Evaluation of the Choice of LLM in a Multi-agent Solution for GUI-Test Generation

Stevan Tomic
Blekinge Institute of Technology
Karlskrona, Sweden
stevan.tomic@bth.se

Emil Alégroth
Blekinge Institute of Technology
Karlskrona, Sweden
emil.alegroth@bth.se

Maycel Isaac
Synteda
Gothenburg, Sweden
mi@synteda.com

***Abstract*—Automated testing, particularly for GUI-based systems, remains a costly and labor-intensive process and prone to errors. Despite advancements in automation, manual testing still dominates in industrial practice, resulting in delays, higher costs, and increased error rates. Large Language Models (LLMs) have shown great potential to automate tasks traditionally requiring human intervention, leveraging their cognitive-like abilities for test generation and evaluation.**

In this study, we present PathFinder, a Multi-Agent LLM (MALLM) framework that incorporates four agents responsible for (a) perception and summarization, (b) decision-making, (c) input handling and extraction, and (d) validation, which work collaboratively to automate exploratory web-based GUI testing.

The goal of this study is to assess how different LLMs, applied to different agents, affect the efficacy of automated exploratory GUI testing. We evaluate PathFinder with three models, Mistral-Nemo, Gemma2, and Llama3.1, on four e-commerce websites. Thus, 27 permutations of the LLMs, across three agents (excluding the validation agent), to test the hypothesis that a solution with multiple agents, each using different LLMs, is more efficacious (efficient and effective) than a multi-agent solution where all agents use the same LLM.

The results indicate that the choice of LLM constellation (combination of LLMs) significantly impacts efficacy, suggesting that a single LLM across agents may yield the best balance of efficacy (measured by F1-score). Hypothesis to explain this result include, but are not limited to: improved decision-making consistency and reduced task coordination discrepancies. The contributions of this study are an architecture for MALLM-based GUI testing, empirical results on its performance, and novel insights into how LLM selection impacts the efficacy of automated testing.

***Index Terms*—Multi-Agent Systems, Large Language Models (LLMs), Automated Testing, MALLM, AI-Assisted Software Testing**

I. Introduction

For decades, completely automated testing has been coveted by both researchers and industrial practitioners, resulting in a myriad of solutions from random testing to meta-heuristics to machine learning and artificial intelligence [1], [2]. These solutions are not restricted to any level of test abstraction, ranging from low-level unit tests to high-level GUI-based tests [2]. For GUI-based testing, automation is particularly appealing since this practice is still, to a large extent, a manual, costly and error-prone practice in industry, e.g. for exploratory testing [3]. These tests, although effective, rely on human cognition for test creation and result evaluation, making complete automation a challenge [4].

However, with the advent of Generative Artificial Intelligence (GenAI), such as Large Language Models (LLMs), great strides have been taken towards this goal [5], [6]. By leveraging the emulated cognitive capabilities of LLMs, tasks that previously required human intervention can now be automated [6]. Whilst results show promise, there are still many hurdles to cross. For instance, while research indicate efficiency gains of these solutions, effectiveness in terms of human-equivalent outputs is still a challenge [7].

The effectiveness of an LLM-based solution is dependent on multiple factors [7], including prompt-engineering [8] and contextualization (e.g. Retrieval Augmented Generation (RAG) or multi-shot prompting [9], [10]). Improvements have also been reported in using LLM agents where the models' general capabilities are contextualized and aligned with a specific purpose, e.g. by setting the LLM in a specific role such as a developer or tester [11]. Further successes have been reported by using multi-agent LLM systems (or MALLMs) where several autonomous agents collaborate to co-produce results [11], [5]. Yet another influential factor is the training data of the models themselves, i.e. the model's knowledge for a specific, or type of, use case, e.g. human-machine interaction, math or document analysis [12]. These capabilities are generally evaluated quantitatively and compared between models using benchmarks, such as HellaSwag or Helm [13]. Inference based on these benchmark tests, which indicate varying capabilities for different models, leads to the assumption that different models are more or less suited in different use cases [14]. Ergo, in a MALLM based solution, the choice of LLM for each agent could have a significant impact on the solution's effectiveness, e.g. measured as precision and accuracy [15]. However, to the best of our knowledge, no study has investigate how the choice of LLM affects the outcome of a MALLM-based solution's ability to generate exploratory GUI tests.

We perform an experiment with three LLMs, Mistral-Nemo, Gemma2 and LLama3.1, integrated to act as agents in a MALLM demonstrator for automated exploration of web-based e-commerce systems. The demonstrator contains four different agents for: (1) summarization, (2) decision making, (3) data extraction and (4) validation, which collaborate in an iterative process to solve the high level goal of purchasing an item. Agents developed for this study are designed with the flexibility to switch between different LLMs rather than being

bound to a single model. In the experiment, the demonstrator is applied on four subject e-commerce web-applications where each permutation of the three LLMs are used for agents 1-3.

Results show that the choice of LLM for a specific agent has an impact on the demonstrator's ability to complete the task, but also that the constellation of LLMs for the different agents plays a role.

The contributions of this work are:

- The architecture and components of a MALLM based demonstrator for automated GUI-based test exploration of e-commerce systems.
- Empirical results about the demonstrator's ability to generate valid exploratory test scenarios for web-based e-commerce systems.
- Results on the effects of the choice of LLM, or constellation of LLMs, in different agent roles on output from a MALLM based demonstrator solution.
- Insights to explain the effects of LLM choice in a MALLM that affect the efficacy (effectiveness and efficiency) of the solution.

The continuation of this paper is structured as follows. In Section II related work is summarized, followed in Section III by a description of the demonstrator and experiment evaluation. Section IV then presents the results that are later discussed in terms of impact and value to industry and academia in Section VI. Finally, the paper is concluded in Section VII.

II. Related work

GenAI has quickly grown as a research field, with multiple avenues of research, and in the following section we provide a brief overview of related topics and studies.

One area of research is the use of agents, where the GenAI (commonly an LLM) is instructed to act as a specific role or perform a specific task. Research into using multi-agents has also been proposed, either with set or dynamic agent architectures [16]. In the study by Liu et al., the researchers propose a dynamic system where agents are activated based on the task to be solved. Several studies have also analyzed using different LLMs as agents to measure their comparative performance [17], [18], [19]. However, common to most of these studies is that all agents use the same model. Examples of multi-agent LLM systems exist, such as the study by Yuan et al. [20], where a multi-agent and multi-LLM system is used for code translation. Similarly, the study of Wang et al. [21] looked at the use of multiple LLMs, using discussion and cmd, as agents. However, unlike our study, these works do not evaluate the impact of varying the constellations of LLM models in different agents.

Furthermore, the field of GenAI-based software engineering is rapidly growing, including in the area of software testing [22], [23]. One example is a study by Yoon et al. [5] where they introduce DroidAgent, a tool for autonomous test generation for Android applications. The study shows that their tool outperforms existing state-of-the-art tools in terms of coverage, achieving 10 percent higher coverage than the baseline. Another, related study, is provided by Liu et al. [6] and their tool GPTDroid, once more a tool for Android test generation. In their study with a sample of 93 applications from Google Play, the tool is shown to outperform the baseline by 32 percent, identifying 53 previously unknown faults in the tested applications.

In conclusion, multi-agent tools have been studied for test generation, but, to the best of our knowledge, our study is the first to investigate the impact of how choosing different LLMs as different agents affect the efficacy of the test generation for web applications.

