Research Statement

Xin Zhang

Overview

My research goal is to fundamentally advance the foundations of program analysis. Over the past five years, I have primarily focused on pioneering Bayesian program analysis, a novel paradigm that embeds probabilistic reasoning within conventional logical analysis. By quantifying uncertainty, this framework empowers analyzers to learn and adapt dynamically. To fully realize this vision, my contributions span the entire research stack: establishing theoretical foundations, designing core algorithms, and building practical applications.

Beyond Bayesian program analysis, my research footprint extends to optimizing domain-specific programming languages (DSLs) to drastically accelerate program synthesis, designing program analyses for explainable artificial intelligence (AI) systems, and abstraction selection for conventional program analysis.

Bayesian Program Analysis

Motivation. While symbolic artificial intelligence has largely been eclipsed by statistical methods, symbolism remains the dominant paradigm in program analysis. This is fundamentally because programs—unlike the noisy, continuous domains of computer vision or natural language processing—are engineered artifacts with precise, unambiguous semantics. Logic-based methods have therefore prevailed, prized for their capacity to encode domain expertise, guarantee explainability, and deliver formal verification. Yet, as modern software systems grow in complexity, this purely logical paradigm has revealed critical limitations in handling uncertainty. Today's software analysis is fraught with unknowns, from partial codebase availability and ambiguous specifications to highly dynamic execution environments. Traditional logic is simply too rigid to model these shifting realities. To bridge this gap, my Ph.D. thesis [Thesis] pioneered the synthesis of logical deduction and probabilistic inference, establishing the foundation of Bayesian program analysis. By embedding probability into classical logic-based frameworks, this paradigm empowers program analyzers to rigorously quantify uncertainty, handle incomplete information, and dynamically evolve by learning from posterior observations. From the user's perspective, Bayesian program analysis enables tools to learn and adapt by continuously updating the confidence levels of results. This adaptability facilitates practical applications such as alarm ranking and automated filtering.

My Contributions. Translating the vision of my Ph.D. thesis into reality, my research over the past few years has comprehensively advanced the applications [POPL'26a, OOPSLA'25a, TSE'25], algorithms [OOPSLA'26, OOPSLA'25b, OOPSLA'24a, ASE'25, ICSE'22], and foundational theories [PLDI'26] of Bayesian program analysis. Specifically, I have expanded the integration of logical and probabilistic reasoning beyond traditional program analysis into critical adjacent domains, including software testing and fault localization. To drive these practical applications, I engineered novel abstraction selection and posterior information acquisition algorithms that maximize learning efficacy, alongside optimized inference algorithms that guarantee analysis scalability and computationally reduce intractable exponential problems to linear time. Finally, I recently established a rigorous theoretical framework to formalize uncertainty quantification and dynamic learning within abstract-interpretation-based analysis. Crucially, these foundational advancements translate into tangible, real-world security improvements: my tools have automatically identified 39 previously unknown vulnerabilities in well-tested open-source software, resulting in 30 newly assigned CVEs. I detail these three pillars of my research on Bayesian program analysis below.

Applications

Program Analysis. To demonstrate the practical impact of Bayesian program analysis, I developed a neural-symbolic framework that systematically incorporates informal, noisy information into rigorous logical analyses [OOPSLA'25a]. Traditional logic-based methods often discard informal clues—such as variable names or string constants—because they are noisy

and offer no formal guarantees. My work overcomes this by leveraging neural networks to evaluate these hints and encoding their outputs as "soft evidences" within a probabilistic model. Following the Bayesian scheme, the analysis produces a list of alarms that are ranked by their probabilities. The logic part guarantees the list does not miss real bugs while the probabilistic part biased the ranking by factoring in the informal information. Moreover, the probabilities in the soft evidences encode how believable the information extracted from the informal clues is. The users can prioritize inspecting alarms with high probabilities. I instantiated this approach across multiple analyses: a pointer analysis for Java extended with predicting variable aliasing based on variable names, and a taint analysis for Android extended with resolving inter-component communication links using string constants. Furthermore, I demonstrated the generality of this soft evidence mechanism by using it to incorporate dynamic test execution data into interval and taint analyses for C programs. Crucially, looking to the future, this soft evidence framework provides a principled, mathematically rigorous foundation for integrating Large Language Models (LLMs) into formal program analysis. By treating LLM inferences—such as extracting knowledge from the repository—as soft evidences, we can harness their vast reasoning capabilities while safely quantifying their inherent uncertainties without breaking formal soundness.

