Automated Generation of Accessibility Test Reports from Recorded User Transcripts

Syed Fatiul Huq
University of California, Irvine
Irvine, California, USA
fsyedhuq@uci.edu

Mahan Tafreshipour
University of California, Irvine
Irvine, California, USA
mtafresh@uci.edu

Kate Kalcevich
Fable Tech Labs Inc.
Toronto, Ontario, Canada
kate@makeitfable.com

Sam Malek
University of California, Irvine
Irvine, California, USA
malek@uci.edu

***Abstract*—Testing for accessibility is a significant step when developing software, as it ensures that all users, including those with disabilities, can effectively engage with web and mobile applications. While automated tools exist to detect accessibility issues in software, none are as comprehensive and effective as the process of user testing, where testers with various disabilities evaluate the application for accessibility and usability issues. However, user testing is not popular with software developers as it requires conducting lengthy interviews with users and later parsing through large recordings to derive the issues to fix. In this paper, we explore how large language models (LLMs) like GPT 4.0, which have shown promising results in context comprehension and semantic text generation, can mitigate this issue and streamline the user testing process. Our solution, called Reca11, takes in auto-generated transcripts from user testing video recordings and extracts the accessibility and usability issues mentioned by the tester. Our systematic prompt engineering determines the optimal configuration of input, instruction, context and demonstrations for best results. We evaluate Reca11's effectiveness on 36 user testing sessions across three applications. Based on the findings, we investigate the strengths and weaknesses of using LLMs in this space.**

***Index Terms*—software accessibility, large language models, crowd-sourced software testing**

I. Introduction

The World Health Organization has reported in 2023 that 16% of the world population, around 1.3 billion people, live with some form of disability [1]. However, recent reports [2]–[5] have indicated that software, both web and mobile, are predominantly inaccessible, containing issues that prove to be nuisances and roadblocks for users with different disabilities. For instance, non-textual UI elements like images that do not have alternative text are inaccessible to visually impaired users who rely on screen readers to announce elements. For users with a motor disability, UI elements that cannot be reached with keyboard navigation means they cannot access those information or features. In a modern world that heavily relies on online applications for everyday tasks, from work and living to health and entertainment, it is imperative to develop applications accessible to all.

With the advent of standardized guidelines [6], government policies [7], and developer activism [8], the initiative to develop accessible software has bolstered in recent times. There are three ways of testing software for accessibility. First, automated testing, where tools can conduct static analysis on web or mobile applications and check for guideline violations [3], [9]–[13], or dynamically assess whether assistive technologies (ATs) like screen readers, switch systems and more are functioning properly on the deployed app [14]–[19]. The second option is manual testing, where developers and in-house testers manually test the app with ATs to check for accessibility issues. While both of these are typically manageable by a development team and streamline the accessibility testing process, they come with their limitations, especially compared to the third option: user testing.

Accessibility user testing [20], [21] involves end users with disabilities evaluating the accessibility of an application using their preferred AT. User testing accommodates the insight from people with varying disabilities, varying states from the same disability, and varying ways they use ATs and experience software. In automated and manual tests, these nuances are usually overlooked. Automated tools, that primarily depend on static rules, have been shown to report 20-40% fewer issues than evaluations with actual disabled people [22]. Manual testing by developers relies on their understanding of the various forms of disability, which is reported to be severely lacking in several prior studies [2], [23]–[28].

Despite the advantages of user testing, it is not widely adopted in the software industry for multiple practical reasons, ranging from recruitment challenges [29] to the cost of human and technical resources [22]. One such challenge comes from the test report generation and validation processes [30]. These processes involve revisiting test sessions through the recordings, deriving and compiling the barriers found by the tester, and validating the derived issues with the software team. Introducing LLMs into report generation can streamline these processes and make it easier for software teams to employ accessibility user testing.

In this paper, we automate report generation from accessibility user tests by analyzing test transcripts through a large language model (LLM) and producing a list of accessibility issues reported. LLMs have proved their aptitude in analyzing and synthesizing natural language, and their usefulness in fields ranging from software development [31] to healthcare [32]. Our study poses three research questions (RQs) and aims to understand how well LLMs can automate the test report generation process. We explore their performance with different prompts (RQ1), assess their effectiveness on real world user tests (RQ2), and analyze where their strengths and weaknesses lie (RQ3).

To run our experiments, we conduct user tests run on three

different mobile and web applications, executed by users with different disabilities — full blindness, partial blindness, and motor disabilities — using different ATs. We use a total of 36 test sessions that range from 10 minutes to 70 minutes. For RQ1, we take a sample of 7 sessions and feed them to the LLM using different prompts, configuring the instructions, contextual information and input transcript. Using the few shot learning approach, we also provide the prompt with input-output demonstrations. As our LLM, we use GPT-4.0, the state-of-the-art model. From our prompt engineering, we find that adding contextual information and demonstrations inform the model how to describe issues. But the biggest factor in determining the model's effectiveness is the input size. We see that feeding the model smaller chunks of the original transcript helps identify more issues.

Based on these insights, for RQ2, we run the final tool on all 36 transcripts. The tool generates a near perfect recall of 98%, attributed to 6 missed issues from a total of 319. The high recall indicates the tool's effectiveness in detecting all possible accessibility barriers reported by users. On the other hand, with a precision of 85%, the tool identifies some issues that are not accurate. For RQ3, we manually inspect these inaccurate issues to understand LLM's shortcomings. We observe that LLM is susceptible to language from contextual information, repeated announcements related to assistive technologies and ambiguity in speech. With these insights, we discuss how best to employ LLMs for test report generation, and how to potentially improve its performance with specific configurations and contextual information.

In summary, this paper makes the following contributions,

- An automated tool, Reca11[1], that takes in accessibility user test recording transcripts and generates a test report.
- A technical analysis of prompt engineering for automated test report generation using LLMs.
- A demonstration and qualitative analysis of LLM's performance in detecting software accessibility issues.
- An assessment of LLM's application in the accessibility user testing process.

The remainder of this paper is structured as follows: Section II provides an overview of accessibility user testing, test report generation and large language models. Section III describes the study design to answer our research questions. Section IV reports the results, the implications therein are delved into in Section V. The paper concludes with Section VII.

II. Background and Related Work

A. The Accessibility User Testing Process

Accessibility user testing is the process of testing the accessibility and usability of a web or mobile application by having a user with disability interact with it. Prior studies have looked into accessibility user tests from an empirical perspective. Aizpurua et al. [21] evaluated the accuracy of user tests, and found that disabled users' experience of a software can differ vastly from the expected interactions with categorical accessibility issues. Brajnik et al. [33] explored how expertise on accessibility can impact user tests, and found that non-experts derive significantly less number of issues than experts. Mateus et al. [34] also looked into the effect of expertise and observed that HCI specialists tend to outperform software developers on reporting accessibility issues. Brajnik et al. [35] also found that collaborative testing, where testers pair up to assess the accessibility of a page, can report more accurately [35]. In an effort to mitigate the dependence on expertise, Song et al. [36] introduce truth inference techniques in crowd-sourced accessibility evaluations.

To understand the contemporary practice of software accessibility testing, we collaborate with Fable [37], an accessibility user testing platform that connects software stakeholders to testers with disabilities. We interview one of their analysts and personally use two of the engagements they provide through their Engage product: User Interviews and Self-Guided Tasks. In these engagements, a disabled tester is provided with an application to test for accessibility and usability. The core difference between Interviews and Self-Guided Tasks is that the former accommodates one or more interviewers in the testing session while in the latter, the tester goes through a task by themselves and provides feedback based on their experience. In Fig. 1, we illustrate the process of these services.

The process begins with a software company or team opening a new request to test their product. These teams can consist of product managers, developers, user researchers, or UI and UX designers. For this research study, an analyst is assigned to the project, who acts as an extension to the team, consulting on accessibility. She helps the team realize the specific test goals for their application or feature, and create a list of tasks. These tasks are what the testers are prompted to complete during the testing session.

A set of testers is selected based on the technical expertise required. Expertise ranges from the assistive technologies used (e.g., screen readers, alternative navigation, screen magnifiers and more) to the platform that hosts the application (e.g., browser, desktop or smartphone).