III. Methodology

1) Goal.: This study aims to explore the performance impact of using different locally hosted LLM models within a MALLM framework for automated GUI testing. To achieve this, we developed PathFinder, a framework that automates GUI-level website testing using local LLMs. The decision to use local models stems from the fact that many companies avoid online models due to privacy concerns. Additionally, we assess whether smaller models, which require fewer resources, can still deliver effective performance for automated testing. This goal was formulated in collaboration with our partner company, Synteda, which is a Swedish SME, developing software products and services using online e-commerce applications.

2) Research questions.: The study is centered on the following research question:

RQ1 *What is the efficacy of combining specialized LLM models within a multi-agent GUI testing framework?*

3) Hypothesis: To answer the research question, we aim to test the following two hypotheses.

- **Hypothesis (H1):** Combining specialized LLM models across agents in a MALLM system will outperform the use of a single LLM across all agents for a *specific* system under test.
- **Hypothesis (H2):** Combining specialized LLM models across agents in a MALLM system will outperform the use of a single LLM across all agents for a *generic* system under test.

A. Demonstrator

PathFinder is a MALLM framework, its architecture visualized in Figure 1, which is designed to autonomously achieve specific goals, such as completing a purchase on an e-commerce web-application. The framework consists of four agents; Perceptron agent (summarization), System 1 agent (decision-making), Entry Bot (data extraction), and Oracle agent (validation), which interact in an iterative process.

1) Perceptron Agent: The Perceptron Agent gathers, processes, and organizes relevant data from web pages, serving as the foundation for subsequent decision-making steps in the MALLM system.

Input. The agent receives two primary inputs: the URL of the web page and a specific task instruction, such as "make an online purchase," that the agents are expected to complete.

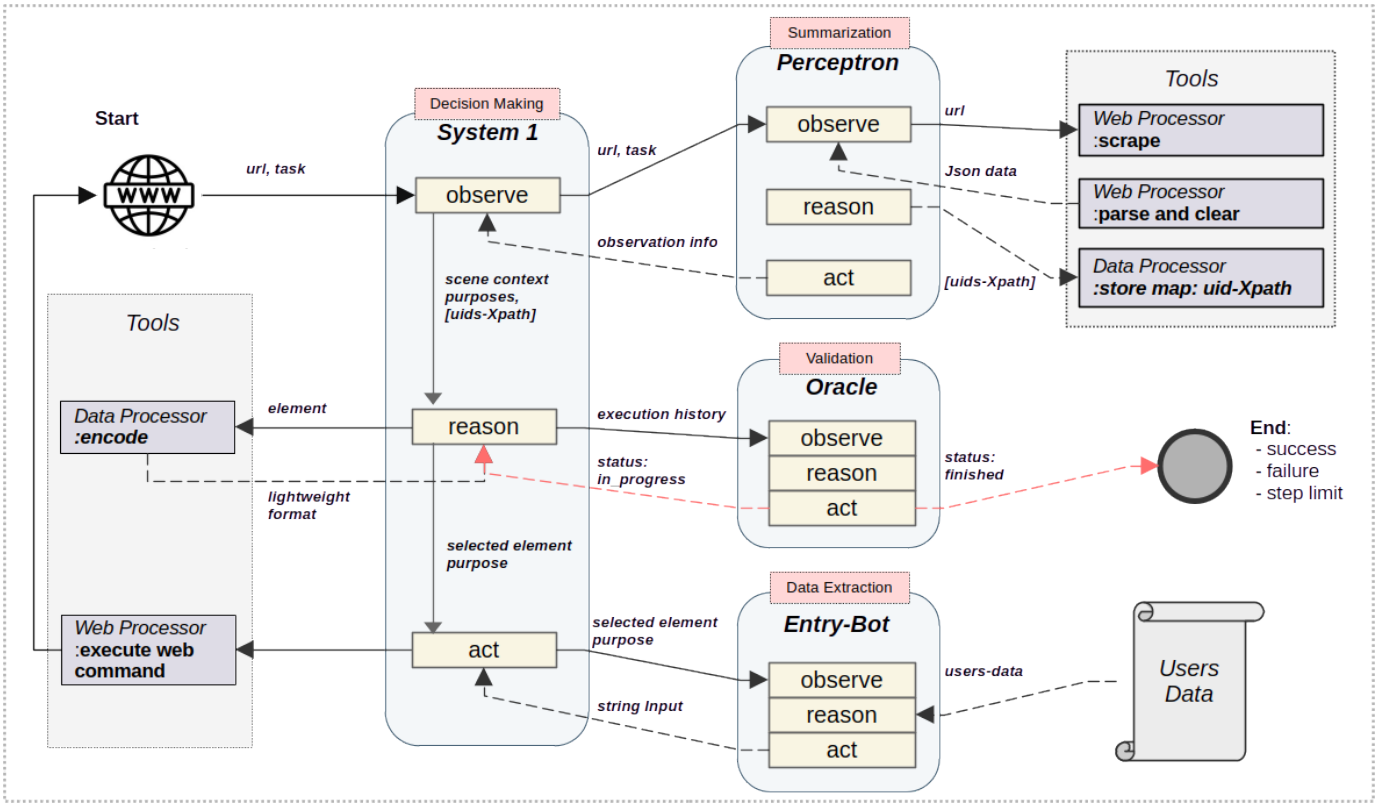

Fig. 1. Agents organization in PathFinder Framework, where Perceptron, System 1, Entry-Bot, and Oracle collaborate to achieve a goal. Perceptron scrapes and parses web data into actionable insights, which System 1 uses to decide and execute the next web action. Entry-Bot provides data input when required, and Oracle validates progress toward the goal. Each agent follows the Observe-Reason-Act cycle, enabling efficient and goal-directed task execution.

Processing. The agent collects and processes data from a web page by using a WebProcessor component. The WebProcessor's scrapes the HTML content, extracts component attributes, creates a unique XPath for each element, and stores the result in a JSON format. This structure is further parsed and filtered by the WebProcessor's parsing component, which analyses each element for visibility and interactability. This analysis ensures that hidden, disabled, or obstructed web elements, which are irrelevant to the user, are excluded. The elements of the JSON structure are then separated into two categories, (a) informational elements, which provide context or content, i.e., elements meant for labeling, visualization, or guiding users on a web page; and (b) actionable elements, i.e. interactive components such as buttons, links, and input fields. Each actionable element is assigned a unique identifier (UID) to maintain traceability and ensure element identification for the downstream execution. The UIDs are created by concatenating key attributes of the HTML elements: `unique_name = f"tag_key_attributes_elements_counter"`. For large web pages, the Perceptron agent trims the processed prompt if it exceeds a predefined size limit and prioritizes actionable elements over other less critical information. The limit is a configurable variable, set to 8192×4 characters, which roughly corresponds to a 8k context window size, assuming each token takes an average of 4 characters.

Prompting. The agent prompts the LLM with a URL, a task, and two lists of actionable and non-actionable web elements. The LLM is tasked with analyzing each actionable element by referencing both lists to determine and summarize their specific purposes within the context of the task.

