Research Statement:
Formal Methods to Ensure Trustworthiness in Deep Learning

Yuhao Zhang, University of Wisconsin–Madison
https://pages.cs.wisc.edu/~yuhaoz/

Deep learning has rapidly emerged as a transformative technology that permeates all modern software, from autonomous driving systems to malware-detection tools. Given the critical role this software plays in infrastructure, the software must behave as intended. However, the complexity introduced by deep-learning components complicates formal reasoning about the behavior of such software, frequently resulting in solutions that offer only empirical or no guarantees. My research contributes techniques and algorithms that increase the trustworthiness of deep-learning-powered software by providing strong **provable guarantees** across the components appearing in the **entire** deep learning pipeline.

The goal of my research is to ensure the safety, security, and reliability of **all** components of deep-learning-powered software by providing **provable guarantees** via formal methods.

I have worked on three essential components of deep-learning-powered software to achieve this research goal. Below, I will describe three ongoing research directions that shaped my PhD research.

Correctness of code built on deep learning platforms: building tools for ensuring the correctness of deep learning training and inference code built on top of TensorFlow/PyTorch.

Robustness of deep learning model inference: proving and improving the robustness of deep learning models against inference-time adversarial examples.

Integrity of deep learning model training: designing certifiable defenses against data poisoning and backdoor attacks during training.

I will summarize some major contributions for each direction. I will then conclude with my future research plans on (1) robustness for large language models, (2) end-to-end verification in deep-learning-based software, and (3) integration of verification and interpretability.

1 Correctness of Code Built on Deep Learning Platforms

Deep learning models have made remarkable advancements in solving complex and diverse problems. The training and inference of these models are coded built on platforms like TensorFlow and PyTorch. This coding requires a programming paradigm that differs significantly from traditional ones. For instance, programming logic relies more heavily on intensive matrix operations and activation functions than traditional programming structures such as branches and loops. As a result, new defects have emerged in deep learning code, and we need to rethink the problem of verifying code correctness and providing provably correct fixes to these defects.

In my research on providing provable guarantees for the correctness of deep learning code, I pose the following questions: *What are these new defects? Which defect type is of most concern to the deep learning community?* The first question led me to conduct a pioneering empirical study of novel defects in deep learning code. This study found that *numerical defects*, which often manifest as `NaN` or `INF` during neural network computations, are prevalent in deep learning code and can significantly impair the model accuracy, potentially leading to system crashes. This finding addressed the second question and guided my work on ensuring deep learning code is free of numerical defects.

Papers: ISSTA 2018 [9], FSE 2020 [10] (**Distinguished Paper Award, Key paper**), ICSE 2023 [3]

Key ideas: (1) A pioneering empirical study of defects in deep learning code. (2) An abstract interpretation technique for verifying the absence of numerical defects. (3) A framework for reliability assurance against numerical defects by providing failure-exhibiting tests and provably correct fix suggestions.

A pioneering empirical study of defects in deep learning code Motivated by the need to investigate emerging defects in deep learning code, I conducted a pioneering empirical study [9] on the root causes, symptoms, and fixing strategies of these new defects. **This empirical study not only serves as a bedrock of my work but also inspires numerous initiatives in the research community.** For example, our study identified new defects such as unaligned-shape defects, where the shape of a tensor, a data structure predominantly used in deep learning, does not align with its expected shape. Subsequently, a corresponding approach [1] has been proposed to detect this new defect. Additionally, Islam et al. [2] conducted a further empirical study and "adapted the classification scheme of root causes and bug effects" from our study.

Verification of the absence of numerical defects My work, DEBAR [10], detects numerical defects in deep learning code. It can either construct a proof confirming the absence of numerical defects within the code or identify suspicious computational graph nodes that may have numerical defects. To construct such proof, I design refined yet scalable abstract domains, *tensor partitioning* and *interval with affine equality relation*, to over-approximate the output range of each node in the computational graph. If a node receives an invalid range or its output overflows, DEBAR will report this node as suspicious. And if no such node exists in the code, DEBAR then gets a correctness proof owing to the soundness of abstract interpretation. DEBAR is highly scalable as it only takes an average of 12.1 seconds for each model, while maintaining a low false positive rate of 7.7%. DEBAR has **received recognition with the ACM SIGSOFT Distinguished Paper Award** at FSE 2020 and it **detected 11 real-world bugs** of 48 models implemented in an official repository maintained by the TensorFlow team[1].