During a user test session, the tester navigates the application to complete the assigned tasks. The tester maintains the Think Aloud protocol [38], vocalizing their thought process as they make decisions to navigate between pages or enter information, and encounter challenges. Thinking aloud is an important step as it helps teams understand how the tester encounters a problem and why they deem it as an issue.

In user interviews, the analyst can guide the tester during navigation, answer the tester's queries and probe for further explanations. During the session, the analyst takes notes, which she would refer to later for compiling a report. These two steps, depicted with dashed borders in Fig. 1, are exclusive to user interviews and skipped in self-guided tasks.

Once the recording of the test session is published, an analyst goes through it and compiles a test report. The report contains accessibility and usability issues encountered by the tester. The final report is submitted to the software team with suggestions for mitigation.

We observe a similar process in the domain of crowd-

[1]The name Reca11 is inspired by the term "a11y", a popular shorthand for accessibility, and our tool's goal to recall the issues from the transcript.

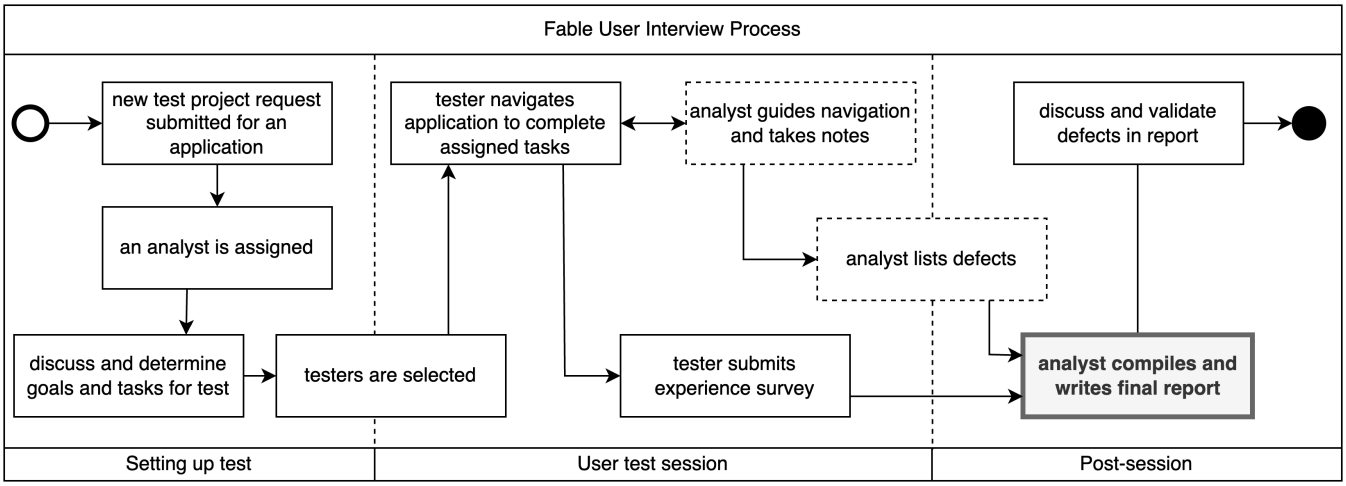

Fig. 1. Software testing process for this study using Fable

sourced software testing (CST) [39], where non-experts evaluate usability, functionality and compatibility of software systems. Instead of an analyst, a project manager coordinates with the testers, and compiles and validates the reported issues. Similar to Self-Guided Tasks, no interviewers are present in these testing sessions.

B. Test Report Generation

Our work focuses on the report generation phase, as highlighted in Fig. 1. A report ideally contains all the issues of the application, where in the app they occur, quotes from the recording transcript for context and corresponding mitigation strategies. It is a manual process with two categories of challenges centered around effort and expertise.

Significant manual effort has to be expended to parse through test recordings and derive issues. A recording can span from 10 minutes to more than an hour, each possibly requiring multiple revisions for better synthesis of information. This effort is multiplied by the number of sessions required for a single project and the variation of assistive technologies, based on target demographics of disability. According to our interviewed analyst, after parsing through the recordings, it can take around 20 minutes to synthesize and write the report, even with notes taken during the interview. To minimize the manual effort, multiple research work have looked into different forms of automation, generating reports from recurrent crash patterns [40], screenshots in test reports [41], relevant metadata from duplicate reports [42], and summarizing existing reports [43]. However, none of these studies regarded accessibility testing or generated reports directly from recordings.

Other than effort, expertise is a necessary component in the effective interpretation of test recordings, to understand the cause and category of tester complaints. Report writing is easier with knowledge of assistive technologies, which is no small feat given the use of a variety of techs for different disabilities on different platforms. We can take screen readers as an example, which are used by people with partial or full blindness. There are four variants of screen readers which are popular: JAWS [44] and NVDA [45] for Windows, VoiceOver [46] for Apple devices and TalkBack [47] for Android phones, each with their distinct characteristics and configurations. Expertise with guidelines like WCAG [6], used as an internationally accepted standard for accessibility is also beneficial. WCAG 2.2, the latest iteration, comes with 13 guidelines under four principles, containing a total of 78 success criteria that elaborates on how to meet the guidelines. Knowing the guidelines helps in explaining the cause of a reported issue and possible ways to fix it.

Transcript

Tester: Not sure. Why, talkbys having trouble entering things in today?
Tester: Or maybe it's me.
Tester: I think 250 got a box budget. Alright, perfect. Alright your text button.
Tester: your next one.
Tester: I didn't like that.
Tester: And this add income button. It'd be helpful. If it said.
add category, button, double tap.
Tester: Keep working
Tester: money, manager, detective icon, add button annotation, button windows, share audio, get it detected alright. We'll do one more
Tester: budget category info. Good morning. Anybody you ask corn, Shawn.

Actual Speech

<tester is filling up form to add "budget category">
Not sure why Talkback is having trouble entering things in today. Or maybe it's me.
<fills up form>
Alright, perfect.
<navigates to submit the form, finds mislabeled button that said "add income">
I didn't like that, this "add income" button. It'd be helpful if it said "add category" button.
<presses the button>
Alright, we'll do one more.

Fig. 2. An excerpt from a transcript to demonstrate noise in the text

LLM, therefore, provides an obvious solution to these challenges. It holds the promise of reducing the manual effort by automatically parsing the test sessions and generating informative test reports for developers. Taeb et al. [48] observed that LLMs can effectively interpret accessibility test

instructions written in natural language. However, automatic report generation presents its own challenges. The transcripts lack systematic formatting and contains a substantial amount of noise for instance, multiple speakers and audio interference from assistive technologies. As illustrated in Fig. 2, not only are the actual statements from the tester inaccurately punctuated and sometimes misspelled, they are interrupted by screen reader announcements. Furthermore, testers may not know what specifically has gone wrong in an interaction and cannot precisely articulate the details of the issue they face, and instead verbalize their frustration caused by the barrier.

C. Large Language Models

With the advent of ChatGPT [49], Large Language Models (LLMs) are becoming an integral part of natural language processing both in the public domain and research space [50]. These models, trained on large-scale corpora and tuned with billions of parameters, are able to generate human-equivalent text. To work with LLMs, the key is to prompt them with the appropriate instructions and contexts that aid them with generating the expected output.

For prompting, LLMs contain four key components. *Instruction* is the task we want the LLM to conduct. Instructions can specify the persona or perspective from which the LLM should view the task, and the format of the output. *Input* is the resource on which the LLM has to conduct the specified task. *Context* is any information that can help the LLM narrow its scope and enhance its lexicon on the specific knowledge space. LLMs have shown to perform better if the prompt lists out *demonstrations* [51]. Termed as few-shot or in-context learning, this process adds input-output pairs to the prompt, usually in the form of $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where x_i is an example input and y_i is the associated output.

III. Study Design

The primary goal of our work is to assess the applicability of LLMs in the space of accessibility user testing, specifically to automate the report generation process. We build an automated tool, Reca11, as illustrated in Fig. 3. The tool is deployed once the tester conducts her assigned test. Using the transcript from the test recording and other contextual information related to the session, our tool prompts an LLM to generate the test report. The resulting report is a list of accessibility and usability issues mentioned by the disabled tester.

