lucia Specia, and Francesca Toni. Find: Human-in-the-loop debugging deep text classifiers, 2020.

[59] Jiwei Li, Alexander H. Miller, Sumit Chopra, Marc'Aurelio Ranzato, and Jason Weston. Dialogue learning with human-in-the-loop, 2017.

[60] Hua Shen, Tiffany Knearem, Reshmi Ghosh, Kenan Alkiek, Kundan Krishna, Yachuan Liu, Ziqiao Ma, Savvas Petridis, Yi-Hao Peng, Li Qiwei, Sushrita Rakshit, Chenglei Si, Yutong Xie, Jeffrey P. Bigham, Frank Bentley, Joyce Chai, Zachary Lipton, Qiaozhu Mei, Rada Mihalcea, Michael Terry, Diyi Yang, Meredith Ringel Morris, Paul Resnick, and David Jurgens. Towards bidirectional human-ai alignment: A systematic review for clarifications, framework, and future directions, 2024.

[61] Kostas Tsiakas and Dave Murray-Rust. Unpacking human-ai interactions: From interaction primitives to a design space, 2024.

[62] Mark Bercasio, Allison Wong, and Dustin Dannenhauer. Human in the loop novelty generation, 2023.

[63] Ruikun Luo, Na Du, and X. Jessie Yang. Enhancing autonomy transparency: an option-centric rationale approach, 2020.

[64] AI Redefined, Sai Krishna Gottipati, Sagar Kurandwad, Clodéric Mars, Gregory Szriftgiser, and François Chabot. Cogment: Open source framework for distributed multi-actor training, deployment operations, 2021.

[65] Brett W Israelsen and Nisar R Ahmed. "dave...i can assure you...that it's going to be all right..." – a definition, case for, and survey of algorithmic assurances in human-autonomy trust relationships, 2018.

[66] Donald R. Honeycutt, Mahsan Nourani, and Eric D. Ragan. Soliciting human-in-the-loop user feedback for interactive machine learning reduces user trust and impressions of model accuracy, 2020.

[67] Chaehan So. Human-in-the-loop design cycles – a process framework that integrates design sprints, agile processes, and machine learning with humans, 2020.

[68] Wangfan Li, Rohit Mallick, Carlos Toxtli-Hernandez, Christopher Flathmann, and Nathan J. McNeese. Leveraging artificial intelligence to promote awareness in augmented reality systems, 2024.

[69] Lujain Ibrahim, Luc Rocher, and Ana Valdivia. Characterizing and modeling harms from interactions with design patterns in ai interfaces, 2024.

[70] Muhammad Raees, Inge Meijerink, Ioanna Lykourentzou, Vassilis-Javed Khan, and Konstantinos Papangelis. From explainable to interactive ai: A literature review on current trends in human-ai interaction, 2024.

[71] Wei Xu and Marvin Dainoff. Enabling human-centered ai: A new junction and shared journey between ai and hci communities, 2023.

[72] Xueyang Feng, Zhi-Yuan Chen, Yujia Qin, Yankai Lin, Xu Chen, Zhiyuan Liu, and Ji-Rong Wen. Large language model-based human-agent collaboration for complex task solving, 2024.

[73] David Abel, John Salvatier, Andreas Stuhlmüller, and Owain Evans. Agent-agnostic human-in-the-loop reinforcement learning, 2017.

[74] Marcel Torne, Max Balsells, Zihan Wang, Samedh Desai, Tao Chen, Pulkit Agrawal, and Abhishek Gupta. Breadcrumbs to the goal: Goal-conditioned exploration from human-in-the-loop feedback, 2023.

[75] Laura Plein, Wendkûuni C. Ouédraogo, Jacques Klein, and Tegawendé F. Bissyandé. Automatic generation of test cases based on bug reports: a feasibility study with large language models, 2023.

