Concept-Guided LLM Agents for Human-AI Safety Codesign

Florian Geissler[1], **Karsten Roscher**[1], **and Mario Trapp**[1, 2]
[1] Fraunhofer IKS, Fraunhofer Institute for Cognitive Systems IKS, Munich, Germany
[2] School of Computation, Information and Technology, Technical University of Munich, Germany
{florian.geissler, karsten.roscher, mario.trapp}@iks.fraunhofer.de, mario.trapp@tum.de

Abstract

Generative AI is increasingly important in software engineering, including safety engineering, where its use ensures that software does not cause harm to people. This also leads to high quality requirements for generative AI. Therefore, the simplistic use of Large Language Models (LLMs) alone will not meet these quality demands. It is crucial to develop more advanced and sophisticated approaches that can effectively address the complexities and safety concerns of software systems. Ultimately, humans must understand and take responsibility for the suggestions provided by generative AI to ensure system safety. To this end, we present an efficient, hybrid strategy to leverage LLMs for safety analysis and Human-AI codesign. In particular, we develop a customized LLM agent that uses elements of prompt engineering, heuristic reasoning, and retrieval-augmented generation to solve tasks associated with predefined safety concepts, in interaction with a system model graph. The reasoning is guided by a cascade of micro-decisions that help preserve structured information. We further suggest a graph verbalization which acts as an intermediate representation of the system model to facilitate LLM-graph interactions. Selected pairs of prompts and responses relevant for safety analytics illustrate our method for the use case of a simplified automated driving system.

Introduction

The advent of transformer-based (Vaswani et al. 2017) large language models (LLMs) has sparked enormous popularity of generative artificial intelligence (AI) for creative, text-based tasks. Representative of this trend is the reported record of OpenAI's ChatPGT for the fastest growing user base of all times in February 2023 (Reuters 2023). The high quality of auto-generated text has inspired the exploration of LLMs for tasks that involve structured data, such as knowledge graphs. In particular, a widely desired use case is the application of LLMs to safety-analytical tasks (Jin et al. 2023; Pan et al. 2023; Wang et al. 2023), for example, reasoning about fault propagation in model graphs and associated risks. Attempts to make use of LLMs for a formal hazard analysis have been made with moderate success (Diemert and Weber 2023).

A particular challenge lies in the transformation of verbalized content to a structured graph information, and vice versa. Since the text generation process is of statistical nature, thus not bound by logical constraints, LLM responses will not necessarily preserve information structures of a given input. Existing strategies to tackle this problem can be broadly categorized as follows (Jin et al. 2023): 1) Prompt-engineering to encourage specific output formats or structural rules; 2) Heuristic or algorithmic reasoning: Encouraging the LLM to perform chain-of-thought (CoT) reasoning, for example, to solve the problem step by step following self-generated or predetermined instructions; 3) Making use of external knowledge using retrieval-augmented generation (RAG), for example via non-AI tools; 4) Fine-tuning or retraining models to empirically minimize the loss of structured information, see e.g., GraphGPT (Tang et al. 2023).

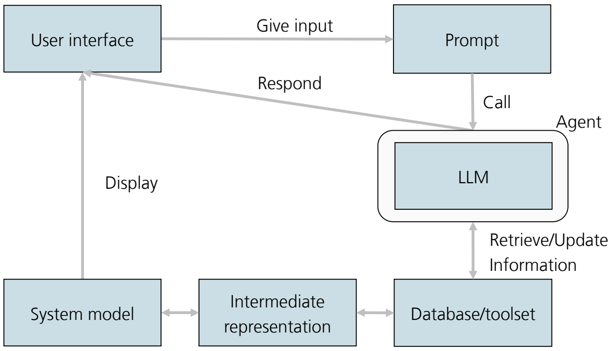


Figure 1: Layout of the Human-AI safety codesign framework: A user interacts with the LLM agent through a chat prompt and receives a text response. The LLM agent interfaces to a database containing a graph description of the system model in IR, as well as safety concepts, and analysis tools. System manipulations can update the database and alter the system model. The system model and its changes are displayed to the user.

