Integrating LLMs With SatSim for Enhanced Satellite Tracking and Identification

Enrique De Alba, Marco de Lannoy Kobayashi, Alexander Cabello
EO Solutions Corp

ABSTRACT

This study introduces SatSim Chat, an innovative system integrating Large Language Models (LLMs) with SatSim, a space scene simulator, to democratize Space Domain Awareness (SDA) through enhanced accessibility. SatSim Chat employs an agent-based architecture built on LangGraph, enabling natural language interaction for complex space simulations. We evaluate various configurations, comparing OpenAI's Structured Outputs and the Instructor library for JSON generation. Experiments across six difficulty levels demonstrate that GPT-4o-mini configurations achieve 70.41% accuracy in translating user inputs into correct SatSim configuration fields, with notable resilience to temperature variations. These findings challenge assumptions about LLM behavior in structured tasks and suggest potential applications beyond SDA. Future research should explore diverse models and include user studies with SDA professionals to further advance the democratization of SDA tools.

1. INTRODUCTION

Space Domain Awareness (SDA) is vital for ensuring the safety and security of space-based assets. However, many existing SDA software tools demand significant technical expertise, which limits their accessibility. As the space domain becomes increasingly congested and contested, SDA tools must be powerful and user-friendly, allowing a wider range of users, including operators, analysts, and decision-makers without deep technical backgrounds, to engage with them effectively. Enhancing the usability of these tools can facilitate better collaboration, improve situational awareness, and enable more timely and informed decision-making across a diverse user base.

Large Language Models (LLMs) have significantly advanced the field of natural language processing, demonstrating unprecedented capabilities in comprehending and generating human-like text. These models, exemplified by architectures like ChatGPT and GPT-4, leverage the Transformer framework and are trained on extensive corpora, enabling them to perform a diverse array of linguistic tasks with remarkable precision and fluency [2]. A key attribute of LLMs is their capacity to simplify complex interfaces, thus enhancing the accessibility of sophisticated systems to users lacking specialized technical expertise.

In the context of SDA, our research demonstrates that LLM-powered interfaces, such as SatSim Chat, allow users to configure and execute simulations by articulating scenarios in natural language, which the model then translates into appropriate configuration parameters. This approach addresses the need for more accessible SDA tools by leveraging the capabilities of LLMs to interpret natural language inputs and translate them into structured commands that are comprehensible to complex systems.

This functionality extends across various domains, including database management, healthcare, finance, and, as our research demonstrates, space domain awareness [9, 14]. By integrating LLMs, developers can create intuitive interfaces that facilitate user interaction with intricate tools and simulations through conversational language, mitigating the need for users to acquire specialized syntax or command knowledge.

Our approach enhances user experience and simultaneously improves operational efficiency. It achieves these outcomes by reducing configuration errors and lowering barriers to entry for advanced technologies. While general research on LLM trustworthiness, as discussed by Li et al. [12], provides a foundation for ensuring reliable AI interactions, our work extends these principles specifically to the domain of space domain awareness. By integrating LLMs with specialized tools like SatSim, we show how the general capabilities of language models can be harnessed to address complex, domain-specific challenges in SDA simulations.

SatSim Chat is our solution to democratize access to SDA simulation and data generation, leveraging the power of LLMs. This approach aligns with recent research exploring the application of LLMs in spacecraft autonomy and

mission planning [7]. While challenges remain in scaling LLM reasoning to complex missions, our work with SatSim Chat demonstrates the potential of LLMs to enhance decision-making and autonomy in SDA. The integration of LLMs into scientific practices raises important ethical considerations, including concerns about accuracy, reliability, and the potential impact on scientific discovery [1]. While our research primarily focuses on the technical implementation and performance of SatSim Chat, we acknowledge these concerns and have implemented measures such as human-in-the-loop workflows to address some of these challenges.

2. BACKGROUND AND RELATED WORK

2.1 Space Domain Awareness and Simulation

Simulation is crucial for SDA because it allows for the comprehensive testing and evaluation of systems in a controlled environment, covering scenarios that real sensor data alone cannot address. This includes modeling and analyzing complex and dynamic threats, such as anti-satellite (ASAT) capabilities, within a synthetic environment that can simulate interactions between various systems and sensors. Through simulation, SDA systems can be rigorously assessed for their performance, resilience, and operational effectiveness in detecting, tracking, and responding to potential space threats, ensuring that they meet the challenges posed by near-peer adversaries and contribute effectively to space control missions. Moreover, simulation provides a critical avenue for developing and refining new technologies, operational strategies, and system integration, ultimately enhancing the United States' ability to maintain space superiority.

2.2 SatSim

SatSim is an open-source high-fidelity, GPU-accelerated space scene simulator designed to generate synthetic electro-optical imagery for training and evaluating machine learning models in SDA. It allows for the rapid creation of large volumes of labeled synthetic data, which is particularly valuable for scenarios that are rare or difficult to capture in real life, such as satellite breakups and collisions. SatSim's architecture is modular and highly configurable, supporting various modes and settings, such as 3D projection and 2D image plane simulations, and it can simulate a wide range of phenomena, including the effects of atmospheric turbulence and sensor noise. By leveraging the computational power of TensorFlow, SatSim integrates seamlessly into machine learning workflows, enhancing the performance of models trained with both real and synthetic data. Its ability to generate annotated images for rare events makes it an essential tool for improving the reliability of space object detection algorithms, especially in edge cases where real data is scarce [3]. Fig. 1 illustrates the high fidelity of SatSim's synthetic imagery by comparing a real telescope image with a SatSim-generated synthetic image.

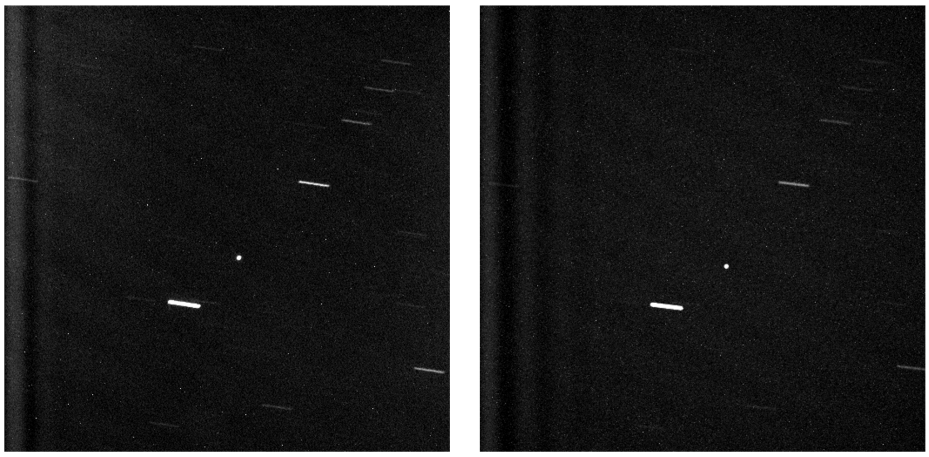

Fig. 1: Comparison of a real image from a Raven telescope (left) and a SatSim-generated synthetic generated (right).

