ORACLE

Program Curriculum

Agentic workflows for cybersecurity

Multi-agent collaboration has emerged as a key AI agentic design pattern that can be built upon LLMs. Given a complex task like detecting/analysing cyber threats a multi-agent approach would break down the task into subtasks to be executed by different roles. For example a cyber threats investigation workflow might involve a team of specialized software agents (e.g. Log Collector Agent, Anomaly Detector Agent, Threat Intelligence Agent etc.) that work together to uncover cyber threats hidden within log data. Log Collector Agent gathers the information and extracts key details, and Anomaly Detector Agent search for suspicious patterns. Meanwhile, a Threat Intelligence Agent keeps watch for known attack methods. If a high-risk threat is identified, alerts are sent to human security analysts for further investigation and mitigation. This multiagent approach can offer improved threat detection accuracy and scalability over state of the art existing techniques.

Networking Language as a new modality

Exploring Networking Language as a new modality beyond Text, Images etc. presents interesting challenges and can serve a variety of use cases. While fine-tuning shows promise, a more promising approach lies in exploring how to make LLMs understand networking grammar as a new modality. This would allow them to grasp not just individual networking concepts (e.g., data and protocols), but also the relationships and interactions between them. Imagine an LLM that can not only understand the definition of BGP (Border Gateway Protocol) but can also analyze configurations, predict routing behavior, and even identify potential security vulnerabilities within BGP implementations. This shift towards understanding networking grammar opens doors to exciting possibilities. LLMs could be used to: Automate Network Security Analysis, Network Planning and Design, Network Troubleshooting and Diagnosis.

Techniques for improving vector similarity search joins.

Vector Index optimized for multi-table join: we would like to understand how we could build a common shared vector index across multiple tables (e.q. job_descriptions, candidates) such that join workloads on vector similarity (given my company needs, find top 3 job description and find top 5 suitable candidates for them) work in an efficient way (better than the naive nested loop join approach).

NL2SQL via Graph

With the recent advance of LLMs many tools that translate natural language to SQL emerged. Nevertheless, the translation accuracy of these tools is far from being great. We would like to explore whether using a more abstract intermediate representation (for example graph schema + SQL/PGQ graph query) into which the natural language can be translated would improve over state of the art existing techniques that translate directly to SQL.

Relational vs. Graph Joins for Converged Databases

Relational joins and graph traversals share many similarities – one can view a graph data structure as the equivalent of a relational join. In this work, we will formalize the equivalence and will show – in practice – when performing "graph joins" is the right way to go in various workloads.

Long term memory & user personalization for assistants

LLMs have made personal assistants already much more useful than the previous generation of assistants. However, they are not yet to the point where they can have long enough memories or adapt well to user preferences to improve the success rate and user satisfaction. We would like to investigate how to make assistants more useful in contexts that require either remembering facts from old conversations or understanding user preferences and user implicit references.

Scalable planning for assistants

LLMs have shown that they can reason to solve tasks, but they are maybe not yet at the point where they can plan and act on complicated tasks requiring plans of many steps, with recursive aspects or challenged by errors / specification changes from users during conversations. We would like to investigate how to improve on such cases.

Scalable Tool Use for Large Language Model

Tool-use by Large Language Models is a crucial piece of the development of helpful assistants which can autonomously search for additional information and take actions for users when instructed to do so. In particular, due to limited context lengths, large sets of tools may be difficult to use accurately, techniques such as tool retrieval, hierarchical sets of tools, or adapting models for tool selection should be investigated and compared.

Search over relational data

LLMs have shown that it is possible to make data accessible with a better user experience by using natural language instead of programming languages such as SQL. We would like to investigate what could be the next technical solutions to make semantic queries more usable over data stored in databases starting from being able to supporting queries over single table, to more complicated settings involving multi-tables with linkage and queries including mixes of semantic predicates as well as logical, range predicates.

Uncertainty in ML

Integrating & optimizing for uncertainty in ML. Evaluating a model's test scores can help inform whether or not the model can be trusted overall; however, it provides little to know information about when a model should be trusted. Integrating uncertainty quantification and anomaly detection technique to identify at inference time when a model is uncertain and when it is being used on out-of-distribution data (which may result in predicts that are confidently wrong). A successful project will not only integrate these techniques, but also introduce mechanisms to optimize for them, thereby improving the reliability of the models.

Fairness & Unintended Bias in LLMs

There are many ways to analyze the fairness or untended biases in LLMs. For example, these include group-based disparity metrics (are protected groups talked about in the same way), representation disparity (are protected groups talked about equally often) and individual, counterfactual fairness (if a specific input was re-written to talk about a different group, would the LLM's output be invariant?). A successful PhD on this topic would review the LLM fairness landscape and provide state-of-the-art methods for evaluating and mitigating one or more types of unintended bias in LLMs.

Explaining LLMs

Understanding LLMs is important not only to gain trust in their abilities, but also to cast doubt on their capabilities where appropriate. It has been shown that humans have a tendency to trust high-quality-sounding LLM output because it is written in plain and understandable language. However, we need to augment LLM output with easy-to-understand explanations about how a generation was created so that humans can review them to decide whether or not a particular generation should be trusted. This likely needs to move beyond simple chain-of-thought-style "explanations", which may themselves be susceptible to hallucinations, towards something that includes grounding in factual, understandable information.

Plagiarism detection of source code

With the increasing reliance on code sharing and reuse in modern software development, the detection of plagiarized source code has become a critical concern. Development teams need to ensure the originality of their code to maintain integrity and avoid legal issues. This project aims to create a system capable of identifying similarities between code bases to detect potential plagiarism. The goal is to develop tools that can accurately identify plagiarized code even when obfuscation or minor modifications are applied. Additionally, the project will explore the development of a risk assessment score for detected plagiarism instances, considering factors such as the extent of similarity, the complexity of the code, and the context in which the code is used. This tool will assist developers and organizations in ensuring the originality and integrity of their source code.

Characterization of third-party open-source libraries

Modern applications heavily depend on open-source software components (dependencies). Developers are often not aware of what capabilities these dependencies need. For example, if the dependency makes network requests, load code at runtime, start new threads or write to the file system. Giving developers insight into the capabilities that their dependencies require could help them make more informed decisions about which dependencies to include. This project aims to develop a system to automatically detect what capabilities an open-source dependency needs based on the available test suite. Another project objective is to evaluate a risk score (in addition to CVSS) for third-party dependency libraries. Different paths can be explored towards this idea, such as a risk score taking into account the maturity of the project and its code repository (project still maintained, license, security policy); the usage of third parties (frequency and ways of usage); and the size, complexity, and use of the library in other projects.

Evaluating the impact of changing a 3rd party library

Vulnerability remediation tools such as OCI's App Dependency Management service or GitHub's dependabot run a CI pipeline of the project to confirm that updating a vulnerable dependency does break the application. This project topic aims to improve this verification, include the historical data about the application, and evaluate the impact of changing a 3rd party by simulating an execution based on a recorded trace of an older version and see if the exercised code path is the same. The high-level idea is to replay the history of the service but replace the version of the application that was used at the time with the updated version. Once this test is run, the verify step would have a better idea of how likely the dependency upgrade is to break the application, and this can be communicated to the user through something like a "disruption score".