This article presents a concept-guided approach aimed at enhancing the capabilities of LLMs for graph analysis and manipulation, particularly in the context of safety-relevant developments. To achieve this goal, we have integrated the strengths of LLMs with the rigorous standards of safety engineering models and analysis. Starting from a structured system model graph, we first establish a verbalized interme-

---

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

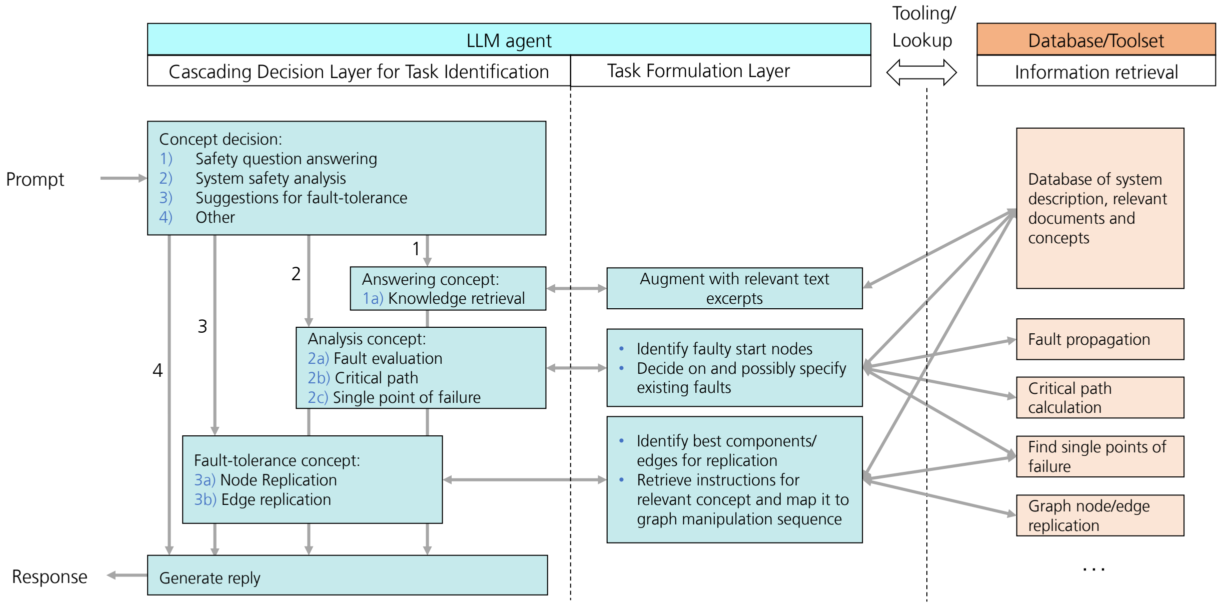

LLM agent
Cascading Decision Layer for Task Identification
Task Formulation Layer
Tooling/
Lookup
Database/Toolset
Information retrieval
Prompt
Concept decision:
1) Safety question answering
2) System safety analysis
3) Suggestions for fault-tolerance
4) Other
1
2
3
4
Answering concept:
1a) Knowledge retrieval
Analysis concept:
2a) Fault evaluation
2b) Critical path
2c) Single point of failure
Fault-tolerance concept:
3a) Node Replication
3b) Edge replication
Augment with relevant text excerpts
Identify faulty start nodes
Decide on and possibly specify existing faults
Identify best components/ edges for replication
Retrieve instructions for relevant concept and map it to graph manipulation sequence
Database of system description, relevant documents and concepts
Fault propagation
Critical path calculation
Find single points of failure
Graph node/edge replication
...
Response
Generate reply

Figure 2: Outline of the workflow of LLM agent decisions and database interactions. The custom agent runs a cascading decision layer to identify the task type, and a subsequent layer to formulate the task for information retrieval. The latter looks up knowledge from a vectorstore database, or uses functional tools to calculate for example the critical path.

diate representation (IR) of the system to facilitate the comprehension of its components and relationships by the LLM. A custom LLM agent is designed who deploys a hybrid strategy featuring the techniques 1) to 3) above: The agent performs a series of LLM calls to categorize and formalize the task at hand according to predefined concepts. Subsequently, RAG techniques are leveraged to offload structural computations to external functions. We test our method for the example of a simplified automated driving architecture, and present selected experimental results for the tasks of *fault propagation*, *critical path finding*, *single-points-of-failure detection*, and *node replication*. Our work provides the basis for an interactive, LLM-based Human-AI safety codesign framework.