SatSim can be accessed through a command line interface and JSON configuration files, or via a Python modular interface. However, using these interfaces requires familiarity with the SatSim JSON schema or Python API, along with domain-specific knowledge in areas like astrodynamics and electro-optical systems. This complexity can limit its accessibility and ease of use for broader audiences.

2.3 LLMs as SDA Simulation Interfaces

To address this accessibility challenge, we propose leveraging LLMs as an interface between users and SatSim. LLMs have demonstrated remarkable capabilities in natural language understanding and generation across diverse technical domains [15, 13]. Their ability to process and generate human-like text makes them ideal candidates for bridging the gap between complex technical systems and non-expert users.

In the context of SDA, LLMs offer a promising solution to democratize access to sophisticated simulation tools like SatSim. By acting as an intermediary, LLMs can interpret natural language queries, translate them into appropriate SatSim configurations, and explain the results in user-friendly terms. This approach aligns with recent research exploring the application of LLMs in spacecraft autonomy and mission planning [7].

The integration of LLMs with specialized tools has shown promise in other technical domains. For instance, the integration of ChatGPT with the PX4/Gazebo simulator for natural language-based drone control [13] demonstrates the potential of LLMs to simplify complex technical interfaces. Our research extends this concept to SDA tools and simulations, specifically focusing on enhancing the accessibility and usability of SatSim.

By combining the high-fidelity simulation capabilities of SatSim with the natural language processing power of LLMs, we aim to create a more intuitive and user-friendly interface for SDA simulations. This integration has the potential to significantly lower the barrier to entry for SDA research and operations, enabling a wider range of users to leverage SatSim's capabilities effectively.

3. SATSIM CHAT: SYSTEM ARCHITECTURE AND DESIGN

3.1 Overview

SatSim Chat's architecture (Fig. 2) integrates LLMs with SatSim, facilitating natural language interactions for space simulations. The system comprises four primary components: the user interface, the LLM-driven agent, the tool ecosystem, and the SatSim interface.

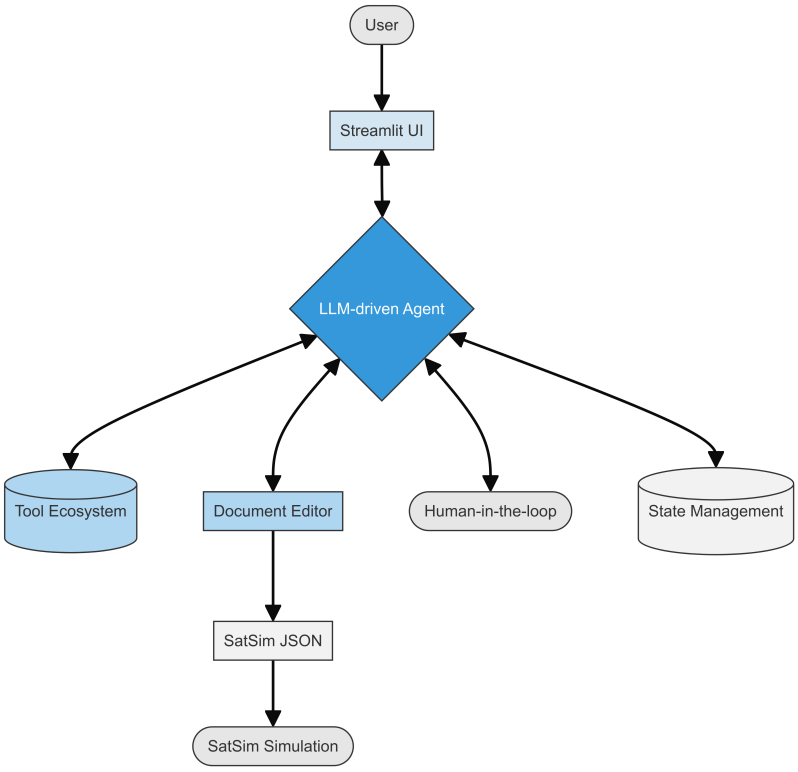

Fig. 2: The architecture of SatSim Chat.

The user interface, implemented using Streamlit, provides an intuitive chat-based interaction platform. Users input natural language queries or commands, which are then processed by the LLM-driven agent. This agent, built on the LangGraph framework, orchestrates the conversation flow and decision-making process.

The LLM-driven agent interfaces with an array of specialized tools, each meticulously designed to manage distinct facets of the simulation configuration. This toolkit encompasses functions for extracting sensor metadata, target information, and SatSim-specific parameters. The modular architecture not only augments the system's extensibility but also serves as a critical mechanism for mitigating the risk of LLM hallucinations [5, 11]. By delegating quantitative operations to deterministic functions, the system effectively addresses potential "factual mirages," particularly in the "numeric nuisance" category, as explained by Rawte et al. [11]. This approach aligns with the recommendations for enhancing the reliability of LLM outputs in domain-specific applications [5], thus fortifying the system's robustness in the context of SDA simulations.

The SatSim Chat interface serves as the bridge between the LLM-driven agent and the SatSim simulation environment. It translates the agent's outputs into SatSim-compatible JSON configurations, enabling seamless execution of user-specified scenarios.

The system's architecture incorporates checkpointing mechanisms for state management and supports human-in-the-loop interventions, allowing for expert oversight and correction when necessary. This design ensures robustness and reliability in generating accurate simulation configurations while maintaining an accessible user experience.

3.2 LLM Integration with SatSim

The integration of LLMs with SatSim represents a significant advancement in the accessibility and usability of simulation tools for SDA. This integration is achieved through a sophisticated architecture that leverages LangGraph for conversation management, LangChain for tool interfaces, and a custom JSON translation layer for SatSim interaction.

