WASP Software Engineering Assignment Course Module 2024

Rishi Hazra

September 1, 2024

1 Introduction

Large Language Models (LLMs) have shown remarkable capabilities in text understanding and generation. This has sparked widespread interest and debate on whether LLMs are capable of reasoning. Recent studies suggest that LLMs are inherently capable of zero-shot reasoning [1] that can emerge and improve with scale [2], and can be further enhanced by using smart prompting techniques [3, 4]. Demonstrations include, planning [5, 6], theorem proving [7], search and optimization [8].

Conversely, a growing body of research presents a more critical view of these emergent abilities [9, 10, 11]. During training, LMs can fit on statistical features [12] or identify reasoning shortcuts [13]. There is also growing concern about dataset contamination from open-source benchmarks [14].

My research seeks to address these discrepancies by asking the following questions:

Can Large Language Models *truly* reason, and if so, to what extent? [15] Current methods for assessing the reasoning abilities of LLMs rely on open-source benchmarks that may be over-represented in LLM training data [14]. We instead provide a computational theory perspective of reasoning, using 3-SAT – the NP-complete problem that underlies logical reasoning [16, 17]. Our experiments show that LLMs cannot perform true reasoning.

How can we enhance the reasoning abilities of LLMs? Recent studies have explored *LLM-Modulo* frameworks [18] to enhance reasoning capabilities by augmenting LLMs with critics and verifiers [19, 6], recognizing them as approximate idea-generators rather than direct problem-solvers. This approach is aligned with neurosymbolic techniques [20] that combine neural networks and symbolic reasoning. Specifically, my research focuses on LLM-Modulo *agents* that perform tasks reliably and autonomously while interacting with their environment, including humans. Some frameworks I have worked on include:

- **EgoTV:** [21] A benchmark and dataset that assesses egocentric agents on tracking and verifying everyday tasks specified in natural language. This requires comprehensive reasoning abilities that Visual Language Models (VLMs) struggle with. We also present a novel framework that combines VLMs with symbolic reasoning to improve their performance.
- **SayCanPay:** [6] Planning with LLMs often leads to infeasible and sub-optimal plans [22]. We combine the power of LLMs and heuristic planning by leveraging the world knowledge of LLMs and the principles of heuristic search to enhance planning.
- **REvolve:** [23] Designing reward functions for reinforcement learning (RL) requires huge human efforts due to the subjective nature of certain tasks that are hard to quantify explicitly [24]. We use LLMs, guided by human feedback, to formulate human-aligned reward functions (in python code) and demonstrate it on Autonomous Driving.

- **Bident:** [25] A framework to integrate LLMs-based robots into shared spaces with humans. Potential applications include personalized education, and healthcare, companionship, and everyday assistance.

Ideas from Robert's lectures

Humans as oracles in Code testing. In code testing, poorly stated problem specifications and incomplete requirements can lead to incorrect tests, ultimately causing the software to exhibit unintended behavior. The concept of "humans as oracles" refers to the practice of relying on human judgment to determine the correctness of the output of a software system during testing. This is the mainly due to the subjective (i.e. not explicit) nature of the output that automated systems may fail to capture. For instance, a voice recognition software whose accuracy can be validated through automated processes, but subjective aspects like politeness or contextual relevance need human intervention. This mirrors the challenge in REvolve, where wrong or vague reward functions can misguide reinforcement learning (RL) agents, leading to behavior that does not align with human expectations. Just like developers can provide insight and judgment to detect issues and correct or refine requirements, REvolve employs human feedback to iteratively refine reward functions, ensuring that the behavior of the RL agent aligns more closely with human standards. In both cases, human involvement is crucial in bridging the gap between poorly defined specifications and the desired outcomes. This parallel highlights the importance of human judgment in refining and guiding automated processes to ensure that they meet real-world needs effectively, whether in software testing or autonomous system training.

