2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)

# LLM-Agents Driven Automated Simulation Testing and Analysis of small Uncrewed Aerial Systems

Venkata Sai Aswath Duvvuru and Bohan Zhang
*Department of Computer Science*
*Saint Louis University*
Saint Louis, USA
{venkatasaiaswath.duvvuru, bohan.zhang.1}@slu.edu

Michael Vierhauser
*Department of Computer Science*
*University of Innsbruck*
Innsbruck, Austria
michael.vierhauser@uibk.ac.at

Ankit Agrawal
*Department of Computer Science*
*Saint Louis University*
Saint Louis, USA
ankit.agrawal.1@slu.edu

***Abstract*—Thorough simulation testing is crucial for validating the correct behavior of small Uncrewed Aerial Systems (sUAS) across multiple scenarios, including adverse weather conditions (such as wind, and fog), diverse settings (hilly terrain, or urban areas), and varying mission profiles (surveillance, tracking). While various sUAS simulation tools exist to support developers, the entire process of creating, executing, and analyzing simulation tests remains a largely manual and cumbersome task. Developers must identify test scenarios, set up the simulation environment, integrate the System under Test (SuT) with simulation tools, formulate mission plans, and collect and analyze results. These labor-intensive tasks limit the ability of developers to conduct exhaustive testing across a wide range of scenarios. To alleviate this problem, in this paper, we propose AUTOSIMTEST, a Large Language Model (LLM)-driven framework, where multiple LLM agents collaborate to support the sUAS simulation testing process. This includes: (1) creating test scenarios that subject the SuT to unique environmental contexts; (2) preparing the simulation environment as per the test scenario; (3) generating diverse sUAS missions for the SuT to execute; and (4) analyzing simulation results and providing an interactive analytics interface. Further, the design of the framework is flexible for creating and testing scenarios for a variety of sUAS use cases, simulation tools, and SuT input requirements. We evaluated our approach by (a) conducting simulation testing of PX4 and ArduPilot flight-controller-based SuTs, (b) analyzing the performance of each agent, and (c) gathering feedback from sUAS developers. Our findings indicate that AUTOSIMTEST significantly improves the efficiency and scope of the sUAS testing process, allowing for more comprehensive and varied scenario evaluations while reducing the manual effort.**

***Index Terms*—Simulation Testing, AI for SE, sUAS**

## I. Introduction

Simulation testing is a critical step in the small Uncrewed Aerial Systems (sUAS) development process, to validate the behavior of sUAS across a variety of real-world scenarios [12]. This includes, for example, adverse weather conditions (e.g., wind, rain, or fog), diverse terrains (e.g., hilly, flat, urban, and open fields), and varying mission profiles (e.g., long-range, short-range, surveillance, and tracking) [16], [28], [45]. Despite the availability of various sUAS simulation tools and software applications [19], [52], simulation testing remains predominantly a manual, or at best partially-automated process. As a result, current sUAS simulation testing practices result in three main pain points for sUAS application developers.

First, identifying and designing simulation test scenarios that clearly specify *(a) operational contexts*, including weather conditions, environmental settings, terrain variability, *(b) sUAS sensor specifications* including how noisy each sensor is expected in the scenario, *(c) mission objectives*, including higher-order tasks, sUAS mission modes, and *(d) core safety properties* to test is a challenging and time-consuming task.

Due to the fact that sUAS operate in complex environments and are expected to perform reliably in unique or even unimaginable real-world scenarios [35], [65], developers' domain knowledge of sUAS use cases plays an important role. An example, where a software issue caused an incident, is a recent case of a drone crashing in the woods while autonomously tracking a person on an electric scooter. The individual was moving at a moderate speed when suddenly, a dog crossed their path. The drone mistakenly switched its tracking from the person to the dog and crashed into a tree. Video of this drone crash is available on Reddit[1]. Such unique scenarios are particularly difficult for sUAS developers to imagine and accurately test during the development phase.

Second, sUAS developers must model each identified scenario to align with the System under Test (SuT) requirements, and configure the simulation tool to simulate environmental context. This process is often manual, repetitive, time-consuming, and prone to human error.

Third, once simulation tests are executed, analyzing flight logs to diagnose abnormal behavior is labor-intensive, requiring a deep understanding of thousands of flight controller parameters [15]. While tools like PX4 Flight Review [62] plot sensor data automatically, developers still need to interpret the plots and identify issues manually, which requires specific skills and in-depth knowledge of the flight controller. Furthermore, there is no automation framework that enables sUAS developers to analyze simulation artifacts automatically.

These challenges highlight the need for automated tools and supporting frameworks capable of systematically *specifying*, *generating*, *executing*, and *analyzing* diverse simulation tests to achieve scalability and sufficient simulation test coverage. Therefore, in this paper, we present a novel framework called

[1]https://www.reddit.com/r/dji/comments/1amf9rg/oops_solid_crash_while_tracking_myself_on_the

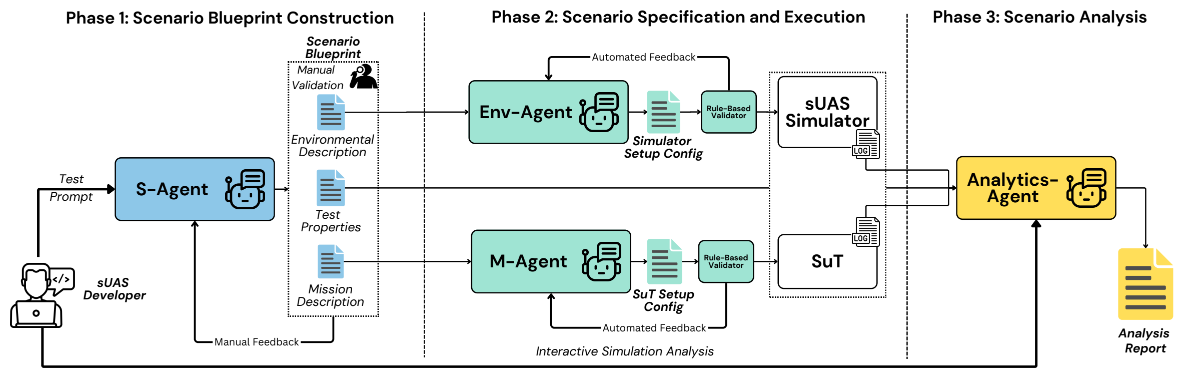


Fig. 1: Overview of our AUTOSIMTEST Framework with the 3 main Phases: Scenario Blueprint Construction and Manual Validation and Feedback (blue), Scenario Specification, Validation, and Execution (green), and Scenario Analysis (yellow).

AUTOSIMTEST as a first step towards automating the simulation testing process for sUAS. We leverage our experience in simulation and field testing within this domain to construct and implement this framework employing recent advancements in Generative AI, more specifically Large Language Models (LLMs) to (a) generate unique test scenarios that incorporate environmental context, mission objectives, and core safety properties to test, (b) model sUAS mission in SuT expected language, (c) configure the simulation environment based on the test scenario's environmental context, and (d) produce a simulation report based on the collected data from scenario execution, as well as provide interactive methods for analyzing simulation results for sUAS developers.

The contributions of this paper are as follows:

- **AUTOSIMTEST:** This is the first work to propose a novel Multi LLM-Agents-based framework to automate the sUAS simulation testing process.
- **Automated Flight Analysis:** As part of the framework, we introduce a novel *Analytics-Agent* enabling automated and interactive analysis of sUAS flight logs.
- **Performance and Developers' Perception:** Extensive experiments have demonstrated that AUTOSIMTEST is capable of testing diverse SuT, effectively analyzing flight logs, and providing valuable support to both novice and experienced developers in better interpreting the simulation data.

The remainder of the paper is laid out as follows. In Section II provide a brief introduction to relevant LLM techniques and then, in Section III introduce our AUTOSIMTEST framework. Subsequently, in Section IV and Section V we discuss the different LLM agents part of our framework. We then, in Section VI, describe our evaluation setup for addressing feasibility, generalizability, and agent performance, and report on results in Section VII. Finally, we discuss several lessons learned in Section VIII, threats to validity in Section IX related work in Section X, and conclude in Section XI.

## II. BACKGROUND

This section briefly introduces the LLM techniques used in the design of our AUTOSIMTEST framework (cf. Section III).

### A. Retrieval Augmented Generation (RAG)

RAG is a method that combines the strengths of pre-trained language models with external knowledge retrieval. Instead of solely relying on the internal knowledge of the language model, RAG retrieves relevant information from external data sources (e.g., a vector database) – serving as a "knowledge library" that the generative AI models can understand. This approach allows the agents to utilize contextually relevant information, especially in specialized domains such as sUAS [49], [78] to generate text.

RAG consists of four primary steps: Query Generation, Document Retrieval, Context Integration, and Response Generation. First, the input query is processed to generate search queries, which are then used to retrieve relevant documents from an external knowledge base. These documents are integrated back into the language model as additional context. Finally, the model produces a final response that combines its internal knowledge with the retrieved information to generate a more precise and contextually relevant output.