3.2.1 LangGraph for State Management

SatSim Chat's conversation management system leverages LangGraph, a graph-based framework that extends existing language model capabilities. This system employs a state graph to define the conversational flow, with nodes representing key operational stages including natural language processing, tool execution, human expert intervention, and document manipulation.

The state of the conversation is encapsulated in a comprehensive data structure that encompasses message history, the current working simulation document, and various control flags. This approach to state management facilitates a context-aware conversation flow, which is essential for navigating the complexities inherent in SDA simulation scenarios.

The conversation graph is constructed programmatically, with each node corresponding to a distinct operational phase. The edges between these nodes are defined conditionally, allowing for adaptive routing based on the current state and user inputs. This flexible architecture, enabled by LangGraph, allows the system to seamlessly transition between different modes of operation as required by the evolving conversation and simulation needs.

This implementation features a persistent storage mechanism for checkpointing via SQLite. This allows for the preservation of conversation state across multiple sessions, enhancing the system's robustness and enabling long-term interaction continuity. Such persistence is particularly valuable in the context of complex, multi-stage simulation setups that may extend over prolonged periods.

3.2.2 LangChain Tools for SatSim Chat

The interface between the LLM and SatSim's JSON API is facilitated through a comprehensive set of LangChain tools. These tools, as outlined in Table 1, provide a structured method for retrieving and manipulating simulation parameters. The table presents all the main tools utilized in SatSim Chat, encompassing functionalities such as retrieving metadata for stars, backgrounds, and sensors, fetching target information, calculating satellite pass events, and performing web searches.

These tools abstract the complexities of the SatSim JSON structure, allowing the LLM to interact with the simulation

Tool	Description
get_stars()	Retrieves star catalog metadata from predefined dictionary.
get_background()	Fetches background metadata for stray light and brightness.
get_sim()	Obtains general simulation settings and parameters.
get_sensor(sensor_name)	Retrieves sensor-specific metadata based on sensor name.
get_target(target_id)	Fetches target satellite information by target ID.
get_target_name(target_name)	Retrieves target metadata based on target name.
get_celestrak_data(target_name)	Fetches Celestrak TLE data for given target.
satellite_pass_events(...)	Calculates satellite pass events over a location.
collision_from_tle(...)	Generates two-object collision configuration from TLE data.
breakup_from_tle(...)	Simulates satellite breakup event from TLE data.
get_current_time()	Retrieves and formats current time as string.
get_website_text(url)	Extracts general text content from given website URL.
web_search(query)	Performs web search and returns first parseable result.

Table 1: Summary of tools used by SatSim Chat

environment through high-level, semantically meaningful operations. The tool outputs are structured to align with the SatSim parameters schema, ensuring type safety and validation at the interface level.

3.2.3 User Input Processing

The translation of user input into SatSim commands is orchestrated through a sophisticated multi-stage process driven by a LLM agent. This process initiates with a natural language interpretation module that contextualizes user input against the current conversation state and the structured schema of SatSim documents. The LLM is primed with a carefully designed prompt that delineates its role as a SatSim assistant and outlines the protocol for requesting document edits.

When document modifications are necessary, a specialized editing module is activated. This module employs one of two approaches to interpret the LLM's output and apply updates to the SatSim document: either OpenAI's Structured Outputs or the Instructor library. The choice between these methods represents a trade-off between comprehensive schema adherence and granular control over the update process. The OpenAI method ensures holistic conformity to the predefined schema, while the Instructor approach segments the update process into discrete top-level fields (e.g., simulation parameters, focal plane array characteristics, background settings, and geometry specifications). This segmentation, while potentially introducing complexity, allows for fine-grained management of the update process and circumvents certain API constraints. In the Instructor method, each field is updated through individual LLM invocations, with schema validations enforced via a custom-patched API client. For a more in-depth overview of these methodologies, please refer to Section 3.3.4.

The resulting structured document, encapsulating the simulation parameters, is then transmitted to the SatSim simulation engine. This engine interprets the document, transforming the parameters into a comprehensive set of simulation instructions. These instructions guide the generation of satellite imagery and associated metadata, forming the core output of the simulation process. Fig. 3 shows an example interaction between the user and SatSim Chat, along with the resulting satellite imagery produced by SatSim.

The presentation of results is facilitated through a web-based user interface, which provides a dual display of the ongoing conversation history and the current state of the SatSim JSON document. A document rendering module serializes the simulation configuration for display, offering users real-time visual feedback on the evolving simulation parameters.

This architecture represents a novel approach to human-AI collaboration in the domain of SDA simulation. It leverages the natural language processing capabilities of large language models while incorporating domain-specific tools for precise management of simulation parameters. The system's capacity to maintain contextual awareness throughout the conversation, handle intricate parameter adjustments, and provide instantaneous feedback marks an advancement in