Model

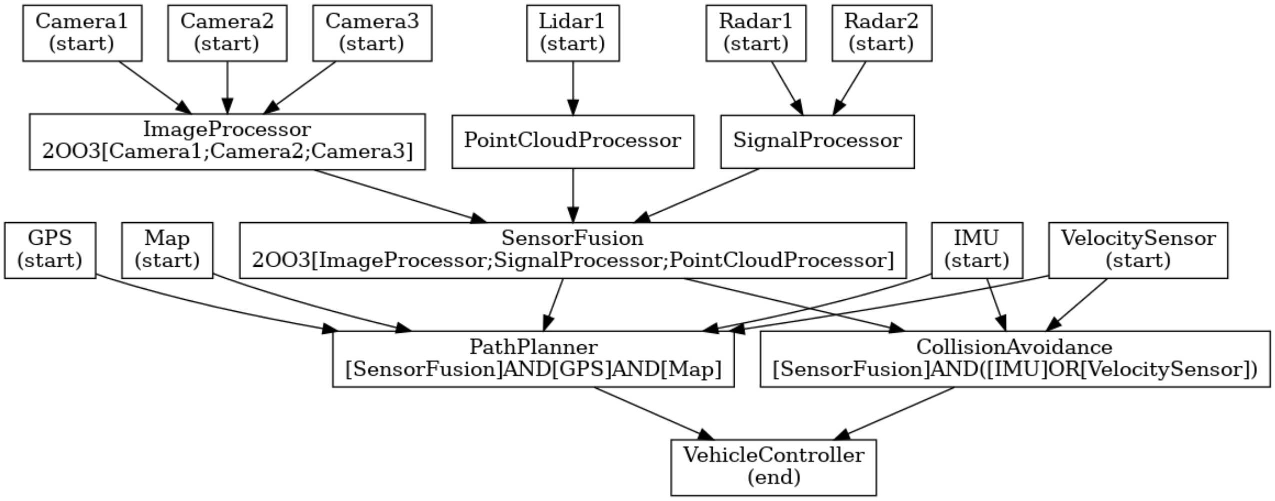
The overall system layout is shown in Fig. 1. Key components of our architecture are explained in the following.

System model and intermediate representation: The system model contains the system architecture, including the system components and their interactions, as well as additional safety-related information required to model fault propagation within the system. To ensure interoperability with established industrial meta model formats, we create the system model with the Eclipse-based OSATE tool (Carnegie Mellon University 2023) and export the resulting *ECore* file as *xml*. The elements of the *xml* model (EClass, EReference, EAttribute) can be directly mapped onto the elements of a generic graph (nodes, edges, attributes). Even though LLMs are capable of reading and interpreting *xml* structures directly, we find that inaccuracies can be reduced when operating with a system description that has a closer resemblance with natural language. Therefore, we further verbalize the *xml* system model to an IR which takes an intuitive list structure:

```
Nodes:
    - Node 1
    - Node 2
    - ...
Edges:
    - Node 1 --> Node 2
    - ...
Attributes:
    - Node 1: Attribute 1
    - ...
```

This IR represents the basis of system-relevant information for the LLM agent. For safety analysis, we assume that each node is subject to possible failure. In order to model the propagation of system faults, we populate graph node attributes with verbalized logic of fault gates (e.g., *AND*, *OR*, *N-out-of-M*) to represent fault trees (Avižienis et al. 2004; Trapp 2016). Further, start and end nodes of the system graph are specified with corresponding attributes. For the current proof-of-concept, we used an integrated system-safety model. However, in model-based safety engineering, long-lasting work has been conducted on integrating system and safety models (Domis and Trapp 2008), which can be used as a scalable basis for further development.