We aim to understand how to prompt an LLM to generate the best results and whether those results are effective enough to automate the report generation phase of accessibility user testing. We pose three research questions.

RQ1. Prompt engineering: How does configuring different prompts affect performance? The performance of LLMs is dictated by the prompt fed to it. We determine the best performing prompt by configuring the input and instructions.

RQ2. Statistical effectiveness: How effective is LLM in producing accurate accessibility issues? Based on the optimal prompt, we statistically assess how accurate and consistent the model is in deriving the issues reported.

RQ3. Strengths and weaknesses: Where does LLM perform consistently and where does it underperform, and why? We manually assess the results to understand the factors in the transcript that determine the LLM's performance.

A. Datasets

Our primary data point is the transcript automatically generated from a video recording of a user test session. A test is done on a single application, on all or some of its features. For our paper, we collect test session transcripts for three different apps, accumulating a total of 36 transcripts. These tests are conducted by testers with disabilities, through Fable's user testing services. The tests are recorded, generating a video recording along with a transcript file. The transcript file is in a *.vtt* format, consisting of a list of "captions", where each caption contains a start and end timestamp, speaker name, and the text being spoken.

1) Money Manager (D_{mm}): Money Manager [52] is an open source Android application with features to add and monitor personal budgetary information. We choose this finance application because of its familiar use case, uncomplicated design and our ability to modify the code. We modify different features and UI elements of the app to inject accessibility issues, as categorized by previous work [2], [15], [19].

We conduct two preliminary tests with screen reader users to assess the effectiveness of injected issues. For the final dataset, we use 5 user interviews and 5 self-guided tasks, with blind testers who use Talkback on Android.

2) Record a Goose Sighting (D_{gs}): Record a Goose Sighting [53] is an educational website used for training people on accessibility testing. Similar to D_{mm}, this website has been injected with accessibility issues [54] by its creators, who are not the authors of this paper. For this dataset, we conduct a total of 10 self-guided tasks, covering every assistive technologies (ATs) offered in Fable, under three AT types:

- Screen readers (for blind users): NVDA [45], JAWS [44], VoiceOver [46], and TalkBack [47]
- Alternative navigation (for users with limited mobility): Dragon NaturallySpeaking [55], On-screen keyboard, Voice Control [56], and Switch system [57]
- Screen magnifiers (for users with low vision): OS Magnification and ZoomText [58]

3) Fable (D_{fb}): D_{fb} contains archival test sessions from Fable where testers are assigned to test different features of Fable's digital products. There are a total of 16 sessions, with 14 self-guided tasks and 2 user interviews. The testers' demographic of these sessions includes 6 blind users, 4 users with low vision and 6 users with limited mobility.

B. Contextual Information

As mentioned in Section II-C, contextual information enhances GPT's performance. Since the knowledge space our prompt is trying to target is software accessibility, we incorporate two relevant guidelines as context:

- WCAG: The Web Content Accessibility Guidelines (WCAG) [6] has been developed to establish an international standard for software accessibility on web and

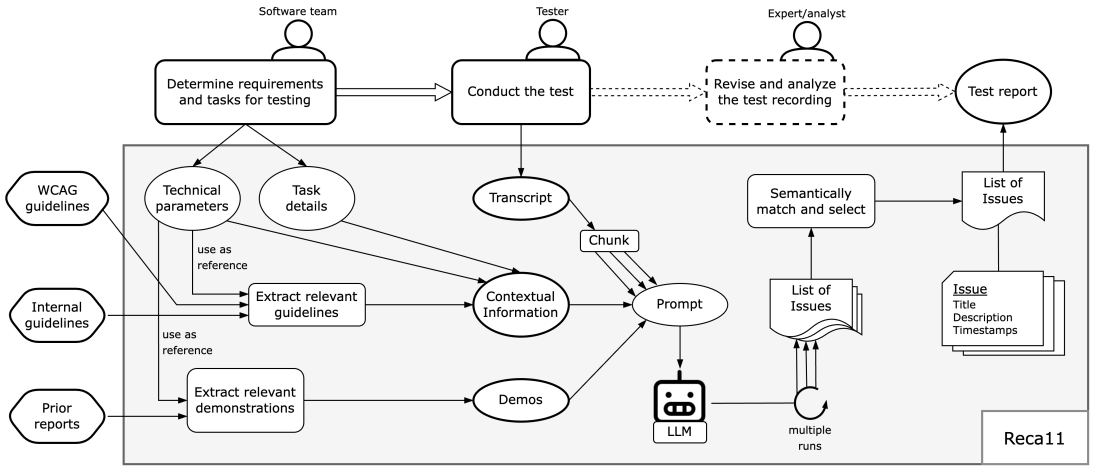

Fig. 3. Overview of Reca11, our automated tool for generating test reports from accessibility user tests

mobile. We insert the website for WCAG guidelines to our prompt as context. It contains 13 guidelines with multiple success criteria that specify how to test whether a guideline is satisfied. For instance, the guideline "2.1 Keyboard Accessible" is related to the operability of web content with keyboard alone, containing three success criteria detailing different ways to test it. Including these guidelines as context can help the LLM map tester statements to guidelines and success criteria, and determine whether it can constitute an accessibility issue.

- Internal Guidelines: The accessibility user testing process has overlap with crowd-sourced software testing, which depends on an intermediary platform to connect software stakeholders with testers. Platforms can include their own guidelines for analyzing test reports. Hence, we include internal guidelines as a contextual information, adopting Fable's "Research Directory". This document lists takeaways and best practices for different concerns (e.g., color contrast) and assistive technologies (e.g., screen reader), and explains how to resolve them. While WCAG caters to designers and developers about best accessible practices, this document is intended to train Fable's analysts on tester behavior and issue categorization.

While the two guidelines are used for enhancing the LLM's knowledge about software accessibility in general, we also use the following two sources of data to elaborate on the test session being analyzed.

- Task description: For a test session, the tester is prompted to follow a set of tasks. These tasks are determined by the analyst and software team. Task descriptions can include names of elements and pages on the screen that the LLM can potentially map to the information contained in the transcripts. For our study, we use task descriptions only for D_{mm} and D_{gs}, as we do not have access to the archival test requests from D_{fb}.
- Technical parameters: Test sessions also contain technology specifications like AT type (e.g., alternative navigation), the specific AT used (e.g., Dragon NaturallySpeaking), and the platform the application under test is deployed on (e.g., Desktop browser). This can further help LLM focus on relevant issues.

C. Output

The expected output from the LLM is a list of issues, making up the test report. Each issue will contain three items: issue title, description and timestamps. The first two are used to describe the reported barrier, the UI elements involved and how the tester encountered the issue. Timestamps are one or more locations in the recording where an issue was mentioned. As per our preliminary study, we know that analysts would include clips from the recording in their report, so that the software team can refer to the recording for validation or explanation. Timestamps serve the same purpose.

Because of the non-deterministic nature of LLM outputs, we run the model multiple times for each test session. The goal is to observe the varied responses LLM can produce, and select the most appropriate issues. We test with two different selection processes: 'all' and 'common'. In Fig. 4, we illustrate our selection function. Given we run the model five times on a transcript, we would generate five test reports Tr_{1-5}, where each test report can detect a different set of issues I_1 to I_n.

In 'all' and 'common', we aim to incorporate all and the most common unique issues reported, respectively. To find and list issues that are unique, we semantically match the issues between different reports. If an issue found in the current run matches an issue found in a prior run, we consider it to be a duplicate, e.g., I_7 matches I_1 in Fig. 4. All contains a list of all the unique issues. From All, $Common$ takes in the subset that occurred in all the reports, e.g., I_1 and I_5.

For semantic matching, we calculate the cosine similarity between the vector embeddings of two issues. From the issues, we concatenate the issue title and description, and encode the resulting text. For encoding, we use a sentence transformer, `all-mpnet-base-v2` [59]. Trained on 1 billion sentence pairs, this model is built for finding semantic similarity between sentences and short paragraphs, and is reportedly the best performing model [60].

