A Retrospective Analysis of Grey Literature for AI-supported Test Automation

Filippo Ricca[1] [0000−0002−3928−5408], Alessandro Marchetto[2] [0000−0002−6833−896X], and Andrea Stocco[3,4] [0000−0001−8956−3894]

[1] Università degli Studi di Genova, Genoa, Italy · filippo.ricca@unige.it
[2] University of Trento, Trento, Italy · alessandro.marchetto@unitn.it
[3] Technical University of Munich, Germany · andrea.stocco@tum.de
[4] fortiss GmbH, Germany · stocco@fortiss.org

Abstract. This paper provides the results of a retrospective analysis conducted on a survey of the grey literature about the perception of practitioners on the integration of artificial intelligence (AI) algorithms into Test Automation (TA) practices.
Our study involved the examination of 231 sources, including blogs, user manuals, and posts. Our primary goals were to: (a) assess the generalizability of existing taxonomies about the usage of AI for TA, (b) investigate and understand the relationships between TA problems and AI-based solutions, and (c) systematically map out the existing AI-based tools that offer AI-enhanced solutions.
Our analysis yielded several interesting results. Firstly, we assessed a high degree of generalization of the existing taxonomies. Secondly, we identified TA problems that can be addressed using AI-enhanced solutions integrated into existing tools. Thirdly, we found that some TA problems require broader solutions that involve multiple software testing phases simultaneously, such as test generation and maintenance. Fourthly, we discovered that certain solutions are being investigated but are not supported by existing AI-based tools. Finally, we observed that there are tools that supports different phases of TA and may have a broader outreach.

Keywords: Test Automation · Artificial Intelligence · Grey Literature.

1 Introduction

The adoption of Artificial Intelligence (AI) and Machine Learning (ML) techniques for Test Automation (TA) is attracting the attention of both researchers and practitioners that are recognizing the potential of AI to fill the gap between human and machine-assisted testing activities. In this paper, we refer to such techniques as Artificial Intelligence supported Test Automation (AIsTA). While AIsTA is increasingly being adopted by companies [25], there is still a limited knowledge about the problems it faces, the solutions it proposes, as well as the existing tools and their connection with the software development process.

Several works tried to review the existing works concerning AIsTA [10,12,16,35]. This work is under the umbrella of these secondary studies but it focuses on the grey literature [6] to capture the perception of the practitioners about the adoption of AIsTA. To this aim, we first extended the grey literature conducted by Ricca et al. [25] by considering more recent sources (+41%). Secondly, we conducted a retrospective analysis into the whole set of collected data, aiming at capturing the underlying relationships among TA problems, AI-based proposed solutions, and existing tools.

Our analysis revealed that existing taxonomies provide a good degree of generalization, also when considering more recent grey literature sources. Also, we found that some TA problems require solutions that involve several testing phases (e.g., test creation, execution and maintenance), with some tools actually supporting this complex scenario. We believe that our work could help practitioners better comprehend the state of the art and practice in AIsTA, for instance, for selecting the most appropriate tools for their testing purposes of problems. Moreover, our work could help researchers in capturing issues that require more investigation and new research directions.

2 Existing Grey Literature Analysis for AI-assisted Test Automation

A previous work by Ricca et al. [25] (referred to as previous work, hereafter) presented a study of the grey literature concerning AI/ML-based testing frameworks and tools for Test Automation. The authors analyzed several hundreds web documents from which they retrieved: (a) existing problems about different aspects of the actual automated testing process, (b) solutions based on AI and ML that are used to mitigate such problems and, (c) the list of most popular frameworks and tools available on the market.

The main contribution of the previous work [25] is the construction of two taxonomies of problems and solutions in AIsTA. Moreover, the authors identified the six most-cited AI/ML testing tools that, according to practitioners, can improve the quality of the TA process and the productivity of testers and developers.