Output. The output from the Perceptron agent is returned in a JSON format and consists of: `SceneIdentification`: The current page primary purpose (e.g. a login page), `WebPageSummary`: A brief summary of the state of the web page, `ActionableElementsPurpose`: Descriptions of each actionable element in the format `'unique_identifier': 'purpose summary'`, which explains the role of each element in the context of achieving the overall task. PathFinder uses UIDs since they strike a balance between being shorter and less complex than XPaths while maintaining semantic associations with elements' tags and attributes. Using web element XPaths as identifiers is impractical in prompts due to size and simpler identifiers, e.g. numeric identifiers, add risk of hallucinations. However, while UIDs are advantageous for prompting, Perceptron can produce them incorrectly. Therefore, PathFinder uses Python's `SequenceMatcher` [24] for fuzzy string matching to align the LLM's output with the most similar original element identifier.

2) System 1 Agent: The System 1 Agent is responsible for taking the decision of what web element to interact with to advance progress towards task completion. *Input.* System 1

takes the outputs from the Perceptron agent, i.e. a scene with actionable and non-actionale components, as input. Additionally, it uses a history of its past choices, denoted as steps. Each step consists of a step number, an action description, including the type of the action (i.e. 'click', or 'input') along with any applicable input text, and the web element's UID. Finally, System 1 has access to a UID-to-XPath Map, which is necessary to locate a web element when executing actions.

Processing. Before prompting the System 1 agent for a decision, each actionable element is encoded using the DataProcessor:encode method, which selects common attributes—such as tag names, IDs, classes, and other relevant properties—while limiting the length of their values to prevent excessive detail. The output of the encoder is a tuple with the structure: `'element_uid': (<lightweight_format>, <purpose>)`. This representation helps optimize the LLM prompt while maintaining enough detail to allow for accurate decision-making.

Prompting. System 1 prompts the LLM to analyze the current scene, actionable elements, and past actions to identify and select the next element for interaction, guiding the task towards completion. The agent is also given guidelines, which focus on objective clarification, repetition avoidance, contextual awareness, etc., which, for the sake of brevity, will not be discussed here. For details, we refer the reader to the provided replication package [25].

Output. The System 1 agent produces a JSON output that includes the following: `CurrentTask`: The LLM is asked to repeat the overall task (completing a purchase), encouraging focus on the goal before providing a final answer. `ActionsHistory`: An updated summary of the actions taken so far. `SelectedElementRationale`: A rationale for why the selected element was chosen, employed in the chain-of-thought [26] technique to promote step-by-step deliberation before making a final decision. `SelectedElement`: The `unique_identifier` (UID) of the selected element. Finally, the parsed UID from `SelectedElement` is passed to the WebProcessor for execution via Selenium. However, if the action is a type action that requires specific textual input, the EntryBot Agent is invoked to provide the appropriate string.

3) EntryBot Agent: The EntryBot agent is responsible for providing text inputs to web elements by extracting relevant information from a user-data document, e.g. username, email, or address. This allows the MALLM to interact with forms, or other text input-based elements, where the LLM cannot generate a suitable or valid input, e.g. a password.

Input. EntryBot receives the outputs from the System 1 agent as input, including `web_element_description`, and `web_element_purpose`. Additionally, it obtains `data_file_content`, which contains YAML formatted data from which the appropriate text for the queried input element can be extracted.

Prompting. The EntryBot agent has access to the user's ID and name, e.g. *"userID: 12345, named: John Doe"*, which it uses to extract corresponding information related to the user.

Output. The EntryBot agent produces a JSON formatted output that includes the following: `CurrentTask`: making the LLM repeating the overall task. `SelectedInput`: The string chosen from the user's data file that best matches the input element's purpose. `Rationale`: A motivation for selecting the particular input, useful for analysis on LLMs decisions. The value obtained by `SelectedInput` is returned to the System 1 agent for execution.

4) Oracle Agent: The Oracle agent operates as the validator within the PathFinder framework, assessing whether the task is completed, is unachievable, or remains in progress. This decision is based on the cumulative history of taken actions and predefined success criteria.

Input. The Oracle agent is invoked every fifth step by the System 1 agent to assess overall progress. In addition to the task and the current URL, the Oracle receives an enriched history of actions from System 1, extended with information about the scene summary of each taken step. The agent also utilizes a set of configurable success criteria that provide a high-level goal for the task, `criteria`, upon which it bases its decision about task attainment.

Prompting. The purpose of the Oracle agent is to determine the task's status. This assessment is iterative, where the agent classifies the task status into one of the following:

- *success* - the task is completed,
- *stuck* - the tested system has reached a state which prohibits task attainment,
- *in_progress* - additional actions are required and possible to achieve task attainment.

Output. The output from the Oracle agent is structured in a JSON format with two key fields: `Rationale` and `TaskStatus`. The `Rationale` field provides a concise motivation for the task's status classification and the `TaskStatus` field categorizes the task's current status. If the classification is *success* or *stuck*, the Oracle terminates the execution. Otherwise, if the classification is *in_progress*, PathFinder begins a new iteration.

5) Error handling: PathFinder is based around JSON formatted data, which is subject to erroneous outputs. To mitigate JSON parsing failures, or incomplete responses from the LLM, all agents employ a recovery mechanism by re-prompting the LLM with an increased temperature parameter (+0.1). This adjustment encourages variability in the generated response, enhancing the likelihood of receiving a more accurate or complete output. Upon a successful response, the temperature and other agent parameters are reset to their initial configurations, ensuring consistency for subsequent steps.

B. Experimental Setup

The core objective of this experiment is to compare the impact of using combinations of LLM models, each with different capabilities, across agents in a MALLM based system, i.e., PathFinder in our case. Specifically, the experiment seeks to verify the hypothesis that a combination of LLMs will be more efficacious, in terms of effectiveness (measured as *F1-score*) and efficiency (measured as *time expenditure* to

complete the task), compared to using a single model in all agents.

1) System Configuration: To optimize PathFinder's performance, each agent of the system was configured with parameters that were acquired during development or through pilot-experiments. The details and specific setup of the parameters are available within our replication package [25]; here, we provide a brief overview of most relevant parameter.

The parameter `num_ctx` sets the context window size for the agents in number of tokens. Perceptron agent used a 8K context window necessary for processing web page information. Others agents were limited to 4K. The models' `temperature:0.1` was kept low for all agents to promote more deterministic behavior, but still above zero to promote flexibility in decision-making. Additionally, each agent has specific parameters tailored to its respective role within the system.

Perceptron agent has three parameters. First, `max_attrib_length:20` limits the number of characters to 20 for attribute values. Second, `max_web_element_length:200` constrains the total length of the description for an actionable element. Finally, `max_children_per_element` restricts the number of inner elements within a web element to a maximum of two, focusing on elements closer to the DOM leaf, as actionable elements are typically found near the lower levels of the DOM hierarchy. A filtering mechanism is applied to select specific tags and attributes that describe actionable elements. For example, the `BUTTON` and `INPUT` tags incorporate attributes such as `tag`, `type`, `id`, `text`, `placeholder`, `class`, `value`. Similarly, the `A` tag shares most of these attributes but includes additional attributes such as `href` (specifying the link URL) and `name` (used for named anchors). During PathFinder's development, it was observed that filtering is essential for reducing prompt size, as longer, irrelevant attributes increase the risk of hallucinations, hence degrading model performance.

System 1 agent also has three parameters. First, `max_trajectory_steps:16`, which specifies how many steps will be saved in the history and thereby controlling the prompt size. Second, `max_number_of_steps:31` represents the maximum number of steps allowed to complete the task, which, if exceeded, terminates the execution with status *expired*. Finally, the `oracle_frequency:5` parameter indicates the number of steps after which the Oracle agent is called to assess the task's progress.