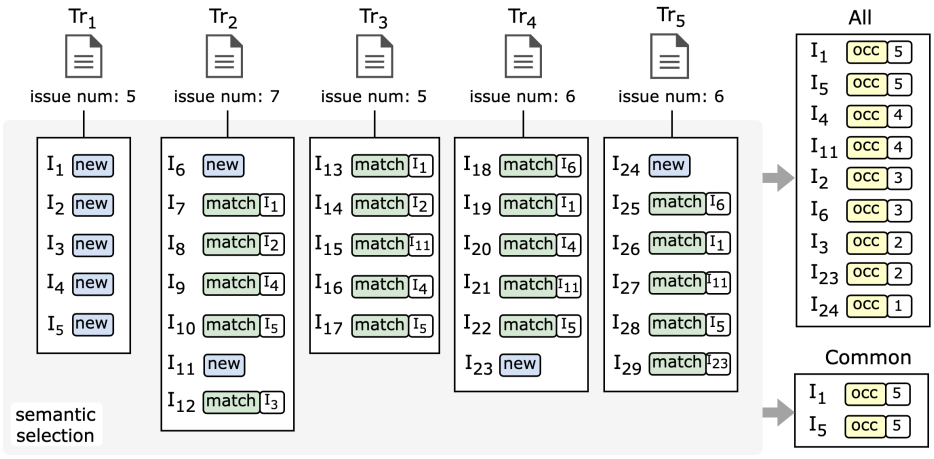

Fig. 4. Output selection process

D. Evaluation Metrics

We evaluate the accuracy of issues reported by the model by manually comparing with all the issues mentioned in the test recording. We categorize each reported issue in one of three groups: confirmed, mentioned, and non-issue.

We consider a reported issue to be 'confirmed' if the issue is included in the ground truth (GT). We create three GTs for the three datasets, as explained in Table I.

TABLE I
GROUND TRUTHS FOR EVALUATION

Label	Dataset	Source
GT_{mm}	Money Manager	The accessibility issues we injected, based on prior work [2], [15], [19].
GT_{gs}	Report a Goose Sighting	The accessibility issues injected by the creators of the website [54].
GT_{fb}	Fable	Accessibility Issues reported by Fable's analysts on D_{fb}.

However, not every issue in the GT is reported by the tester during the test, because either the tester did not realize the issue taking place or did not consider it to be an issue. Conversely, not all issues reported by the tester is in the GT. For the latter case, we include the category 'mentioned' issues. For each issue not categorized as 'confirmed', we manually check the recording, on the timestamp reported with the issue. If the tester mentioned the reported issue, then we label it as 'mentioned'. Otherwise, we label it as a 'non-issue'.

We consider 'confirmed' (I_c) and 'mentioned' (I_m) issues as True Positive, and 'non-issues' (I_n) as False Positive. If an issue is mentioned by the tester but not reported by our model, that is considered a False Negative (FN). Therefore, the formulas for precision and recall are:

$$Precision = \frac{\sum I_c + \sum I_m}{\sum I_c + \sum I_m + \sum I_n} \quad (1)$$

$$Recall = \frac{\sum I_c + \sum I_m}{\sum I_c + \sum I_m + \sum FN} \quad (2)$$

E. GPT

For our LLM, we employ GPT-4 [61], the latest of the GPT models. We use this model because it outperforms contemporary ones in natural language processing tasks [62], and its Turbo update provides a larger input length of 128K tokens, necessary for accommodating large input such as transcripts. Since our work requires deterministic output, we run the model on 0 temperature, which reduces LLM's creativity and increases, if not ensures, determinism [63], [64].

IV. RESULTS

A. RQ1: Prompt engineering

The goal of this research question is to configure the different components of the prompt — instruction, contextual information, input and demonstrations — to derive the optimal prompt that generates the most accurate results. To do so, we take a sample of 7 transcripts from our dataset of 36, and analyze the results from prompt engineering.

We semi-randomly select the sample dataset so that transcripts of different characteristics are represented. From D_{mm}, we randomly select one self-guided task and one user interview. Since D_{mm} are all screen reader tests, from D_{gs} we randomly pick two transcripts that use alternative navigation and screen magnification. Lastly, from D_{fb}, we choose three transcripts of the largest, average and smallest sizes.

For each variant, we run our tool Reca11, as illustrated in Fig. 3. As input, we provide the transcript and the corresponding contextual information when needed. As output, the tool generates the test report, a list of issues. For analysis, we manually inspect each issue, comparing it with the GT and the original recording. Labeling the issues as 'confirmed', 'mentioned' or 'non-issue', we calculate the precision and recall. We also assess the qualitative characteristic of the title and description, and the accuracy of the timestamps. We report the quantitative and qualitative findings, and compare with previous variants to choose the better prompt. Once we derive the optimal variant of a component, we employ that as the

baseline for analyzing the next component. We add prompts for the new variants onto the existing one.

1) RQ1.1 Instruction: Inspired by previous study [65], we write instructions of variant levels of detail: simple instruction, with scenario, with persona and with warning against forced results. We list the variants in Fig. 5. We add one variant at a time to check their effect on the prompt.

TABLE II
COMPARING SAMPLE RESULTS FOR DIFFERENT INSTRUCTIONS (RQ1.1)

		$\sum I_c$	$\sum I_m$	$\sum I_n$	FN
D'^1_{mm}	Simple	3	0	0	7
	With scenario	4	0	0	6
	With persona	4	0	0	6
	With warning	4	0	0	6
D'^2_{mm}	Simple	2	1	0	14
	With scenario	3	0	0	14
	With persona	3	1	0	13
	With warning	3	1	0	13

After executing the prompts with subsequent instructions on all 7 samples, we find that there is no distinguishable differences in the performance of the model. In Table II we show the results for two D'_{mm} sessions as examples, employing the *all* selection criteria. We see that the results remain nearly stagnant for all the variants. While adding scenario helped in D'^1_{mm}, it did the opposite for D'^2_{mm}.

Descriptively, the issue specifications remain similar across variants. The corresponding timestamps were mostly accurate. However, longer transcripts suffered from what we call clustered targeting. The tool extracted issues from the end of the transcript, missing issues mentioned in earlier parts.

2) RQ1.2 Contextual information: The number of False Negatives from the previous prompt indicates that the tool is failing to understand statements as issues. Therefore, we look to improve the results by adding contextual information, informing our model more about software accessibility. As described in Section III-B, we incorporate four contextual information in our prompt: WCAG guidelines, internal guidelines from Fable, task description for specific test sessions, and technological parameters the session was conducted on. Additionally, we filter the internal guidelines using the technological parameters to automatically select relevant guidelines. As shown in Fig. 5, we include some preamble text before introducing each context to better differentiate the information.

The results, however, do not demonstrate any improvements numerically. As exemplified in Table III, adding each contextual information did not significantly improve the results.

In analyzing the issues descriptively, we find that they are more verbose and use guideline-specific language. Generic wordings are replaced by more technical terms like "heading structures", "error messages", "notifications" and more. However, the reports still suffer from clustered targeting. The increased prompt size, attributed to the inclusion of contextual information, amplifies this challenge.

3) RQ1.3 Input: In analyzing the reports so far, we noticed that the reported issues originated from the last sections of the transcript, especially for longer transcripts. In general, the model's performance degraded as transcripts got larger. Therefore, we decide to create multiple chunks from a single transcript, injecting smaller sized subsets of the transcript into the prompt and combining the resulting reports.