In terms of TA problems that are addressed with AI, the taxonomy reports six main categories related to (1) test planning, (2) test design, (3) test authoring, (4) test execution, (5) test closure, and (6) test maintenance [25]. Among the AI-supported solutions to such problems, the taxonomy reports four main categories related to (1) test generation, (2) test oracles, (3) debugging, and (4) test maintenance [25]. Previous work also identified the most popular AIsTA tools, those being Functionize,[5] Applitools,[6] Mabl,[7] Testim,[8] Test.ai,[9] and Appvance.ai.[10] Although being an important first step in understanding this evolving domain, the previous work falls short in providing critical details concerning an examination of the connections between TA problems and investigated solutions and a systematic mapping of AIsTA tools and solutions.

In this work, we first assessed the generalizability of the proposed taxonomy using a set of recent studies not used during its development. After having validated and extended the set of data analyzed by the previous work, aiming at considering more sources, we make a step ahead in collecting and presenting these important relationships based on an analysis of the grey literature. This information can be useful for both practitioners and researchers. Practitioners can determine which solutions to employ based on their specific requirements and challenges, as well as identify the tools that can assist them in implementing the identified solution. At the same time, researchers can focus on areas in which AI is not yet used and that require further investigation.

[5] https://www.functionize.com
[6] https://applitools.com
[7] https://www.mabl.com
[8] https://www.testim.io
[9] https://www.test.ai
[10] https://www.appvance.ai

3 Retrospective Analysis

The objective of our investigation is to gain insights into how practitioners perceive TA problems addressed by AI-enhanced tools, as well as their effectiveness. We focus our retrospective analysis on the grey literature analysis conducted by previous work [25], that we also extended. With this goal in mind, we aim to unravel the connections between existing TA issues and AI-driven solutions, as well as the interplay between current TA tools and these AI-based solutions. We aim to address the following research questions:
RQ_0 (Generalizability). *What is the generalizability of the previous taxonomies [25]?*
RQ_1 (Problems vs Solutions). *How are TA problems and solutions linked in AIsTA?*
RQ_2 (Tools vs Solutions). *How do AIsTA tools and solutions relate to each other?*

RQ_0 assesses the degree of generalizability of the existing taxonomies by validating them on a set of studies that we not used during its development. RQ_1 investigates the solutions provided by AI-powered test automation designed to address specific TA problems. RQ_2 analyzes what solutions can be obtained from AIsTA tools.

3.1 Procedure

Data Extension. Concerning the generalizability of the existing taxonomy (RQ_0), we adopted the same experimental procedure outlined in previous work [25], which is composed by four distinct phases: (1) Google/arXiv search, (2) document selection, (3) data extraction, and (4) taxonomy creation. In short, the authors searched a list of possible web documents with a Google-based search to collect as many documents as possible regarding TA conducted with AI/ML solutions (details are available in our replication package [24]). Second, web documents not related to AI/ML-enhanced TA were filtered out adopting some specific inclusion/exclusion criteria. Third, the authors read and analyzed the candidate documents, filling out a form with the information gathered from each source [11]. Finally, following a systematic process [8,7], they created two taxonomies, one for problems and one for solutions, as well as a list of the most adopted AIsTA tools.

By considering the same search strings of the original paper, updated to the years 2022 and 2023, we broadened the investigation of grey literature to include also the most current works released up to February 2023. We followed the same procedure to extract the information from the sources and update the existing taxonomies. A tabular data extraction form was used to keep track of the extracted information [24] (e.g., the list of all the information sources). The authors proceeded with the analysis independently on separate sets of documents, reusing existing labels previously created, should an existing label apply to the document under analysis.
Data Collection. Our pool of data contains the newly collected data and the existing data that were made available in the replication package of the previous work [25]. The extracted information includes: (1) the link of the document, (2) the author(s) of the document, (3) the nature of the document (e.g., blog post, interview transcript, white papers), (4) the data of the document, (5) the TA tool(s), (6) the testing level (e.g., acceptance, unit), (7) the problem addressed, and (8) the solution offered.
Data Preprocessing. We used `Pandas` data-frames [19] to filter the information related to problems addressed and solutions offered (RQ_1) and tools (RQ_2). We discarded the

entries for which either the problem, solution, and tool was unspecified or for which it was too generic. On the interesting entries, we counted each pairwise combination of `<problem, solution>` and `<tool, solution>`. Particularly, every source mentioning a problem instance and a solution instance is counted as one connection. From the original pool of 541 pairs, we retained 335 pairs for our analysis (62%).