### B. Prompt Engineering

Prompt Engineering is a critical technique in the field of Natural Language Processing (NLP) that involves designing and crafting input prompts to elicit desired outputs from language models [39], [54], [72], [73], [79]. The fundamental principle of prompt engineering is to provide clear and context-rich instructions that align with the underlying model's training data. Key strategies to write quality prompts include (a) providing examples to illustrate the request, (b) specifying the desired output format to ensure clarity, and (c) incorporating relevant contextual information to help the LLM understand the prompt and produce the intended output. Prior research in the domain of NLP and Generative AI has shown that the quality of prompt to LLM model greatly influences the quality of generated content [31], [68].

## III. AUTOSIMTEST- FRAMEWORK OVERVIEW

Our AUTOSIMTEST framework supports three main phases of performing sUAS tests. Phase 1, the *Scenario Blueprint Construction*, phase 2, the *Scenario Specification and Execution*, and finally phase 3, the subsequent *Scenario Analysis* and interpretation of results. These phases aim to enhance the sUAS simulation testing process by (a) reducing manual effort in designing and executing scenarios, (b) providing automated analytics support for better understanding of simulation results, while at the same time minimizing reliance on domain-specific knowledge, and finally (c) achieving faster and more frequent simulation testing cycles during development.

Figure 1 provides an overview of the three phases of the framework. Each phase consists of specialized LLM-based AI agents to automate parts of the testing process. All agents share the generated data to provide the necessary automation support. In the following, we provide a brief overview of the main functionality of each phase and discuss the role of each AI agent, before we provide further design and implementation details in Section IV and Section V.

```
{
  "SimulatorSettings": {
    "Weather": {
      "RainIntensity": 0.5,
      "WindSpeed": 5,
      "WindDirection": 0,
      "Visibility": 0.7
    }
  },
  "Vehicles": {
    "Drone_1": {
      "VehicleType": "Quadrotor",
      "Pose": {
        "X": 0,
        "Y": 0,
        "Z": 10,
        "Roll": 0,
        "Pitch": 0,
        "Yaw": 0
      },
      "HomeLocation": {
        "Latitude": 47.641468,
        "Longitude": -122.140165,
        "Altitude": 10
      }
    }
  }
}
```

Fig. 2: *Env-Agent* output

```
{
    "mission": {
        "cruiseSpeed": 15,
        "hoverSpeed": 5,
        "items": [
            {
                "AMSLAltAboveTerrain": null,
                "Altitude": 50,
                "AltitudeMode": 1,
                "autoContinue": true,
                "command": 22,

                "frame": 3,
                "params": [15,0,0,
                    null,
                    47.398039859999997,
                    8.5455725400000003,
                    50
                ],
                "type": "SimpleItem"
            }, {...}, {...}, {...}
        ],
        "plannedHomePosition": [
            47.397742,
            8.545594,
            488
        ],
    },}
```

Fig. 3: *M-Agent* output

## A. *Phase 1 – Scenario Blueprint Construction*

As a starting point, the sUAS developer provides the high-level objectives of the simulation tests that should be created in natural language, as textual input. Depending on the focus of the simulations to be performed, this input can range from very specific goals, such as *"testing the computer vision model of the system under foggy weather conditions"*, to more general objectives, such as *"evaluating the navigation capabilities of the system"*. This crucial input then serves as a part of the prompt for the first agent, the Scenario-Gen-AI-Agent (*S-Agent*) which is tasked with generating a scenario blueprint based on its knowledge of past sUAS incidents in the real world. The *S-Agent* utilizes a Retrieval-Augmented Generation approach to consider real-world sUAS incidents to generate relevant scenario blueprints (cf. Section IV).

The output of the *S-Agent* includes the textual specifications of the environment, the mission that the SuT must execute, and importantly, test properties to evaluate the SuT. For instance, a developer might provide input to the *S-Agent* as *"Test the ability of a drone to track a missing person during a search-and-rescue mission"*. In response, the *S-Agent* generates:

- **Environmental Description**: A densely forested area with tall trees, uneven terrain, and potential obstacles like fallen logs, branches, and wildlife activity, alongside dynamic conditions such as varying light levels, fog, and light rain.
- **sUAS Mission Description**: The SuT should be tasked with locating a missing hiker in a forest environment, navigating through waypoints including search patterns, obstacle avoidance maneuvers, and target identification.
- **Test Properties**: Specific metrics, such as target detection and identification accuracy, obstacle avoidance efficiency, flight stability in varying weather, sensor performance under different lighting, and overall mission completion time.

Before being fed into the second phase of the pipeline, a stage-gate is added where the scenario blueprints undergo manual inspection and validation by a human, for example, developers, testers, or safety engineers. This ensures that the generated scenarios are accurate and do align with testing needs and respective safety-related requirements (e.g., operational boundaries of sUAS related to environmental conditions). Developers can provide feedback to *S-Agent* to refine and adjust specific components of the blueprint such as *"increase the mission complexity"* or *"add lighting variability to the environment"*. This feedback process ensures early identification and correction of scenarios that do not meet developers' objectives.

## B. *Phase 2 – Scenario Executable Script Generation*

This phase focuses on automatically generating the scenario execution scripts as per the sUAS developer's testing infrastructure. This phase consists of two specialized agents each taking over and creating one crucial part required for scenario execution: the SuT Mission Execution Script, and Simulation Tool Configuration Script

*Env-Agent*: Simulation tools such as AirSim [66], Gazebo [59], and DroneReqValidator [13], [76] require configuration scripts, typically in JSON or XML format, to initialize the simulation environment. These scripts define various parameters necessary for the simulation, such as weather conditions, the origin of the simulation environment, drone characteristics (e.g., quad-copter or fixed-wing), and their home or starting geolocations. Therefore, the *Env-Agent* takes the high-level textual environment description from phase 1, provided by the *S-Agent*, as input and utilizes it as a primary prompt to generate a configuration script.

This generated output is used to initialize the 3D environment of the simulation tool according to the scenario execution requirements. For instance, if a scenario needs to be executed in foggy weather, or an urban environment, the *Env-Agent* will generate the necessary tool configuration to simulate these conditions. However, the fidelity of the weather conditions and the realism of the environment, e.g., buildings, depends on the fidelity of the simulation tool itself. Figure 2 shows a sample script generated by the agent to initialize the AirSim simulation tool with weather conditions and a vehicle.

*M-Agent*: Missions define the tasks and actions that sUAS must perform, serving as inputs to the SuT. These missions specify tasks like searching, tracking, or flying in patterns to capture imagery. Typically, sUAS developers use open-source tools such as QGroundControl [63] or MissionPlanner [23] for creating these missions. These tools allow users to specify waypoints, set altitudes, define actions at each waypoint (e.g., taking photos or adjusting speed), and incorporate safety protocols, such as return-to-home triggers, or no-fly zones. However, the manual process is time-consuming and prone to human error due to the vast number of settings and configurations required. Moreover, the complexity of these settings increases with the increasing complexity of the SuT. Therefore, similar to the *Env-Agent*, the *M-Agent* takes the high-level textual mission description from the *S-Agent* and transforms it into specific, executable missions for the SuT. Figure 3 shows a sample output to execute a PX4 mission.

*Validation and Error Correction*: Given the susceptibility of current LLMs to issues such as hallucinations [43], it is important to validate the generated scripts. For this purpose, we introduce an automated validation step, before scripts are executed, using a set of pre-defined rules. Developers can create these rules based on their sUAS mission and simulation tool requirements. For instance, rules can be designed to detect basic syntactical errors in the generated scripts, such as incorrect formatting, missing fields, or invalid/out-of-range parameter values in mission. When issues are detected, the validator provides immediate feedback to the respective agent, prompting it to regenerate a corrected script with specific error details. This feedback loop ensures that any errors introduced by the LLMs are resolved within the framework and do not propagate to subsequent stages.

Finally, after the mission is validated for execution and the simulation environment is initialized with environmental context as per the scenario, either a developer can manually trigger the simulation execution in SuT or it can also be automated using automation scripts.

### C. Phase 3 – Scenario Analysis Generation

The execution of a simulation scenario typically generates simulation results in the form of SuT logs, which developers record in their code base for debugging purposes. These SuT logs contain high-level explanatory messages, such as `Mission Dispatched Successfully` and `Moving to the next waypoint`, as well as low-level log messages that include event names and their timestamps. These logs are crucial for developers to understand and debug any issues related to SuT execution as per the system requirements. In addition to SuT logs, the underlying flight controller used by developers, such as PX4 [61] or ArduPilot [22], generate their own logs called flight logs. These flight logs contain time-series sensor data, and warnings or messages generated by flight controllers. This data is crucial for conducting lower-level analysis and understanding the behavior of the vehicle. Developers analyze these logs to interpret the simulation results. Analyzing time-series data from hundreds of flight controller parameters, however, requires extensive domain expertise and knowledge. Therefore, the primary objectives of *Analytics-Agent* are twofold:

- *Automated Scenario Analysis*: Generate scenario analysis reports from data produced as part of the scenario execution.
- *Bridge Developer's Knowledge Gap in Analysis*: Provide developers with interactive methods to explore and understand complex simulation logs. This interactive analysis aims to bridge the knowledge gap by allowing developers to ask high-level analytics questions and receive automated analysis from the *Analytics-Agent* in the form of a small set of relevant parameters to investigate, out of hundreds of parameters.

## IV. Agents' Knowledge-Base and Prompts

In this section, we discuss two main components of our framework: the knowledge base and custom prompt design.