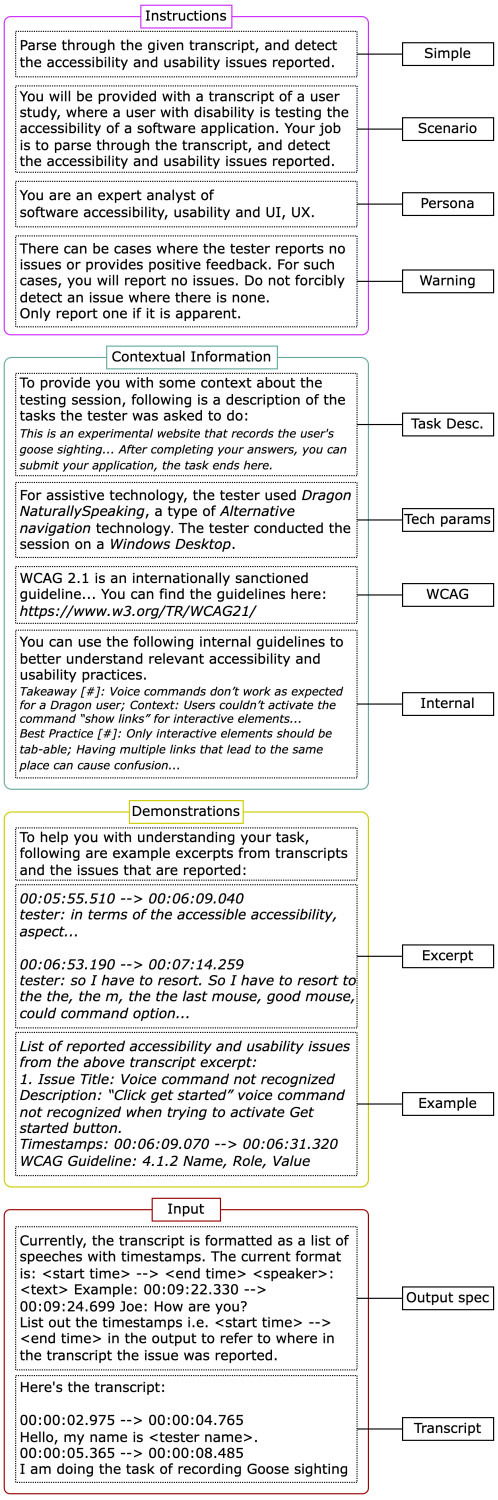

Fig. 5. An example prompt from our experiments

TABLE III
COMPARING SAMPLE RESULTS FOR DIFFERENT CONTEXTS (RQ1.2)

		$\sum I_c$	$\sum I_m$	$\sum I_n$	FN
D'^1_{mm}	No context	4	0	0	6
	Task description	4	0	0	6
	Tech params	4	0	0	6
	WCAG	4	0	0	6
	Internal guidelines	4	1	0	5
	All contexts	4	0	0	6
D'^2_{mm}	No context	4	2	0	11
	Task description	5	2	3	10
	Tech params	4	2	0	11
	WCAG	3	2	0	12
	Internal guidelines	4	1	1	12
	All contexts	5	1	0	10

We split a transcript by the number of captions. In our sample dataset, the number of captions ranged from 77 to 566. The larger transcripts, D'^1_{mm}, D'^2_{mm} and D'^3_{fb} contain 566, 359 and 503 transcripts respectively, all of which suffer from clustered targeting. We choose to divide transcripts by 250 captions, since that halves these transcripts. We also choose chunk sizes of 100 captions, as the smallest transcript, D'^1_{fb}, has 77 captions. From our experiments so far, the results for D'^1_{fb} have generated perfect scores.

During the chunking process, we include one extra caption from the previous chunk, to provide context to the current chunk and mitigate an issue being ignored. We observe from prior variants that the reported issues contain an average of 2 captions. Hence we choose one caption as our padding.

We find that the results improve significantly. We illustrate our findings in Table IV. The number of reported issues increases significantly with chunked transcripts, with 100 caption chunks resulting in zero false negatives. We observe that the number of non-issues also increased. However, we deem it an acceptable trade-off as missing issues are more critical for reports.

TABLE IV
COMPARING SAMPLE RESULTS FOR DIFFERENT INPUTS (RQ1.3)

		$\sum I_c$	$\sum I_m$	$\sum I_n$	FN
D'^1_{mm}	Full transcript	4	0	0	8
	250 captions	7	1	1	4
	100 captions	10	2	3	0
D'^2_{mm}	Full transcript	6	0	0	10
	250 captions	7	5	1	4
	100 captions	7	9	4	0
D'^3_{fb}	Full transcript	6	1	1	9
	250 captions	1	4	4	4
	100 captions	1	8	3	0

4) RQ1.4 Demonstrations: Lastly, we introduce demonstrations into the prompt as a form of in-context or one/few-shot learning [51]. As shown in Fig. 5, we structure a demonstration in two parts: first, an excerpt of a transcript and second, the example issues reported from that excerpt. To create demonstrations, we take excerpts from D_{fb}. These reports, compiled by Fable analysts, represent how issues are interpreted and written by industry experts.

We experiment with two types of demonstrations:

- Single example: Following one-shot learning, we include only one transcript excerpt and one resulting issue as a demonstration.
- Multiple examples: Following few-shot learning, we include multiple excerpts with multiple resulting issues.

In both cases, we select demonstrations so that they match the input transcript's technological parameters. From the set of demonstrations, we only choose ones with similar type of AT used. AT types represent the tester's disability (e.g., screen reader users have full or partial blindness while alternative navigation users have motor disability), and therefore can provide similar language when conveying issues faced.

TABLE V
COMPARING SAMPLE RESULTS FOR DEMONSTRATIONS (RQ1.4)

		$\sum I_c$	$\sum I_m$	$\sum I_n$	FN
D'^1_{gs}	No examples	1	7	0	0
	Single example	1	7	0	0
	Multiple examples	1	7	0	0
D'^2_{gs}	No examples	1	2	1	2
	Single example	1	3	1	1
	Multiple examples	1	3	1	1

To exemplify, we take two sessions from D'_{gs}: the first uses screen magnification and second alternative navigation. Table V shows that the results do not indicate any significant improvement from using no examples. Specifically, in D'^1_{gs}, all the variants performed similarly. For D'^1_{gs}, the examples detected an extra issue each. Qualitatively, the reported issues show clear influence in terms of verbosity. The demonstration issues were very on point and concise, compared to the model's descriptions. However, in terms of accuracy, descriptions for all three performed similarly.

B. RQ2: Statistical effectiveness

Based on the results of RQ1, we decide to use the following configuration for our tool's final prompt:

- Instruction: We include a detailed instruction with scenario, persona, and a warning to mitigate false positives.
- Contextual information: We include all the sources for contextual information — task description, technological parameters, WCAG and internal guidelines. As mentioned in Section III-B, we do not include task description for D_{fb} in the absence of validated information.
- Input: We use transcript chunks with 100 caption size as that produced the highest amount of correct issues.
- Demonstrations: While adding demonstrations did not hamper the results, we found the improvement not significant enough to include demonstrations in the final prompt. Especially because these require a significant amount of tokens and we found that increasing token size can be detrimental to report quality.

We run this final prompt on all 36 transcripts. For each transcript, we run the model five times and create the final report using *all* and *common* methods, as described in Section III-C. For each issue in the final report, we manually assess its validity by parsing the recording and ground truth answers for each of the datasets.

The results are listed in Table VI, grouped by datasets, AT types and category of user test. In the final column, we show the cumulative results. When considering *all* issues, our tool generated a precision of 85% and a near perfect recall of 98%. The precision indicates that most of the issues reported are correct, as in these have been mentioned by the tester, as opposed to false interpretation. The high recall indicates that almost all the issues mentioned by the tester have been reported by the model, missing only 6 out of 319. For *common* issues, precision slightly increases, but recall drops to 85%. In filtering out issues mentioned less frequently, valid issues were skipped. The slight increase in precision indicates that the filtering process helped remove some false interpretations.

Looking into the different datasets, we see that the model underperforms for D_{fb} compared to the other two. The recall suffers from 4 False Negative issues, which can be attributed to cases where Fable analysts derived issues from the recording that was not apparent in the transcript. We look into the False Positives in the next section.

In terms of recall, reports from the three different AT types perform near equally. However, precision falls for sessions conducted with alternative navigation technology, with screen reader sessions also accumulating comparatively more False Positives. We detail in the next section how the nature of these two technologies can cause the model to misinterpret text in the transcript as an issue where none exists.