Data Visualization. We used the `Plotly` library [9] to create different Sankey diagrams [27]. We decided to use Sankey diagrams since they are used to visualize the flow of data from an input set to an output set. The elements of both input and output sets are called *nodes*. Such *nodes* are connected by means of *links*, where each *link* connects an input node to an output node, thus showing that a given relationship exists between input and output. The width of a *link* indicates the absolute values of the matching instances. By using Sankey diagrams in our retrospective analysis, we aim at identifying the connections between TA problems and AI-enhanced solutions, and between the existing AIsTA tools and such solutions, with the goal to surface trends and patterns. For RQ_1, we considered the pairs pertaining to each software development phase separately (e.g., maintenance). It is worthy notice that the Sankey diagrams are built by considering all the collected sources, while the ones shown in this paper focus only on the most evident connections, only for space and readability reason.

3.2 Results

RQ_0 (Generalizability). Table 1 (*Problem*) presents the list of problems in TA that are faced with AI/ML, according to [25], and their occurrences. The table also includes the updated entries and occurrences derived from our extension.

The two most represented sub-categories are *Manual code development* and *Maintenance of test scripts*, two well-known cumbersome activities concerning the development and maintenance of test scripts. More in detail, test development requires both domain knowledge and programming expertise. Domain knowledge is required to identify the test data and test oracle, whereas the programming expertise is required when using script-based testing frameworks to programmatically develop test code (e.g., with Selenium WebDriver [5]). Test maintenance is critical in both mobile and web contexts [3,13], because these kinds of applications are subject to a rapid evolution, thus requiring a non-trivial effort to evolve and maintain existing test code. Other relevant problems are related to *Untested code*, i.e., the inability of automated test suites to test the entire application under test (AUT); *Manual data creation*, i.e., the unavailability of high-quality input data for testing [26]; *Visual analysis*, related to the assessment of the visual correctness of the AUT graphical user interface; *Test scripts Flakiness*, i.e., non-deterministic test scripts that either pass or fail when executed on the same AUT due to environmental factors such as the network traffic [17]); and *Manual debugging overhead*, which concerns the search of the root-cause behind test script failures and breakages.

Table 1 (*Solution*) lists the solutions provided by AI/ML, as presented in [25], as well as the updated entries and occurrences derived from our extension (highlighted in the figure). The most represented category is *Automated test generation*, which includes different sub-categories, such as the usage of machine translation (test cases are described using natural language and the testing framework interprets and translates them in executable test scripts) or crawling [21] (automated AUT model inference at support of

Table 1. Original and Extended Taxonomy (in green) of TA problems and solutions.

PROBLEM	[25]	#	SOLUTION	[25]	#
Test Planning	22	32	**Test Generation**	125	192
Critical paths identification	13	13	Aut. test generation	29	48
Planning what to test	7	9	Aut. generation using machine translation	11	13
Planning long release cycles	2	2	Aut. generation from user behaviour	11	22
Test process management	-	8	Aut. test generation from API calls	6	6
Test Design	8	22	Aut. test generation from mockups	3	3
Programming skills required	5	11	Aut. test generation using crawling	2	11
Domain knowledge required	3	9	Declarative testing	-	2
Test-cases	-	1	Predict faulty-areas	-	4
Programming-skills required	-	1	Aut. data generation	22	22
Test Authoring	109	132	Robust element localization	13	13
Manual code development	52	58	Dynamic user-behaviour properties recognition	8	8
Manual API test development	7	11	Automated exploratory testing	7	10
Manual data creation	19	24	Object recognition engine	6	13
Test object identification	13	13	Mock generation	3	3
Cross-platform testing	10	15	Self-learning	2	3
Costly exploratory testing	5	5	Automated API generation	1	9
Locators for highly dynamic elements	1	1	Page object recognition	1	2
Test code modularity	1	1	**Test Execution**	-	8
Accessibility testing	1	1	Cloud execution	-	2
Adequacy-focus on faulty-areas	-	3	Decoupling test framework from host	-	2
Test Execution	71	103	Smart test execution	-	3
Untested code	29	34	Anomaly detection	-	1
Flakiness	18	22	**Oracle**	38	46
Slow execution time	14	16	Visual testing	38	46
Useless test re-execution	4	7	**Debugging**	62	82
Scalability	2	3	Intelligent test analytics	17	22
Parallelization	2	3	Automated coverage report	14	22
Low user responsiveness	1	1	Noticeable code changes identification	12	13
Slow execution time	14	16	Runtime monitoring	10	11
Platform independence	1	1	Flaky test identification	7	8
Test Closure	47	59	Bad smell identification	1	1
Manual debugging overhead	18	22	Decoupling test framework from host	1	1
Costly result inspection	10	11	Root-cause-analysis	-	1
Visual analysis	19	25	Prediction of failures	-	3
Data-quality	-	1	**Maintenance**	81	141
Test Maintenance	82	102	Self-healing mechanisms	43	43
Manual test code migration	3	3	Self-healing test scripts	24	32
Bug prediction	11	13	Smart locators	19	21
Fragile test script	10	12	Intelligent fault prediction	12	13
Regression faults	2	7	Intelligent selective test re-execution	12	12
Costly visual GUI regression	8	10	Intelligent waiting sync	5	5
Maintenance overhead	48	57	Intelligent test prioritization	4	6
			Aut. identification environment configurations	3	6
			Pattern recognition	1	1
			Remove unnecessary test cases	1	1
			Reduce UI testing	-	1
Unspecified	60	77	Unspecified	91	99
Generic	23	33	Generic	25	30
Total	339	560	Total	422	598