Testing. To demonstrate the broad applicability of Bayesian program analysis, I developed BAYZZER [POPL'26a], a framework that systematically leverages this paradigm to guide large-scale target-guided greybox fuzzing (LTGF). Traditional LTGF approaches rely on coarse, heuristic target prioritization strategies, leading to wasted effort on unreachable targets or false alarms. My work fundamentally resolves this by formulating target prioritization as a reachable fuzzing targets problem. By building a Bayesian model on top of static analysis derivations, the framework predicts the probability that each target is reachable by the fuzzer. Crucially, the system systematically learns from the fuzzer's dynamic execution feedback as posterior information, allowing the guidance to adapt as the fuzzer's capabilities evolve. This approach provides semantics-aware, fine-grained, and adaptive guidance to the fuzzing process. Empirically, BAYZZER drastically outperforms state-of-the-art fuzzers, discovering 3.25x to 13x more unique bugs on a suite of real-world programs. **Furthermore, it automatically identified 39 previously unknown vulnerabilities in well-tested open-source programs, 30 of which have been assigned CVEs**. Ultimately, this work successfully extends the application of Bayesian program analysis from simply ranking static alarms to fully automated bug discovery.

Fault Localization. A critical challenge in automated debugging is localizing faults with both high precision and practical scalability. Existing paradigms are inherently limited: spectrum-based techniques ignore program semantics and suffer from imprecision, while mutation-based or angelic debugging approaches model full semantics but routinely fail to scale. To bridge this gap, I designed a semantics-based probabilistic fault localization approach [TSE'25, ICSE'22]. The core methodological innovation lies in abstracting away intractable full semantics and instead probabilistically modeling only the correctness of program values. By combining this highly efficient semantic approximation with rigorous probabilistic inference over passing and failing test executions, the system dynamically pinpoints faulty program elements. Evaluated against the real-world Defects4J 2.0 dataset, this probabilistic model dramatically advanced the state of the art, yielding a 130% improvement over the best spectrum-based baselines and the best mutation-based baselines in top-1 statement-level accuracy. This work clearly demonstrates that carefully calibrated probabilistic approximations can unlock entirely new levels of scalability and precision in automated debugging.

Algorithms

Abstraction Selection. To operationalize the applications described above, I tackled a fundamental challenge in program analysis: abstraction selection. In traditional logic-based frameworks, the choice of semantic abstraction dictates the critical trade-off between precision and scalability—a historically extensively studied problem. However, in the context of Bayesian program analysis, this challenge introduces a profound new dimension of complexity. Here, the abstraction serves as the core representation of the underlying learning system, fundamentally governing its ability to generalize from posterior observations. To solve the complex trilemma of balancing precision, scalability, and learnability, I designed a comprehensive framework that systematically navigates this space. My solution couples a data-driven offline prediction phase [OOPSLA'24a] with a dynamic, refinement-based online adjustment phase [OOPSLA'25b].

In the offline phase, the system learns an abstraction selection policy by analyzing labeled training programs. Unlike traditional methods that only optimize for precision and scalability, this data-driven phase explicitly optimizes for generalization ability. By rigorously evaluating derivation graph properties both before and after modifying candidate abstractions, the framework automatically predicts configurations that will maximize statistical learning efficacy. Empirically, this offline approach is highly effective; when evaluated on a suite of real-world Java programs in an alarm ranking setting, it reduced

the total inspection burden required to find all true alarms by an average of 37.53% for datarace analysis, and 12.34% for thread-escape analysis compared to random abstraction selection compared to the coarsest abstraction.

However, because offline learning can occasionally yield sub-optimal performance on new, out-of-distribution programs, I designed a complementary online phase inspired by counterexample-guided abstraction refinement (CEGAR). During the interactive analysis, this phase dynamically refines the abstraction on the fly. The core innovation lies in leveraging the theory of conditional independence to map information transmission within the Bayesian network. By formulating the refinement as a maximum satisfiability (Max-SAT) problem, the system guarantees that incorrect, over-generalized inferences are systematically removed while correct generalizations are strictly preserved. This dynamic refinement yields substantial real-world improvements: in extensive evaluations, it further reduced the total number of false alarms inspected before finding all true alarms (Rank-100%-FP) by 18.86% for datarace analysis and 33.01% for thread-escape analysis over the offline-only approach. Notably, when the offline training set was sub-optimal, the online refinement successfully compensated, boosting the improvement to nearly 47%.