Core AI research vs. AI/ML system development. My research completely focuses on the core AI research which involves laying mathematical foundations and formulating new algorithms. A key challenge in this area is that the frameworks I create are primarily designed as proof-of-concepts and are not immediately deployable in real-world applications. My models often rely on assumptions such as deterministic (or minimally stochastic) and ergodic environments (all relevant states are reachable by the agent), whereas real-world systems are typically highly stochastic and non-ergodic. Consequently, a significant effort in pipelining and integration is necessary to make these systems operational in practical settings. Despite these challenges, both core AI research and AI/ML system development are essential and complementary in advancing AI capabilities.

Ideas from Guest lectures

Behavioral Software Engineering. A critical challenge in this field is understanding and mitigating the cognitive load that developers experience, which can directly impact their productivity, job satisfaction, and the quality of the software they produce. For instance, programming, adn understanding and navigating large codebases, which require cricial and creative thinking. A hybrid approach that integrates Behavioral Software Engineering with AI tools can significantly reduce cognitive load and enhance developer productivity – i.e. LLMs to automate routine tasks and assist developers in more complex activities. Frameworks like Github Copilot and Amazon CodeWhisperer are already being used [26]. Developers can interact with the software development environment using natural language commands, significantly reducing the cognitive burden associated with remembering and typing specific syntax or commands. LLMs can generate code snippets, suggest refactorings, and even draft documentation based on natural language inputs, making the development process more intuitive. Much like LLM-Modulo frameworks, LLMs can serve as idea generators, helping developers brainstorm solutions or explore multiple approaches to a problem. Moreover, memory-based tools are being developed that can be customized to the needs of the specific user by learning patterns from interactions. This can further simplify the cognitive load and enhance efficiency.

How AI/ML will affect Software Engineering products In the previous discussion, I emphasized how integrating AI tools with software engineering can benefit developers. However, AI can also play a crucial role in simplifying and enhancing the experience for (non-expert) end-users by helping them refine and customize software.

Similar to how frameworks like REvolve translate implicit human concepts into explicit code snippets, Large Language Models (LLMs) can serve as intuitive interfaces that leverage human feedback to improve code generation. These AI-driven interfaces can interpret subjective and abstract natural language input from end-users, allowing them to tailor software outputs to their specific needs without requiring technical expertise. This not only empowers users to have greater control over the software but also bridges the gap between user intent and technical implementation.

Paper Discussions

Welcome Your New AI Teammate: On Safety Analysis by Leashing Large Language Models, Nouri et al., CAIN 2024 [27]

(a) This paper explores the potential application of Large Language Models (LLMs), particularly GPT-4, in the field of safety engineering, specifically for performing Hazard Analysis and Risk Assessment (HARA) in Autonomous Driving. Traditional methods for HARA are time-consuming and require significant intellectual input, which slows down the SafetyOps process. Since several key steps in HARA –, e.g. identifying and describing hazards, outlining safety goals and requirements, documentation and communication, etc. – involve the use of natural language, LLMs offer a potential means to automate some of these tasks, thereby speeding up the process. The authors aim to accelerate the iterative loops in SafetyOps by automating parts of the safety analysis process using LLMs. Specifically, the authors designed a pipeline consisting of multiple prompts, each corresponding to a specific sub-task within the HARA process. The prompts guide the LLM in generating outputs such as hazardous event identification, scenario generation, severity classification, and safety goal formulation. The paper concludes that while LLMs can contribute significantly to accelerating the HARA process, their outputs should be used with caution and always reviewed by experts.

(b) In LLM-Modulo frameworks, while LLMs are used to generate ideas, candidate plans, or approximate models, humans play a crucial role in validating and refining these outputs. For instance, in REvolve, we argue that LLMs, by themselves, lack the ability to ensure correctness and reliability, especially in complex reward design for autonomous driving. Human experts are needed to help refine the outputs for better alignment. Both approaches advocate for a collaborative interaction between AI and human experts. In the HARA context, the AI teammate provides a "version zero" of the HARA, which human engineers then refine. This is analogous to the LLM-Modulo Framework, where LLMs generate candidate solutions or plans that are refined through human intervention or external model-based critiques.

(c) Autonomous driving systems in the real world involve multiple complex components such as route planning [28], energy optimization [29], real-time decision-making [30], and safety analysis [24]. Each of these components can have varying levels of automation. An effective strategy to manage this complexity is to systematically break down these tasks into manageable subtasks. Each sub-task can then utilize dedicated prompts, as proposed in the AI teammate paper, to guide AI models (like LLMs) in generating relevant outputs.