LLM: At the core, we use OpenAI's $GPT3.5 - turbo$ (OpenAI 2023) model for LLM inferences. As our concept-guided approach requires the LLM to solve a series of rather simple micro-decisions, we expect to see in further work, that also much smaller and simpler models such as LLama2 (Touvron et al. 2023) or Mistral-7B (Jiang et al. 2023) suffice for this purpose.



Camera1 (start)
Camera2 (start)
Camera3 (start)
Lidar1 (start)
Radar1 (start)
Radar2 (start)
ImageProcessor 2OO3[Camera1;Camera2;Camera3]
PointCloudProcessor
SignalProcessor
GPS (start)
Map (start)
SensorFusion 2OO3[ImageProcessor;SignalProcessor;PointCloudProcessor]
IMU (start)
VelocitySensor (start)
PathPlanner [SensorFusion]AND[GPS]AND[Map]
CollisionAvoidance [SensorFusion]AND([IMU]OR[VelocitySensor])
VehicleController (end)

Figure 3: Example use case of a simplified automated driving system. The node labels denote the component name (top row) and the fault gate attribute below (if none is given, an AND fault gate of all inputs is assumed). $2OO3$ means that two out of three of the listed inputs are required. Start and end nodes are further labeled explicitly. The figure is a *pydot* visualization of the *ECore* file.

LLM agent: LLMs are conveniently orchestrated by functional wrappers called *agents*, which can be configured to self-induce an iterative chain of thoughts, or to interface automatically to external tools and sources of information. We here use the *LangChain* library (Harrison Chase 2022) to design a customized agent. Importantly, we find that equipping an agent with multiple tools can quickly lead to inaccurate tool usage, unless the verbalized trigger conditions are well-separated in linguistic space. At the same time, for self-induced CoT flows, it is challenging to incorporate and assert procedural safety guardrails.

We therefore implement a different workflow, as illustrated in Fig. 2: The agent cascades the input prompt through a network of micro-decisions, where a single LLM call associates the input with the best match among only 2-4 predefined concepts in each decision. For example, as shown in Fig. 2, in the first decision node the agent associates the input with one of four possible task concepts of *Safety question answering*, *System safety analysis*, *Suggestions for fault-tolerance*, and *Other*. To improve the quality of the micro-decision, we use few-shot learning (Brown et al. 2020) by augmenting the prompts with around ten hand-crafted example triggers for each concept. The last layer of our decision network is a *Task formulation layer*, where the fully identified task concept is used together with the given prompt to refine a specific task prompt. Finally, the external tools or information lookup mechanisms that are provided for this specific task are called and the result is rephrased as a user response. We find that this method overcomes some of the challenges mentioned above: The CoT flow is tightly guided by predefined concepts, yet allows sufficient flexibility in the prompt formulation, and tool triggers cannot overlap since they are strictly associated with specific task concepts.

Database/toolset: We provide a collection of information that is retrieved from a knowledge base, or generated on demand with deterministic functions. This includes, but is not limited to the following: The IR of the system, documentation about safety practices, a function to calculate fault propagation using the fault gates retrieved from the system IR, a function to calculate the critical path using Dijkstra's algorithm (Dijkstra 1959) for the shortest path and start and end nodes from the IR, a function to find the single points of failure, a function to replicate a given node or edge in the model IR. The usage of these tools is triggered in a predefined way depending on the identified task category.

Experiments

We test our model with the use case of a simplified automated driving system modeled in OSATE, see Fig. 3, here visualized with *pydot* (Carrera 2021). Fault gates are implemented as respective node attributes and given in the figure below the node label, for example, the *ImageProcessor* node will turn faulty if at least two out of three ($2OO3$) camera input nodes are faulty. To verify the effectiveness of our concept-guided agent, we test our approach with sample questions in the domain of system safety design. Representative examples and results are given in Tab. 1. In all examples, we find that the tasks have been accurately identified and formulated, and the correct information was retrieved with the appropriate tools. For example, the agent can make suggestions about modifying the graph to improve fault tolerance. In our implementation, this is based on the predefined concept of redundancy and the usage of the tool which finds single points of failures to select the best replication candidate. To modify the system graph, the agent uses the graph replication tool and updates the *xml* model accordingly. We further note that, in the response, the agent leverages general knowledge to relate the functionality of the relevant components with its impact on safety.

