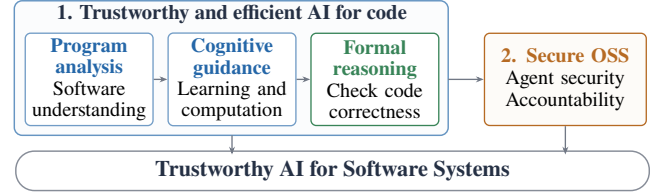


Trustworthy AI for Software Systems

Research Statement — Yueke Zhang

Research Overview. AI can help developers understand code, generate changes, and maintain software. Making this assistance trustworthy requires more than producing a plausible solution. Models must use relevant program information, learn efficiently, and satisfy the requirements of the task. Agents must also act within the authority granted by the developer and the project. These responsibilities become closely connected when agents retain information and revise their strategies, because lessons from one task can shape later decisions. My research studies **how learning, reasoning, and security can work together to support reliable AI participation in software development**.



My research develops trustworthy AI for software systems. It connects two themes: **trustworthy and efficient AI for code** and **secure open-source systems**. In the first, program analysis provides information about software behavior; cognitive modeling and neural measurements guide which information shapes learning and where adaptation occurs; and ideas from formal reasoning support checking and correcting generated solutions. In the second, my impersonation and authorship studies examine how code and repository history inform trust in a contribution. Together, these themes connect reliable software behavior with trust in a contribution’s origin and context.

My goal is to make complex technical work easier to delegate while preserving human control over the goal, the process, and the result. I will pursue **AI that improves with experience while preserving required behavior and respecting limits on its authority**. Progress must be assessed through task fulfillment, computational cost, and the effort people need to inspect and correct the work. Software is a practical foundation because its structure, execution, and history make actions and outcomes observable. This foundation will support reusable memory, adaptation, and verification infrastructure, with applications to web interaction, scientific analysis, and other technical domains.

Recognition. My research spans software engineering, security, cognitive modeling, and AI for code, with publications in **ICSE, FSE, ACL, CSCW, ACM TOSEM, and TMLR**. My record includes my first-author paper **Unclaimed Territory (ICSE 2027)**, GitHub impersonation and authorship studies, **ScanCoder**, **EyeMulator**, **ContraBin**, and open-source software for social good. Other publications include **CodeACT-R (ASE NIER)**, **Stop the Retrieval Thrash (FSE Companion)**, work at **ESEM and EASE**, **VERITAS (ICML AI4Math workshop)**, and the **GECCO Genetic Improvement workshop**.

Research Contributions

(1) Developing Trustworthy and Efficient AI for Code. My first theme connects three foundations of software AI: understanding program behavior, organizing learning and computation, and reasoning about correctness. These foundations support models that can interpret complex software, adapt effectively, and produce solutions whose behavior can be assessed against explicit requirements.

(a) Program Analysis for Reliable Software Understanding. Program analysis makes information about software behavior available for diagnosis and repair. In **fault localization**, I developed **a method that accounts for uncertainty when combining complementary testing-based rankings** [3]. To assess whether analysis results guide a repair, my **static-analyzer study evaluated warning locations against the code developers changed** [4]. Using five analyzers and vulnerabilities and patches from 243 open-source projects, I measured warning-to-patch distances. These studies provide a foundation for assessing the program information supplied to AI, including whether it helps address the underlying software problem.

(b) Cognition-Informed Learning and Efficient AI. Program structure describes what code contains; developer cognition offers guidance about how its information is processed. I developed **CodeACT-R** with my collaborators as **a reusable computational model of human attention during code reading** [5]. Informed by data from 48 programmers, it uses program structure and reading strategies within the ACT-R cognitive architecture to generate simulated attention sequences.

ScanCoder, the central methodological contribution of this line, turns simulated reading patterns into **scalable guidance for training code models** [6]. I co-developed a framework that weights token-level training by attention coverage and reading order, without collecting gaze traces for every example or changing the model architecture. Across four models with one to eight billion parameters, guidance from C++ reading behavior improved Java code completion and summarization by up to 41.66 CrystalBLEU points and 21.93 BERTScore points, respectively. In a related collaboration, I contributed to **EyeMulator**, which uses statistical gaze patterns to guide code-model training [7].

ScanCoder guides what a model learns from; my new **CogAdapt** manuscript extends cognitive guidance to **where learning occurs inside the model** [8]. I use EEG and gaze measurements to guide which transformer blocks are adapted for each task in mixture-of-experts code models. On LiveCodeBench, CogAdapt improves pass@1 by 10.86 and 6.29 percentage points for Qwen and GLM over matched all-block fine-tuning, while reducing the adaptation parameters updated per example by 86–87%. The full backbone still runs at inference, without EEG or gaze input. An arXiv posting is forthcoming.