### A. Knowledge Base

*S-Agent knowledge base*: To automatically generate scenarios that aim to detect specific vulnerabilities in the SuT, especially the ones that are common in the real world, we created a robust database drawing from publicly available information on sUAS real-world incidents. The sources we used to build the S-Agent's knowledge base are listed in Table I. We used web scrapers to collect the data from sources and semi-autonomously cleaned them to create the knowledge base. This knowledge base provides the agent with realistic scenarios and contexts that have previously caused issues in sUAS operations. This data-driven approach enables the *S-Agent* to generate simulation testing scenario blueprints that are informed by actual events and historical data. This data-driven approach is also extensible to other domains, e.g., autonomous vehicles (further discussed in Section VIII).

| No. | Incident Source | Incident # | Token # |
|---|---|---|---|
| 1 | UK Air Accident Investigation Rep. [9] | 81 | 6280 |
| 2 | Wikipedia List of UAV incidents [4] | 74 | 1203 |
| 3 | NASA ASRS Report [7] | 50 | 11316 |

TABLE I: Scenario incident sources to build *S-Agent* knowledge base.

*Analytics-Agent knowledge base*: With extensive simulation data and hundreds of parameters in each log, the *Analytics-Agent* must select the relevant parameters and log sections to generate relevant responses. This requires the agent to have domain knowledge to interpret the log data and understand its content and meaning.

To build this second knowledge base, we developed a dataset that encodes the meaning and interpretation of hundreds of parameters from the two most popular open-source flight controllers: PX4 and ArduPilot. The PX4 flight controller codebase[2] provides a comprehensive description of each logged parameter, with detailed comments explaining the meaning and interpretation of each parameter. We used a Python script to parse the data structure of each message, which contains the parameter name and a brief description as comments. This domain-specific information enables the *Analytics-Agent* to leverage RAG techniques to identify and analyze contextually relevant parameters within the flight logs based on user queries and scenario test properties.

[2]https://github.com/PX4/PX4-Autopilot/tree/main/msg

| Part \ Agent | S-Agent | M-Agent | Env-Agent | A-Agent (Automated) | A-Agent (Interactive) |
|---|---|---|---|---|---|
| **Agent Goals** | Act as a Test Engineer to generate scenario blueprints that include<br>- Environment<br>- Mission<br>- Test Props | Act as an Automation Engineer to generate a SuT mission script based on scenario blueprint | Act as an Automation Engineer to generate sim tool script based on scenario blueprint | Act as a Data Analyst to analyze simulation log data and explain how the test properties are affected | N/A |
| **User Goals** | *Example*: Generate a search-and-rescue mission scenarios | N/A | N/A | N/A | *Example*: Explain all PX4 parameters that might have an impact of high wind |
| **Sample** | cf. Example blueprint in Section III-A | cf. Figure 3 | cf. Figure 2 | N/A | N/A |
| **Rules** | - Blueprint Completeness | - Script Format Validity<br>- Valid Geo-loc.<br>- Valid Waypoints<br>- Velocity = [0,30] mph<br>- Altitude $\leq 400ft$ | - Script Format Validity<br>- Wind = [0,50] mph<br>- Light Intensity = (0,10) | - Analysis Completeness | - Analysis Completeness |
| **Ext. Data** | List of past real world sUAS incidents | N/A | N/A | Flight Controller Codebase and Documentation | |

TABLE II: Overview of the prompt design for four agents part of AUTOSIMTEST.

## B. Prompt Design

We use a pattern-based approach [73] to design prompts for each agent. Our prompt design for agents in the framework consists of four essential patterns:

- *Agent's Goals*: We utilize the *Persona Pattern* [58] to instruct the agents about their primary goals when generating output. For instance, we use the "Act as a Simulation Testing Engineer" persona to design sUAS scenarios, or "Act as a Data Analyst" to analyze the simulation logs. The Persona Pattern helps the agent to identify the details it should focus on when generating the output.
- *Sample of Expected Output*: We utilize the *Template Pattern* to instruct the agent about the format of the expected response. The *S-Agent*, *M-Agent*, and *Env-Agent* include this component, especially because the output is directly fed into another sub-system and must match the sub-system's input format for automated execution.
- *User Goals*: We utilize the *Context Manager* Pattern to instruct the LLM about the User Goals to fulfill when generating the output. The prompt includes instructions from the user's perspective to tailor the output, such as generating a scenario blueprint for a complex search-and-rescue mission or a simpler package-delivery mission. The *S-Agent* and *Analytics-Agent* include this component in their prompt since the user initiates the process using *S-Agent* and consumes/interacts with the output of the framework using the *Analytics-Agent*. Further, these agents utilize RAG to include the contextual information from their respective knowledge base, based on user goals, to augment targeted and relevant information.
- *Rules*: Additionally, we utilize the *Error Identification Pattern* to force the LLM to validate the generated output against a set of predefined rules or conditions. For instance, the sUAS mission script should not allow sUAS to fly over 400ft (safe altitude). This validation process ensures that the generated output meets the required criteria and maintains consistency in the LLM output.

Table II summarizes the prompt design for each agent, including the four components and the data utilized from their knowledge base to generate a response. The following sections provide details about the design and architecture of each agent.

## V. AGENTS' DESIGN

### A. S-Agent

First, the agent uses an embedding model [48] to vectorize the knowledge documents (cf. Table I) and store them in a vector database (*RAG Knowledge Store*) to perform search queries on it. Second, when an sUAS developer provides input to the agent to generate a scenario blueprint as per their testing needs, the agent vectorizes the textual input (*Prompt Vector*) and performs a cosine similarity search in the *RAG Knowledge Store* to identify a set of real-world sUAS incidents relevant to the user's testing context. For example, if a user wishes to test a search-and-rescue operation, the agent will find the details of accidents during search-and-rescue operations and utilize that to formulate User Goals. In addition to the *User Goals*, the agent incorporates other components (*Agent's Goals*, *Sample of Expected Output*, and *Rules* as described in Table II) of the prompt to form a comprehensive prompt and sends it as input to the LLM to generate a scenario blueprint.

### B. M-Agent and Env-Agent

The *M-Agent* and *Env-Agent* agents utilize specific parts of the scenario blueprint as their goals to generate scripts. The *M-Agent* uses the Mission Description section as its goals, while the *Env-Agent* uses the Environment section. Since these agents act as translators, converting textual descriptions into a language that the SuT and simulation tool can interpret, they don't require *User Goals*. However, to ensure their generated output is compatible with the SuT and simulation tool, they include *Sample of Expected Output* and *Rules* to follow when generating output. The detailed breakdown of the prompt design is shown in Table II.

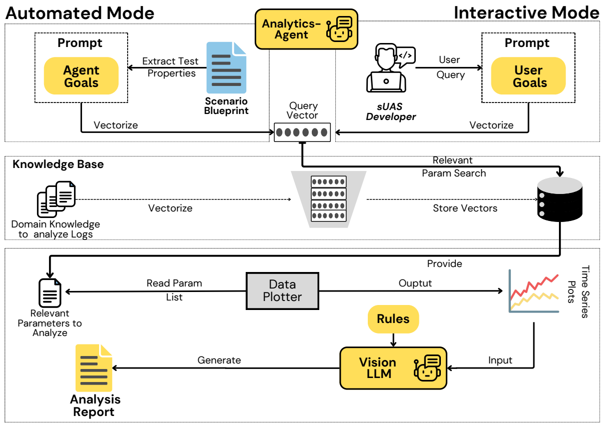


Fig. 4: Schematic overview of the main parts of the *Analytics-Agent*.

## C. *Analytics-Agent*

The design of the *Analytics-Agent* is divided into two main, logically separated components, as shown in Figure 4. The *Automated Mode* component is responsible for analyzing simulation logs based on the test properties generated by the *S-Agent* and producing a comprehensive scenario analysis report without direct interaction with the user. Complementing this first part, the second part of the *Analytics-Agent*, the *Interactive Mode* component allows sUAS developers to start new analyses that may not be part of *Automated Mode* analysis.

Both components utilize a shared knowledge base to identify the essential simulation log data to analyze. The primary distinction between the two modes lies in the formulation of prompts. In Automated mode, the agent uses *Agent's Goals* and *Rules*, whereas Interactive mode relies on user input, designated as *User Goals* and *Rules*, in the prompt. Based on the Analysis Goals, whether from the User or the Agent, the agent performs a semantic search within the knowledge base to identify flight controller parameters to analyze such as velocity, pitch angles, GPS position, and altitude.

Upon identifying the core flight parameters, the agent initiates an automated script to plot these parameters, with the x-axis representing the timestamp and the y-axis representing the parameter values. These plots, along with the prompt, serve as input for the Vision LLM model to analyze the plot images and generate an analysis report. *Analytics-Agent* saves this report for developers' debugging and analysis purposes.

## VI. Evaluation

We used our AutoSimTest framework to investigate the following research questions:

– *RQ1: To what extent is our framework applicable for conducting real-world sUAS simulation testing, and how well can it be applied across diverse SuT that use different flight controllers and simulation tools for testing?*

To answer this research question, we conduct experiments to first evaluate (a) the feasibility of AutoSimTest to test different SuT that utilize widely-used flight controllers such as PX4 [61] and ArduPilot [22], and second (b) its ability handle diverse common sUAS use cases (cf. Table III).

