Research Interest Statement:
Advancing Reliable Autonomous Agents for Software Engineering and Verification

Chenchen Ye

ccye@cs.ucal.edu, Department of Computer Science, UCLA

1 Proposed Research Direction

Software engineering represents one of the most challenging domains for developing **autonomous systems** capable of multi-step reasoning, quality assurance, and self-verification—capabilities that are fundamental to reliable engineering across all disciplines. The development of intelligent agents that can not only solve complex problems but also **verify their own solutions** addresses a critical need in modern engineering: creating trustworthy autonomous systems that can maintain quality standards without constant human oversight.

Software engineering with LLMs has emerged as a critical testbed for next-generation AI agents (Jimenez et al., 2024; Yang et al., 2024), demanding sophisticated integration of natural language understanding, code generation,s reasoning, and iterative problem-solving. **Accurate** and **reliable** software engineering agents are essential for enhancing developer productivity and accelerating software innovation across industries. This research proposes developing a comprehensive framework for intelligent agents that can autonomously perform both **Software Engineering (SWE)** task: generating code patches to fix repository issues, and **Software Engineering Test Generation (SWET)** task: creating high-quality, discriminative test suites for code verification (Wang, 2025). The framework enables these complementary capabilities to continuously improve through co-evolutionary reinforcement learning and self-play mechanisms (Liu et al., 2025).

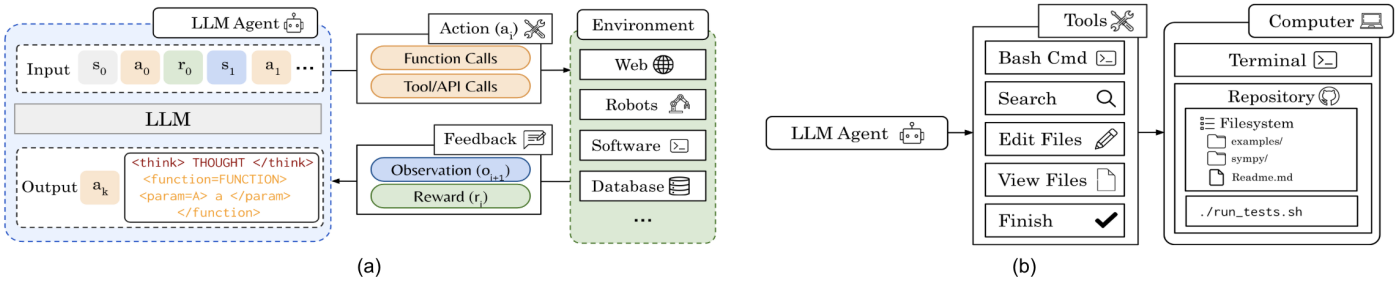

Figure 1: (a) LLM agents generate thought-guided actions, in the form of function or tool calls, to interact with an environment, which returns the next observation and reward. (b) Software Engineering (SWE) Agents.

2 Research Interests and Background

My research focuses on developing intelligent LLM agents capable of solving complex real-world problems, with particular emphasis on **agentic AI systems** that can effectively plan, reason, and execute multi-step tasks. I work at the intersection of machine learning and natural language processing, exploring how LLM autonomous agents can be enhanced through tool use, reasoning, and self-improvement mechanisms. My research also encompasses heterogeneous knowledge integration, leveraging knowledge graphs, text, code, and visual data to enhance model capabilities (Deng et al.; Ye et al., 2022a,b).

Through my previous work, I have developed comprehensive expertise in agentic AI across multiple domains. I created **MIRAI** (Ye et al., 2024), a systematic benchmark for evaluating LLM agents as temporal forecasters through code generation-based tool use and ReAct-style iterative reasoning processes. During my internship at Microsoft Research, I developed **Hierarchical Multi-Agent GraphRAG** (Microsoft, 2024), which advanced complex query answering through dynamic decomposition and collaborative planning among multiple agents. My recent work at Scale AI on **SWE-Level Unit Tests Generation Agents** (Ye et al., 2025) provided direct experience with optimizing agentless pipelines for both patch generation and unit test generation. Additionally, my **AgentGrow** (Cui et al., 2024) framework established scalable, customizable approaches for language agent training in environment simulator, enabling targeted skill acquisition in agent training.