Question Selection. In Bayesian program analysis, online learning relies on acquiring posterior observations—whether through manual user inspection, dynamic test execution, or querying expensive formal oracles—to continuously refine the probabilistic model. Because acquiring this ground truth is costly, it is critical to select the most informative targets. Historically, systems have employed a purely greedy, exploitation-only strategy, constantly prioritizing the highest-confidence facts. However, I identified that this approach inherently traps the learning system in local optima, causing it to stagnate on redundant evaluations and delaying the discovery of underlying truths. To fundamentally resolve this, I introduced the Exploration-Exploitation paradigm to Bayesian program analysis. I designed BEER [OOPSLA'26], a novel active learning framework that systematically estimates the expected information gain of exploring different variables. By clustering analysis facts based on their structural correlations—specifically, the proximity and number of shared root causes within the derivation graph—the system accurately identifies candidates that maximize new information acquisition. Evaluated on the task of interactive alarm resolution across datarace, thread-escape, and taint analyses, this domain-specific exploration mechanism proved highly effective, improving the inversion count (the metric measuring false alarms inspected before true alarms) by an average of 32.10% compared to traditional greedy strategies.

Probabilistic Inference. A fundamental bottleneck in scaling Bayesian program analysis to real-world software is the computational cost of the underlying probabilistic inference. While approximate algorithms like loopy belief propagation (LoopyBP) are standard, their message-passing updates require enumerating all possible variable assignments within a constraint. For the large factors typically generated by program semantics, this leads to an intractable exponential time complexity of $O(n2^n)$. To break this computational barrier, I developed an optimized LoopyBP algorithm that systematically exploits the inherent local structures of probabilistic constraints [ASE'25]. Rather than relying on standard tabular encodings, my framework represents factors using "if-then" rules, algebraically grouping variable assignments that yield identical probabilities. This structural exploitation allows the algorithm to extract common terms during message updates, mathematically reducing the inference complexity for prevalent constraints—such as probabilistic **Horn clauses**—from exponential to strictly linear $O(n)$ time. Implemented across diverse applications, this foundational algorithmic shift accelerated state-of-the-art fault localization and interactive analysis frameworks by average factors of 5.11x and 2.31x respectively, with peak speedups exceeding 289x on highly complex constraints.

Theory

Abstract Interpretation with Confidence. To provide a rigorous mathematical foundation for Bayesian program analysis, I developed a theoretical framework that formally quantifies the precision of abstract-interpretation-based analysis using probabilities [PLDI'26]. Instead of relying on empirical intuitions for alarm probabilities, my work explicitly defines the source of randomness: I formalized analysis confidence as the exact probability that an abstract interpreter is *locally complete* for a concrete program sampled from a distribution of programs matching the abstract interpretation. To compute these probabilities systematically, I designed a novel compositional denotational semantics that derives the distribution of concrete program outputs. Furthermore, to solve the inherent inefficiency of computing probabilities across all possible concrete programs, I introduced a secondary, tractable denotational semantics. This semantics systematically under-approximates the original by conservatively tracking only locally complete states, allowing for the efficient calculation of a rigorous lower bound for alarm confidence. By mathematically proving the correctness and soundness of both semantics, my work establishes the definitive theoretical foundation for quantifying abstract interpretation precision with probabilities. The related paper is currently conditionally accepted to PLDI'26.

Other Works

Optimizing DSLs for Accelerating Program Synthesis. A recent work of mine is to optimize domain-specific languages (DSLs) to accelerate different program synthesis algorithms [POPL'26b]. Syntax-guided synthesis fundamentally relies on DSLs to constrain intractable search spaces, but manually designing an optimal DSL that balances expressiveness with efficiency is notoriously difficult. To automate and solve this, I developed AMAZE, a novel optimization framework that iteratively refines a DSL without enduring the prohibitive computational cost of repeatedly invoking the underlying synthesizer. The core methodological breakthrough is the identification of "feature components"—key program fragments whose enumeration ranks strongly correlate with total synthesis time. By engineering a dynamic-programming algorithm to calculate these ranks and coupling it with a machine learning predictive model, AMAZE accurately estimates synthesis costs on the fly. Empirically, this automated DSL optimization achieved massive performance gains across multiple domains, yielding up to a 4.35x speedup on state-of-the-art synthesizers including EUSOLVER, DRYADSYNTH, DUET, and POLYGEN.