For EntryBot agent, the only parameter is `filepath`, which defines the relative path to a YAML file that stores user data, while Oracle has two parameters that serve as criteria to determine whether the execution is deemed successful or failed, respectively: `success_condition` and `stuck_condition`.

2) Used LLMs: When selecting models, we chose offline models with perceived different capabilities that, given our hypothesis, could have synergistic effects. For this purpose, we chose Gemma2 [27], Llama3.1 [28],, and MistralNemo [29]. The models used in the experiments are 4-bit quantizations of the originals, enabling them to run efficiently on local hardware without sacrificing too much accuracy and improving both inference speed and resource efficiency (see Section III-B5 for details).

TABLE I
MODEL PERFORMANCE BY CATEGORY

Category & Test	Gemma2	Llama 3.1	Mistral-Nemo
General Knowledge (MMLU, AGIEval, TruthfulQA, Boolq)	71.3, 52.8 –, 84.2	73.0, – –, –	68.0, – 50.3, –
Reasoning (ARC-C, Winogrande, OpenBookQA, ARC-e)	68.4, 80.6 –, 88.0	83.4, – –, –	–, 76.8 60.6, –
Math (GSM8K, MATH)	68.6, 36.6	84.5, 51.9	NA
Code (HumanEval, MBPP)	40.2, 52.4	72.6, –	NA
Commonsense/Logic (HellaSwag, PIQA, SIQA)	81.9, 81.7 53.4	NA	83.5, 70.4 –, –
Question Answering (DROP, GPQA, CommonSense QA)	69.4, – –	–, 32.8 –	–, – 70.4

Table I presents the results of our analysis of the LLMs different capabilities, taken from LLM's corresponding websites [27], [28], [29]. *Gemma2* demonstrates strong performance in commonsense reasoning and logic tasks, scoring highly on benchmarks like HellaSwag and PIQA. This makes it a solid choice for tasks requiring efficient text output and good reasoning capabilities. However, in complex tasks such as advanced mathematics and coding skills, Gemma2 tends to underperform compared to larger models. *Mistral-Nemo* exhibits strong common sense reasoning and high truthfulness, achieving top scores in HellaSwag and TruthfulQA among the models. This makes it suitable for tasks where these qualities are paramount. However, its performance in advanced reasoning and general knowledge tasks is lower than that of the other models, which may limit its effectiveness in tasks requiring extensive knowledge or precise reasoning. Despite this, it maintains solid performance in commonsense and logic tasks. *Llama 3.1* consistently ranks as a top performer across the considered benchmarks. It excels in mathematics and coding tasks, outperforming other models in GSM8K, MATH, and HumanEval. Its versatility and strong performance across various tasks make it a robust choice for handling multistep reasoning and detailed response generation.

3) Task and Website Selection: To evaluate PahtFinder's performance in a valid and generalizable way, both the task and web application subjects were carefully selected.

Task Description: The task chosen for this experiment was *'make an online purchase'*, which is a critical function in any e-commerce system. This task also includes inherent complexities as it requires navigating product listings, adding items to the shopping cart, interacting with forms, and completing purchases. Hence, features that are common to all e-commerce systems but which are implemented with different appearance, user flows and prerequisites. By focusing on this task, the

experiments challenge PathFinder's capability to handle realistic and meaningful interactions in an environment where effectiveness and efficiency are important. This evaluation was also relevant to our partner company for the experiment.

Websites: The experiments were conducted on four e-commerce websites selected to progressively increase in complexity. The selected websites included two demo platforms; saucedemo.com and demo.spreecommerce.org, which are widely used by testers and developers to experiment with automation and navigation scripts, providing a controlled environment for initial trials. To complement these sites, ikea.com and netonnet.se were used to represent real-world e-commerce platforms with higher complexity in their design and functionality.

4) LLM Permutation Strategy and Execution: In the experiment, we evaluated all possible permutations of the three LLM models, Mistral-Nemo, Llama3.1, and Gemma2, across three of the four agents in PathFinder, namely Perceptron, System 1, and EntryBot. All permutations were tested on each of the four chosen e-commerce websites and run 10 times with each permutation to acquire robust results. Thereby, 1080 runs were performed in total, where each run, depending on site complexity, varied between 10 to 20 steps. The maximum step limit for the experiment was set to 31 (allowing execution to terminate on step 32), which was required to keep execution time for the experiment manageable whilst still providing PathFinder some room to recover from incorrect decisions during a run. Furthermore, again to make the experiment manageable in terms of time, the LLM model for the Oracle agent was fixed across all experimental runs. Based on insights gained from development and preliminary pilot experiments, Gemma2 was selected as the Oracle model. This decision was informed by the observation that other models tended to prematurely report task completion, interrupting execution before the task was actually completed. By contrast, Gemma2 demonstrated more consistent and reliable task evaluation, reducing the occurrence of false positive task completions and ensuring a more accurate assessment of when a task was fully achieved or irrevocably stuck. Whilst presenting a limitation to our study, we do not perceive this design choice as a threat to answer the study's research question.

5) Hardware and Execution Environment: To ensure consistency and reproducibility of the results, the experiments were performed in a controlled hardware and software environment. *Hardware:* The experiments were conducted on a MacBook Pro with an Apple M1 Max chip, 32 GB of memory, running macOS Sonoma 14.6.1. The device features a built-in Liquid Retina XDR display with a resolution of 3456x2234 and a 1 TB SSD for storage. *Software:* For the software setup, we used Ollama [30] version `0.3.10` to run the three LLM models. The agents were implemented in Python, with all experiments executed using Python `3.12.3` and developed within the PyCharm IDE. *PathFinder Implementation:* The PathFinder framework consists of several packages, with key components including `aiframe` for agent implementation, `webhandler` for managing experiment results and facilitating human evaluations, and `data_processor` and `jupyter` for data formatting and evaluation through notebooks. Full implementation details and analysis processes are available in the provided software package [25].

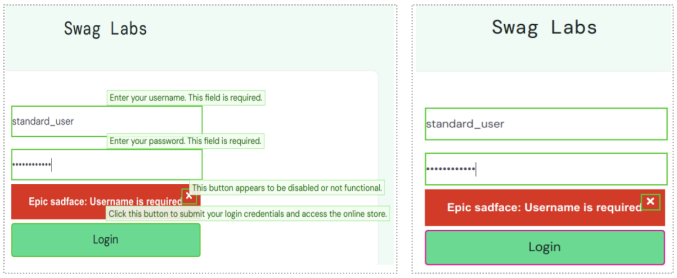

Fig. 2. Interface for Human Evaluation (first two columns for a step)

C. Experimental Procedure and Methodology

This experimental study was inspired by the guidelines for performing experiments in software engineering by Wohlin et al. [31].

1) Experimental Runs: The experiments were initiated by configuring PathFinder and setting agent parameters as specified in Section III-B1. PathFinder systematically permuted the LLMs across agents, with each combination of models referred to as a *constellation*. Each constellation was executed 10 times, with each individual run termed a *session*. While the Perceptron and System 1 agents executed each step, the Oracle agent was invoked every fifth step to assess task status. The EntryBot agent was triggered when System 1 selected an element requiring text input, making EntryBot's activation frequency variable across the experiments. Upon completion of each experiment, the experimenter updated the URL, hence changing the SUT.