– *RQ2: To what extent can our framework with RAG be utilized for generating relevant test scenarios blueprints across sUAS use cases and investigating sUAS failures?*

We conducted a second set of experiments to quantitatively assess the performance and effectiveness of our RAG approach, for both the *S-Agent* and the *Analytics-Agent*. We used the standard Retrieval-Augmented Generation Assessment (RAGAs) framework [36], a reference-free RAG pipeline evaluation method that utilizes a separate LLM as a critic to judge the responses generated by our agents. We collect the following metrics.

- *Context Precision and Recall*: These metrics measure the effectiveness of RAG in retrieving relevant information from the knowledge base. *Context Precision* evaluates the proportion of retrieved information that is relevant, while *Context Recall* evaluates the proportion of relevant information.
- *Response Faithfulness*: This metric assesses whether an AI-agent generate outputs that are grounded in the information gathered from their knowledge bases.
- *Response Relevancy*: This metric evaluates the completeness of the final responses generated by an agent.

To compute Response Relevancy and Faithfulness, we employ the Llama 3 LLM [57], a 3 trillion parameter model, as a critic to evaluate responses of *S-Agent* and *Analytics-Agent*. Further, we also investigate the quality and correctness of the *Analytics-Agent* output using flight logs as described in Section VI-A.

– *RQ3: How do developers perceive the generated scenarios, and does the interactive analysis support developers in better understanding the simulation results?*

To address this research question, we conducted interviews (each approx. lasting 1 hour) with sUAS developers to understand current challenges in sUAS simulation testing and their perception about AutoSimTest and especially how the interactive analysis can support them during simulation log analysis. We deployed our entire framework on the cloud and asked participants to especially interact with *S-Agent* and *Analytics-Agent*. Participants provided feedback and opinions on the framework's usability in sUAS simulation testing.

Next, we discuss our strategy, tools, and techniques we utilized to implement the framework that we utilize for conducting experiments.

## A. *Evaluation Setup*

**AutoSimTest Framework Implementation:** We leverage the advanced capabilities of the open-source Phi-3 family of LLMs [10] designed by Microsoft to implement our framework. Specifically, we utilize the Phi-3-medium-128k-instruct LLM [5], a model with 14 billion parameters. In order to support analysis of image data, we use the Phi-3-vision-128k-instruct model [6], which integrates an image encoder, connector, projector, and a Phi-3 Mini language model, all within 4.2 billion parameters. For embedding and similarity matching, we use sentence-transformers/all-mpnet-base-v2 model [64] and FAISS [46] for vector database management. We deployed our framework on Gradio.App [3], a cloud platform for ML models. We use this deployment in our perception study to gather feedback from sUAS developers. Figure 5 shows the interface our participants used in the study.

**SuT:** We designed two sUAS, SuT-Px4 and SuT-Ardu, based on Px4 and Ardupilot flight controllers, respectively. These flight controllers were chosen due to their widespread use, comprehensive documentation, and open-source license. For simulation, we employed AirSim (compatible with Px4) and Ardupilot-SITL (compatible with Ardupilot). We developed our SuT for city surveillance use case with two features:

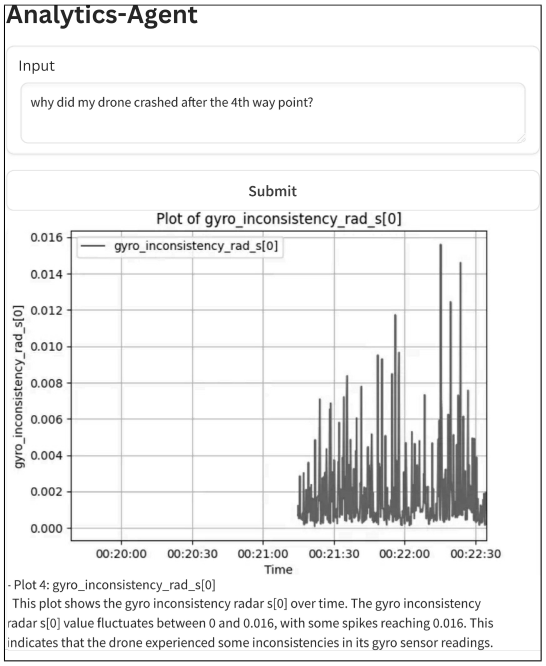
- Way-point-based Navigation - SuT follows predefined way-points to surveil an area.
- Autonomous Navigation - SuT determine its grid-based flight path based on the shape of surveillance area.

**sUAS Use Cases:** We examined the effectiveness of AUTOSIMTEST across popular sUAS use cases, as summarized in Table III. These specific use cases were selected due to their popularity in both academic research and industrial products.

| sUAS Use Case | References |
|---|---|
| UC1: Search-and-Rescue | [2], [11], [56] |
| UC2: Precision Agriculture | [34], [44], [74] |
| UC3: Env. Monitoring | [20], [38], [40] |
| UC4: Surveillance | [17], [25], [53] |
| UC5: Package Delivery | [18], [27], [50] |

TABLE III: Common sUAS use cases with references in the literature

**SuT Flight Logs:** We created a dataset of 7 PX4 flight logs with artificially injected sensor failures in our PX4-SuT. We used PX4 utilities to inject these errors [8], as they are widely used during the simulation testing phase to model common real-world sensor failures, such as sudden GPS loss. Each flight log contains one of sensor failure including GPS, Accelerometer, Gyro, Magnetometer, Battery, Barometer, and


Analytics-Agent

Input

why did my drone crashed after the 4th way point?

Submit

- Plot 4: gyro_inconsistency_rad_s[0]
This plot shows the gyro inconsistency radar s[0] over time. The gyro inconsistency radar s[0] value fluctuates between 0 and 0.016, with some spikes reaching 0.016. This indicates that the drone experienced some inconsistencies in its gyro sensor readings.

Fig. 5: User Interface for developers to interact with *Analytics-Agent*. Top: User Query; Middle: Generated Plot; Bottom: Generated Analysis Report

| Category | Description |
|---|---|
| Environment | • *Location*: New York City<br>• *Weather*: Wind = 15m/s<br>• *GPS Quality*: High<br>• *Obstacles*: Buildings, PowerLines |
| Mission | Monitor traffic patterns, and assist law enforcement in maintaining public safety. |
| Test Property | Flight Stability in Wind |

TABLE IV: Scenario Blueprint for City Surveillance Use Case

AirSpeed. We use these flight logs to evaluate the ability of *Analytics-Agent* in detecting sensor failures (Section VII-B3).

## VII. RESULTS AND ANALYSIS

### A. RQ1 - Applicability and Generalizability

*1) **Applicability**:* To demonstrate the general applicability of AUTOSIMTEST for sUAS simulation testing, we apply each phase to our two SuT, for a common use case and analyze the resulting output. Starting in phase 1, we used the *S-Agent* to design a scenario blueprint for a city surveillance use case. We used a very simple "Generate a city surveillance" scenario as *User Goals* of the prompt. This blueprint serves as input for phase 2 to generate executable scripts for our SuT, specifically SuT-PX4, and SuT-Ardu, using the mission agent, and for simulation tools, specifically AirSim and ArduPilot-SITL, using the environment agent. We provide the *Sample of expected outputs* to these agents to generate valid executable scripts. The generated scripts were used to automatically execute SuT-PX4 missions in AirSim [66] and SuT-Ardu missions ArduPilot-SITL [21].

*Results and Analysis*: The scenario blueprint, presented in Table IV, outlines the environment, sUAS mission, and test properties. The executable scripts generated by the *M-Agent* and *Env-Agent* were validated, and confirmed to contain accurate geolocation waypoints for New York City (Latitude: 40.7128, Longitude: -74.0060) and wind speed settings of 15m/s in the AirSim simulation script (see supplementary materials for complete scripts). We were able to use generated mission scripts in both simulation tools without making any changes and collected the flight logs for analysis. This initial set of tests demonstrates the capability of our framework to generate executable scenarios in simulation. In the next step, we discuss how the framework can be applied to test SuT in an extended more broader manner.

*2) **Generalizability across sUAS Use Cases**:* As a first step, to demonstrate generalizability, we applied our framework to 5 different sUAS use cases, each of which requiring different properties to be tested and simulations to be configured accordingly (cf. Table III). For each use case, we used the *S-Agent* to generate 5 scenario blueprints using a simple prompt such as *"Create a scenario to test my sUAS in delivering emergency supplies during a natural disaster."* To validate our framework's ability to generate correct executable mission and simulation scripts, we adopted a model-based approach using a JSON-schema to define valid scripts for hypothetical SuTs in each use case. This choice aligns with the widespread use of structured JSON messages as a standard for mission inputs in both commercial [1] and academic sUAS systems [14], [32].

When generating mission scripts where we leveraged this model to specify the *Sample of Expected Output* component of the prompt for the *M-Agent*. To further enhance the prompt, we introduced new rules that capture specific requirements for each use case. For instance, we added a rule that "Missing Person" should be a valid geolocation when generating mission scripts for search-and-rescue missions. Similarly, we included a rule that the distance to the delivery location should be fewer than 2 miles from the home location of a drone when generating missions for package delivery. We followed a similar approach to generate scripts using *Env-Agent* for AirSim simulation tool settings. As a result, using our framework, we created a total of 25 Scenario Blueprints, 25 mission scripts, and 25 AirSim setting scripts.