[76] Doris Xin, Litian Ma, Jialin Liu, Stephen Macke, Shuchen Song, and Aditya Parameswaran. Accelerating human-in-the-loop machine learning: Challenges and opportunities, 2018.

[77] Faria Huq, Masum Hasan, Mahim Anzum Haque Pantho, Sazan Mahbub, Anindya Iqbal, and Toufique Ahmed. Review4repair: Code review aided automatic program repairing, 2020.

[78] Thomas Hirsch and Birgit Hofer. What we can learn from how programmers debug their code, 2021.

[79] Amr Gomaa and Bilal Mahdy. Unveiling the role of expert guidance: A comparative analysis of user-centered imitation learning and traditional reinforcement learning, 2024.

[80] Yoshiharu Ikutani, Nishanth Koganti, Hideaki Hata, Takatomi Kubo, and Kenichi Matsumoto. Toward imitating visual attention of experts in software development tasks, 2019.

[81] James R. Kirk, Robert E. Wray, Peter Lindes, and John E. Laird. Improving knowledge extraction from llms for task learning through agent analysis, 2024.

[82] Qi Xin, Haojun Wu, Steven P. Reiss, and Jifeng Xuan. Towards practical and useful automated program repair for debugging, 2024.

[83] Fernanda Madeiral and Thomas Durieux. A large-scale study on human-cloned changes for automated program repair, 2021.

[84] Zelin Zhao, Zhaogui Xu, Jialong Zhu, Peng Di, Yuan Yao, and Xiaoxing Ma. The right prompts for the job: Repair code-review defects with large language model, 2023.

[85] Shrestha Mohanty, Negar Arabzadeh, Andrea Tupini, Yuxuan Sun, Alexey Skrynnik, Artem Zholus, Marc-Alexandre Côté, and Julia Kiseleva. Idat: A multi-modal dataset and toolkit for building and evaluating interactive task-solving agents, 2024.

[86] Rudolf Laine, Bilal Chughtai, Jan Betley, Kaivalya Hariharan, Jeremy Scheurer, Mikita Balesni, Marius Hobbhahn, Alexander Meinke, and Owain Evans. Me, myself, and ai: The situational awareness dataset (sad) for llms, 2024.

[87] Thomas Dupriez, Steven Costiou, and Stéphane Ducasse. First infrastructure and experimentation in echo-debugging, 2020.

[88] Manish Motwani and Yuriy Brun. Better automatic program repair by using bug reports and tests together, 2023.

[89] Nishat Raihan, Mohammed Latif Siddiq, Joanna C. S. Santos, and Marcos Zampieri. Large language models in computer science education: A systematic literature review, 2024.

[90] Sergey Titov, Agnia Sergeyuk, and Timofey Bryksin. What writing assistants can learn from programming ides, 2023.

[91] McKane Andrus, Sarah Dean, Thomas Krendl Gilbert, Nathan Lambert, and Tom Zick. Ai development for the public interest: From abstraction traps to sociotechnical risks, 2021.

[92] Tathagata Chakraborti and Subbarao Kambhampati. Algorithms for the greater good! on mental modeling and acceptable symbiosis in human-ai collaboration, 2018.

[93] Thanh Tung Khuat, David Jacob Kedziora, and Bogdan Gabrys. The roles and modes of human interactions with automated machine learning systems, 2022.

[94] Vivek Nallur, Karen Renaud, and Aleksei Gudkov. Nudging using autonomous agents: Risks and ethical considerations, 2024.

Disclaimer:

SurveyX is an AI-powered system designed to automate the generation of surveys. While it aims to produce high-quality, coherent, and comprehensive surveys with accurate citations, the final output is derived from the AI's synthesis of pre-processed materials, which may contain limitations or inaccuracies. As such, the generated content should not be used for academic publication or formal submissions and must be independently reviewed and verified. The developers of SurveyX do not assume responsibility for any errors or consequences arising from the use of the generated surveys.