2) Human Evaluation of Results: After completing the experiments for all SUTs, the `webhandler` package was used to generate a local web server that displayed the results for all constellations and their corresponding sessions. Each session was represented in a report, where rows corresponded to steps and columns provided detailed information for each step:

- The first column presented a screenshot of the actual web page (SUT), where actionable elements were highlighted and a small text label, indicating the purpose of each element, was placed over each element.
- The second column displayed the selected element, highlighted on the screenshot (see Figure 2).
- The third column provided textual data, including the execution state, total step time, the purpose of the selected element, and any entered text inputs.

At the end of each session, the experimenter entered a comment on the session/run, which were stored in the corresponding JSON files.

Human evaluators classified each outcome according to our defined criteria, as follows:

- **True Positive (TP)**: The Oracle agent reports success, and the task is successfully completed.
- **False Positive (FP)**: The Oracle agent reports success, but the task is not actually completed.
- **True Negative (TN)**: The Oracle agent reports failure, and the task was indeed impossible to complete.
- **False Negative (FN)**: The Oracle agent reports failure or timeout, but the task could theoretically be completed.

The experimenter manually evaluated all 1,080 runs (27 constellations across 4 websites with 10 sessions each), assigning each session as TP, FP, TN or FN. Assignment of TP, FP was based on visual inspection of the final state of each session. For TN and FN, the experimenter looked at the complete session and used their expert judgement to classify the session's status and probability of success. To reduce risk of researcher bias or error, the rubric for classification, as well as ambiguous cases, were discussed among the authors to reach consensus.

3) Data Processing and Analysis: After collecting agent data and human evaluations, the post-processing tool from the `data_processor` package was used to generate Pandas dataframes. The extracted data is then compiled, which are subsequently saved as .csv files that were further analyzed in Jupyter notebooks. For transparency, the post-processing script, including data extraction and formatting steps, is provided in the replication software package [25].

4) Analysis Methodology: After classifying all sessions, standard formulas were applied to calculate F1-score based on standard formulas for precision and recall. F1-score was chosen for the evaluation because it's a standardized metric for evaluating an LLM-based application's ability to return relevant and not missing relevant information. The formula for F1-score is presented below:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Since LLMs are regarded as black-box, our analysis used both qualitative and quantitative methods to explore potential hypotheses explaining the results. For the qualitative analysis, we analyzed the generated reports of successful and unsuccessful sessions manually to detect patterns or anomalies that might explain the performance. The objective was to identify behaviors or trends that were specific to certain LLM constellations.

To strengthen the qualitative findings, we employed quantitative analysis and descriptive statistics that went beyond correlating hypotheses with experimental data. Instead, we extracted statistical and syntactic text features from the LLM-generated text to validate our assumptions. Features, such as text length, part-of-speech counts and common phrases used by the model, were compared across the different sessions. Furthermore, decision tree models were employed to confirm the importance of specific attributes. This allowed us to quantify the influence of these features in the experiment, offering data-driven support of the qualitative insights.

IV. Results

Table II presents a summary of the results acquired from experiment, divided into four metrics for each constellation: F1-score for each website individually, F1-score across all websites, the average (median) number of steps required to complete the task, and the average (median) execution time per session. The best-performing constellation of LLMs per-website and across websites are highlighted in bold. The `Step Number` and `Duration` columns remain unmarked, as a low number of steps or shorter duration may suggest efficiency but can be misleading if the model does not perform effectively.

The following text provides a detailed explanation of the table and its resulting columns.

A. Effectiveness

Per-Site Evaluation: When applying PathFinder with different constellations of LLMs on individual websites, results indicate that using the same LLM model across all agents yielded the best effectiveness. For demo1 this was achieved using *gemma2 gemma2 gemma2* (*F1-score: 0.889)*, for demo2 it was achieved with 12 constellations (*F1-score: 1.0)*, for Ikea it was achieved with *lama3.1_llama3.1_llama3.1* (*F1-score: 0.947)* and finally, for Netonnet, with two one-model LLMs, *gemma2 gemma2 gemma2* and *lama3.1_llama3.1_llama3.1* (*F1-score: 0.667)*. Thus, we cannot accept Hypothesis H1 (see Section III-3), which proposes that a mixed model approach would outperform the use of a single model across all agents.

Cross-Site Evaluation: When taking a cross-site perspective to applying PathFinder, we get different results, i.e. that using different (mixed) LLM model across all agents yielded the best effectiveness. This was concluded using the average F1-score over the tested websites that shows that *llama3.1_mistral-nemo_mistral-nemo* is the best (*Average F1-score: 0.703*. Hence, we can accept the H2 hypothesis that using mixed LLM for all agents will produce a better result compared to using the same LLM model.

B. Efficiency

Efficiency was a secondary focus of this study, measured as number of steps and execution time and therefore, due to page restrictions, we only highlight some of our more interesting observations below. For the complete dataset, we refer the reader to the replication package [25].

The results revealed a huge variance in number of steps required to complete the task between different constellations on different websites. For the less complex sites, the median number of steps was lower, `demo1:15.0, demo2:17.5`, compared to the median of the more complex websites, `ikea:20.0, netonnet:25.0`. The complexity of the websites influenced both the number of steps and the time taken per step. For instance, demo1 averaged about one minute per step (15 minutes total over 15 steps), while Netonnet took around two minutes per step (56 minutes total over 28.5 steps). This finding underscores the differences in website complexity, including the number and type of web elements that the agents need to analyze to achieve task completion.

TABLE II
SUMMARY OF THE RESULTS OBTAINED FROM THE EXPERIMENTS FOR ALL PERMUTATIONS OF LLMS AND WEBSITES.