Lastly, in comparing the difference of performance between self-guided tasks and user interviews, we see that they scored similar precision and recall. This indicates that the model works well for both types of user tests, despite the transcripts for the latter containing extra speech from the interviewer in the transcripts.

C. RQ3: Strengths and weaknesses

For this Research Question, we manually inspect the reports to understand where the model succeeded and underperformed, and why that might have happened.

With a recall of 98%, the quantitative results showed that the model is able to grasp reported issues from a noisy transcript. From manual observation, we also see that the issues reported are described accurately and in detail. It interprets the tester's complaint and relates it to common accessibility issues, aided by the guidelines used as contextual information.

In Fig. 6, we illustrate a reported issue, specifically one derived from the excerpt in Fig. 2. This demonstrates how the model can parse through the noise and understand the tester's speech, and place that issue within the context of the larger transcript and elaborate on the issue.

However, we also observed that the guidelines can lead to generic descriptions. For instance, we observed the predominance of terms related to "error messages" and "focus management". As shown in Fig. 7, while not incorrect, these often convoluted straight-forward descriptions, when comparing with ground truth. Looking into the internal guidelines, we found that there were multiple entries with these terms and hypothesize its influence on the model.

Reported Issue

Title	Inconsistent Button Labeling
Desc	[Tester] expressed confusion due to inconsistent labeling of buttons. For example, the 'Add Income' button was labeled correctly in one screen but not in another, which led to uncertainty about how to interact with the UI elements.
Time	00:10:41.730 --> 00:10:47.159, 00:13:12.430 --> 00:13:34.189

Fig. 6. Example of a reported issue with title, description and timestamps

Ground Truth

Title	No feedback when bookmarking lesson
Description	[Tester] finds the bookmark function, but gets no feedback in their screen reader when they bookmark the page

Reported Issue

Title	Focus Management Issue
Description	[Tester] encountered an issue with focus management after attempting to add an activity to bookmarks using VoiceOver. After activating the button, there was no spoken feedback or indication of a change in state, which left [tester] uncertain if the action was successful. This lack of immediate feedback and focus movement can lead to confusion and a break in the workflow for screen reader users.

Fig. 7. Example of reports being affected by guideline language

For the model's output, we instructed it to generate the timestamps where the issues originated. In most cases, the model has been able to make accurate mapping of reported issues with the corresponding timestamps, helpful for revisiting issues in the recording.

In some rare cases, as shown in Fig. 8, the issues would span long stretches of timestamps and generate vague descriptions. While not always inaccurate, these are redundant and may lead to extra effort to understand an issue.

Reported Issue

Title	Screen Reader Navigation Issues
Desc	The tester experienced issues with screen reader navigation, as indicated by their repeated attempts to locate and interact with UI elements. This suggests that the screen reader focus was not guiding the user well, leading to confusion and difficulty in performing tasks.
Time	["00:43:04.030 --> 00:43:06.140", "00:43:15.210 --> 00:43:19.469", "00:44:06.830 --> 00:44:16.510", "00:44:53.090 --> 00:44:58.539", "00:45:01.290 --> 00:45:07.650", "00:45:45.820 --> 00:45:49.419", "00:46:01.110 --> 00:46:09.429", "00:46:28.330 --> 00:46:34.109", ..., "00:52:06.810 --> 00:52:09.319", "00:52:23.570 --> 00:52:27.349", "00:52:27.940 --> 00:52:35.870", "00:52:36.080 --> 00:52:45.430", "00:52:46.160 --> 00:52:47.750"]

Fig. 8. Example of a report with vague description and too many timestamps

TABLE VI
RESULTS FOR RQ2

		D_{mm}	D_{gs}	D_{fb}	Screen Reader	Alternative Navigation	Magnification	Self-guided	Interview	Total
No. of sessions		10	10	16	20	10	6	29	7	36
All	TP	177	30	108	240	29	46	198	117	315
	FP	25	4	26	37	13	5	36	19	55
	FN	0	2	4	3	3	0	3	1	4
	Precision	88%	88%	81%	87%	69%	90%	85%	86%	85%
	Recall	100%	93%	96%	99%	91%	100%	98%	99%	98%
Common	TP	140	25	84	188	24	37	156	93	249
	FP	18	1	20	25	10	4	24	15	39
	FN	21	5	18	32	4	8	29	15	44
	Precision	89%	96%	81%	88%	71%	90%	87%	86%	86%
	Recall	87%	83%	82%	85%	86%	82%	84%	86%	85%

We observed that the model is very thorough in analyzing issues. It was able to generate a total of 374 issues from 36 transcripts. A strength of being thorough is that it can extract issues from minute details that may be missed by an analyst. In D_{fb}, where the 'confirmed' issues were issues derived by analysts, the 'mentioned' issues can represent issues they have missed. Of the total unique True Positives detected in D_{fb}, almost 60% are 'mentioned' issues. This indicates that our model is able to detect issues that human analysis may miss.

The downside of being thorough is that the model would sometimes forcefully interpret statements as issues. There are three types of such cases. First, when the tester is remembering past issues experienced with other applications. The model is not always able to differentiate that with issues on the current app. Second, when the tester is talking about issues that are not related to the app, for instance, issues with the keypad or their AT. The model sometimes interprets them as the app's issues. Third, when the tester is complimenting on a certain feature. The model would still try to reframe the statement as a possible issue with the app under test. We exemplify the last case in Fig. 9, where the tester appreciating the inclusion of a transcript is framed as them complaining about video controls.

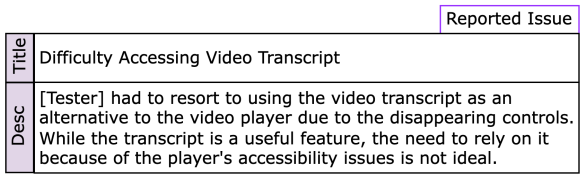

Fig. 9. Example of an inaccurate report that misconstrued a compliment

Lastly, as touched on in the last section, a weakness of the model is related to the nature of screen readers and alternative navigation tools. The former announces elements on the screen, while the latter usually works through voice commands from the user. Both of these audio cues are present in the transcript. In most cases, the model is able to parse the noise, and exclude screen reader announcements and voice commands. However, when the announcement or command contains repeat phrases, the model interprets it as a problem. As shown in Fig. 10, the event of a tester using mouse grids to pinpoint elements and clicking through them is flagged as an issue because the tester used the "click" voice commands multiple times in succession.

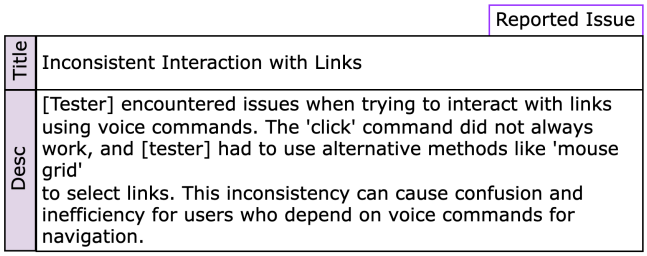

Fig. 10. Example of an inaccurate report that misinterpreted repeat voice commands as a possible issue

The points listed in this section are valuable in understanding the limitations of LLMs in this space, but also direct further investigation to be done to improve this line of work.

V. DISCUSSION

In this study, we look into the applicability of LLMs for automated report generation in accessibility user testing processes. We develop an automated tool, called Reca11, that takes in transcripts from user test sessions and generates a test report containing the accessibility and usability issues encountered by the tester with disability. To develop the tool, we first conduct prompt engineering to configure the most optimal prompt for the LLM. We observe the performance of our tool on real world user testing transcripts and investigate the properties of the reported issues. In this section, we discuss our findings from these steps and what implications these have for the relevant communities.

From our prompt engineering runs, we derive two elements that influence the results the most: input size and the language of contextual data. We observe that dividing the transcripts, especially larger ones, into smaller chunks leads to better issue extraction. Secondly, the model takes significant influence in writing its output from the contextual information provided

to it, e.g., internal guidelines. This can be beneficial when organizations want to adhere to a predetermined format.