test generation). Another relevant category is *Maintenance* of test scripts, often performed with self-healing mechanisms that are able to perform automatic fixes in case of breakage, or via smart locators that are resilient to common breakages causes. Other solutions concern *Debugging* with intelligent test analytics able to reveal the portions of the AUT with high probability of bugs, and *Oracles*, for enabling visual testing, an approach able to automatically check the visual appearance and behavior of a AUT graphical user interface.

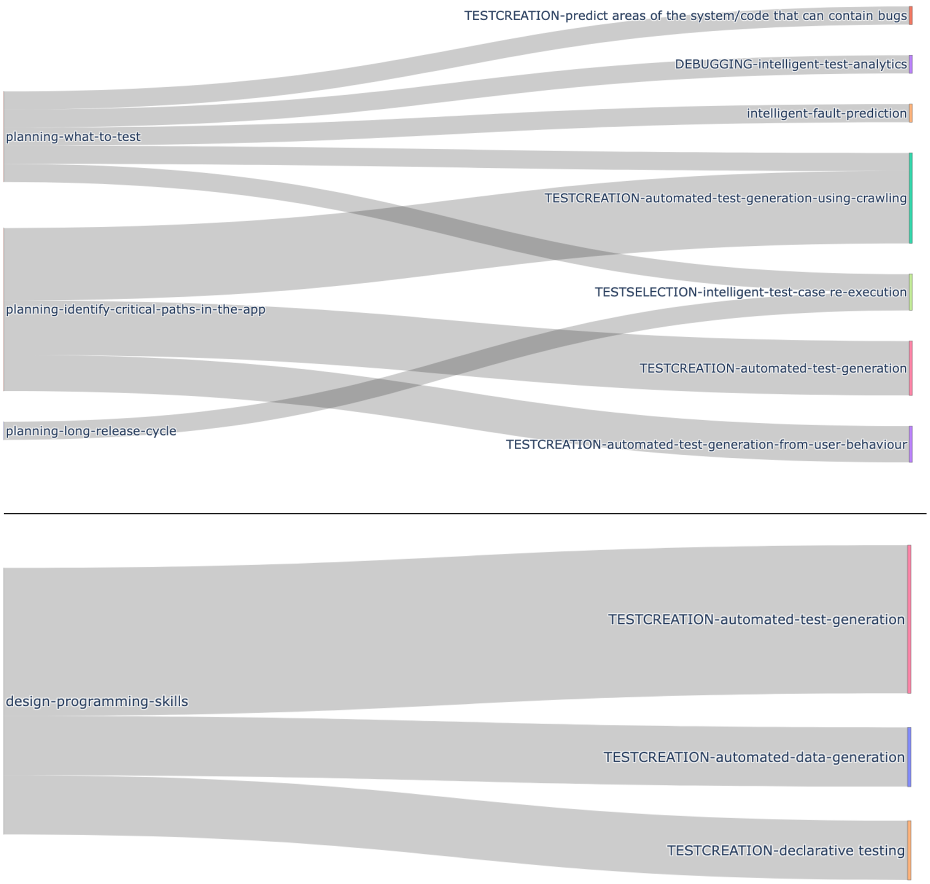

Fig. 1. Problems vs Solutions in the Planning (top) and Design phases (bottom).