Constellation	**F1-score** demo1, demo2, ikea, netonnet	**F1-score accross sites**	**Step Number** demo1, demo2, ikea, netonnet	**Duration (minutes)** demo1, demo2, ikea, netonnet
gemma2_gemma2_gemma2	**0.889**, 0.75, 0.333, **0.667**	0.660	15.0, 17.5, 32.0, 28.5	15.05, 27.63, 54.93, 56.0
gemma2_gemma2_llama3.1	0.462, 0.571, 0.0, 0.333	0.342	15.0, 32.0, 32.0, 28.5	14.56, 50.68, 72.49, 61.22
gemma2_gemma2_mistral-nemo	0.824, 0.824, 0.462, 0.571	0.670	17.5, 20.0, 28.5, 28.5	16.88, 32.26, 49.78, 56.66
gemma2_llama3.1_gemma2	0.0, 0.947, 0.462, 0.333	0.436	5.0, 20.0, 20.0, 27.5	2.98, 27.62, 34.93, 47.42
gemma2_llama3.1_llama3.1	0.0, 1.0, 0.462, 0.571	0.508	12.5, 20.0, 15.0, 17.5	10.47, 24.39, 27.23, 33.55
gemma2_llama3.1_mistral-nemo	0.0, 0.947, 0.571, 0.571	0.523	10.0, 17.5, 28.5, 30.0	11.12, 24.62, 36.32, 58.75
gemma2_mistral-nemo_gemma2	0.667, 1.0, 0.462, 0.333	0.615	15.0, 25.0, 32.0, 31.0	12.13, 31.56, 59.96, 68.86
gemma2_mistral-nemo_llama3.1	0.75, 0.889, 0.75, 0.333	0.681	14.5, 25.0, 17.5, 17.5	12.0, 35.43, 34.07, 40.05
gemma2_mistral-nemo_mistral-nemo	0.75, 0.947, 0.571, 0.462	0.683	15.0, 22.5, 22.5, 22.5	10.71, 27.3, 38.96, 50.2
llama3.1_gemma2_gemma2	0.182, 1.0, 0.333, 0.182	0.424	10.0, 10.0, 32.0, 32.0	7.64, 9.12, 51.22, 57.56
llama3.1_gemma2_llama3.1	0.0, 1.0, 0.571, 0.462	0.508	12.5, 10.0, 25.0, 31.0	8.94, 8.8, 29.6, 56.97
llama3.1_gemma2_mistral-nemo	0.182, 1.0, 0.571, 0.182	0.484	10.0, 10.0, 17.5, 32.0	8.06, 9.0, 28.95, 53.82
llama3.1_llama3.1_gemma2	0.182, 1.0, 0.824, 0.182	0.547	12.5, 15.0, 20.0, 25.0	9.9, 18.59, 27.1, 37.16
llama3.1_llama3.1_llama3.1	0.0, 1.0, **0.947**, **0.667**	0.654	20.0, 17.5, 17.5, 20.0	19.87, 20.52, 25.7, 26.78
llama3.1_llama3.1_mistral-nemo	0.0, 1.0, 0.667, 0.182	0.462	15.0, 15.0, 20.0, 20.0	14.32, 19.21, 24.48, 32.47
llama3.1_mistral-nemo_gemma2	0.462, 1.0, 0.75, 0.182	0.598	10.0, 20.0, 20.0, 30.0	8.19, 24.43, 32.31, 55.43
llama3.1_mistral-nemo_llama3.1	0.462, 1.0, 0.462, 0.0	0.481	10.0, 17.5, 15.0, 25.0	6.45, 20.68, 22.83, 46.84
llama3.1_mistral-nemo_mistral-nemo	0.462, 1.0, 0.889, 0.462	**0.703**	12.5, 15.0, 20.0, 22.5	8.2, 15.68, 30.72, 43.06
mistral-nemo_gemma2_gemma2	0.0, 0.889, 0.333, 0.0	0.306	15.0, 22.5, 15.0, 32.0	9.78, 24.84, 33.43, 51.09
mistral-nemo_gemma2_llama3.1	0.5, 0.947, 0.333, 0.333	0.529	15.0, 20.0, 30.0, 32.0	6.81, 22.83, 60.43, 63.35
mistral-nemo_gemma2_mistral-nemo	0.333, 0.947, 0.0, 0.462	0.436	15.0, 15.0, 27.5, 32.0	9.01, 18.05, 55.06, 64.05
mistral-nemo_llama3.1_gemma2	0.0, 0.947, 0.462, 0.182	0.398	20.0, 15.0, 26.0, 15.0	17.4, 21.93, 52.54, 36.02
mistral-nemo_llama3.1_llama3.1	0.0, 0.824, 0.889, 0.462	0.543	25.0, 15.0, 20.0, 17.5	29.41, 23.74, 31.04, 36.74
mistral-nemo_llama3.1_mistral-nemo	0.0, 0.889, 0.75, 0.571	0.553	15.0, 25.0, 20.0, 22.5	19.07, 35.5, 31.42, 42.66
mistral-nemo_mistral-nemo_gemma2	0.571, 1.0, 0.667, 0.571	0.702	15.0, 15.0, 22.5, 25.0	14.17, 25.99, 47.35, 51.31
mistral-nemo_mistral-nemo_llama3.1	0.333, 0.875, 0.75, 0.571	0.632	5.0, 20.0, 15.0, 22.5	2.92, 35.52, 29.94, 52.98
mistral-nemo_mistral-nemo_mistral-nemo	0.333, 0.947, 0.333, 0.0	0.404	10.0, 15.0, 17.5, 17.5	11.07, 28.58, 39.05, 39.05

C. Conclusion

We cannot accept our hypothesis H1, as one-model MALLMs tend to perform better on specific websites. This finding does not imply that we have identified a universally optimal LLM model for this use case, nor does it suggest that such a model exists. Rather, our experiments indicate that the effectiveness of a single LLM is largely dependent on the unique structure and demands of each individual website. Conversely, we accept hypothesis H2: mixed-model MALLMs show stronger performance across multiple websites, supporting adaptability in broader applications. While this does not point to a specific ideal model combination, the results suggest that, in cases where the optimal model is unknown, using a mix of LLMs generally enhances performance across diverse, generic websites. Our finding underscores how increased website complexity, reflected in the number and intricacy of web elements, impacts efficiency by requiring agents to perform more steps and spend more time analyzing elements to achieve task completion. Finally, our study indicates that mixed-model LLM constellations are more efficacious when considering both effectiveness and efficiency across diverse websites. For instance, the constellations *lama3.1_mistral-nemo_mistral-nemo* provides a balance of high f1-scores and operational efficiency, demonstrating adaptability to different web environments without adding significant additional steps or execution time.

V. ANALYSIS

In this section, hypotheses are presented to explain the results presented in Section IV.

One model to rule them all hypothesis. The core hypothesis for this study was that a combination of models would yield better results, but that the quality of the model would also matter, i.e. a model with high scores on comparative benchmarks for different tasks, e.g. summarization or decision-making, could outperform any combination of lesser models. This would imply that a single model should consistently outperform any other constellation for all websites. However, this assumption was not supported by the experimental results, since the top performing constellation varied across the web pages. Of course, for this study, the models were comparative in terms of size and parameters, implying that using one model could still be the best option if using models are GPT-4o, LLama-3 400B, etc.

Mutual unintelligibility hypothesis. In our investigation of the demo1 site, we noticed distinct linguistic patterns in the models outputs. All the models were using imperative tone for describing web elements, and the phrases conveyed a clear message. We notice a seemingly slight variation in wording usage and their length, and that each LLM tends to repeat specific phrases, indicating patterns in their language generation. Additionally, llama was sometimes generating additional text like: *'This button is required'*, while other models were using brackets to indicate necessity for the action, i.e., *(required)*. Such differences lead us to the hypothesis that since the

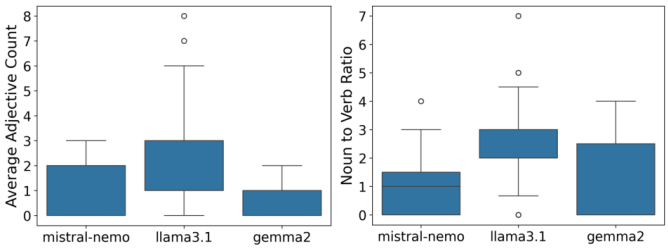

Fig. 3. Comparison of Noun-to-Adjective Ratios and Average Adjective Count Across Models.

generated text differs, the models communicate better with models of their own type, i.e., share a common linguistic pattern as an emergent property of their training. Whilst the casual relationship of this hypothesis cannot be determined from the experimental data, there is correlation to support it. Further support for the hypothesis is given by research into prompt engineering, which stipulates the importance of using correct wording in prompts [32], [33].