*Generated Scenario Blueprint Quality*: We analyzed all 25 generated scenarios to assess their relevance and their usefulness for testing sUAS. Two authors, who have both several years of experience in developing sUAS applications and testing sUAS systems, acted as evaluators independently evaluating each scenario, and provided binary ratings on (a) its relevance to the use cases and (b) its potential usefulness for testing sUAS. Both evaluators agreed that all 25 generated scenarios are relevant for the respective use case and useful for testing. This agreement resulted in a perfect Cohen's Kappa [55] score of 1. To expand our qualitative analysis, we used inductive coding [69] to identify recurring themes and potential gaps in the scenarios. Both evaluators first independently commented on the scenarios' unique aspects and areas for improvement, then discussed their comments to identify common themes.

As part of this qualitative analysis, we found that scenarios contain *Rich and Diverse Environmental Details*, such as changing wind, lighting, and temperature conditions, with realistic nuances like moderate GPS signal loss due to foliage [47] in the area. Such subtle environmental details are often overlooked in simulation testing as they require expertise in GPS sensor signal reception and processing. Both evaluators also agreed that the scenarios for the Search and Rescue use cases were the most diverse ones. This finding is consistent with the diversity analysis presented in Table VI.

On the other hand, we also observed that many scenarios, while containing different mission profiles and test properties, often share a *"Similar Flavor"*. For example, navigating through flooded terrain and avoiding obstacles, and navigating through smoke-filled terrain and avoiding obstacles are two flavors of the collision avoidance test property. To avoid redundancy, human reviewers in Phase 1 must provide precise feedback to the *S-Agent*. Our current framework allows incorporating this feedback by updating the input prompt and allowing to focus on either a broader range of scenarios or smaller nuances of more specific scenarios depending on the required testing context. In the future, this process could involve additional guidance for developers interacting with the framework, for example, by providing support for refining specific sections of the scenario blueprint by targeting and modifying selected snippets of text in *S-Agent*'s output. Our perception study with sUAS developers, discussed in Section VII-C, further provides more insights on the usability of these scenarios in sUAS testing.

*Validation of Generated Executable Scripts*: Besides the qualitative analysis, we further assessed the validity (i.e., executability) of each script. We used *Rule-based Validators* for both the *Env-Agent* and *M-Agent* to verify if the generated scripts are valid and executable using the *Rules* (cf. Table II) that were enforced as part of the agents' prompt. We found that all scripts were valid and passed the validity checks, indicating that *M-Agent* and *Env-Agent* can generate valid mission scripts and simulator settings for each scenario blueprint. In particular, we found that the combination of *Rules* and *Expected Output* in the prompt ensured that the values in each JSON property accurately reflect the mission and environmental description in the scenario blueprint. These initial results indicate that 1) the *S-Agent* generates relevant and useful scenario blueprints, 2) the output of the *M-Agent* and *Env-Agent* can be used to test SuT without any human intervention, 3) the design of the *S-Agent*, *M-Agent*, and *Env-Agent* is flexible to support sUAS testing across diverse use cases, SuT, and simulation tools, and finally 4) script generation is fast enough to be used in large-scale automated testing.

**RQ1: AUTOSIMTEST Applicability & Generalizeability**

Based on our experiments, AUTOSIMTEST can be used to test two different SuT based on PX4 and ArduPilot flight controllers and also found that it can be applied to test across diverse sUAS use cases, and that valid simulation and test scripts were generated in all 25 test executions.

## B. RQ2 - RAG Agents' Performance

We use the standard RAGAs framework [36] to quantitatively analyze the performance of the *S-Agent* and *Analytics-Agent*. For the *S-Agent*, we analyzed the 25 scenario blueprints across 5 sUAS use cases to evaluate if our knowledge base helps in generating contextually relevant and diverse blueprints. Additionally, to evaluate the *Analytics-Agent*'s ability to produce the contextually relevant analysis report for 7 common sensor failures as well as its ability to identify issues.

*1) **S-Agent RAG Performance**:* The RAG evaluation metrics scores, as presented in Table V, show the effectiveness of the *S-Agent* across five use cases. Notably, the *S-Agent* achieved high faithfulness scores (0.8-0.9), indicating that it generates factually accurate scenario blueprints and utilized real-world sUAS incident data in the knowledge-base to generate scenario blueprints. The high context precision scores (0.7-0.9) also demonstrate the agent's ability to retrieve information from the knowledge base, relevant to the sUAS use case, with minimal accident data for a different use case. Furthermore, the high response relevancy scores (0.7-0.9) confirm that the agent generates responses relevant to the *User Goals*.

However, the moderate context recall scores (0.6-0.8) suggest that the *S-Agent* retrieves a significant portion of relevant information, but also misses some. The lower scores in *Precision Agriculture* can be attributed to the relatively limited number of related incident in our knowledge-base.

| sUAS Use Case | Faith-Fulness | Context Precision | Response Relevancy | Context Recall |
|---|---|---|---|---|
| Search-and-Rescue | 0.90 | 0.80 | 0.90 | 0.80 |
| Precision Agriculture | 0.80 | 0.70 | 0.70 | 0.60 |
| Env. Monitoring | 0.90 | 0.80 | 0.90 | 0.80 |
| Surveillance | 0.90 | 0.80 | 0.90 | 0.80 |
| Package Delivery | 0.90 | 0.90 | 0.90 | 0.80 |

TABLE V: *S-Agent* RAG performance across sUAS use cases

2) ***Generated Scenario Blueprint Diversity:*** While the RAGA metrics offer valuable insights into the contextual relevance of the generated output, they do not evaluate the diversity of the scenario blueprints. To address this limitation, we calculated the Jaccard similarity score [26], a measure of similarity between two sets ranging from 0 (completely dissimilar, which is desirable in our context) to 1 (identical), across all 5 scenarios generated for each use case. The low similarity scores, ranging between 0.24 and 0.53, presented in Table VI, show that the generated scenario blueprints contain a notable degree of diversity.

| sUAS Use Case | Avg. Jaccard Similarity |
|---|---|
| Search-and-Rescue | 0.24 |
| Precision Agriculture | 0.41 |
| Env Monitoring | 0.44 |
| Surveillance | 0.48 |
| Package Delivery | 0.54 |

TABLE VI: Jaccard Similarity across sUAS use cases

3) ***Analytics-Agent RAG Performance:*** For each flight log, we generated an analysis report across 5 different test properties. For instance, test properties related to the Accelerometer included Peak Acceleration, Unexpected Acceleration Change, Average Acceleration in Each Axis, Duration of Highest Sustained Acceleration, and Consistency of Accelerometer Readings. As a result, for the seven sensor failures and flight logs, this process yielded a total of 35 responses from the *Analytics-Agent*, which we then analyzed using RAGAs to evaluate *Analytics-Agent*'s performance in generating contextually relevant output. Table VII presents the results.

Our findings indicate that the agent identified and utilized relevant parameters for each sensor to generate the analysis report. For example, in the analysis of accelerometer data, the agent selected parameters such as $accel\_bias[0]$, $failsafe$, $yawspeed$, $has\_low\_throttle$, and $accel\_fault\_detected$, and used their time-series data to construct the analysis. It is important to note that identifying relevant parameters goes beyond simple keyword matching with sensor and parameter names; the agent also utilizes the description of each parameter to identify items like $failsafe$ and $yawspeed$ for analyzing test properties related to the Accelerometer sensor.

4) ***Generated Analysis Report Correctness:*** While our previous analysis indicates that the agent can generate contextually appropriate content, assessing the accuracy of the generated report is equally important. For our dataset of 7 log files, we evaluated whether the *Analytics-Agent* could accurately detect the injected failures and correctly describe the actual issues in the analysis report.

| Sensor Analysis | Faith-Fulness | Context Precision | Response Relevancy | Context Recall |
|---|---|---|---|---|
| Accelerometer | 0.86 | 0.88 | 0.82 | 0.82 |
| Gyroscope | 0.82 | 0.85 | 0.82 | 0.82 |
| Magnetometer | 0.74 | 0.80 | 0.72 | 0.75 |
| Air Speed | 0.93 | 0.87 | 0.93 | 0.83 |
| Barometer | 0.80 | 0.80 | 0.80 | 0.72 |
| Battery | 0.84 | 0.85 | 0.87 | 0.72 |
| GPS | 0.90 | 0.86 | 0.90 | 0.80 |

TABLE VII: *Analytics-Agent* RAG performance across sensor failures

| Sensor Fail. Injected | Issue Detected | *Analytics-Agent* (Automated Mode) Snippet of Analysis Report |
|---|---|---|
| Accelerometer | ✓ | *Inconsistent accelerometer bias values* suggest issue, which can lead to inaccurate motion sensing and stability. |
| Gyroscope | ✓ | *Spikes in gyro readings* indicate potential instability. |
| Magnetometer | ✓ | *The magnetometer is occasionally experiencing faults*, This could impact the drone's stability and control. |
| Air Speed | ✓ | the *warning messages related to 'airspeed off'* suggest that there were some issues with controlling the airspeed. |
| Barometer | ✓ | *The sudden spike in altitude readings at around 150 seconds* in the baro alt meter plot could be due to a sensor error. |
| Battery | ✓ | *failsafe was activated* due to low battery capacity. |
| GPS | ✓ | *shows sudden and erratic fluctuations* that could impact the drone's navigation and stability. |

TABLE VIII: Sensor Issue Analysis and Agent Report

We found that the *Analytics-Agent* correctly identified sensor failures in all 7 flight logs as shown in Table VIII. The generated report provides clear textual descriptions of data indicating a failure based on *Analytics-Agent*'s interpretation of time-series plots. For instance, the agent was precisely able to find an issue with the barometer sensor by identifying a sudden or unusual spike in the $baro\_alt\_meter$ reading at second 150 of the flight. These results highlight the effectiveness of *Analytics-Agent* in automatically analyzing simulation logs and supporting developers in analysis.