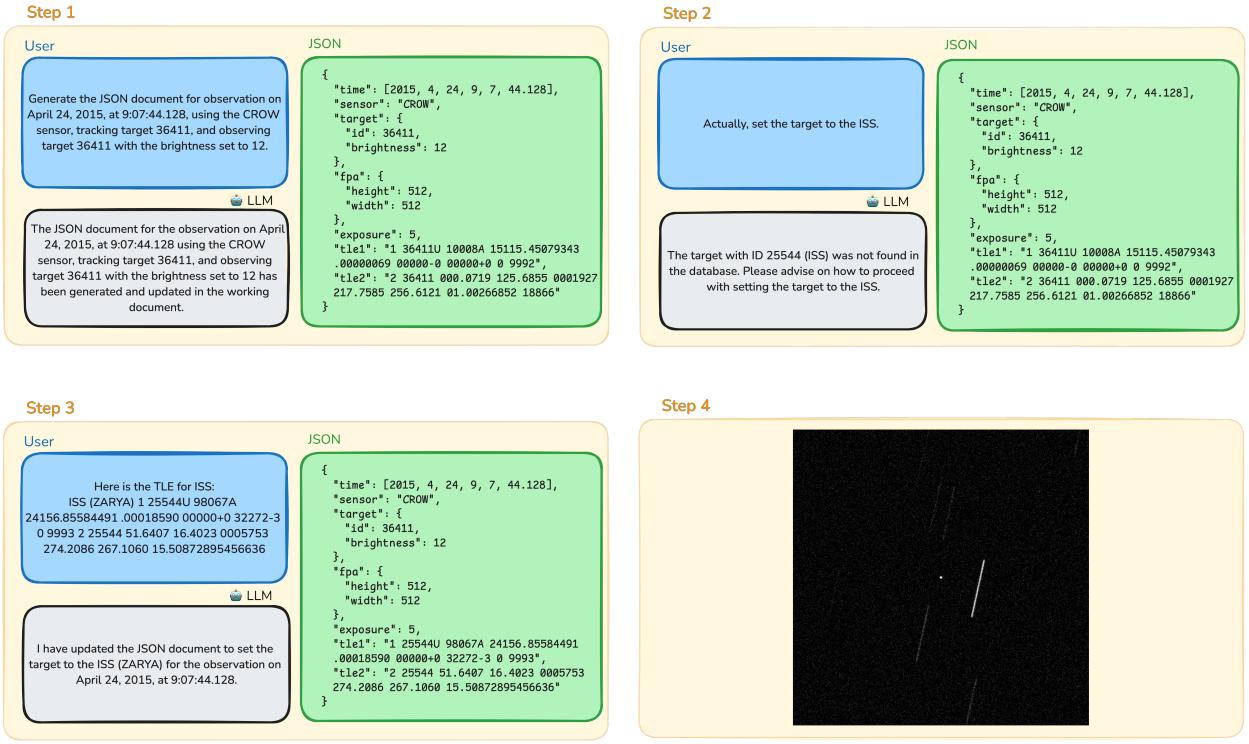

Fig. 3: Step-by-step user interaction with SatSim Chat

the accessibility of space simulation tools. This integration of AI-driven natural language interaction with specialized simulation software opens new avenues for efficient and intuitive SDA modeling.

3.3 Key Design Principles

Our implementation of SatSim Chat incorporates several key design decisions to enhance robustness and usability, with a particular focus on maintaining system integrity and facilitating complex interactions in the context of SatSim.

3.3.1 Agent-based Architecture

Our first key design decision focuses on the overall system architecture, employing an agent-based approach to enhance robustness and flexibility. The agent-based architecture employed in SatSim Chat leverages the LangGraph framework to encapsulate domain-specific operations within discrete functional units. This design significantly reduces the risk of hallucination, a common concern with LLMs, by delegating quantifiable domain-specific tasks to specialized, non-heuristic procedures.

The architecture is implemented through a StateGraph object, which defines the structure of our chatbot as a state machine. Nodes in this graph represent LLM invocations and domain-specific functions, while edges specify transitions between these functions. For example, the ‘chatbot’ node handles LLM interactions, while the ‘editor’ node manages document updates using strongly-typed Pydantic models. This separation of concerns allows for precise control over the flow of information and actions within the system.

A key benefit of this approach is the ability to integrate domain-specific tools seamlessly. Functions like `get_sensor` and `get_target` serve as interfaces to predefined catalogs, ensuring that the LLM operates with accurate, up-to-date information rather than potentially outdated or incorrect data from its training set.

3.3.2 Persistent State Management

To facilitate complex, multi-turn interactions crucial for our application, we utilize a persistent state management using SQLite through LangGraph's SqliteSaver. This enables our system to save and resume conversation states, maintaining context across multiple interactions and even system restarts.

The SQLite-based checkpointing mechanism, initially implemented with an in-memory database, is designed for seamless adaptation to persistent storage in production environments. This feature is crucial for maintaining conversation state across multiple interactions, allowing for interruption and resumption of the dialogue flow, particularly when human expertise is required.

Our architecture incorporates a sophisticated human-in-the-loop paradigm, which is essential for maintaining the accuracy and reliability of complex space simulation scenarios like those pertinent to SatSim. The system is designed to recognize its limitations and escalate queries to human experts when encountering scenarios beyond its current knowledge. This approach ensures that the LLM-driven interface collaborates effectively with users to define and refine simulation parameters. Please refer to Section 3.3.5 for a more in-depth overview.

The persistent state management system encapsulates both the conversation history and the evolving simulation configuration, represented as a nested data structure. This design facilitates the incremental refinement of simulation parameters across multiple interactions, a critical feature for developing complex space scenarios for SatSim. By integrating checkpointing capabilities with a mechanism for expert intervention, the system achieves a high degree of accuracy and flexibility in handling intricate space simulation tasks, while mitigating the risk of erroneous extrapolations or hallucinations.

SatSim Chat achieves a robust, flexible system capable of handling the intricate requirements of SatSim while maintaining a user-friendly interface. The agent-based architecture ensures accurate domain-specific operations, while the persistent state management facilitates complex, multi-turn interactions necessary for comprehensive scenario development.

3.3.3 Interactive Chat Interface

The SatSim Chat interface interprets user inputs and generates contextually relevant responses. This enables the translation of complex space domain scenarios into structured simulation parameters for SatSim. The interface dynamically adapts its query strategy based on the current state of the simulation configuration, identifying and addressing information gaps. To enhance user comprehension, the system incorporates visual feedback mechanisms, rendering real-time updates of the SatSim document as a JSON. This allows users to verify the results of their inputs. The interface also implements a sophisticated error handling system, providing clear, domain-specific explanations for invalid inputs or inconsistencies in the simulation parameters. This approach simultaneously educates users about SatSim's constraints and guides them towards constructing valid and meaningful simulation scenarios. Consequently, it effectively bridges the gap between natural language interaction and technical SDA requirements.

3.3.4 Structured LLM Outputs

To ensure consistent and reliable outputs from the LLM, our implementation leverages OpenAI's Structured Outputs feature [8]. This approach makes use of a context-free grammar (CFG) derived from a predefined JSON schema, which constrains the model's output to conform to the specified structure. Specifically, we utilize a Pydantic model of the SatSim parameters as the schema, encompassing the nested structure of SatSim configurations. Pydantic is a Python library that leverages Python type annotations to perform data validation and parsing [4]. It allows for defining the data model via Python model classes and automatically generates the respective JSON Schema. Virtually every LLM Python framework containing structured output utilities integrates with Pydantic.

The CFG-based decoding technique used in OpenAI's Structured Outputs determines valid tokens during the generation process, effectively eliminating the possibility of malformed JSON outputs. This method not only guarantees syntactic correctness but also ensures semantic adherence to the SatSim parameter specifications. By employing this technique, we mitigate the risk of hallucinated or inconsistent outputs. This can be particularly challenging for complex schemas like those of SatSim. Furthermore, this approach facilitates the integration of LLM-generated content

into our system's workflow, as the structured output can be directly parsed and utilized without intermediate validation or correction steps.