To support this analysis with quantitative evidence, we focused on the scene descriptions for the demo1 login page at step 1, with the same data consistent across models. For each model, we extracted various statistical and syntactic features to determine whether correlations existed between the model choice and the basic text characteristics. The extracted features included: `avg_length`: Average text length, `vb_count`: Number of verbs in the analyzed text, `nn_count`: Number of nouns, `prp_count`: Use of pronouns (e.g., he, she, it), `noun_to_verb_ratio`: Ratio of nouns to verbs, offering insight into the balance between actions and entities, `avg_adjective_count`: Average number of adjectives per sentence, reflecting the descriptive richness of the text. Additionally, we identified the `most_common_phrases` for each model to observe stylistic tendencies. One notable observation, aside from the variation in text length, was the difference in noun-to-adjective ratios, which is visualized in Figure 3. To determine whether the generated texts exhibit distinguishable characteristics based on the LLM model, we used the aforementioned features as inputs to train a decision tree classifier (RandomForestClassifier). The model aimed to predict which LLM produced the text, with the hypothesis that distinctive linguistic properties exist between generated text. The classifier achieved over 94% accuracy with the most important features being `avg_length` (0.27), `noun_to_verb_ratio` (0.22), and `avg_adjective_count` (0.21), indicating that these aspects play a critical role in distinguishing the LLM models, and confirming that different LLMs produce text with identifiable differences. Finally, we analyzed the usage of most common 2-3 word phrases. Distinct patterns emerged, allowing clear distinctions to be made between the models. Gemma2 emphasized "login page" $(2,544)$ and "inventory page" $(1,740)$, focusing on authentication and product listings, while Llama3.1 leaned heavily on "online purchase" $(1,552)$, highlighting its e-commerce focus. Mistral-Nemo showed a balanced distribution with "login page" $(1,420)$, "inventory page" $(1,089)$, and "online purchase" $(1,540)$, reflecting versatility across contexts.

No prompt fits them all hypothesis. A single prompting strategy is unlikely to work equally well across all models. Since PathFinder was primarily developed using Gemma2 and Mistral-Nemo for Perceptron, there may be prompt-related biases that make certain prompts more effective for specific models. For example, with the `SCENE_IDENTIFICATION` request on Llama3.1, the model consistently repeated the instruction, *"Determine the current high-level scene of the web page"* instead of producing the expected output.

This could have influenced the decision-making process in later stages, partly explaining the lack of good results for Llama on demo1. This observation indicates that the prompt, which works for other agents, may not be ideal for Llama. Thus, giving credence to our hypothesis of the impact of linguistic differences (e.g. wording) of prompts for different LLMs, regardless of type of agent (i.e. human or LLM).

The Hammer-Nail Hypothesis. This hypothesis reflects the importance of the underlying context (SUT) in decision-making of tested LLMs. In summary of our observation of decision-making, demo1 and demo2 had more straightforward, fixed paths, allowing models to perform well with a focused, more linear approach. In contrast, sites like Netonnet and Ikea presented more variable paths and information content, requiring the models to be flexible and adapt their decision-making to return to the correct path when deviations occurred.

This behavior can be associated with each model's handling of information complexity. Gemma displayed signs of diminishing returns, as its decision-making seemed optimal on simpler sites but became less effective as information complexity increased, leading to redundant actions and longer steps. Llama, on the other hand, showed a complexity bias, thriving on more complex sites where its advanced summarization abilities allowed it to make more informed decisions. Mistral performed between these extremes, showing moderate adaptability and effectiveness across both simpler and more complex environments.

VI. Discussion

A. On Results

The results of this study show two interesting findings when it comes to the selection of LLMs for MALLMs for GUI test generation. Firstly, if the MALLM is intended to generate tests for a specific application, the best result is achieved when applying the same LLM to all agents. However, and secondly, if the MALLM is intended to be generally applicable to different websites, using different LLMs as different agents provides the best result. Whilst this study is delimited to generation of GUI test cases of web applications, further research is warranted to explore this phenomenon for other tasks, e.g. code generation or document summarization.

The implications of this work for industrial practice is that the constellation of LLMs need to be considered when developing a MALLM system. Depending on the MALLM's

purpose, a single LLM or a combination of LLMs, may be optimal. This observation is of interest also from a monetary perspective, since providing a solution with fewer models requires less maintenance and, potentially, less licensing costs. Furthermore, whilst the multi-LLM solution outperformed the single LLM solution in the general case, the results only showed a marginal difference (0.703 for the best multi-LLM constellation, 0.660 for Gemma2 and 0.653 for $Llama3.1$. Thus, implying that the choice of model may only be required for optimization purposes. From the efficiency perspective, two main factors play a role: parsing time, which depends on data volume, and prompt size generated from site data. Mitigating these require optimized scraping and parsing techniques, as well as a dedicated server for LLM processing.

B. Future Work

Our study highlights that while LLM configurations impact performance, it remains unclear which specific models or configurations work best for particular applications. This insight leads to two key directions for future research: evaluation of other MALLM architectures and testing a broader range of LLMs. Additionally, exploration of the hypothesis that one model's responses, when used as prompts, can enhance system effectiveness. Although this study focuses on responses from LLMs, fine-tuning for specific domains, like web data and user interactions, may improve results. Integrating approaches such as Retrieval-Augmented Generation (RAG) could further enhance decision-making, especially with human-validated data. Future directions also include testing multi-modal LLMs with visual capabilities to refine scene summarization and understanding. For efficiency, dedicating servers to LLM processing and working directly with webpage source data, rather than scraping, may offer faster and more reliable outcomes.

C. Threats to validity

In this section we discuss the possible threats to validity divided into the four categories proposed by Wohlin et al. [31].

Internal validity. Concerns the study's ability to answer the research question. Although our experiment failed to accept hypothesis H1, and can thereby be classified as a failed experiment, the study provides interesting insights that allow us to answer the study's research question. From a technical perspective we identify the following aspects. *Scraping/Parsing:* Web scraping and parsing can fail due to dynamic content, malformed HTML, or complex JavaScript, hindering data extraction. *Ollama Model Switching:* Potential issues like incorrect model invocation or API failures in Ollama's API could be a potential source of disrupting agent coordination. *Workflow Errors:* Subtle bugs or workflow inconsistencies are difficult to track due to the system's complexity. Continuous monitoring and robust error handling are necessary to minimize disruptions. *Framework Architecture:* The architecture could be improved by fine-tuning models or using approaches like RAG. Still, such enhancements would introduce new dependencies and variables, making it harder to isolate the effects of LLM cooperation.

External validity. Concerns the generalizability of the study's results to other contexts and domains. Firstly, only one, high-level, but carefully chosen, task was given to PathFinder, which delimits the result's representativeness. However, given the task's complexity, in terms of number and amount of dependent steps, we perceive the results to be representative for many advanced user scenarios.

While similar offline LLMs of comparable size and capability, such as *Qwen* or *Phi*, could also have been selected for the experiment, we believe that our findings are generalizable to other models with similar capabilities, given the diversity in the benchmark scores of the chosen LLMs. However, this generalization may not hold with significantly more powerful LLMs. Therefore, replicating this study using state-of-the-art, high-performance models, such as GPT-4o or GPT-o1, is warranted. Thirdly, the sample of websites is limited but cover both small and complex, real-world, applications.

Construct validity. Concerns the study designs ability to capture realistic data in relation to a real-world context. The study made use of models and websites used in industrial practice, instrumented by a common task.