Analyses for Explaining AI Systems. I have also worked on program analyses for explaining artificial intelligence systems with the focus of exploiting structural information in the inputs. Mainstream local model-agnostic explanation techniques—such as LIME, Anchors, and SHAP—typically treat inputs as flat collections of basic features, such as individual words or isolated image superpixels. This lack of structure often yields explanations that are unfaithful to the model's actual decision-making process and confusing to end-users. To address this, I developed a suite of frameworks that fundamentally redefine the explanation language through structured logical predicates. First, to capture the sequential structure of inputs for models like Large Language Models (LLMs) and Recurrent Neural Networks (RNNs), I designed ReX [AAAI'25]. ReX systematically incorporates temporal information into explanations by augmenting underlying perturbation models and utilizing 1-D and 2-D temporal predicates to represent absolute and relative feature positions. Second, to bridge the semantic gap between low-level features and human reasoning, I proposed ConLUX [In Submission]. ConLUX elevates the explanation language by generating concept predicates—high-level semantic abstractions, such as topics or visual objects, extracted automatically via large pre-trained models. Lastly, to make these explanation techniques scalable to models as large as LLMs, I proposed a proxy framework that uses efficient models to approximate the decision boundaries of the target LLMs [ACL'26]. By instantiating these frameworks across diverse state-of-the-art vision and language models, my research demonstrates that exploiting both temporal and conceptual input structures significantly improves the mathematical fidelity of local explanations while making them substantially more interpretable and actionable for humans.

Abstraction Selection for Conventional Program Analyses. Besides developing Bayesian program analysis, I have also addressed abstraction selection problem in conventional logic-based program analysis. In purely logical frameworks, such as those formulated in Datalog, tuning the sensitivity of the analysis typically involves a severe trade-off between precision and scalability. To enable fine-grained sensitivity tuning without triggering state-space explosions, I proposed a novel data-driven technique that automatically learns to merge domain values for library code [SAS'21]. By extracting optimal merging strategies from training programs, this approach significantly accelerates the analysis while rigorously minimizing precision loss.

Furthermore, to tackle the inherent scalability bottlenecks of counterexample-guided abstraction refinement (CEGAR), I introduced a transformative integration of Graph Neural Networks (GNNs) into the refinement pipeline [OOPSLA'24b]. Existing CEGAR techniques cast abstraction refinement as complex constraint-solving problems, which routinely fail to scale on large industrial codebases. My framework bypasses computationally prohibitive SMT solvers by utilizing GNNs to predict high-quality abstractions directly from the topological structures of the derivation graphs. This integration effectively scales CEGAR, dramatically expanding the horizon of programs that can be practically and rigorously verified.

Future Directions

Looking ahead, my research over the next five years will be driven by two primary objectives. The first is to elevate Bayesian program analysis to its full potential, scaling its practical applications across broader domains while continuing to solidify its underlying mathematical theories. The second direction focuses on the critical intersection of formal methods and artificial intelligence: I aim to embed formal verification directly into Large Language Models (LLMs) to mathematically enforce and improve the correctness of their reasoning capabilities.

Enhancing Bayesian Program Analysis. My first objective is to continue scaling the impact and rigor of Bayesian program analysis. I plan to achieve this by advancing three highly synergistic research thrusts across applications and

theory:

Application to Operating System Testing. Building upon my work on BAYZZER, which successfully guided large-scale target-guided greybox fuzzing for regular programs, I will expand Bayesian guidance to comprehensive operating system testing. The core scientific challenge I will address is bridging the profound semantic gap between generating regular, flat inputs and synthesizing valid, state-dependent sequences of system API invocations. By modeling the probabilistic dependencies of system states, I aim to create an adaptive fuzzer that systematically navigates complex API protocols to uncover deep, system-level vulnerabilities.

Neural-Symbolic LLM Integration. To resolve the inherent imprecision of traditional static analyses, I am investigating a hybrid neural-symbolic approach that intimately couples Bayesian reasoning with Large Language Models (LLMs). Rather than wastefully querying LLMs on entire codebases, I will leverage the Bayesian model to precisely pinpoint suspicious internal derivation facts for targeted LLM inspection. This dramatically reduces token consumption while simultaneously boosting the LLM's analytical accuracy. Crucially, by quantifying the LLM's unreliability as probabilistic uncertainty, I will seamlessly integrate its responses through my established soft evidence mechanism. This mathematical grounding safely neutralizes the risk of LLM hallucinations and strictly limits the impact of their generative imprecision.