An agent also needs to carry useful information from one step to the next. In **Stop the Retrieval Thrash**, I proposed **brain-guided episodic memory that organizes repository work into experiences an agent can retain and retrieve** [9]. This early proposal extends cognitive guidance from attention and adaptation to memory across a longer task.

(c) *Formal Reasoning and Software Verification.* Cognitive guidance shapes learning and adaptation; reliable software also requires explicit reasoning about what its behavior must satisfy. I use structured feedback and ideas from formal reasoning to guide generation and repair. **CodeGrad** provides **structured feedback that connects a missing requirement to a targeted revision** [10]. I organize feedback around correctness, input and output behavior, efficiency, and completeness, then use it to direct iterative refinement.

To explain why a solution misses a requirement, **SCOPE** uses **a prover-initialized critic that turns implicit requirements into actionable subgoal critiques** [11]. The critic identifies subgoals, explains implementation gaps, and constructs a robustness checklist. It learns from both feedback quality and the effect of its feedback on the revised program’s execution results. SCOPE achieved 39.4% pass@1 on LiveCodeBench V6, compared with 36.6% for Reflexion, and 42.6% on BigCodeBench Hard, compared with 36.5%.

Feedback helps revise a solution; I also ask how a model can learn these relationships before it writes one. **GraphAlign-Coder** provides **structural supervision that aligns program regions with proof-related obligations** [12]. Together, these projects connect feedback, subgoal reasoning, and program structure to intended behavior. Formal structure guides learning and revision; establishing correctness still requires appropriate specifications and independent checks. This foundation motivates agents that maintain requirements throughout a complete task.

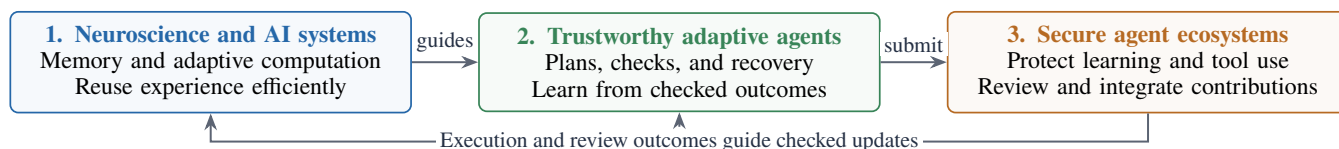
(2) **Securing Open-Source Systems through Contribution Accountability.** Producing the requested change leads to the next question: should an agent’s contribution be accepted by the project? An agent can act under a legitimate account while being redirected by malicious repository content, exposing private information or making changes the maintainer never requested [13]. My second theme examines how to connect contributions to their origin and to the authority behind them.

A contributor’s identity helps maintainers decide how to review and accept a change. My **ICSE study of GitHub impersonation characterized the risks hidden in repository attribution practices** [1]. Combining interviews with 17 open-source contributors and an analysis of 12.5 million commits, I found limited awareness of impersonation risks and showed that commit records can reflect both legitimate collaboration and misleading attribution. A contribution associated with a familiar identity therefore still requires contextual interpretation.

Building on this empirical study, I developed **automated authorship attribution: a method that combines code representations with commit context to guide authorship review** [2]. The approach identifies contributions that warrant closer inspection and evaluates attribution as a probabilistic signal across repository and contributor settings. In **Unclaimed Territory (ICSE 2027)**, I combined repository history with DNS and HTTP checks to confirm 284 vulnerable GitHub Pages subdomains and studied domain trust with 288 participants [14]. These studies motivate checking both contribution provenance and project infrastructure.

Future Research

My future lab will study **how AI agents can learn from experience while remaining reliable, efficient, and under human control**. I will connect three directions: neuroscience-informed systems that retain and reuse experience; trustworthy agents that check their work and improve their strategies; and security mechanisms that govern learning and deployment. Software engineering provides a foundation for this agenda, with extensions to web interaction and scientific workflows. The progression remains connected: experience supports better decisions, checked decisions lead to contributions, and execution and review outcomes improve subsequent behavior.



(1) Neuroscience-Informed Systems for Adaptive AI. I will turn insights from human attention and memory into **reusable systems for adaptive AI**. ScanCoder guides which information receives learning emphasis, CogAdapt selects where adaptation occurs, and Stop the Retrieval Thrash organizes experience across tasks [8, 9]. I will extend these foundations to memory consolidation, selective replay, and computation scheduling: deciding what to retain, which experiences to revisit, and when deeper reasoning is worth its cost. EEG and gaze will provide training-time guidance alongside execution traces, without requiring neural measurements at runtime. A shared runtime will coordinate retrieval, task-dependent adapters, and inference-time computation under explicit memory and latency budgets. I will study how stale context and conflicting experiences affect adaptation. Comparisons with trace-only designs under matched resources will measure task success, forgetting, latency, and energy use across software, web, and scientific workflows. The goal is infrastructure that **helps agents reuse experience efficiently as tasks and environments change**.