From the quantitative results, the tool performs near perfectly, with 98% recall, in deriving all issues mentioned by the tester. However, the tool suffers from false positives, where the model labeled as issues instances that were either not mentioned as an issue originally or not mentioned at all. Despite that, the precision of the tool is 85%.

Lastly, in analyzing the descriptive properties of the reported issues, we derive multiple categories of negative cases.

- Unintuitive issues: The overuse of guideline jargon where it is not necessary can create unintuitive issues.
- Generic issues: The model can merge multiple consecutively mentioned issues together to report a generic issue, often associated with long timestamps.
- Misinterpreted issues: The model can misunderstand neutral and positive statements or statements that are not related to the app under test as issues.
- Distracted issues: Due to the interruption in the transcript from the screen readers, the model is confused by the transcript and derives an issue from it.

These findings and observations help us understand the applicability of this tool and LLMs at large in the context of accessibility user testing.

Implications for Intermediaries: Organizations like Fable who connect software teams to disabled testers work as intermediaries in the user testing process. Similarly, in the CST world, there exist intermediaries like Amazon Mechanical Turk [66] and others [39]. Our tool is built to streamline their processes by automatically generating the test report.

Intermediaries have the opportunity to improve the performance of the tool by solving its limitations. For instance, intermediaries can use data from their archival sessions as demonstrations to the prompt, or to even create a dedicated fine tuned model. Intermediaries can also update processes for conducting test sessions to better adopt Reca11. For example, a separate audio input for screen readers can reduce noise from the transcript, hence improving the tool's performance.

Implications for Software Teams: Software practitioners attribute the disinterest in user testing to its time-consuming processes and lack of expertise [8]. Reca11 tackles both of these challenges, by reducing the time to analyze test recordings, and incorporating timestamps and guideline specifications for easier understanding.

However, this tool cannot work as a complete replacement for expertise and awareness on software accessibility. User tests do not always provide the most accurate feedback [33], [34] and solving the issues derived still require technical understanding of accessible programming.

Implications for Researchers: The field of LLM applications is ever expanding, and newer methods of improving model performance are being experimented with and discovered. Based on our prompt engineering results, there is opportunity to modify and expand on the methods, and improve upon the reported limitations.

Future work can also look into more advanced outputs. The generated reports can rank the issues based on severity, determined by its impact during the user test session or how negatively the tester talks about it. The reported issues can include steps for resolution, with the inclusion of interaction data or the source code as reference.

VI. Threats to Validity

Internal Validity: LLMs typically generate non-deterministic results, causing a potential threat to validity. To mitigate this, we ran the model five times for each transcript, combined duplicate issues and reported all the unique issues.

Using a sentence transformer to detect duplicate issues can pose a threat. To mitigate, we manually experiment with one report from each dataset, checking whether the reported duplicates were reliable and configuring the threshold accordingly.

External Validity: To increase the generalizability of our results, we experiment with test sessions from three different sources. One of these sources, D_{fb}, was conducted in real world setting by external clients, with analysis from experts. While the sessions for D_{gs} were conducted by the authors, the website itself, along with the issues injected, were developed by accessibility experts. We conducted D_{mm} on an open-source application that has been experimented with for accessibility in prior papers. The issues we injected were in accordance to categories derived by prior studies.

We distributed our test sessions to represent all three popular AT types: screen readers, alternative navigation and magnification, and both user testing methods: self-guided tasks and user interview. The numeric distribution of the latter is based on D_{fb}, a real world project.

VII. Conclusion

In this study, we aimed to automatically generate test reports from recorded user transcripts, in an effort to streamline the accessibility user testing process. We employ GPT 4.0, a large language model capable of comprehending, synthesizing and generating natural language, to derive accessibility and usability issues from test transcripts. From our findings, we observe that the model performs well when the transcript is split into smaller chunks and is paired with detailed instructions and contextual information about the test. Our tool secures a precision of 85% and a recall of 98%, indicating its success in finding the great majority of reported issues. We also investigate its inaccuracies and derive limitations such as forced interpretation and susceptibility to guideline language.

To ensure reproducibility of our study, we publish our artifacts on a companion website [67].

VIII. Acknowledgments

This work has been supported, in part, by award numbers 2211790, 1823262, and 2106306 from the National Science Foundation. We thank Fable for their collaboration in making this research possible. We are grateful for the detailed feedback from the anonymous reviewers of this paper, which helped improve this work.

REFERENCES

[1] "Disability fact sheet - who," Mar 2023. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/disability-and-health

[2] A. Alshayban, I. Ahmed, and S. Malek, "Accessibility issues in android apps: state of affairs, sentiments, and ways forward," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1323–1334.

[3] S. Chen, C. Chen, L. Fan, M. Fan, X. Zhan, and Y. Liu, "Accessible or not an empirical investigation of android app accessibility," *IEEE Transactions on Software Engineering*, vol. 48, pp. 3954–3968, 2021.

[4] S. Yan and P. Ramachandran, "The current status of accessibility in mobile apps," *ACM Transactions on Accessible Computing (TACCESS)*, vol. 12, no. 1, pp. 1–31, 2019.

[5] WebAIM, "Webaim: The webaim million - the 2022 report on the accessibility of the top 1,000,000 home pages," https://webaim.org/projects/million/, 2022, (Accessed on 08/24/2022).

[6] W3C, "Web content accessibility guidelines (wcag) 2.1." [Online]. Available: https://www.w3.org/TR/WCAG22/

[7] A. S. Compliance. (2022) A recap of 2022 website accessibility lawsuits. Ada Site Compliance. [Online]. Available: https://adasitecompliance.com/recap-2022-website-accessibility-lawsuits/

[8] S. F. Huq, A. Alshayban, Z. He, and S. Malek, "#a11ydev: Understanding contemporary software accessibility practices from twitter conversations," in *International Conference on Human-Computer Interaction*, ser. CHI '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3544548.3581455

[9] KIF. (2023) Keep it functional - an ios functional testing framework. [Online]. Available: https://github.com/kif-framework/KIF

[10] Android. (2023) Improve your code with lint checks. Google. [Online]. Available: https://developer.android.com/studio/write/lint?hl=en

[11] G. Android. (2023, March) Accessibility scanner. [Online]. Available: https://support.google.com/accessibility/android/answer/6376570?hl=en

[12] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "Puma: programmable ui-automation for large-scale dynamic analysis of mobile apps," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. Bretton Woods, New Hampshire, USA: ACM New York, NY, USA, 2014, pp. 204–217.

[13] M. M. Eler, J. M. Rojas, Y. Ge, and G. Fraser, "Automated accessibility testing of mobile apps," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation*. Västerås, Sweden: ICST, 2018, pp. 116–126.

[14] N. Salehnamadi, A. Alshayban, J.-W. Lin, I. Ahmed, S. Branham, and S. Malek, "Latte: Use-case and assistive-service driven automated accessibility testing framework for android," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Virtual, Okohama, Japan: ACM New York, NY, USA, 2021, pp. 1–11.

[15] N. Salehnamadi, F. Mehralian, and S. Malek, "Groundhog: An automated accessibility crawler for mobile apps," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–12.

[16] A. S. Alotaibi, P. T. Chiou, and W. G. Halfond, "Automated detection of talkback interactive accessibility failures in android applications," in *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*, IEEE. Virtual: IEEE, 2022, pp. 232–243.

[17] A. Alshayban and S. Malek, "Accessitext: Automated detection of text accessibility issues in android apps," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 984–995. [Online]. Available: https://doi.org/10.1145/3540250.3549118

[18] F. Mehralian, N. Salehnamadi, S. F. Huq, and S. Malek, "Too much accessibility is harmful! automated detection and analysis of overly accessible elements in mobile apps," in *2022 37th IEEE/ACM International Conference on Automated Software Engineering*, IEEE. Rochester, Michigan, USA: ACM New York, NY, USA, 2022.

[19] N. Salehnamadi, Z. He, and S. Malek, "Assistive-technology aided manual accessibility testing in mobile apps, powered by record-and-replay," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–20.

[20] G. Brajnik, "A comparative test of web accessibility evaluation methods," in *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*, 2008, pp. 113–120.