The extended version of the taxonomies includes 95 new contributions (out of 231 sources considered, i.e., +41.1% with respect to previous work [25]), which we arranged into categories based on the dimensions described in the prior work. This procedure yielded 419 additional instances to the original taxonomies, of which five new TA problems out of 40 (e.g., test process management, test adequacy on application fault-prone application areas, and data quality), 10 new types of solution out of 44 (e.g., cloud-based test execution, smart test execution, and failure prediction) and 22 new tools out of 71 (e.g., Tricentis Tosca, Google OSS-Fuzz, ChatGPT). For space reasons, we entire list of tools in available in our replication package [24].

Overall, about the generalizability of the existing taxonomies of TA problems and AI-based solutions (RQ_0), we can observe that in our analysis, we have been only partially extended the taxonomies presented by Ricca et al. [25].

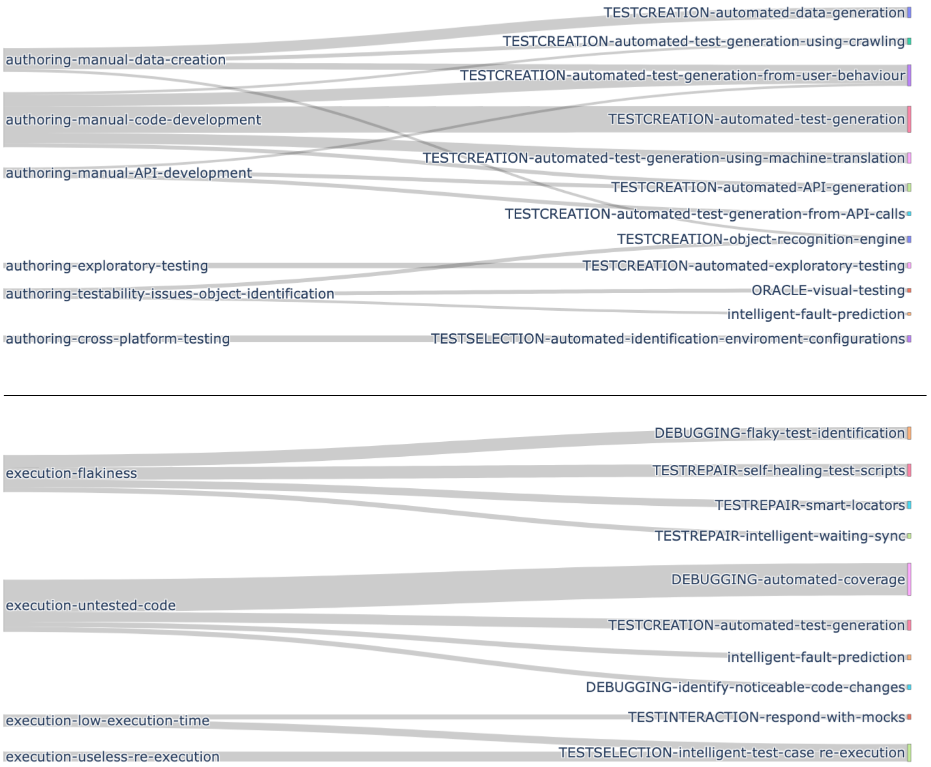
authoring-manual-data-creation
authoring-manual-code-development
authoring-manual-API-development
authoring-exploratory-testing
authoring-testability-issues-object-identification
authoring-cross-platform-testing
TESTCREATION-automated-data-generation
TESTCREATION-automated-test-generation-using-crawling
TESTCREATION-automated-test-generation-from-user-behaviour
TESTCREATION-automated-test-generation
TESTCREATION-automated-test-generation-using-machine-translation
TESTCREATION-automated-API-generation
TESTCREATION-automated-test-generation-from-API-calls
TESTCREATION-object-recognition-engine
TESTCREATION-automated-exploratory-testing
ORACLE-visual-testing
intelligent-fault-prediction
TESTSELECTION-automated-identification-enviroment-configurations
execution-flakiness
execution-untested-code
execution-low-execution-time
execution-useless-re-execution
DEBUGGING-flaky-test-identification
TESTREPAIR-self-healing-test-scripts
TESTREPAIR-smart-locators
TESTREPAIR-intelligent-waiting-sync
DEBUGGING-automated-coverage
TESTCREATION-automated-test-generation
intelligent-fault-prediction
DEBUGGING-identify-noticeable-code-changes
TESTINTERACTION-respond-with-mocks
TESTSELECTION-intelligent-test-case re-execution