In addition to OpenAI's Structured Outputs, we explored an alternative method using the Instructor library [6]. Instructor attempts to provide structured output guarantees using Pydantic combined with a retry mechanism. Specifically, Instructor prompts the LLM to generate outputs conforming to a Pydantic model, validates the outputs against the Pydantic model, and if the output does not meet the schema's requirements, the generation process is retried automatically, transparent to the user. This ensures one of two outcomes: the final output adheres to the required structure; or an error is raised explicitly notifying that the LLM model has failed to generated proper output, which is preferable to the workflow continuing with a latent malformed LLM output.

The instructor-based approach decomposes the complex SatSim document into more manageable sub-models, addressing challenges with processing the entire structure simultaneously. Attempting to process the entire SatSim document in a single request results in an API error. We avoid the error by iterating through each sub-model (Sim, Fpa, Background, Geometry), generating more targeted LLM prompts and parsing the responses using their corresponding Pydantic models. This method offers increased robustness and flexibility due to reducing the scope of the task per request for the LLM model, allowing for efficient updates to specific document sections. However, it introduces additional complexity and potential for inconsistencies between sub-models, since the LLM model no longer receives the full schema. Moreover, since we're not parsing the entire schema at once, processing issues with any sub-model could result in missing chunks of the schema, an issue entirely avoided by OpenAI's Structured Outputs approach.

Here we show an example of a baseline SatSim schema generated by our system in response to the prompt: "Generate the SatSim parameters for September 18, 2024, at 16:45:00.000, tracking no target and observing no targets". This shows the structural integrity and semantic coherence maintained by our methods. The listing showcases how SatSim Chat correctly interprets temporal information, initializes simulation parameters, and appropriately represents the absence of tracking and observation targets. Notably, both OpenAI's Structured Outputs method and the Instructor approach generate identical schemas given this prompt.

Listing 1: Example Baseline SatSim Schema

```
{
  "version": {
    "version": 1
  },
  "sim": {
    "mode": "fftconv2p",
    "spacial_osf": 15,
    "temporal_osf": 100,
    "padding": 100,
    "samples": 1
  },
  "fpa": {},
  "background": {
    "stray": {
      "mode": "none"
    },
    "galactic": 19.5
  },
  "geometry": {
    "time": [2024, 9, 18, 16, 45, 0.0],
    "stars": {
      "mode": "sstr7",
      "path": "/app/server/sstrc7",
      "motion": {
        "mode": "none"
      }
    },
    "obs": {
```

```
      "mode": "list",
      "list": []
    }
  }
}
```

3.3.5 Guardrails for LLM Output

While LLMs have revolutionized natural language processing and enabled promising integration across various domains, developing effective LLM applications capable of meeting the stringent reliability requirements of mission-critical applications remains a significant challenge, in large part due to the inherent unpredictability and occasional unreliability of LLM outputs. For example, LLMs may generate hallucinations—plausible-sounding but factually incorrect or irrelevant information—or produce subtle inaccuracies in output that otherwise appears correct. They may also generate content that is incoherent or inconsistent with the desired format or domain-specific context, all of which undermine the dependability required for critical applications.

Given these challenges, the assumption when building LLM applications should be that the models themselves are unreliable. As such, designing a reliable LLM agent necessitates implementing robust guardrails around the model to ensure the generated content aligns with the intended purpose, accuracy, and quality standards of the application. In this context, guardrails refer to strategies and mechanisms designed to constrain, validate, and guide the output of the model, thereby avoiding or significantly mitigating the outlined reliability issues inherent to LLMs.

During our early experimentation with SatSim Chat, we realized that even the most advanced LLM models struggled to reliably generate the SatSim JSON input beyond very basic scenarios. To address this limitation, we designed SatSim Chat to support an iterative, human-in-the-loop workflow, with the user contributing both in validation and assistance to the model, thereby acting as a critical guardrail.

Unlike a one-shot prompt-to-SatSim image generation approach, which would result in wasted time and compute generating invalid images (if images could be generated at all), our human-in-the-loop workflow involves an "editor" node responsible for iteratively refining a "working document"—the SatSim parameters JSON object. The iterative process allows for human oversight and incremental adjustments, ensuring that the final simulation parameters are accurate and well-aligned with the user's intent. This method leverages human judgment as a critical guardrail, mitigating the risk of erroneous or inappropriate output by enabling continuous validation and refinement. By involving the user at critical steps *in the loop*, the design ensures that the LLM's suggestions are filtered, corrected, and augmented with human expertise.

A critical element of the human-in-the-loop workflow is the integration of a "human" node, which interfaces the user with the LLM as a tool within the LLM's operational loop and serves as another guardrail. This node allows the user to provide necessary information, clarification, or corrections to the LLM in real-time, effectively guiding the model's behavior. The ability to interact dynamically with the LLM prevents the model from diverging into irrelevant or incorrect outputs and ensures that the final JSON object matches the intent of the user. This design enables a kind of "interactive guardrails," where the model's outputs are continuously shaped and constrained by human input, making it more adaptive and resilient to terminal errors.

Generating valid JSON that complies with the SatSim parameter schema is essential for ensuring the LLM's output can be directly used by the simulation tool. This compliance is the key to bridging the gap between the LLM's suggestions and the accurate, reliable simulated image generation with SatSim. As outlined in 3.3.4, both OpenAI's Structured Outputs and the Instructor-based approach provide structured output guarantees that are critical for maintaining this compliance. By integrating strong validation mechanisms and human oversight, SatSim Chat effectively mitigates the inherent unreliability of LLMs, ensuring that outputs are both structurally and semantically correct, thereby enhancing the practical utility of SatSim in user workflows.

4. EVALUATION

4.1 Experimental Setup

To evaluate the performance and capabilities of SatSim Chat, we designed a comprehensive experimental framework. This setup simulates a range of user interactions with the system, varying in complexity and domain-specific requirements.