Provably correct fixes to numerical defects Once the defects are identified, developers need to fix them. We found that developers either provided no fixes or designed heuristic fixes that did not eliminate these defects, such as reducing the learning rate. Such heuristic fixes often delay the triggering of numerical defects during training and further obscure the defects. Our work, RANUM [3], automatically synthesizes provably correct fixes to these defects for developers. The fixes take the form of clipping the input ranges of some computational graph nodes. Striking the right balance with these input ranges is critical; overly tight ranges can hinder the model accuracy, whereas overly wide ranges may still result in numerical defects. We propose an abstraction optimization algorithm to find the tightest input ranges that provably eliminate the defects.

2 Robustness of Deep Learning Model Inference

A unique characteristic of deep learning models is their vulnerability to malicious attacks, even when the underlying code implementations are correct. Among the various types of attacks, inference-time (or test-time) attacks have been extensively studied as they directly affect the performance and reliability of the model. These attacks craft a human-imperceptible perturbation to the test input to deceive the model into making incorrect predictions.

Test-time defenses and attacks on deep learning models have been a never-ending cat-and-mouse game. My research aims to end this game by providing deep learning model inference with well-defined and provable guarantees. I focus on the robustness verification of language models, an area previously

[1]See e.g., `https://github.com/tensorflow/models/pull/8221`, `https://github.com/tensorflow/models/pull/8223`

unexplored due to the challenge of the discreteness of the inputs.

Papers: ICML 2020 [5], EMNLP 2021 [6] (**Oral Presentation, Key paper**)

Key ideas: (1) Languages for describing test-time robustness for deep learning models. (2) Training approaches for improving model robustness. (3) An abstract interpretation technique for verifying model robustness.

Programmable perturbation space Existing work on robustness for deep learning model inference employs ad-hoc perturbations tailored to specific attacks, such as synonym substitutions. However, these perturbations do not apply to a wide range of scenarios. To address this limitation, I introduced the concept of a programmable perturbation space [5] and designed a language for defining attacks/perturbations to input sequences for language models. The versatile language allows users to express their specific robustness requirements as user-defined string transformations and combinations. For example, it can express a perturbation that removes stop words and duplicates exclamation and question marks in a movie review. Furthermore, this language enables robustness verification and training approaches [5, 6] to compile and understand users' needs seamlessly.

Verifying robustness of recursive models Given a robustness specification as a programmable perturbation space, my approach, ARC [6], generates proofs of robustness for recursive models, such as LSTMs or Tree-LSTMs. The key idea underlying ARC involves symbolically recursive memoization and abstraction of sets of possible hidden states, a task that becomes infeasible for enumeration due to its exponential growth with the input length. As ARC over-approximates the sets of all possible outcomes, it captures the worst-case scenario, thus establishing proofs for model robustness.

Robust training approaches When given a programmable perturbation space, the challenge of training robust models against the space lies in accurately approximating the worst-case loss. Traditional approximation methods provide loose approximations, such as the under-approximation by adversarial training or the over-approximation by provable training [6]. To overcome this challenge, I proposed A3T [5], an innovative approach that approximates the worst-case by decomposing the programmable perturbation space into two subsets: one that can be explored using adversarial training and another that can be abstracted using provable training. This novel idea of decomposition **has been adopted by the state-of-the-art robust training method, SABR [4]**.

3 Integrity of Deep Learning Model Training

High-quality, abundant data is crucial for training deep learning models to address complex problems. However, the integrity of this data is threatened by data poisoning attacks, where an attacker can subtly modify the training set to manipulate the model's predictions. Such attacks have been successfully utilized to surreptitiously insert backdoors into deep learning models.

This concern for data-poisoning attacks on deep learning models has led to my research on certified defenses ensuring the integrity of deep learning model training.

Papers: Neurips 2022 [7] (**Key paper**), Under Submission [8]

Key ideas: (1) Probabilistic and deterministic certified defenses against training-time attacks. (2) A holistic view of handling test-time and training-time threats.

Certified defenses against test-time and training-time attacks in a holistic view I have proposed two certified defenses, BagFlip [7] and PECAN [8], against data poisoning attacks that can modify both the training set and test inputs. These defenses construct a verifiable training algorithm $\bar{A}$ over the original algorithm A by creating an ensemble of models, each trained on subsets of the training data. These defenses adopt a holistic view of inference and training processes by regarding these processes as a closed box, abstracting away the intricate details of the training algorithm, which can pose challenges for verification techniques in establishing meaningful bounds. Leveraging this holistic view, BagFlip employs randomized smoothing to construct *probabilistic* proofs. In contrast, PECAN generates *deterministic* proofs by seamlessly integrating training-time and test-time proofs derived from corresponding techniques.