Fig. 2. Problems vs Solutions in the Authoring (top) and Execution phases (bottom).

RQ$_1$ (Problems vs Solutions). Figure 1 illustrates the relations between problems and solutions identified in the planning (top) and design (bottom) phases. Problems are shown on the left, while solutions are shown on the right.

In the planning phase, the figure depicts the identification of critical paths (e.g., sequences of clicks, links and functionalities to test) in the AUT as the main problem. In the design phase, the need for programming skills required for developing and implementing test scripts is shown instead as the main issue. It is worth noting that deciding on which aspects to focus test automation involves a variety of solutions, such as fault prediction, test analytics, test generation, and test selection. Both Sankey diagrams emphasize the relationship with various automated test creation activities, ranging from crawling-based to behavior-driven development (BDD) solutions. In particular, it is interesting to observe the contribution of crawling (in the planning phase) as a way to identify critical paths in the AUT and that of declarative testing [34] (in the design phase) as a way to separate the design of test cases from their technical implementation. This may be reflective of the separation of testers and developers as two distinct roles in industry.

In Figure 2, the relationships between problems and solutions in the authoring (top) and execution (bottom) phases are depicted. The primary challenge addressed in the authoring phase is manual code development, which is tackled through various automated test creation activities, including crawling-based and machine-translation-based solutions. Additionally, the generation of automated tests from user behavior and API calls is explored as a potential solution to the problem of manual data creation. Interestingly, the diagram shows that the object identification is a testability problem related to four possible major solutions such as: (i) object recognition engines (i.e., tools that can identity testing elements of the AUT GUI); (ii) exploratory testing (e.g., based on taxonomies of past-discovered bugs); (iii) visual testing (i.e., automated visual checks of the AUT GUI by means of computer vision approaches); and (iv) intelligent fault prediction. Regarding the test execution phase, two primary issues arise, namely test flakiness and identifying portions of code untested. The former can be tackled through various automated techniques such as self-healing test scripts and smart locators or using intelligent synchronization methods able to insert waiting commands in the right place of the flaky test script and with the right waiting time. The latter involves mainly approaches like coverage analysis, automated test generation and to a minimal extent also fault prediction.

There are two primary concerns related to the closure (top) and maintenance (bottom) phases (Figure 3). The first concern, which is the visual oracle problem, can be addressed using visual testing solutions. The second concern, which involves the debugging overhead, requires a combination of various automated debugging techniques such as self-healing methods (e.g., to autonomously decide the corrective actions to apply in case of a broken test), automated coverage toolsets, test analytics (e.g., methods for failure analysis), and change impact analysis (e.g., to identify what has been tested). These techniques can also benefit the task of inspecting test results. This Sankey diagram highlights the fact that in some cases there are unique and specific solutions to a problem (e.g., visual oracle problem) while in other cases the problems are more complex (e.g., debugging overhead) and require multiple distinct solutions. Finally, concerning the maintenance phase, the biggest problem is the overhead of maintaining test scripts and the associated cost, which also needs different and multi-faceted solutions. The greatest contribution is given by smart locators and self healing test scripts, being the most cited solutions for this specific problem. Another important problem is that of the fragility of test scripts. A test script is fragile if it can no longer identify a web element as the AUT evolves. The proposed solutions, in this case, are the usage of smart locators and robust web element localization.