Our experiments comprise a series of test cases, each consisting of one or more interaction steps. These test cases are stratified into six difficulty levels, allowing for a nuanced assessment of the system's performance across varying degrees of complexity. Each step in a test case includes the user prompt, the expected JSON output, and a specification of critical fields to monitor for changes. Table 2 provides an overview of these difficulty levels along with example prompts:

Table 2: Difficulty Levels

Level	Description	Example Prompt
1	Modify specific JSON fields explicitly specified by the user.	Change the "exposure" time from 5.0 to 10.0 in the JSON configuration.
2	Retrieve information from a local database or memory and populate the appropriate fields in the JSON.	Retrieve and fill in sensor characteristics, such as field of view and noise characteristics, based on the sensor name provided by the user.
3	Modify fields not explicitly specified by the user, inferring the required changes based on a short user prompt.	Adjust sensor parameters based on terms like "megapixel" without the user explicitly specifying the fields to modify.
4	Modify fields based on a more complex and detailed user prompt, requiring inference across multiple aspects of the configuration.	Adjust sensor specifications based on a manufacturer's specification sheet.
5	Requires the use of a general-purpose tool, such as web search, to fulfill the user's request.	Retrieve TLE data from an external public catalog like CelesTrak.
6	Requires the use of domain-specific tools, such as a physics-based astrodynamics library, to fulfill the user's request.	Import TLE data and propagate the satellite's orbit to generate a visibility trajectory.

The primary evaluation metrics focus on the system's ability to generate and modify JSON documents representing SatSim simulation parameters. We analyze the differences between the generated and expected JSON outputs, tracking various metrics, including the number and types of differences (e.g., missing keys, unexpected keys, value mismatches), as well as the system's ability to make appropriate modifications in multi-step scenarios.

To simulate realistic usage patterns, we leveraged a diverse set of external tools, as detailed in Table 1. This toolkit extends beyond the core simulation capabilities, enabling the system to handle a wide range of user queries that require real-time data or complex calculations. By integrating these tools, we can use a more dynamic and responsive environment that closely mimics real-world scenarios in satellite operations and space situational awareness (SSA). The inclusion of web search capabilities allows the system to provide up-to-date information, enhancing the contextual relevance of responses. This integration broadens the scope of possible interactions and also tests SatSim Chat's ability to incorporate external data into its decision-making process.

We investigate the following variables as different configurations in our experiments:

1. The choice of language model (e.g., GPT-4o-mini vs. GPT-3.5-turbo) for managing tools
2. Temperature settings for the language models
3. JSON parsing and generation methods (OpenAI's beta parser vs. Instructor)

We developed custom metrics to assess the integrity of the generated JSONs and to analyze changes between consecutive steps in multi-prompt scenarios. These metrics provide insights into SatSim Chat's ability to maintain overall structure while making the necessary modifications.

The test cases are designed to cover a wide spectrum of scenarios, ranging from simple parameter updates to complex multi-step processes involving web searches and integration of external data. This allows us to evaluate the technical accuracy of the system and its practical applicability in SDA.

Our experimental framework incorporates a data collection mechanism that records detailed performance metrics for each test case and step. This approach enables longitudinal analysis of the system's performance across various task types and difficulty levels. By maintaining a structured record of performance data, we can track the system's behavior over time, identify trends in its capabilities, and assess its effectiveness over our test suite.

4.2 Results and Analysis

In assessing SatSim Chat's performance, we developed a metric quantifying the accuracy of generated SatSim configurations: the percentage of correct critical fields. This metric evaluates the proportion of essential, user-input-dependent fields in the generated JSON that align with the ground truth. Critical fields are dynamically identified based on user requirements; for example, in satellite tracking scenarios, these would include sensor specifications and temporal constraints. This approach allows for a nuanced evaluation of the system's ability to translate natural language inputs into precise, task-specific SatSim configurations, thus assessing the fidelity of human-AI interaction within the SatSim Chat framework (see Fig. 4).

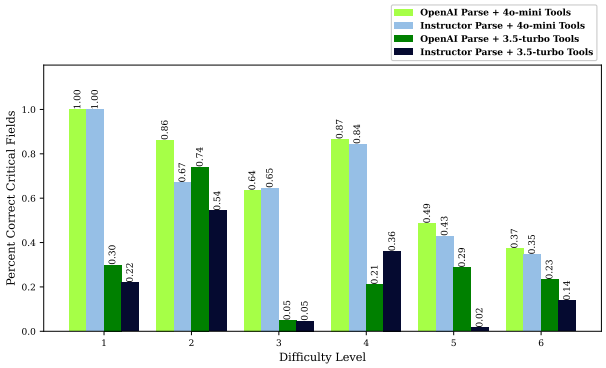

Fig. 4: JSON Critical Fields Correctness

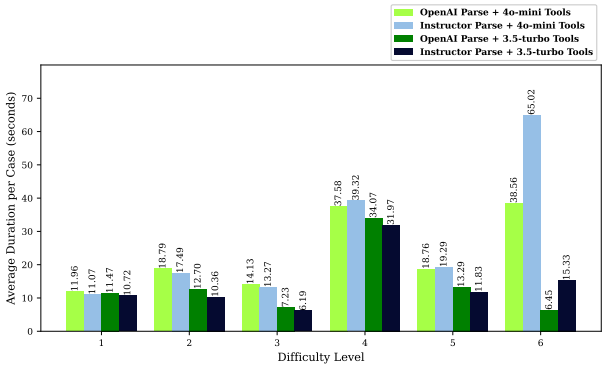

Fig. 5: Average Duration per Test Case

Fig. 4 illustrates the performance of four distinct SatSim Chat configurations across six increasing difficulty levels. These configurations represent different combinations of parsing methods and language models: OpenAI's Structured Outputs or the Instructor library for parsing, paired with either GPT-4o-mini or GPT-3.5-turbo for managing tools. This design allows us to evaluate the impact of both parsing method and language model on system performance. The OpenAI parser leverages a context-free grammar derived from a predefined JSON schema, while Instructor prompts LLMs to generate Pydantic-model-compliant outputs, validating and auto-retrying if necessary. GPT-4o-mini and GPT-3.5-turbo serve as the underlying language models for interpreting user inputs and managing the tool ecosystem.

Statistical analysis shows significant performance differences among the tested configurations. The OpenAI Parse with GPT-4o-mini Tools configuration demonstrated superior performance, achieving a mean accuracy of 70.41% with a standard deviation of 24.51% across all difficulty levels. [1] To assess the statistical significance of the observed differences, we conducted a one-way ANOVA, yielding an F-statistic of 6.906 and a p-value of 0.00225, indicating a statistically significant difference among the group means at the $\alpha = 0.05$ level.