**RQ2: AUTOSIMTEST Performance**

The *S-Agent* generated relevant and diverse scenarios for each sUAS use case. The *Analytics-Agent* generated contextually relevant analysis and automatically identify common sensor failures.

## C. RQ3 - End-Users perception of AUTOSIMTEST

We recruited 4 sUAS developers for interviews from our network. Details of each participant is shown in Table IX.

| P.# | sUAS Dev Exp. | Flight Log Analysis Experience |
|---|---|---|
| P1 | 1 Year | 10+ flight logs |
| P2 | 5 Years | 50+ flight logs |
| P3 | 3 Years | 10+ flight logs |
| P4 | 7 Years | 10+ flight logs |

TABLE IX: Participants' Details

All participants interacted with *Analytics-Agent*, while three participants interacted with *S-Agent* to evaluate and provide feedback on the perceived usefulness of the framework. P4 could not engage with *S-Agent* due to logistical issues.

*1) **Problem Validity**:* P1 validated the inherent difficulty in manually generating thousands of diverse scenarios and highlighted the value of providing automation support to this process. P2, P3, and P4 confirmed that analyzing hundreds of flight log parameters is a time-consuming task and heavily depends on the developer's experience. According to P2, *"these drones output hundreds of time-series data, and when something goes wrong, analyzing that is a real challenge"*. The participants unanimously agreed that automation support would greatly simplify simulation testing for them.

*2) **Perceived Usefulness of Generated Scenarios**:* P1, P2, and P3 highlighted that generated scenario blueprints are contextual as per their input prompt and include details such as *high winds*, *temperature*, *gps quality*, and *locations*. P3 specifically mentioned: "the details in the generated scenario are pretty impressive such as flight duration of 30 minutes and payload of 1.5 kg". While our study participants agreed that the generated scenarios are highly detailed and read like realistic situations, they also mentioned that current state-of-the-art sUAS simulation tools do not support direct execution of such complex scenarios. They emphasized that executing these scenarios would require significant technical effort and substantial enhancements to the simulation tools.

*3) **Perceived Usability of Analytics-Agent**:* All participants appreciated the inclusion of plots for relevant parameters extracted from flight logs in the reports and mentioned *significant time-saver* (to analyze thousands of parameters). They highlighted that the automated identification and plotting of relevant time-series data is particularly valuable, as it addresses a common challenge: determining where to begin their analysis. For example, P2, an expert in flight log analysis, entered the prompt: *"Was the satellite count low?"* In response, the agent correctly plotted data for parameters such as *satellite_used* and additionally reported a supporting parameter, *rx_message_lost_count*, which P2 found interesting. Based on this interaction with *Analytics-Agent*, P2 remarked: *"the analytics agent provided really good analysis."*.

*4) **Improvement Opportunities**:* When interacting with *S-Agent*, P2 recommended that the agent should generate "*incremental test scenarios*". For instance, the first scenario could involve a simple flight path without obstacles, while subsequent scenarios could gradually increase complexity by adding obstacles, limited battery availability, and varying weather conditions. This structured progression would help developers build a comprehensive test suite.

On the other hand, when interacting with *Analytics-Agent*, P2, suggested incorporating knowledge from public forums (e.g., https://discuss.ardupilot.org) into *Analytics-Agent*'s knowledge base. This would enable *Analytics-Agent* to also propose potential fixes to issues. P1 and P3 recommended displaying additional information on the User Interface, such as flight paths, to facilitate targeted queries about specific flight duration or timestamps. Finally, P4 stated that the entire framework could be best taken advantage of during a CI/CD pipeline to get an analysis report after every commit to SuT.

**RQ3: sUAS Developers' Perception of AutoSimTest**

sUAS Developers expressed enthusiasm for utilizing the framework, and indicated that AutoSimTest would serve as a valuable tool during their simulation analysis activities.

## VIII. Discussion and Lessons Learned

*1) **Application to CPS Testing**:* The proposed framework's multi-agent design allows for broad application to various CPS, including Autonomous Ground Vehicles (AGV) such as self-driving cars and delivery robots. Adapting the agents' Knowledge Base and Prompt Design is key to applying the AutoSimTest to AGVs. For instance, to test the navigation capabilities of an autonomous car in rainy weather, the *S-Agent* can generate a scenario blueprint using instances of road accidents stored in its knowledge base. This blueprint can be utilized by the *M-Agent* to produce a list of destinations, specifying the locations the car must visit during simulation. For widely used simulation tools such as CARLA [52], the *Env-Agent* can automatically generate the necessary configuration files (e.g., *CarlaSettings.ini*) to initialize the CARLA simulation environment [52] with rainy conditions. ultimately, the generated simulation logs can be analyzed by the *Analytics-Agent*, leveraging its comprehensive domain knowledge of analyzing parameters of autopilots of autonomous cars. Therefore, the concept of AutoSimTest is not limited to sUAS and has the potential to be applied for more general CPS testing.

*2) **Realism of sUAS Simulations**:* While our AutoSimTest was successful in generating scenario blueprints and executable scripts, we observed a huge gap between the scenario blueprint we generated and the realism of actual simulation execution. For instance, a scenario blueprint generated for river search-and-rescue use case included simulating the movement pattern of a drowning person in the river. However, simulating such dynamism is not supported by Airim or similar sUAS simulation tools. This was also emphasized by participant P3. Therefore, our study also highlights the need for more advanced sUAS simulation tools that allow simulating context-specific elements of the environment.

*3) **Swarm sUAS Analysis**:* The *Analytics-Agent* is a valuable tool in detecting sensor failures and potential end-users, both novices and experts, perceive this as a useful tool in their simulation analysis process. However, real-world flights suffer from failures that extend beyond sensors, particularly those arising from the human interaction with sUAS during the flight [70]. Additionally, state-of-the-art flight analytics tools are limited to supporting single sUAS analysis, which falls short for swarm sUAS applications that require simultaneous analysis of multiple sUAS. Therefore, we will expand the scope of *Analytics-Agent* in the future to analyze a variety of sUAS failures and support swarm sUAS analysis.

## IX. Threats to Validity

- Our feasibility and generalizability tests indicate the broad applicability of the framework. However, these tests are based on assumptions about inputs to more complex SuTs. To mitigate this limitation, we are working to collaborate with our industry partners to apply our framework to more advanced and complex SuTs in future work.
- The standard approach to compute relevance and faithfulness metrics for evaluating the RAG performance of the *S-Agent* and *Analytics-Agent* relies on another LLM, introducing a construct validity threat. To address this, we will construct a database of testing scenarios across major sUAS use cases that serve as representative benchmarks. These scenarios will act as ground truth, allowing us to compute scores such as BLEU [60] to fairly compare the quality of LLM-generated scenarios with human-generated ones.
- Our perception study offers preliminary insights into the framework's usefulness from the perspective of sUAS experts. However, a formal user study is needed to obtain deeper insights into the framework's effectiveness compared to existing sUAS simulation testing and analysis practices.

## X. Related Work

Testing CPS is a rather broad research area, ranging from assessing the accuracy of onboard AI models [29] applying fuzzing and metamorphic testing for generating test cases [71]. In this section, we focus on (1) sUAS simulation tools and testing, (2) the use of LLMs for testing, and (3)Automated Analysis of sUAS simulation results.

• **sUAS Simulation Testing:** In the domain of sUAS, researchers have developed various different simulation tools that support both Software-in-the-Loop (SITL) and Hardware-in-the-Loop (HITL) testing [37] such as Gazebo [59], AirSim [28], [66], and RFlySim [33]. These platforms serve as the foundation for testing their performance under controlled conditions. Besides these core simulation capabilities, the research community has further enhanced and adapted these platforms to meet specific sUAS testing needs, including LiDAR sensor simulation for testing sensor-based algorithms, environmental conditions like wind for evaluating sUAS navigational capabilities [13], [75], [76]. In this context, our framework can also be leveraged to generate initialization/configuration files, for example, specifying LiDAR sensor sensitivity or noise levels, and wind configurations. Furthermore, while these tools provide valuable support to developers and researchers, our framework offers an end-to-end simulation testing solution that covers scenario specification, execution, and analysis.