(2) Trustworthy Self-Improvement in AI Agents. With this infrastructure, I will develop **agents that learn from success and failure while preserving user intent**. Building on CodeGrad, SCOPE, and GraphAlignCoder, I will connect each plan and revision to the requirements it must satisfy. Independent checks will assess execution results and guide decisions to continue, replan, or recover. A scientific-analysis agent, for example, should preserve the specified study population while revising its methods. I will use checked task trajectories to improve planning strategies, reusable skills, and critique routines. The central challenge is to distinguish a reusable improvement from one that exploits an evaluation rule or fails when the task changes. Candidate updates will be tested in isolated environments on held-out tasks before adoption, including checks for unintended behavior and loss of previously learned capabilities. Infrastructure from the first direction will retain useful lessons and support selective adaptation; security mechanisms from the third will govern which updates are permitted. Across software, web, and scientific tasks, comparisons with agents that do not update their strategies, or only reflect within a task, will measure goal fulfillment, recovery, and the reliability of continued learning under matched tool and computation budgets.

(3) Securing Adaptive Agents and Open-Source Ecosystems. Adaptive agents must remain secure both while learning and when their work enters shared systems. Building on my impersonation and authorship-attribution research, I will study **how trust can be maintained across an agent’s memory, updates, tool actions, and contributions**. Malicious repository content could redirect an action or be retained as a misleading lesson; compromised feedback could also steer subsequent self-improvement. I will connect retained experiences and consequential actions to their source and authorized purpose, distinguish observations from instructions, and restrict which components an agent may modify. These controls will keep task definitions and security checks outside the self-improvement loop. Open-source repositories provide a concrete setting for studying submission, review, dependency changes, and coordination among human and AI contributors [13]. Controlled attack-and-recovery studies will measure immediate compromise, persistence across tasks, recovery cost, and legitimate task completion. Connecting memory from the first direction with requirement checks from the second will support **AI systems that learn and contribute without allowing untrusted content to rewrite their goals or authority**.

Selected References

- [1] Yueke Zhang et al. Who’s Pushing the Code? An Exploration of GitHub Impersonation. *ICSE*, 2025.
- [2] Yueke Zhang et al. Understanding and Detecting GitHub Impersonation through Automated Authorship Attribution. *ACM Transactions on Software Engineering and Methodology*, 2026.
- [3] Yueke Zhang, Kevin Leach, and Yu Huang. Leveraging Evidence Theory to Improve Fault Localization: An Exploratory Study. *ESEM*, 2023.
- [4] Yueke Zhang et al. Locating Software Vulnerabilities With Static Analyzers: How Far Are We? *EASE*, 2026.
- [5] Yueke Zhang et al. CodeACT-R: A Cognitive Simulation Framework for Human Attention in Code Reading. *ASE, New Ideas and Emerging Results*, 2025.
- [6] Yueke Zhang, Yifan Zhang, et al. ScanCoder: Leveraging Human Attention Patterns to Enhance LLMs for Code. *Proceedings of the ACM on Software Engineering (FSE)*, 2026. Equal first authors.
- [7] Yifan Zhang, Chen Huang, Yueke Zhang, et al. EyeMulator: Improving Code Language Models by Mimicking Human Visual Attention. *ACL*, 2026.
- [8] Yueke Zhang, Zihan Fang, Kevin Leach, and Yu Huang. CogAdapt: Cognition-informed Sparse Adaptation of Code LLMs. Manuscript, 2026; arXiv posting forthcoming.
- [9] Yueke Zhang, Zihan Fang, and Kevin Leach. Stop the Retrieval Thrash: Brain-Guided Episodic Memory for Repository-Scale Coding Agents. *FSE Companion*, 2026, pp. 1793–1794.
- [10] Yueke Zhang et al. CodeGrad: Integrating Multi-Step Verification with Gradient-Based LLM Refinement. *arXiv:2508.10059*, 2025. Preprint.
- [11] Yueke Zhang et al. SCOPE: Leveraging Subgoal Critiques for Code Generation. *arXiv:2607.05810*, 2026. Preprint.
- [12] Yueke Zhang et al. GraphAlignCoder: Aligning Program and Proof Graphs for Code Generation. *arXiv:2608.11394*, 2026. Preprint.
- [13] GitHub. GitHub Agentic Workflows and Security Architecture. *GitHub Blog*, 2026.
- [14] Yueke Zhang, Zihan Fang, Yuxin Hu, Kevin Leach, and Yu Huang. Unclaimed Territory: Analyzing Subdomain Takeover Risks in GitHub Pages. (*ICSE* 2027).