Post-hoc analysis using Tukey's Honest Significant Difference (HSD) test revealed specific pairwise differences. The contrast between GPT-4o-mini and GPT-3.5-turbo configurations was particularly notable, with effect sizes quantified using Cohen's d:

$$1.48 \leq d \leq 2.22 \quad \text{for GPT-4o-mini vs. GPT-3.5-turbo comparisons} \tag{1}$$

These large effect sizes highlight the substantial improvement in performance achieved by employing the more advanced GPT-4o-mini model, far exceeding the conventional threshold for a large effect (Cohen's $d > 0.8$). To account

[1] All language model configurations were tested with temperature set to 0 for consistent comparisons.

for the nested structure of our data, with difficulty levels nested within configurations, we employed a mixed linear model. This model corroborated our findings, showing significant improvements for both GPT-4o-mini configurations over the baseline (coefficients: $\beta_{\text{Instructor GPT-4o}} = 0.441$, $\beta_{\text{OpenAI GPT-4o}} = 0.489$, $p < 0.001$ for both).

These initial results suggest a notable advantage for GPT-4o-mini configurations in generating accurate SatSim JSON schemas, particularly for critical fields. However, a comprehensive evaluation requires consideration of computational efficiency. Fig. 5 illustrates the average duration per test case across configurations and difficulty levels. While GPT-4o-mini configurations exhibit longer processing times ($\mu = 23.30$ s, $\sigma = 11.75$ s for OpenAI Parse), the difference is not statistically significant ($F = 1.327$, $p = 0.294$). Notably, the weak correlation ($r = 0.1141$) between duration and accuracy suggests that increased processing time does not directly translate to improved performance, and vice versa. This finding is particularly noteworthy given the potential for GPT-3.5-turbo configurations to terminate prematurely due to task failure, potentially skewing these duration metrics.

We analyzed temperature, a hyperparameter that modulates the randomness in token selection during text generation by scaling logits before the softmax application. Originally conceptualized as a "creativity dial," temperature controls the trade-off between deterministic and diverse outputs. However, recent research challenges this oversimplified interpretation, revealing only weak correlations with novelty and moderate associations with incoherence [10]. The basis for temperature's effect is expressed as:

$$p(\text{token}_i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{2}$$

where z_i represents the logit for token i, and T is the temperature. This shows how temperature modulates the probability distribution, with lower values amplifying differences in logits and higher values flattening the distribution.

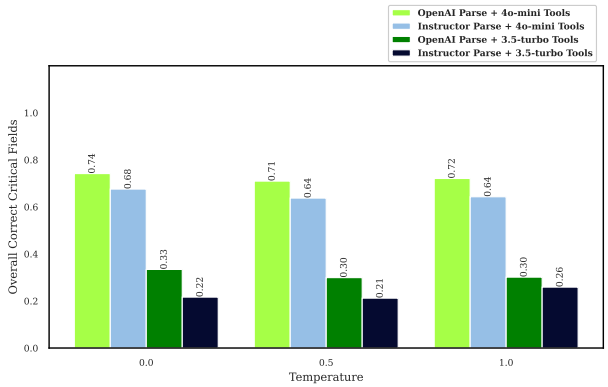

Fig. 6: Temperature vs. Critical Fields Correctness

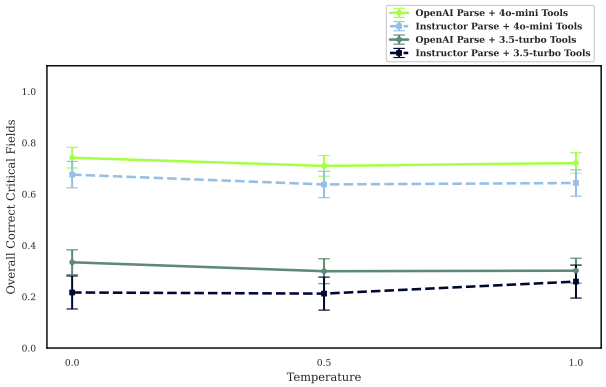

Fig. 7: Temperature Impact with Confidence Intervals

Our analysis of temperature's impact on SatSim Chat's performance provides insights into the system's robustness. Fig. 6 illustrates the relationship between temperature settings (0.0, 0.5, and 1.0) and the percentage of correct critical fields across four different configurations. Notably, the OpenAI Parse + GPT-4o-mini Tools configuration consistently outperforms others, maintaining the highest accuracy across all temperature settings. Fig. 7 provides a more detailed view with confidence intervals, highlighting the stability of performance across temperatures for each configuration.

Despite slight variations in overall mean performance across temperatures (49.20% at 0.0, 46.46% at 0.5, and 48.11% at 1.0), these differences are not statistically significant. A Shapiro-Wilk test ($p = 0.0119$) indicated non-normal distribution, necessitating non-parametric analysis. The Kruskal-Wallis H-test yielded $H = 0.5000$ and $p = 0.7788$, far exceeding our significance threshold ($\alpha = 0.05$). The small effect size ($\varepsilon^2 = 0.0455$) further corroborates the lack of significant temperature impact, accounting for only 4.55% of performance variance.

The consistent performance across temperatures challenges the notion of temperature as a critical hyperparameter in this context, suggesting that the structured nature of the SatSim JSON generation task and the use of specialized tools may constrain the LLM's outputs regardless of temperature. These findings highlight temperature's limited impact on SatSim Chat's performance.

4.3 Limitations

While our research provides valuable insights into integrating LLMs with SDA tools, several limitations warrant consideration. Our evaluation primarily focused on JSON generation accuracy, which serves as a proxy for simulation quality due to the challenges involved in directly evaluating the quality of simulations. The study focused exclusively on OpenAI's LLMs, which may limit the broader applicability of our findings across different models. Although we utilized 30 test cases with multiple repetitions, the relatively small number of unique scenarios does not fully capture the breadth of real-world SDA applications. Additionally, the lack of user studies limits our understanding of SatSim Chat's practical usability and acceptance among SDA professionals. Future work should address these limitations by expanding the range of evaluated LLMs, diversifying test cases, and incorporating user feedback to enhance the system's real-world applicability.

5. DISCUSSION AND FUTURE WORK

5.1 Impact and Implications

The integration of LLMs with SatSim marks a significant advancement in SDA. Our research indicates that LLM-driven interfaces can effectively convert natural language inputs into complex SatSim configurations, offering a significant advancement in space simulation development and manipulation. Our system's consistent accuracy across varying difficulty levels suggests its adaptability to diverse SDA scenarios, from routine operations to more complex tasks.