Expanding the Theoretical Foundation. To govern these advanced applications, I will substantially expand my foundational theory on abstract interpretation with confidence. First, I plan to broaden its scope beyond traditional data-flow analysis by mathematically formulating the imprecision incurred through control-flow abstractions. Second, I will extend the framework from quantifying the confidence of over-approximations to rigorous under-approximations, thereby yielding a precise metric for the degree of analysis incompleteness. Finally, I will develop a data-driven offline framework to meticulously learn the probabilistic distributions of atomic abstract transferrers. Together, these theoretical pillars will enable the development of an entirely new generation of Bayesian program analyses governed by a unified, rigorous mathematical foundation.

Integrating Formal Verification into LLMs. While people typically think and write in natural languages, they need mathematical and formal tools when navigating tricky domains, such as science and formal verification. Therefore, it is important to teach LLMs to reason in formal languages. I am planning to investigate two directions.

Empowering Existing models with Auto Formalization. The first direction is to empower existing LLMs with formal verification. The approach I am investigating is auto-formalization, where the overall scheme is to ask an LLM to formalize its reasoning process into a formal program, which can then be rigorously checked using a verifier. I built a guess-and-check framework based on this concept [TOPLAS'26]. Specifically, when an LLM proposes an incorrect solution for a complex task like loop invariant synthesis, my framework prompts the model to articulate its step-by-step reasoning as a natural language proof. To avoid the fragility of generating complete formal proof scripts, I utilize an auto-formalization pipeline to translate only the core local reasoning steps into lightweight first-order logic implications. By evaluating these implications with an SMT solver, the system can automatically pinpoint the exact logical flaws or hallucinations in the LLM's thought process. This enables the generation of highly precise, constructive feedback that actively guides the LLM to refine its logic. My initial implementation of this paradigm has already demonstrated exceptional promise, achieving a 93.1% success rate on rigorous loop invariant synthesis benchmarks, proving that verifying a model's internal thinking process can dramatically elevate its reasoning capabilities.

White-Box Neuro-Symbolic Reasoning with a Verifier-in-the-Loop. My second major direction focuses on teaching Large Language Models to natively reason in formal languages by embedding a symbolic verifier directly into both the training and inference pipelines. While recent approaches using Reinforcement Learning with Verifiable Rewards (RLVR) show promise, they are fundamentally limited by treating the verifier as a "black box" that yields only sparse, binary pass/fail rewards or surface-level text errors. To overcome this, I will pioneer a "white-box" neuro-symbolic architecture that deeply integrates the verifier's internal data structures into the neural network's learning loop. Specifically, to solve the critical *credit assignment problem* in reinforcement learning, I will draw on my foundational work in Bayesian program analysis to formulate the verification process probabilistically. This shift allows the system to automatically pinpoint the exact logical steps responsible for failures, translating them into dense, highly localized gradient penalties that actively correct the model's reasoning.

Beyond training, this verifiable-by-design foundation fundamentally transforms online inference. At generation time, the model will produce a "Formal Chain-of-Thought," with each incremental step compiled and checked by the background verifier. If the model attempts an invalid logical leap, the verifier serves as a deterministic guardrail: it halts generation, injects precise structural feedback from its internal state directly into the context window, and prompts the LLM to self-correct on the fly. Ultimately, this interactive, white-box feedback loop dynamically prunes incorrect reasoning paths, ensuring the final output is mathematically rigorous and free of hallucinations.

My Publication (See CV for a Complete List)

[Thesis] X. Zhang. *COMBINING LOGICAL AND PROBABILISTIC REASONING IN PROGRAM ANALYSIS*. PhD thesis, Georgia Institute of Technology, 2017.

[TOPLAS'26] T. Li, Z. Yan, J. Liu, P. Di, and X. Zhang. Guiding LLM-based loop invariant synthesis via feedback on local reasoning errors. *ACM Trans. Program. Lang. Syst.*, 2026. DOI: 10.1145/3806652. URL: https://doi.org/10.1145/3806652.

[ACL'26] J. Liu, H. Yu, Z. Yan, and X. Zhang. Revitalizing black-box interpretability: actionable interpretability for LLMs via proxy models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics, ACL 2026*, 2026. arXiv: 2505.12509. URL: https://arxiv.org/abs/2505.12509. To appear.