3 Research Goals and Problem Statement

Building upon this foundation in agentic AI, I have identified that current autonomous software engineering agents face critical limitations in **self-verification capabilities**. While these agents can generate code patches for bug fixing, they struggle with creating reliable verification mechanisms that can distinguish between correct and incorrect implementations. Recent analysis in R2E-Gym (Jain et al., 2025; Luo et al., 2025) reveals that execution-based verifiers suffer from **limited distinguishability** (less than 20% of generated tests provide discriminative signal) and **test toxicity** issues (up to 10% of tests erroneously favor incorrect solutions).

These limitations stem from three key challenges: (1) **Absence of comprehensive test generation benchmarks and training data** that systematically evaluate SWET generation and enable scalable SWET training, (2) **Limited reward design and training methodologies for SWET**, with only supervised fine-tuning available with bounded performance, and (3) **Heavy reliance on external training labels** such as human-written unit tests, suffer from limited availability and unclear data distribution.

4 Research Approach and Methodology

Our approach leverages the OpenHands scaffold (Wang et al., 2025a) with **multi-turn reasoning SWE agents** that operate in computer-based environments equipped with terminal access and filesystem navigation capabilities. As shown in Figure 1(b), similar to how human developers interface with IDEs, these agents utilize a comprehensive toolkit including bash execution, code search, and file viewing/editing to navigate codebases and resolve software engineering tasks. The integration of **reasoning** LLMs enhances the agents' ability in dynamic tool selection and self-reflection (Luo et al., 2025). Building upon this foundation, I propose a **three-fold methodology** that leverages AWS's computational infrastructure for large-scale development:

4.1 SWET Benchmark: Software Engineering Test Generation Framework

Building upon recent advances in SWE agent evaluation such asSWE-Bench (Jimenez et al., 2024) and R2E-Gym (Jain et al., 2025), I propose developing **SWET-Bench (Software Engineering Test Generation)**, a comprehensive framework that fills the critical gap in both SWET evaluation and training data. For evaluation, SWET will assess models' ability to generate high-quality test cases with strong distinguishability between good and bad patches by checking if the generated tests aligned with gold tests on various code patches collected from different SWE agents. This approach draws inspiration from RewardBench (Lambert et al., 2024), featuring test generation tasks where models must create tests capable of distinguishing between subtly different code implementations with clear correctness judgment. Beyond evaluation, SWET-Bench can also establish a training data generation pipeline that converts existing SWE training datasets into SWET training data. This dual-purpose framework addresses the fundamental gap in both assessing and training test generation capabilities.

4.2 Verifiable Reward Reinforcement Learning for Test Generation

Current test generation models rely exclusively on **supervised fine-tuning (SFT)** (Xia et al., 2024), which suffers from inability to handle out-of-distribution problems and bounded performance ceilings. I propose a novel **verifiable reward reinforcement learning** framework specifically designed to address these challenges through systematic reward mechanisms (Shao et al., 2024). The approach will leverage the SWET training data generation pipeline to create diverse code patch candidates for each repository issue. The RL reward signal will be computed by evaluating generated tests against multiple diverse code patches, providing positive rewards for tests that align with gold-standard test behaviors in terms of (1) **reproduction capability**—effectively reproducing reported bugs, and (2) **distinguishability**—successfully differentiating between correct and incorrect patches. This RL approach **learn how to learn**, incentivizing the development of discriminative test generation strategies rather than simply memorizing expert patterns.