The observed resilience to temperature variations challenges conventional assumptions about LLM behavior in structured tasks. This finding implies that for domain-specific applications like SDA, task constraints may play a more significant role in output consistency than previously thought, potentially informing future LLM implementations in specialized fields.

5.2 Future Work

Future research should focus on generalizing the principles developed in SatSim Chat to other complex technical domains. The successful integration of LLMs with domain-specific tools could be extended to fields such as climate modeling or bioinformatics, potentially democratizing access to a wide array of complex analytical tools.

A key avenue for investigation is the development of a generalizable framework for coupling LLMs with specialized software. This framework could leverage SatSim Chat's architecture, particularly the use of structured outputs and domain-specific tool integration, reducing development time for similar interfaces in other fields.

The structured output and JSON generation techniques developed in this research have broader implications for LLM-based agent communication. These methods could serve as the foundation for a standardized "language" enabling agents to interact with each other and with various tools via APIs. In the context of SDA and SSA, this could facilitate collaborative AI systems capable of handling multifaceted space environment analyses.

6. CONCLUSION

This research introduces SatSim Chat, an innovative integration of LLMs with SatSim, advancing the field of SDA. By bridging natural language interaction and complex SDA requirements, SatSim Chat demonstrates the potential to democratize access to sophisticated space simulation tools.

Our findings show that GPT-4o-mini configurations achieve a mean accuracy of 70.41% in generating correct critical fields across diverse test scenarios. Furthermore, the system's performance exhibited resilience across temperature settings, challenging conventional assumptions about LLM behavior in structured tasks and suggesting that domain-specific constraints may play a more substantial role in output consistency than previously thought.

While these results are promising, limitations such as the focus on OpenAI models and the use of JSON generation accuracy as a proxy for simulation quality warrant further investigation. Future research should expand the range of evaluated LLMs, diversify test cases to better represent real-world SDA scenarios, and incorporate user studies to assess practical usability among SDA professionals.

The implications of this work extend beyond SDA, potentially serving as a model for integrating LLMs with specialized tools in other complex domains. Furthermore, the structured output techniques developed here may contribute to standardizing LLM-based agent communication.

In conclusion, SatSim Chat represents a significant step towards enhancing SDA capabilities through AI integration. As this field evolves, addressing ethical implications and developing robust frameworks for AI-software integration will be crucial to realizing the full potential of these technologies in critical domains like SSA and SDA.

7. REFERENCES

[1] Abeba Birhane, Atoosa Kasirzadeh, David Leslie, and Sandra Wachter. Science in the age of large language models, 5 2023.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[3] Alexander Cabello and Justin Fletcher. Satsim: a synthetic data generation engine for electro-optical imagery of resident space objects. In *Defense + Commercial Sensing*, 2022.

[4] Samuel Colvin, Eric Jolibois, Hasan Ramezani, Adrian Garcia Badaracco, Terrence Dorsey, David Montague, Serge Matveenko, Marcelo Trylesinski, Sydney Runkle, David Hewitt, and Alex Hall. Pydantic, 2024.

[5] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, 2023.

[6] Jason Liu. Welcome to instructor - instructor. `https://python.useinstructor.com/`, 2024.

[7] David Maranto. Llmsat: A large language model-based goal-oriented agent for autonomous space exploration, 2024.

[8] OpenAI. Introducing structured outputs in the api. `https://openai.com/index/introducing-structured-outputs-in-the-api/`, 2024.

[9] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano,

Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

[10] Max Peeperkorn, Tom Kouwenhoven, Dan Brown, and Anna Jordanous. Is temperature the creativity parameter of large language models?, 2024.

[11] Vipula Rawte, Swagata Chakraborty, Agnibh Pathak, Anubhav Sarkar, S. M Towhidul Islam Tonmoy, Aman Chadha, Amit P. Sheth, and Amitava Das. The troubling emergence of hallucination in large language models – an extensive definition, quantification, and prescriptive remediations, 2023.

[12] Lichao Sun, Yue Huang, Haoran Wang, Siyuan Wu, Qihui Zhang, Yuan Li, Chujie Gao, Yixin Huang, Wenhan Lyu, Yixuan Zhang, Xiner Li, Zhengliang Liu, Yixin Liu, Yijue Wang, Zhikun Zhang, Bertie Vidgen, Bhavya Kailkhura, Caiming Xiong, Chaowei Xiao, Chunyuan Li, Eric Xing, Furong Huang, Hao Liu, Heng Ji, Hongyi Wang, Huan Zhang, Huaxiu Yao, Manolis Kellis, Marinka Zitnik, Meng Jiang, Mohit Bansal, James Zou, Jian Pei, Jian Liu, Jianfeng Gao, Jiawei Han, Jieyu Zhao, Jiliang Tang, Jindong Wang, Joaquin Vanschoren, John Mitchell, Kai Shu, Kaidi Xu, Kai-Wei Chang, Lifang He, Lifu Huang, Michael Backes, Neil Zhenqiang Gong, Philip S. Yu, Pin-Yu Chen, Quanquan Gu, Ran Xu, Rex Ying, Shuiwang Ji, Suman Jana, Tianlong Chen, Tianming Liu, Tianyi Zhou, William Wang, Xiang Li, Xiangliang Zhang, Xiao Wang, Xing Xie, Xun Chen, Xuyu Wang, Yan Liu, Yanfang Ye, Yinzhi Cao, Yong Chen, and Yue Zhao. Trustllm: Trustworthiness in large language models, 2024.

[13] Man-Fai Wong, Shangxin Guo, Ching-Nam Hang, Siu-Wai Ho, and Chee-Wei Tan. Natural language generation and understanding of big code for ai-assisted programming: A review. *Entropy*, 25(6):888, June 2023.

[14] Huaqin Zhao, Zhengliang Liu, Zihao Wu, Yiwei Li, Tianze Yang, Peng Shu, Shaochen Xu, Haixing Dai, Lin Zhao, Gengchen Mai, Ninghao Liu, and Tianming Liu. Revolutionizing finance with llms: An overview of applications and insights, 2024.

[15] Zihuai Zhao, Wenqi Fan, Jiatong Li, Yunqing Liu, Xiaowei Mei, Yiqi Wang, Zhen Wen, Fei Wang, Xiangyu Zhao, Jiliang Tang, and Qing Li. Recommender systems in the era of large language models (llms), 2024.