As stated in the AI teammate paper, *LLMs can contribute significantly . . . , their outputs should be used with caution and always reviewed by experts.* On one hand, LLM-Modulo frameworks can address

this problem by integrating external critics and verifiers to improve the reliability and robustness of the outputs. The flexible nature of the framework also allows for human-in-the-loop approaches [31, 32, 33] for situations where subjective evaluation is necessary and where performance indicators are difficult to quantify explicitly. A practical example of this is REvolve [23].

On the other hand, the pipelined approach consisting of multiple prompts from the AI teammate paper could be used to expand LLM-Modulo frameworks to the real-world. In this setup, each module would function as an LLM-Modulo instantiation, with dedicated prompts automating specific subtasks within a larger, more complex task.

(d) Suppose the LLM-Modulo framework is used in an autonomous driving scenario where it needs to plan a route. The AI Teammate approach could contribute by providing a series of structured prompts that guide the LLM through the decision-making process. Instead of relying on a single, broad prompt, the AI Teammate's structured approach would allow the LLM-Modulo framework to break down the route planning task into smaller, more manageable prompts. For example, one prompt could be focused on identifying potential routes, another on evaluating these routes based on traffic and weather data, and a final prompt on selecting the optimal route. This is also akin to chain-of-thought reasoning used for LLM prompting [3].

Mutation-based Consistency Testing for Evaluating the Code Understanding Capability of LLMs, Li et al., CAIN 2024 [34]

(a) The paper emphasizes the importance of evaluating LLMs beyond code generation, focusing on their ability to understand and reason about code semantics. Traditional benchmarks for code generation (HumanEval [35]) do not adequately assess the code understanding capabilities of Large Language Models (LLMs). The paper presents a novel method called Mutation-based Consistency Testing (MCT), wherein, given a pair of inputs – program description and implementation – LLMs are asked to identify whether the implementation is consistent with the description. The authors produce subtle inconsistencies between code and its corresponding natural language descriptions by introducing code mutations. These mutations (e.g., Arithmetic Operator Replacement, Relational Operator Replacement) are small changes that alter the semantics of the original code, thus creating mismatches with the descriptions. The findings suggest that LLMs show significant variation in their code understanding performance and that they have different strengths and weaknesses depending on the mutation type and language.

(b) Code understanding plays a critical role in the design and refinement of reward functions. In REvolve, the process involves feeding the LLM with the reward function from the previous generation along with performance feedback from the trained agents. The LLM must not only parse and comprehend the syntax of the reward function code but also deeply understand its semantics—how the code implements specific behavioral incentives and constraints. This understanding enables the LLM to make informed modifications and refinements to the reward function, ensuring that the changes are both syntactically correct and semantically aligned with the desired outcomes.

The proposed idea can also be extended to other domains like AI Planning wherein, given a plan, the LLM must understand whether the plan achieves the goal. This is akin to our work on EgoTV [21]. For instance, for a hypothetical task description *heat and clean a bowl*, the LLM must understand that heating and cleaning can be done in any order, while for another task *heat then clean a bowl*, the heating must be performed before cleaning. This also requires semantic understanding. Task and Plan verification, as proposed in EgoTV could immensely benefit from training with hard negatives constructed through mutation. This is also useful for evaluating and improving the self-reflection abilities

of LLMs [36, 37] wherein LLMs are required to analyze their own outputs and propose refinements. Moreover, this would also lead to improved heuristics for AI reasoning and decision-making as proposed in our work SayCanPay [6]. Existing research shows that LLMs struggle to self-reflect [11].

(c) Code testing is an area that could be significantly automated with the help of AI, particularly through the use of LLMs. LLMs, with their ability to understand and generate natural language, have been increasingly applied to code-related tasks, such as code generation, documentation, and debugging. In the context of code testing, LLMs can automate the creation of test cases, predict potential bugs, and even suggest fixes. However, the effectiveness of this automation hinges on their understanding of the underlying code. An LLM that lacks a deep, innate understanding of code may produce outputs that appear correct but fail to address the nuances and complexities of real-world programming.