Prompt	Response
What happens if Radar1, Radar2 and IMU have a fault?	In this scenario, the faulty components are IMU, Radar1, Radar2, and SignalProcessor. When these components have a fault, it can have several consequences for the system. The IMU (Inertial Measurement Unit) is responsible for measuring the orientation, velocity, and acceleration of the vehicle. If the IMU has a fault, it can lead to inaccurate measurements, which can affect the overall navigation and control of the vehicle. The radars (Radar1 and Radar2) are used for detecting and tracking objects in the environment. If these radars have a fault, it can result in missed or incorrect detections, which can impact the perception and decision-making capabilities of the system. The SignalProcessor is responsible for processing the sensor data and extracting relevant information. If the SignalProcessor has a fault, it can lead to incorrect or incomplete data processing, which can affect the overall performance and reliability of the system. Overall, when these components have a fault, it can lead to degraded performance, reduced safety, and potentially compromised functionality of the system.
Show me the critical path.	The critical path in the given scenario includes the following components: Camera1, Camera2, CollisionAvoidance, GPS, IMU, ImageProcessor, Lidar1, Map, PathPlanner, PointCloudProcessor, SensorFusion, and VehicleController. The critical path represents the sequence of components that are essential for the system's operation and have the longest duration in terms of time. These components are crucial for tasks such as perception, sensor fusion, planning, and control, and any faults or failures in these components can significantly impact the overall performance and safety of the system.
Explain the critical path, given the last fault.	The critical path in this scenario, considering the last reported faulty system components, consists of the following components: Camera1, Camera3, CollisionAvoidance, GPS, ImageProcessor, Lidar1, Map, PathPlanner, PointCloudProcessor, SensorFusion, VehicleController, and VelocitySensor. The critical path represents the sequence of components and processes that are essential for the system's operation and performance. Any delay or failure in these components can significantly impact the overall functionality and reliability of the system. It is important to note that the previously reported system faults, including IMU, Radar1, Radar2, and SignalProcessor, can also affect the critical path and further exacerbate the potential consequences to the system.
What are the single points of failure?	The single point of failure components in the given scenario are PathPlanner, VehicleController, Map, SensorFusion, CollisionAvoidance, and GPS. These components are considered single points of failure because if any of them were to fail, it could result in a complete system failure or significant degradation in the system's performance. For example, if the PathPlanner fails, the system may not be able to generate safe and efficient trajectories for the vehicle. Similarly, if the VehicleController fails, the system may lose control over the vehicle's movements. The Map, SensorFusion, CollisionAvoidance, and GPS components are also critical for accurate perception, situational awareness, and navigation, and their failure can have serious consequences for the overall system functionality.
How would you make my system safer?	To make your system safer, I would suggest performing replication on the component SensorFusion. SensorFusion should be replicated because it is a critical component that relies on multiple inputs from ImageProcessor, SignalProcessor, and PointCloudProcessor. If SensorFusion fails, it can lead to incorrect data being sent to PathPlanner and CollisionAvoidance, compromising the overall safety of the system. By replicating SensorFusion, the system can continue to function even if one instance of SensorFusion fails, ensuring that accurate and reliable data is provided to the other components.

Table 1: Selected example prompts and responses. Refer to Fig. 3 for the system description. The example in row three references the agent's memory, as *last fault* refers to the last mentioned system fault in the chat history.

Conclusion and Outlook

Our approach of a concept-guided LLM agent overcomes two challenges typically encountered when solving generative text tasks on structured data: 1) Chain-of-thought processes derail and no longer comply with a set of required rules, or miss their goals, 2) Trigger conditions to retrieve external information via tools become inaccurate when the number of possible options increases. Instead, we here guide the thought process with a series of micro-decisions to ensure that the right tool triggers are met. We envision to extend our setup to more concepts and larger decision cascades, such that more complex tasks can be tackled. Our work provides the basis for an interactive framework where LLMs assist in Human-AI safety codesign.