4 End-to-end Verification to Large-Scale Models and Their Interpretability

I envision a world where deep-learning-based software is universally employed, allowing everyone to enjoy the substantial benefits it offers and where everyone trusts such software due to its thoroughly verified trustworthiness. My work has shown that we can design new formal-method approaches to provide provable guarantees for deep learning code built on deep learning platforms, as well as for the inference and training of deep learning models. Therefore, over the next 5-10 years, I will persist in pursuing the thoroughly verified trustworthiness of deep-learning-based software, focusing on three key objectives:

Robustness for Large Language Models As deep learning models continue to grow in scale, exemplified by the prevalence of large language models (LLMs) with billions of parameters, the concept of their robustness needs reformulation. This need arises from the transition from classification to generation tasks, driven by the remarkable generative capabilities of LLMs. In the context of generation, robustness goes beyond single predictions to LLM-generated sequences and, in some cases, even code snippets. For example, I aim to define the robustness properties of LLMs that generate code and to develop techniques that can provide rigorous robustness guarantees for LLMs. This goal encompasses addressing the following two challenges. First, it is challenging to define the functional equivalence of code snippets generated by LLMs given different inputs in the programmable perturbation space. Second, the enormous model size challenges the scalability of verification techniques. My research [5] on designing a language for defining perturbations to input sequences of language models has laid a foundation for this transition of robustness.

End-to-end Verification in Deep-Learning-Based Software Ensuring the correctness and reliability of deep-learning-based software requires a comprehensive perspective, as a flaw in any component can lead to the failure of the entire software. This perspective necessitates expanding the verification beyond just the code built on deep learning platforms to include the deep learning platforms themselves, deep learning compilers, and the underlying deep learning systems. I seek to expand the scope of verification to encompass all the components in deep-learning-based software. An illustrative example of this involves guaranteeing numerical stability when applying the gradient scaling technique and mixed-precision training, both essential components of LLM training platforms. My research [10] on detecting numerical defects in deep learning code can serve as the initial endeavor for verifying these essential components in LLM training platforms.

Integration of Verification and Interpretability The opacity of deep learning models is a major reason for verifying model robustness. Simultaneously, another research community is addressing this challenge by developing methods to explain model behaviors. My vision involves the synthesis of deep learning robustness and interpretability, giving rise to the concepts of *interpretable robustness* and *robust interpretation*, which center around two questions, respectively: (1) can we use interpretability techniques to improve the precision of robustness verification techniques? And (2) can we prove the robustness of model interpretations?

These three objectives reflect my current interests, but other practical problems will continue to contribute to my main goal of *providing provable guarantees to ensure the safety, security, and reliability of deep-learning-based software*.

References

[1] M. Hattori, S. Sawada, S. Hamaji, M. Sakai, and S. Shimizu. Semi-static type, shape, and symbolic shape inference for dynamic computation graphs. In *4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 11–19, 2020.

[2] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan. A comprehensive study on deep learning bug characteristics. In M. Dumas, D. Pfahl, S. Apel, and A. Russo, editors, *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, pages 510–520. ACM, 2019.

[3] L. Li, **Y. Zhang**, L. Ren, Y. Xiong, and T. Xie. Reliability assurance for deep neural network architectures against numerical defects. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 1827–1839. IEEE, 2023.

[4] M. N. Müller, F. Eckert, M. Fischer, and M. T. Vechev. Certified training: Small boxes are all you need. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[5] **Y. Zhang**, A. Albarghouthi, and L. D'Antoni. Robustness to programmable string transformations via augmented abstract training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 11023–11032. PMLR, 2020.

[6] **Y. Zhang**, A. Albarghouthi, and L. D'Antoni. Certified robustness to programmable transformations in lstms. In M. Moens, X. Huang, L. Specia, and S. W. Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 1068–1083. Association for Computational Linguistics, 2021.

[7] **Y. Zhang**, A. Albarghouthi, and L. D'Antoni. Bagflip: A certified defense against data poisoning. In *NeurIPS*, 2022.

[8] **Y. Zhang**, A. Albarghouthi, and L. D'Antoni. PECAN: A deterministic certified defense against backdoor attacks. *CoRR*, abs/2301.11824, 2023.

[9] **Y. Zhang**, Y. Chen, S. Cheung, Y. Xiong, and L. Zhang. An empirical study on tensorflow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*, pages 129–140, 2018.

[10] **Y. Zhang**, L. Ren, L. Chen, Y. Xiong, S. Cheung, and T. Xie. Detecting numerical bugs in neural network architectures. In P. Devanbu, M. B. Cohen, and T. Zimmermann, editors, *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pages 826–837. ACM, 2020.