While LLM-Modulo frameworks are designed to enhance output reliability by integrating critics and verifiers, the approach discussed here aims to directly strengthen the inherent code understanding of LLMs. By improving this foundational understanding, even greater advancements can be achieved when applied within an LLM-Modulo framework. One effective method to enhance this understanding is through training the LLM using code mutations and evaluating its performance to assess its comprehension of code. This approach is elaborated in the next section. The same approach can similarly be applied to other domains like task and plan verification.

(d) LLM-Modulo frameworks – which are currently limited to inference systems – could be extended to incorporate training where the train data is generated from specialized modules that mutate input code. This approach would establish a continuous feedback loop for training LLMs during code testing. In this loop, LLM-generated pseudo labels—indicating whether the code meets its intended goal—can be verified by experts or through automated test cases. Failed cases can be immediately added back into the training data, while successful cases can be further mutated (automatically) to create additional training examples. This process would continuously expand and diversify the training dataset, leading to more robust and reliable LLM performance.

References

[1] Takeshi Kojima et al. "Large Language Models are Zero-Shot Reasoners". In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 22199–22213.

[2] Jason Wei et al. "Emergent Abilities of Large Language Models". In: *Transactions on Machine Learning Research* (2022). ISSN: 2835-8856. URL: https://openreview.net/forum?id=yzkSU5zdwD.

[3] Jason Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 24824–24837.

[4] Denny Zhou et al. "Least-to-Most Prompting Enables Complex Reasoning in Large Language Models". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=WZH7099tgfM.

[5] Wenlong Huang et al. "Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents". In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. 2022, pp. 9118–9147.

[6] Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. "SayCanPay: Heuristic Planning with Large Language Models using Learnable Domain Knowledge". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 2024, pp. 20123–20133.

[7] Sean Welleck et al. “NaturalProver: Grounded Mathematical Proof Generation with Language Models”. In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 4913–4927.

[8] Chengrun Yang et al. “Large Language Models as Optimizers”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=Bb4VGOWELI.

[9] Konstantine Arkoudas. “GPT-4 Can’t Reason”. In: *arXiv* 2308.03762 (2023). URL: https://api.semanticscholar.org/CorpusID:260704128.

[10] Karthik Valmeekam et al. “Large Language Models Still Can’t Plan (A Benchmark for LLMs on Planning and Reasoning about Change)”. In: *NeurIPS 2022 Foundation Models for Decision Making Workshop*. 2022. URL: https://openreview.net/forum?id=wUU-7XTL5XO.

[11] Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. “GPT-4 Doesn’t Know It’s Wrong: An Analysis of Iterative Prompting for Reasoning Problems”. In: *NeurIPS 2023 Foundation Models for Decision Making Workshop*. 2023. URL: https://openreview.net/forum?id=PMtZjDYB68.

[12] Honghua Zhang et al. “On the Paradox of Learning to Reason from Data”. In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*. Aug. 2023, pp. 3365–3373. DOI: 10.24963/ijcai.2023/375. URL: https://doi.org/10.24963/ijcai.2023/375.

[13] Gregor Bachmann and Vaishnavh Nagarajan. “The pitfalls of next-token prediction”. In: *arXiv* 2403.06963 (2024). URL: https://api.semanticscholar.org/CorpusID:268364153.

[14] Hugh Zhang et al. “A Careful Examination of Large Language Model Performance on Grade School Arithmetic”. In: *arXiv* 2405.00332 (2024).

[15] Rishi Hazra et al. *Can Large Language Models Reason? A Characterization via 3-SAT*. 2024. arXiv: 2408.07215 [cs.AI]. URL: https://arxiv.org/abs/2408.07215.

[16] Carla P Gomes and Bart Selman. “Satisfied with physics”. In: *Science* 297.5582 (2002), pp. 784–785.

[17] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

[18] Subbarao Kambhampati et al. “LLMs Can’t Plan, But Can Help Planning in LLM-Modulo Frameworks”. In: *arXiv* 2402.01817 (2024). URL: https://api.semanticscholar.org/CorpusID:267413178.