[In Submission] J. Liu, H. Yu, and X. Zhang. Conlux: concept-based local unified explanations. *CoRR*, abs/2410.12439, 2024. DOI: 10.48550/ARXIV.2410.12439. arXiv: 2410.12439. URL: https://doi.org/10.48550/arXiv.2410.12439.

[PLDI'26] Y. Shi, Z. Jin, and X. Zhang. Abstract interpretation with confidence. *PLDI*, 2026. To appear.

[OOPSLA'26] H. Lin, Z. Yan, and X. Zhang. Beer: interactive alarm resolution in bayesian program analysis via exploration-exploitation. *Proc. ACM Program. Lang.*, (OOPSLA1), 2026. To appear.

[POPL'26a] Y. Zhang and X. Zhang. Fuzzing guided by bayesian program analysis. *Proc. ACM Program. Lang.*, 10(POPL):476–506, 2026. DOI: 10.1145/3776659. URL: https://doi.org/10.1145/3776659.

[POPL'26b] Z. Ye, R. Ji, Y. Xiong, and X. Zhang. Accelerating syntax-guided program synthesis by optimizing domain-specific languages. *Proc. ACM Program. Lang.*, 10(POPL):1066–1093, 2026. DOI: 10.1145/3776679. URL: https://doi.org/10.1145/3776679.

[OOPSLA'25a] T. Li and X. Zhang. Combining formal and informal information in bayesian program analysis via soft evidences. *Proc. ACM Program. Lang.*, 9(OOPSLA1):1774–1801, 2025. DOI: 10.1145/3720508. URL: https://doi.org/10.1145/3720508.

[OOPSLA'25b] Y. Shi, Y. Zhang, and X. Zhang. On abstraction refinement for bayesian program analysis. *Proc. ACM Program. Lang.*, 9(OOPSLA2):3232–3258, 2025. DOI: 10.1145/3763166. URL: https://doi.org/10.1145/3763166.

[TSE'25] Y. Wu, Y. Liu, Y. Yin, M. Zeng, Z. Ye, X. Zhang, Y. Xiong, and L. Zhang. Smartfl: semantics based probabilistic fault localization. *IEEE Trans. Software Eng.*, 51(7):2161–2180, 2025. DOI: 10.1109/TSE.2025.3574487. URL: https://doi.org/10.1109/TSE.2025.3574487.

[AAAI'25] J. Liu and X. Zhang. Rex: A framework for incorporating temporal information in model-agnostic local explanation techniques. In T. Walsh, J. Shah, and Z. Kolter, editors, *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 18888–18896. AAAI Press, 2025. DOI: 10.1609/AAAI.V39I18.34079. URL: https://doi.org/10.1609/aaai.v39i18.34079.

[ASE'25] Y. Wu, Y. Chen, Y. Xiong, and X. Zhang. Belief propagation with local structure and its applications in program analysis. In *40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16-20, 2025*, pages 1566–1577. IEEE, 2025. DOI: 10.1109/ASE63991.2025.00132. URL: https://doi.org/10.1109/ASE63991.2025.00132.

[OOPSLA'24a] Y. Zhang, Y. Shi, and X. Zhang. Learning abstraction selection for bayesian program analysis. *Proc. ACM Program. Lang.*, 8(OOPSLA1):954–982, 2024. DOI: 10.1145/3649845. URL: https://doi.org/10.1145/3649845.

[OOPSLA'24b] Z. Yan, X. Zhang, and P. Di. Scaling abstraction refinement for program analyses in datalog using graph neural networks. *Proc. ACM Program. Lang.*, 8(OOPSLA2):1532–1560, 2024. DOI: 10.1145/3689765. URL: https://doi.org/10.1145/3689765.

[ICSE'22] M. Zeng, Y. Wu, Z. Ye, Y. Xiong, X. Zhang, and L. Zhang. Fault localization via efficient probabilistic modeling of program semantics. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 958–969. ACM, 2022. DOI: 10.1145/3510003.3510073. URL: https://doi.org/10.1145/3510003.3510073.

[SAS'21] Y. Chen, C. Yang, X. Zhang, Y. Xiong, H. Tang, X. Wang, and L. Zhang. Accelerating program analyses in datalog by merging library facts. In C. Dragoi, S. Mukherjee, and K. S. Namjoshi, editors, *Static Analysis - 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17-19, 2021, Proceedings*, volume 12913 of *Lecture Notes in Computer Science*, pages 77–101. Springer, 2021. DOI: 10.1007/978-3-030-88806-0_4. URL: https://doi.org/10.1007/978-3-030-88806-0%5C_4.