• **Testing using LLMs:** Recently, various ML-based approaches have emerged, particularly LLMs, for testing. For example, Arora *et al.* [24] used LLMs for test scenario generation. While their approach performs well and tests various functional aspects, it lacks a concrete structure to automatically execute the scenarios. For autonomous vehicles, Chang *et al.* [30] propose LLMScenario, using prompt engineering for scenario generation, while Zhang *et al.* [77] employ an LLM with Scenic Programming Language for creating safety-critical CARLA scenarios, though limited by its reliance on a specific language. In contrast, AutoSimTest is a language-agnostic, end-to-end framework for sUAS simulation testing, enabling seamless scenario generation, execution, and analytics across various simulation tools. Further, we used a RAG-based approach to decouple our framework from the underlying LLM. This provides flexibility to switch between state-of-the-art or objective-specific models such as Trustworthiness [41] or Functional Safety [67] to capture diverse perspectives when generating scenario blueprints.

• **Automated sUAS Flight Analytics:** Recent studies have demonstrated the efficacy of deep learning techniques in anomaly detection for automated analysis of sUAS flights [15], [42], [51]. By identifying unexpected spikes in sensor data, these methods provide valuable insights into abnormal sUAS behavior. Traditional anomaly detection approaches typically employ a bottom-up analysis, where expert developers leverage their prior knowledge to perform an in-depth examination of specific sensor data and diagnose the root cause of sUAS abnormal behavior. However, this approach has a significant limitation: it requires extensive domain-specific knowledge and expertise to analyze the complex sensor data. In contrast, our proposed *Analytics-Agent* employs a top-down approach, where developers initiate analysis with high-level investigation questions, identifying relevant flight controller parameters and drilling down to determine underlying causes of issues.

## XI. Conclusion

In this paper, we have presented a novel multi-LLM-agents-based framework, automating the sUAS simulation testing process. Our extensive experiments have demonstrated the framework's capabilities in testing diverse SuT, generating contextually relevant and diverse scenario blueprints, effectively analyzing flight logs, and providing valuable support to both novice and experienced developers in analyzing and understanding simulation results. The implications of AutoSimTest are twofold: (a) enabling exhaustive simulation testing and rapid iteration between development and simulation tests, and (b) facilitating the development and safe deployment of sUAS by substantially reducing the time and effort required for testing and validation, while achieving high simulation test coverage. For future work, we have planned additional studies with industry partners to validate the usability of AutoSimTest in testing more advanced and complex SuT. Furthermore, we are planning to extend our *Analytics-Agent* to support automated analysis of swarm sUAS.

## XII. Data Availability

All supplementary materials, including the framework codebase, are available at https://github.com/UAVLab-SLU/AutoSimTestFramework.

## XIII. Acknowledgment

The work in this paper was funded under USA National Aeronautics and Space Administration (NASA) Grant Number: 80NSSC23M0058

## References

[1] Dronelink. drone flight control for dji, autel drones. https://www.dronelink.com/. (Accessed on 07/29/2024).

[2] Drones for search and rescue — skydio. https://www.skydio.com/solutions/public-safety/search-and-rescue-drones. (Accessed on 07/01/2024).

[3] Gradio. https://www.gradio.app/. (Accessed on 07/21/2024).

[4] List of unmanned aerial vehicle-related incidents - wikipedia. https://en.wikipedia.org/wiki/List_of_unmanned_aerial_vehicle-related_incidents. (Accessed on 07/17/2024).

[5] microsoft/phi-3-mini-128k-instruct · hugging face. https://huggingface.co/microsoft/Phi-3-mini-128k-instruct. (Accessed on 07/21/2024).

[6] microsoft/phi-3-vision-128k-instruct · hugging face. https://huggingface.co/microsoft/Phi-3-vision-128k-instruct. (Accessed on 07/21/2024).

[7] NASA Unmanned Aircraft Systems (UAS) Reports. https://asrs.arc.nasa.gov/docs/rpsts/uas.pdf. (Accessed on 07/17/2024).

[8] System failure injection — px4 guide (main). https://docs.px4.io/main/en/debug/failure_injection.html. (Accessed on 11/07/2024).

[9] UAS - Air Accidents Investigation Branch reports - GOV.UK. https://www.gov.uk/aaib-reports?keywords=UAS. (Accessed on 07/17/2024).

[10] M. Abdin, S. A. Jacobs, A. A. Awan, J. Aneja, A. Awadallah, H. Awadalla, N. Bach, A. Bahree, A. Bakhtiari, H. Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[11] A. Agrawal, S. J. Abraham, B. Burger, C. Christine, L. Fraser, J. M. Hoeksema, S. Hwang, E. Travnik, S. Kumar, W. Scheirer, et al. The next generation of human-drone partnerships: Co-designing an emergency response system. In *Proc. of the 2020 CHI Conf. on Human Factors in Computing Systems*, pages 1–13, 2020.

[12] A. Agrawal, P. Zech, and M. Vierhauser. Coupled Requirements-Driven Testing of CPS: From Simulation to Reality . In *Proc. of the 32nd International Requirements Engineering Conference*, pages 337–344, Los Alamitos, CA, USA, Jun 2024. IEEE Computer Society.

[13] A. Agrawal, B. Zhang, Y. Shivalingaiah, M. Vierhauser, and J. Cleland-Huang. A requirements-driven platform for validating field operations of small uncrewed aerial vehicles. In *Proc. of the 31st Int'l Requirements Engineering Conf.*, pages 29–40. IEEE, 2023.

[14] M. N. Al Islam, M. T. Chowdhury, A. Agrawal, M. Murphy, R. Mehta, D. Kudriavtseva, J. Cleland-Huang, M. Vierhauser, and M. Chechik. Configuring mission-specific behavior in a product line of collaborating small unmanned aerial systems. *Journal of Systems and Software*, 197:111543, 2023.

[15] M. N. Al Islam, Y. Ma, P. Alarcon, N. Chawla, and J. Cleland-Huang. Resam: Requirements elicitation and specification for deep-learning anomaly models with applications to uav flight controllers. In *Proc. of the 30th Int'l Requirements Engineering Conf.*, pages 153–165. IEEE, 2022.

[16] A. Al-Mousa, B. H. Sababha, N. Al-Madi, A. Barghouthi, and R. Younisse. Utsim: A framework and simulator for uav air traffic integration, control, and communication. *International Journal of Advanced Robotic Systems*, 16(5):1729881419870937, 2019.

[17] B. S. Ali. Traffic management for drones flying in the city. *International Journal of Critical Infrastructure Protection*, 26:100310, 2019.

[18] Amazon.com. Amazon Drone Delivery. https://www.aboutamazon.com/news/operations/amazon-delivering-the-future-2023-announcements, 2023. (Accessed on 07/01/2024).

[19] H. Anand, S. A. Rees, Z. Chen, A. J. Poruthukaran, S. Bearman, L. G. P. Antervedi, and J. Das. Openuav cloud testbed: a collaborative design studio for field robotics. In *Proc. of the 17th Int'l Conf. on Automation Science and Engineering*, pages 724–731. IEEE, 2021.

[20] S. Anweiler and D. Piwowarski. Multicopter platform prototype for environmental monitoring. *Journal of Cleaner Production*, 155:204–211, 2017.

[21] ArduPilot. SITL Simulator. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html, 2018. (Accessed on 07/01/2024).

[22] Ardupilot. Ardupilot – open source autopilot software. https://ardupilot.org, 2020. (Accessed on 06/01/2024).

[23] Ardupilot. MissionPlanner. https://ardupilot.org/planner, 2022. (Accessed on 07/17/2024).

[24] C. Arora, T. Herda, and V. Homm. Generating test scenarios from nl requirements using retrieval-augmented llms: An industrial study. *arXiv preprint arXiv:2404.12772*, 2024.

[25] E. Atkins, A. Khalsa, and M. Groden. Commercial low-altitude uas operations in population centers. In *Proc. of the 9th Aviation Technology, Integration, and Operations Conf. and Aircraft Noise and Emissions Reduction Symposium*, page 7070, 2009.

[26] S. Bag, S. K. Kumar, and M. K. Tiwari. An efficient recommendation generation using relevant jaccard similarity. *Information Sciences*, 483:53–64, 2019.

[27] T. Benarbia and K. Kyamakya. A literature review of drone-based package delivery logistics systems and their implementation feasibility. *Sustainability*, 14(1):360, 2021.

[28] E. Bondi, D. Dey, A. Kapoor, J. Piavis, S. Shah, F. Fang, B. Dilkina, R. Hannaford, A. Iyer, L. Joppa, et al. Airsim-w: A simulation environment for wildlife conservation with uavs. In *Proc. of the 1st ACM Conf. on Computing and Sustainable Societies*, pages 1–12, 2018.

[29] J. Chandrasekaran, Y. Lei, R. Kacker, and D. R. Kuhn. A combinatorial approach to testing deep neural network-based autonomous driving systems. In *Proc. of the 2021 Int'l Conf. on Software Testing, Verification and Validation WS*, pages 57–66. IEEE, 2021.

[30] C. Chang, S. Wang, J. Zhang, J. Ge, and L. Li. Llmscenario: Large language model driven scenario generation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2024.

[31] B. Chen, Z. Zhang, N. Langrené, and S. Zhu. Unleashing the potential of prompt engineering in large language models: a comprehensive review. *arXiv preprint arXiv:2310.14735*, 2023.

[32] J. Cleland-Huang, A. Agrawal, M. N. A. Islam, E. Tsai, M. Van Speybroeck, and M. Vierhauser. Requirements-driven configuration of emergency response missions with small aerial vehicles. In *Proc. of the 24th ACM Conf. on Systems and Software Product Line: Volume A-Volume A*, pages 1–12, 2020.