[19] Hunter Lightman et al. “Let’s Verify Step by Step”. In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=v8L0pN6EOi.

[20] Luc De Raedt et al. “From Statistical Relational to Neuro-Symbolic Artificial Intelligence”. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Survey track. International Joint Conferences on Artificial Intelligence Organization, July 2020, pp. 4943–4950. DOI: 10.24963/ijcai.2020/688. URL: https://doi.org/10.24963/ijcai.2020/688.

[21] R. Hazra et al. “EgoTV: Egocentric Task Verification from Natural Language Task Descriptions”. In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, 2023, pp. 15371–15383. DOI: 10.1109/ICCV51070.2023.01414. URL: https://doi.ieeecomputersociety.org/10.1109/ICCV51070.2023.01414.

[22] Karthik Valmeekam et al. *On the Planning Abilities of Large Language Models (A Critical Investigation with a Proposed Benchmark)*. 2023. arXiv: 2302.06706 [cs.AI].

[23] Rishi Hazra et al. *REvolve: Reward Evolution with Large Language Models for Autonomous Driving*. 2024. arXiv: 2406.01309 [cs.NE]. URL: https://arxiv.org/abs/2406.01309.

[24] W Bradley Knox et al. "Reward (mis) design for autonomous driving". In: *Artificial Intelligence* 316 (2023), p. 103829.

[25] Tim Schreiter et al. *Bidirectional Intent Communication: A Role for Large Foundation Models*. 2024. arXiv: 2408.10589 [cs.RO]. URL: https://arxiv.org/abs/2408.10589.

[26] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG]. URL: https://arxiv.org/abs/2107.03374.

[27] Ali Nouri et al. "Welcome Your New AI Teammate: On Safety Analysis by Leashing Large Language Models". In: *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*. CAIN '24. Lisbon, Portugal: Association for Computing Machinery, 2024, 172–177. URL: https://doi.org/10.1145/3644815.3644953.

[28] Mohamed Reda et al. "Path planning algorithms in the autonomous driving system: A comprehensive review". In: *Robotics and Autonomous Systems* 174 (2024), p. 104630. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2024.104630. URL: https://www.sciencedirect.com/science/article/pii/S0921889024000137.

[29] Dewant Katare et al. "A Survey on Approximate Edge AI for Energy Efficient Autonomous Driving Services". In: *IEEE Communications Surveys & Tutorials* 25.4 (2023), pp. 2714–2754. DOI: 10.1109/COMST.2023.3302474.

[30] Zhenjie Yang et al. *LLM4Drive: A Survey of Large Language Models for Autonomous Driving*. 2024. arXiv: 2311.01043 [cs.AI]. URL: https://arxiv.org/abs/2311.01043.

[31] Andrea L. Thomaz and Cynthia Breazeal. "Teachable robots: Understanding human teaching behavior to build more effective robot learners". In: *Artificial Intelligence* 172.6 (2008), pp. 716–737. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2007.09.009. URL: https://www.sciencedirect.com/science/article/pii/S000437020700135X.

[32] Paul F Christiano et al. "Deep reinforcement learning from human preferences". In: *Advances in neural information processing systems* 30 (2017).

[33] W Bradley Knox and Peter Stone. "Interactively shaping agents via human reinforcement: The TAMER framework". In: *Proceedings of the fifth international conference on Knowledge capture*. 2009, pp. 9–16.

[34] Ziyu Li and Donghwan Shin. "Mutation-based Consistency Testing for Evaluating the Code Understanding Capability of LLMs". In: *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*. CAIN '24. Lisbon, Portugal: Association for Computing Machinery, 2024, 150–159. URL: https://doi.org/10.1145/3644815.3644946.

[35] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG]. URL: https://arxiv.org/abs/2107.03374.

[36] Aman Madaan et al. "Self-Refine: Iterative Refinement with Self-Feedback". In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023, pp. 46534–46594. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

[37] Noah Shinn et al. “Reflexion: language agents with verbal reinforcement learning”. In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023, pp. 8634–8652. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.