4.3 Co-Evolutionary Self-Play for Enhanced Software Engineering and Test Generation

Drawing inspiration from recent advances in co-evolutionary training frameworks (Liu et al., 2025; Wang et al., 2025b), I propose a novel **co-evolutionary self-play architecture** where test generation and code generation capabilities evolve together through continuous adversarial and collaborative interactions. The framework will implement a **two-agent zero-sum game** where a single model alternates between test generator and code generator roles, with adversarial interactions designed to expose weaknesses and collaborative interactions aimed at solving complex software engineering problems. The training process will construct a pairwise reward matrix based on interactions between generated code and generated tests, enabling mutual supervision through interaction outcomes rather than external labels. This continuous co-adaptation ensures that **both SWE and SWET capabilities evolve together**.

5 Technical Implementation and AWS Integration

This research requires computational resources for large language modeling training scaling: (1) **GPU-intensive training** for verifiable reward RL and co-evolutionary self-play, and (2) **CPU-intensive environment scaling** for parallelizing Docker containers during SWE and SWET rollouts.

Overall, these research questions would advance academic understanding of agentic AI capabilities and self-play RL training methodologies, while providing practical open-source tools that enhance software reliability, improve development workflows through better code verification, and contribute to more trustworthy autonomous systems across engineering applications.

References

Yuedong Cui, Yiming Wang, Da Yin, Di Wu, Chenchen Ye, Zongyu Lin, Xueqing Wu, and Chang Kai-Wei. 2024. AgentGrow: LLMs as Scalable, Customizable General-Purpose Simulators For Language Agent Training.

Yihe Deng, Chenchen Ye, Zijie Huang, Mingyu Derek Ma, Yiwen Kou, and Wei Wang. GraphVis: Boosting LLMs with Visual Knowledge Graph Integration.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. R2E-Gym: Procedural Environments and Hybrid Verifiers for Scaling Open-Weights SWE Agents.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? ArXiv:2310.06770 [cs].

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, L. J. Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. 2024. RewardBench: Evaluating Reward Models for Language Modeling.

Mickel Liu, Liwei Jiang, Yancheng Liang, Simon Shaolei Du, Yejin Choi, Tim Althoff, and Natasha Jaques. 2025. Chasing Moving Targets with Online Self-Play Reinforcement Learning for Safer Language Models.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Colin Cai, Tarun Venkat, Manan Roongta, Li Erran Li, Raluca Ada Popa, Koushik Sen, Ion Stoica, Ameen Patel, Qingyang Wu, Alpay Ariyak, Shang Zhu, Ben Athiwaratkun, and Ce Zhang. 2025. DeepSWE: Training a Fully Open-sourced, State-of-the-Art Coding Agent by Scaling RL.

Microsoft. 2024. GraphRAG.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models.

Xingyao Wang. 2025. SOTA on SWE-Bench Verified with Inference-Time Scaling and Critic Model.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. 2025a. OpenHands: An Open Platform for AI Software Developers as Generalist Agents.

Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. 2025b. Co-Evolving LLM Coder and Unit Tester via Reinforcement Learning.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2024. Agentless: Demystifying LLM-based Software Engineering Agents.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering.

Chenchen Ye, Jeff Da, Sean Hendryx, and Brad Kenstler. 2025. SWE-Level Unit Tests Generation Agents.

Chenchen Ye, Ziniu Hu, Yihe Deng, Zijie Huang, Mingyu Derek Ma, Yanqiao Zhu, and Wei Wang. 2024. MIRAI: Evaluating LLM Agents for Event Forecasting.

Chenchen Ye, Lizi Liao, Fuli Feng, Wei Ji, and Tat-Seng Chua. 2022a. Structured and Natural Responses Co-generation for Conversational Search. ACM SIGIR.

Chenchen Ye, Lizi Liao, Suyu Liu, and Tat-Seng Chua. 2022b. Reflecting on Experiences for Response Generation. ACM MM.