[33] X. Dai, C. Ke, Q. Quan, and K.-Y. Cai. Rflysim: Automatic test platform for uav autopilot systems with fpga-based hardware-in-the-loop simulations. *Aerospace Science and Technology*, 114:106727, 2021.

[34] M. Dileep, A. Navaneeth, S. Ullagaddi, and A. Danti. A study and analysis on various types of agricultural drones and its applications. In *Proc. of the 5th Int'l Conf. on Research in Computational Intelligence and Communication Networks*, pages 181–185. IEEE, 2020.

[35] M. Erdelj and E. Natalizio. Uav-assisted disaster management: Applications and open issues. In *Proc. of the 2016 Int'l Conf. on Computing, Networking and Communications*, pages 1–5. IEEE, 2016.

[36] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert. Ragas: Automated evaluation of retrieval augmented generation. *arXiv preprint arXiv:2309.15217*, 2023.

[37] J. Giovagnola, J. B. M. Megías, M. M. Fernández, M. P. Cuéllar, and D. P. M. Santos. Airloop: A simulation framework for testing of uav services. *IEEE Access*, 11:23309–23325, 2023.

[38] J. González-Rocha, L. Bilyeu, S. D. Ross, H. Foroutan, S. J. Jacquemin, A. P. Ault, and D. G. Schmale. Sensing atmospheric flows in aquatic environments using a multirotor small uncrewed aircraft system (suas). *Environmental Science: Atmospheres*, 3(2):305–315, 2023.

[39] J. Gu, Z. Han, S. Chen, A. Beirami, B. He, G. Zhang, R. Liao, Y. Qin, V. Tresp, and P. Torr. A systematic survey of prompt engineering on vision-language foundation models. *arXiv preprint arXiv:2307.12980*, 2023.

[40] S. Guan, H. Sirianni, G. Wang, and Z. Zhu. suas monitoring of coastal environments: A review of best practices from field to lab. *Drones*, 6(6):142, 2022.

[41] W. Hua, X. Yang, M. Jin, Z. Li, W. Cheng, R. Tang, and Y. Zhang. Trustagent: Towards safe and trustworthy llm-based agents through agent constitution. In *Trustworthy Multi-modal Foundation Models and AI Agents (TiFA)*, 2024.

[42] M. N. A. Islam, J. Cleland-Huang, and M. Vierhauser. Adam: Adaptive monitoring of runtime anomalies in small uncrewed aerial systems. In *Proc. of the 19th Int'l Symp. on Software Engineering for Adaptive and Self-Managing Systems*, pages 44–55, 2024.

[43] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[44] Z. A. Jibon, M. A. Adnan, N. T. Nora, M. M. H. Akash, and F. Ahammed. Development of an autonomous uav for seed and fertilizer distribution in precision agriculture. In *Proc. of the 14th Int'l Conf. on Computing Communication and Networking Technologies*, pages 1–5. IEEE, 2023.

[45] E. Johnson and S. Mishra. Flight simulation for the development of an experimental uav. In *Proc. of the AIAA Modeling and Simulation Technologies Conference and Exhibit*, page 4975, 2002.

[46] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[47] G. Lachapelle, J. Henriksen, and T. Melgara. Seasonal effect of tree foliage on gps signal availability and accuracy for vehicular navigation. In *Proc. of the 7th Int'l Technical Meeting of the Satellite Division of The Institute of Navigation*, pages 527–532, 1994.

[48] C. Lee, R. Roy, M. Xu, J. Raiman, M. Shoeybi, B. Catanzaro, and W. Ping. Nv-embed: Improved techniques for training LLMs as generalist embedding models. *arXiv preprint arXiv:2405.17428*, 2024.

[49] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[50] D. Locascio, M. Levy, K. Ravikumar, B. German, S. I. Briceno, and D. N. Mavris. Evaluation of concepts of operations for suas package delivery. In *Proc. of the 16th AIAA Aviation Technology, Integration, and Operations Conf.*, page 4371, 2016.

[51] Y. Ma, M. N. Al Islam, J. Cleland-Huang, and N. V. Chawla. Detecting anomalies in small unmanned aerial systems via graphical normalizing flows. *IEEE Intelligent Systems*, 38(2):46–54, 2023.

[52] S. Malik, M. A. Khan, Aadam, H. El-Sayed, F. Iqbal, J. Khan, and O. Ullah. Carla+: An evolution of the carla simulator for complex environment using a probabilistic graphical model. *Drones*, 7(2):111, 2023.

[53] V. C. Martinez, B. Ince, P. K. Selvam, I. Petrunin, M. Seo, E. Anastassacos, P. G. Royall, A. Cole, A. Tsourdos, and S. Knorr. Detect and avoid considerations for safe suas operations in urban environments. pages 1–10, 2021.

[54] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende. Prompt engineering in large language models. In *Proc. of the Int'l Conf. on Data Intelligence and Cognitive Informatics*, pages 387–402. Springer, 2023.

[55] M. L. McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282, 2012.

[56] M. Messmer, B. Kiefer, L. A. Varga, and A. Zell. Uav-assisted maritime search and rescue: A holistic approach. *arXiv preprint arXiv:2403.14281*, 2024.

[57] Meta. Llama 3. https://llama.meta.com/llama3/, 2024. (Accessed on 08/01/2024).

[58] C. Olea, H. Tucker, J. Phelan, C. Pattison, S. Zhang, M. Lieb, and J. White. Evaluating persona prompting for question answering tasks. In *Proc. of the 10th Int'l Conf. on Artificial Intelligence and Soft Computing*, 2024.

[59] Open Robotics. Gazebo. https://gazebosim.org, 2023. (Accessed on 07/29/2024).

[60] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[61] PX4. Open Source Flight Controller. https://px4.io, 2021. (Accessed on 06/01/2024).

[62] PX4 Autopilot. PX4 Flight Review. https://review.px4.io, 2024. (Accessed on 08/01/2024).

[63] QGroundControl – Drone Control. QGroundControl. http://qgroundcontrol.com, 2022. (Accessed on 07/17/2024).

[64] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proc. of the 2019 Conf. on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[65] J. Scherer, S. Yahyanejad, S. Hayat, E. Yanmaz, T. Andre, A. Khan, V. Vukadinovic, C. Bettstetter, H. Hellwagner, and B. Rinner. An autonomous multi-UAV system for search and rescue. In *Proc. of the 1st Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pages 33–38, 2015.

[66] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics: Results of the 11th Int'l Conf.*, pages 621–635. Springer, 2018.

[67] L. Shi, B. Qi, J. Luo, Y. Zhang, Z. Liang, Z. Gao, W. Deng, and L. Sun. Aegis: An advanced llm-based multi-agent for intelligent functional safety engineering. *arXiv preprint arXiv:2410.12475*, 2024.

[68] H. Strobelt, A. Webson, V. Sanh, B. Hoover, J. Beyer, H. Pfister, and A. M. Rush. Interactive and visual prompt engineering for ad-hoc task adaptation with large language models. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):1146–1156, 2022.

[69] D. R. Thomas. A general inductive approach for qualitative data analysis. 2003.

[70] M. Vierhauser, M. N. A. Islam, A. Agrawal, J. Cleland-Huang, and J. Mason. Hazard analysis for human-on-the-loop interactions in suas systems. In *Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering*, pages 8–19, 2021.

[71] J. Wang, J. Chen, Y. Sun, X. Ma, D. Wang, J. Sun, and P. Cheng. Robot: Robustness-oriented testing for deep learning systems. In *Proc. of the Int'l Conf. on Software Engineering*, pages 300–311. IEEE, 2021.

[72] J. Wang, Z. Liu, L. Zhao, Z. Wu, C. Ma, S. Yu, H. Dai, Q. Yang, Y. Liu, S. Zhang, et al. Review of large vision models and visual prompt engineering. *Meta-Radiology*, page 100047, 2023.

[73] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*, 2023.

[74] M. Yadav, B. Vashisht, N. Vullaganti, S. Jalota, S. Yadav, G. Singh, A. Kumar, and S. Kothiyal. Iot-enabled unmanned aerial vehicle: An emerging trend in precision farming. *Artificial Intelligence and Smart Agriculture: Technology and Applications*, pages 271–292, 2024.

[75] B. Zhang and A. Agrawal. DroneWiS: Automated simulation testing of small unmanned aerial system in realistic windy conditions. In *Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering*, page 2358–2361. ACM, 2024.

[76] B. Zhang, Y. Shivalingaiah, and A. Agrawal. Dronereqvalidator: Facilitating high fidelity simulation testing for uncrewed aerial systems developers. In *Proc. of the 38th IEEE/ACM Int'l Conf. on Automated Software Engineering*, pages 2082–2085. IEEE, 2023.

[77] J. Zhang, C. Xu, and B. Li. Chatscene: Knowledge-enabled safety-critical scenario generation for autonomous vehicles. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, pages 15459–15469, 2024.

[78] Y. Zhao, P. Singh, H. Bhathena, B. Ramos, A. Joshi, S. Gadiyaram, and S. Sharma. Optimizing llm based retrieval augmented generation pipelines in the financial domain. In *Proc. of the 2024 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 279–294, 2024.

[79] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.