Acknowledgments

This work was funded by the Bavarian Ministry for Economic Affairs, Regional Development and Energy as part of a project to support the thematic development of the Institute for Cognitive Systems.

References

Avižienis, A.; Laprie, J. C.; Randell, B.; and Landwehr, C. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1): 11–33.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020-Decem.

Carnegie Mellon University. 2023. OSATE 2.13. https://osate.org/. Accessed: 2023-12-01.

Carrera, E. 2021. pydot. https://pypi.org/project/pydot/. Accessed: 2023-12-01.

Diemert, S.; and Weber, J. H. 2023. Can Large Language Models Assist in Hazard Analysis? In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 14182 LNCS, 410–422. ISBN 9783031409523.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numer. Math.*, 271: 269–271.

Domis, D.; and Trapp, M. 2008. Integrating Safety Analyses and Component-Based Design. In Harrison, M. D.; and Sujan, M.-A., eds., *Computer Safety, Reliability, and Security*, 58–71. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-87698-4.

Harrison Chase. 2022. LangChain. https://github.com/langchain-ai/langchain. Accessed: 2023-12-01.

Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; de las Casas, D.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; Lavaud, L. R.; Lachaux, M.-A.; Stock, P.; Scao, T. L.; Lavril, T.; Wang, T.; Lacroix, T.; and Sayed, W. E. 2023. Mistral 7B. arXiv:2310.06825.

Jin, B.; Liu, G.; Han, C.; Jiang, M.; Ji, H.; and Han, J. 2023. Large Language Models on Graphs: A Comprehensive Survey. arXiv:2312.02783.

OpenAI. 2023. ChatGPT 3.5-turbo. https://openai.com/. Accessed: 2023-12-01.

Pan, S.; Luo, L.; Wang, Y.; Chen, C.; Wang, J.; and Wu, X. 2023. Unifying Large Language Models and Knowledge Graphs: A Roadmap. arXiv:2306.08302.

Reuters. 2023. ChatGPT sets record for fastest-growing user base - analyst note. https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/. Accessed: 2023-12-01.

Tang, J.; Yang, Y.; Wei, W.; Shi, L.; Su, L.; Cheng, S.; Yin, D.; and Huang, C. 2023. GraphGPT: Graph Instruction Tuning for Large Language Models. arXiv:2310.13023.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; Bikel, D.; Blecher, L.; Ferrer, C. C.; Chen, M.; Cucurull, G.; Esiobu, D.; Fernandes, J.; Fu, J.; Fu, W.; Fuller, B.; Gao, C.; Goswami, V.; Goyal, N.; Hartshorn, A.; Hosseini, S.; Hou, R.; Inan, H.; Kardas, M.; Kerkez, V.; Khabsa, M.; Kloumann, I.; Korenev, A.; Koura, P. S.; Lachaux, M.-A.; Lavril, T.; Lee, J.; Liskovich, D.; Lu, Y.; Mao, Y.; Martinet, X.; Mihaylov, T.; Mishra, P.; Molybog, I.; Nie, Y.; Poulton, A.; Reizenstein, J.; Rungta, R.; Saladi, K.; Schelten, A.; Silva, R.; Smith, E. M.; Subramanian, R.; Tan, X. E.; Tang, B.; Taylor, R.; Williams, A.; Kuan, J. X.; Xu, P.; Yan, Z.; Zarov, I.; Zhang, Y.; Fan, A.; Kambadur, M.; Narang, S.; Rodriguez, A.; Stojnic, R.; Edunov, S.; and Scialom, T. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Trapp, M. 2016. Assuring Functional Safety in Open Systems of Systems. https://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-44221.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-Decem: 5999–6009.

Wang, H.; Feng, S.; He, T.; Tan, Z.; Han, X.; and Tsvetkov, Y. 2023. Can Language Models Solve Graph Problems in Natural Language? arXiv:2305.10037.