[21] A. Aizpurua, M. Arrue, S. Harper, and M. Vigo, "Are users the gold standard for accessibility evaluation?" in *Proceedings of the 11th Web for All Conference*, 2014, pp. 1–4.

[22] D. A. Mateus, C. A. Silva, A. F. De Oliveira, H. Costa, and A. P. Freire, "A systematic mapping of accessibility problems encountered on websites and mobile apps: A comparison between automated tests, manual inspections and user evaluations," *Journal on Interactive Systems*, vol. 12, no. 1, pp. 145–171, 2021.

[23] Y. Inal, K. Rızvanoğlu, and Y. Yesilada, "Web accessibility in turkey: awareness, understanding and practices of user experience professionals," *Universal Access in the Information Society*, vol. 18, no. 2, pp. 387–398, 2019.

[24] Y. Inal, F. Guribye, D. Rajanen, M. Rajanen, and M. Rost, "Perspectives and practices of digital accessibility: A survey of user experience professionals in nordic countries," in *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*, 2020, pp. 1–11.

[25] T. Bi, X. Xia, D. Lo, J. Grundy, T. Zimmermann, and D. Ford, "Accessibility in software practice: A practitioner's perspective," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 4, pp. 1–26, 2022.

[26] S. L. Christopher Vendome, Diana Solano and M. Linares-Vásquez, "Can everyone use my app? an empirical study on accessibility in android apps," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019.

[27] M. V. R. Leite, L. P. Scatalon, A. P. Freire, and M. M. Eler, "Accessibility in the mobile development industry in Brazil: Awareness, knowledge, adoption, motivations and barriers," *Journal of Systems and Software*, vol. 177, p. 110942, Jul. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412122100039X

[28] S. Cao and E. Loiacono, "The state of the awareness of web accessibility guidelines of student website and app developers," in *International Conference on Human-Computer Interaction*. Springer, 2019, pp. 32–42.

[29] H. Petrie and N. Bevan, "The evaluation of accessibility, usability, and user experience." *The universal access handbook*, vol. 1, pp. 1–16, 2009.

[30] S. Alyahya, "Crowdsourced software testing: A systematic literature review," *Information and Software Technology*, vol. 127, p. 106363, 2020.

[31] A. Beganovic, M. A. Jaber, and A. Abd Almisreb, "Methods and applications of chatgpt in software development: A literature review," *Southeast Europe Journal of Soft Computing*, vol. 12, no. 1, pp. 08–12, 2023.

[32] J. Li, A. Dada, B. Puladi, J. Kleesiek, and J. Egger, "Chatgpt in healthcare: a taxonomy and systematic review," *Computer Methods and Programs in Biomedicine*, p. 108013, 2024.

[33] G. Brajnik, Y. Yesilada, and S. Harper, "The expertise effect on web accessibility evaluation methods," *Human–Computer Interaction*, vol. 26, no. 3, pp. 246–283, 2011.

[34] D. A. Mateus, S. B. L. Ferreira, M. R. de Almeida Souza, and A. P. Freire, "Accessibility inspections of mobile applications by professionals with different expertise levels: An empirical study comparing with user evaluations," in *IFIP Conference on Human-Computer Interaction*. Springer, 2023, pp. 135–154.

[35] G. Brajnik, M. Vigo, Y. Yesilada, and S. Harper, "Group vs individual web accessibility evaluations: effects with novice evaluators," *Interacting with Computers*, vol. 28, no. 6, pp. 843–861, 2016.

[36] S. Song, J. Bu, A. Artmeier, K. Shi, Y. Wang, Z. Yu, and C. Wang, "Crowdsourcing-based web accessibility evaluation with golden maximum likelihood inference," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–21, 2018.

[37] Feb 2024. [Online]. Available: https://makeitfable.com/

[38] M. Van Someren, Y. F. Barnard, and J. Sandberg, "The think aloud method: a practical approach to modelling cognitive," *London: AcademicPress*, vol. 11, no. 6, 1994.

[39] W.-T. Tsai, L. Zhang, S. Hu, Z. Fan, and Q. Wang, "Crowdtesting practices and models: An empirical approach," *Information and Software Technology*, vol. 154, p. 107103, 2023.

[40] M. Gómez, R. Rouvoy, B. Adams, and L. Seinturier, "Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring," in *Proceedings of the international conference on mobile software engineering and systems*, 2016, pp. 88–99.

[41] D. Liu, X. Zhang, Y. Feng, and J. A. Jones, "Generating descriptions for screenshots to assist crowdsourced testing," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 492–496.

[42] X. Chen, H. Jiang, Z. Chen, T. He, and L. Nie, "Automatic test report augmentation to assist crowdsourced testing," *Frontiers of Computer Science*, vol. 13, pp. 943–959, 2019.

[43] H. Jiang, X. Li, Z. Ren, J. Xuan, and Z. Jin, "Toward better summarizing bug reports with crowdsourcing elicited attributes," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 2–22, 2018.

[44] F. Scientific. (2023, March) Jaws. [Online]. Available: https://www.freedomscientific.com/products/software/jaws/

[45] N. Access. (2023, March) Nvda. [Online]. Available: https://www.nvaccess.org/about-nvda/

[46] Apple. (2023, March) Introducing voiceover. [Online]. Available: https://www.apple.com/voiceover/info/guide/_1121.html

[47] A. Talkback. (2023, March) Get started on android talkback. [Online]. Available: https://support.google.com/accessibility/android/answer/6283677?hl=en

[48] M. Taeb, A. Swearngin, E. Schoop, R. Cheng, Y. Jiang, and J. Nichols, "Axnav: Replaying accessibility tests from natural language," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, ser. CHI '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3613904.3642777

[49] OpenAI. (2024) Introducing chatgpt. [Online]. Available: https://openai.com/blog/chatgpt

[50] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, 2023.

[51] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM computing surveys (csur)*, vol. 53, no. 3, pp. 1–34, 2020.

[52] N. Nasser. (2024) Money manager app. [Online]. Available: https://github.com/Nada-Nasser/Money-Manager-App

[53] Instructions - record a goose sighting. [Online]. Available: https://record-a-goose-sighting.apps.live.cloud-platform.service.justice.gov.uk/

[54] B. Newing, "Answers - record a goose sighting." [Online]. Available: https://record-a-goose-sighting.apps.live.cloud-platform.service.justice.gov.uk/steps/answers

[55] Nuance. (2023, March) Get more done by voice. [Online]. Available: https://www.nuance.com/dragon.html

[56] A. Support. (2023, March) Use voice control. [Online]. Available: https://support.apple.com/en-us/111778

[57] Fable. (2023, March) What is a switch system. [Online]. Available: https://makeitfable.com/glossary-term/switch-system/

[58] F. Scientific. (2023, March) Zoomtext. [Online]. Available: https://www.freedomscientific.com/products/software/zoomtext/

[59] Hugging Face. [Online]. Available: https://huggingface.co/sentence-transformers/all-mpnet-base-v2

[60] N. Reimers, "Pretrained models - sentence-transformers documentation." [Online]. Available: https://www.sbert.net/docs/pretrained_models.html

[61] OpenAI. (2023, March) Gpt-4. [Online]. Available: https://openai.com/research/gpt-4

[62] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[63] Z. Cheng, T. Xie, P. Shi, C. Li, R. Nadkarni, Y. Hu, C. Xiong, D. Radev, M. Ostendorf, L. Zettlemoyer *et al.*, "Binding language models in symbolic languages," *arXiv preprint arXiv:2210.02875*, 2022.

[64] N. Nashid, M. Sintaha, and A. Mesbah, "Retrieval-based prompt selection for code-related few-shot learning," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2450–2462.

[65] Q. Guo, J. Cao, X. Xie, S. Liu, X. Li, B. Chen, and X. Peng, "Exploring the potential of chatgpt in automated code refinement: An empirical study," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[66] (2024, June) Amazon mechanical turk. [Online]. Available: https://www.mturk.com/

[67] (2025, January) Reca11 reproducible package. [Online]. Available: https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/SyedFatiulHuq/Reca11-reproducible-pkg