Reliability. Concerns the study designs transparency and replicability. For this purpose, all the necessary steps of the experiment have been presented, complemented with a rigorous replication package [25].

VII. Conclusions

This study investigates the application of our Multi-Agent LLM (MALLM) framework, PathFinder, for automating exploratory GUI testing of web-based e-commerce systems. By deploying three models (Mistral-Nemo, Gemma2, and Llama 3.1) across four e-commerce websites, we explored 27 LLM configurations to determine their efficacy within a multi-agent architecture. The findings suggest two primary applications:

Industrial Application: Using a single LLM across all agents yields stronger performance for tests on a specific website. This result aligns well with industrial settings where the testing domain is fixed or known. Although our results do not indicate which specific LLM to use, they suggest that simpler models may be preferable for less complex sites, enabling a balance between efficacy and computational demands. However, for complex website, more capable models may still be needed.

Research Application: When testing across a variety of unknown or complex sites, mixed-LLM configurations perform better, validating the hypothesis that different models bring complementary strengths, adapting well to diverse website structures. This versatility makes mixed-LLM setups advantageous for exploratory testing. Despite the modest performance gap between single and mixed LLM configurations, a practical approach is to begin with a single LLM and fine-tune as needed for optimal performance.

Acknowledgment

This project was financed by Vinnova (T.A.R.G.E.T. (2024-00242)), Kunskapsstiftelsen (KKS) (SERT research profile (2018-01-22)), and our research partner Synteda.

References

[1] V. Garousi and M. V. Mäntylä, "A systematic literature review of literature reviews in software testing," *Information and Software Technology*, vol. 80, pp. 195–216, 2016.

[2] D. Amalfitano, S. Faralli, J. C. R. Hauck, S. Matalonga, and D. Distante, "Artificial intelligence applied to software testing: A tertiary study," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–38, 2023.

[3] A. Bauer, J. Frattini, and E. Alégroth, "Augmented testing to support manual gui-based regression testing: An empirical study," *Empirical Software Engineering*, vol. 29, no. 6, p. 140, 2024.

[4] M. Nass, E. Alégroth, and R. Feldt, "Why many challenges with gui test automation (will) remain," *Information and Software Technology*, vol. 138, p. 106625, 2021.

[5] J. Yoon, R. Feldt, and S. Yoo, "Intent-driven mobile gui testing with autonomous large language model agents," in *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2024, pp. 129–139.

[6] Z. Liu, C. Chen, J. Wang, M. Chen, B. Wu, X. Che, D. Wang, and Q. Wang, "Make llm a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[7] M. Pavlovic and M. Poesio, "The effectiveness of llms as annotators: A comparative overview and empirical analysis of direct representation," *arXiv preprint arXiv:2405.01299*, 2024.

[8] C. Shah, "From prompt engineering to prompt science with human in the loop," *arXiv preprint arXiv:2401.04122*, 2024.

[9] K. Wu, E. Wu, and J. Zou, "How faithful are rag models? quantifying the tug-of-war between rag and llms' internal prior," *arXiv preprint arXiv:2404.10198*, 2024.

[10] T. Ahmed and P. Devanbu, "Few-shot training llms for project-specific code-summarization," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5.

[11] M. A. Sami, M. Waseem, Z. Rasheed, M. Saari, K. Systä, and P. Abrahamsson, "Experimenting with multi-agent software development: Towards a unified platform," *arXiv preprint arXiv:2406.05381*, 2024.

[12] R. Kirk, I. Mediratta, C. Nalmpantis, J. Luketina, E. Hambro, E. Grefenstette, and R. Raileanu, "Understanding the effects of rlhf on llm generalisation and diversity," *arXiv preprint arXiv:2310.06452*, 2023.

[13] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar *et al.*, "Holistic evaluation of language models," *arXiv preprint arXiv:2211.09110*, 2022.

[14] I. de Zarzà, J. de Curtò, G. Roig, P. Manzoni, and C. T. Calafate, "Emergent cooperation and strategy adaptation in multi-agent systems: An extended coevolutionary theory with llms," *Electronics*, vol. 12, no. 12, p. 2722, 2023.

[15] T. Hu and X.-H. Zhou, "Unveiling llm evaluation focused on metrics: Challenges and solutions," *arXiv preprint arXiv:2404.09135*, 2024.

[16] Z. Liu, Y. Zhang, P. Li, Y. Liu, and D. Yang, "A dynamic llm-powered agent network for task-oriented agent collaboration," in *First Conference on Language Modeling*, 2024.

[17] J. Xie, K. Zhang, J. Chen, T. Zhu, R. Lou, Y. Tian, Y. Xiao, and Y. Su, "Travelplanner: A benchmark for real-world planning with language agents," *arXiv preprint arXiv:2402.01622*, 2024.

[18] Y. Wu, X. Tang, T. M. Mitchell, and Y. Li, "Smartplay: A benchmark for llms as intelligent agents," *arXiv preprint arXiv:2310.01557*, 2023.

[19] M. Shao, B. Chen, S. Jancheska, B. Dolan-Gavitt, S. Garg, R. Karri, and M. Shafique, "An empirical evaluation of llms for solving offensive security challenges," *arXiv preprint arXiv:2402.11814*, 2024.

[20] Z. Yuan, W. Chen, H. Wang, K. Yu, X. Peng, and Y. Lou, "Transagent: An llm-based multi-agent system for code translation," *arXiv preprint arXiv:2409.19894*, 2024.

[21] Q. Wang, Z. Wang, Y. Su, H. Tong, and Y. Song, "Rethinking the bounds of llm reasoning: Are multi-agent discussions the key?" *arXiv preprint arXiv:2402.18272*, 2024.

[22] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, 2023.

[23] H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, "From llms to llm-based agents for software engineering: A survey of current, challenges and future," *arXiv preprint arXiv:2408.02479*, 2024.

[24] P. S. Foundation, *Python Standard Library Documentation*, 2024, accessed: 2024-10-30. [Online]. Available: https://docs.python.org/3/library/difflib.html

[25] S. Tomic, "Pathfinder software package," 2024, data collected in September. Available at: https://symatiq.com/resources/pack_pathfinder/software_pack_pathfinder.zip.

[26] G. Bao, H. Zhang, L. Yang, C. Wang, and Y. Zhang, "Llms with chain-of-thought are non-causal reasoners," *arXiv preprint arXiv:2402.16048*, 2024.

[27] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé *et al.*, "Gemma 2: Improving open language models at a practical size," *arXiv preprint arXiv:2408.00118*, 2024.

[28] M. AI, "Llama 3.1: Large language model meta ai," 2024, accessed: 2024-10-14. [Online]. Available: https://ai.meta.com/blog/meta-llama-3-1/

[29] M. A. Team, "Mistral-nemo model," 2024, accessed: 2024-10-14. [Online]. Available: https://mistral.ai/news/mistral-nemo/

[30] O. Inc., "Ollama: Lightweight language models for local inference," 2024. [Online]. Available: https://ollama.com

[31] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén *et al.*, *Experimentation in software engineering*. Springer, 2012, vol. 236.

[32] S. Arvidsson and J. Axell, "Prompt engineering guidelines for llms in requirements engineering," *Gothenburg University*, 2023.

[33] Z. J. Wang, A. Chakravarthy, D. Munechika, and D. H. Chau, "Wordflow: Social prompt engineering for large language models," *arXiv preprint arXiv:2401.14447*, 2024.