Overall, about the link between TA problems and AI-based solutions (RQ_1), we can observe that in our analysis, we captured existing links as perceived by practitioners. In particular, we observed that some TA problems require solutions that involves several testing phases and tasks.

RQ_2 (Tools vs Solutions). The relations between tools and solutions are depicted in Figure 4. The figure highlights three commonly used tools—Applitools, Mabl, and Functionize—each associated with distinct solutions.

Applitools specializes in visual GUI testing (see thicker connection with ORACLE-visual-testing), which involves automating visual testing and creating visual assertions. The creators of Applitools claim that their tool is able to replicate the human vision system to spot functional and visual regressions. Applitools also provides functionality during the

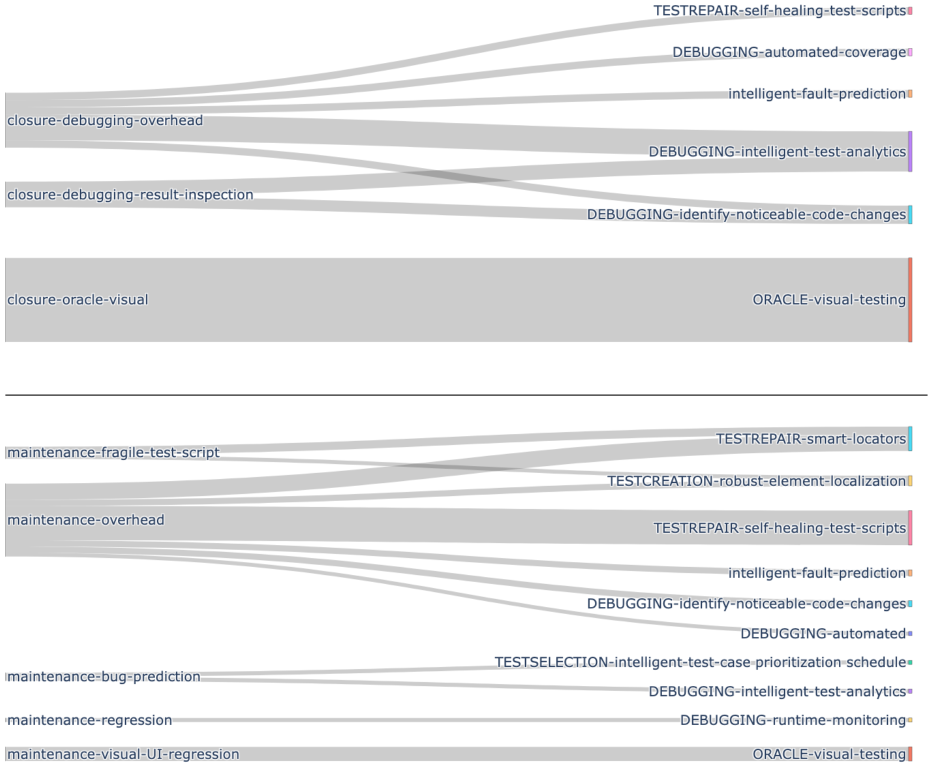

Fig. 3. Problems vs Solutions in the Closure (top) and Maintenance phases (bottom).

debugging phase, allowing developers to identify code changes that have affected the visual appearance of the application. This is helpful for regression testing, to verify that changes to the production code have not caused unintended changes to the GUI. As such, the figure illustrates three primary links for Applitools: visual testing, usage of an object recognition system for test creation, and change impact analysis.

Mabl serves as a flexible test automation platform designed for continuous integration and deployment purposes. Its primary function is to facilitate the creation, execution, and maintenance of test scripts. Mabl is associated with a vast range of solutions that include assistance to test generation, automated repair of locators (with a smart element locators strategy), automated visual testing, and debugging. Although the strengths of Mabl are on test creation and maintenance, an important aspect is that of visual testing. Indeed, Mabl is able to identify visual changes in AUT by comparing screenshots from the current test script run to a visual baseline.

Functionize also turns out to be a fairly complete test platform. Functionize performs test script creation using natural language processing (NLP) and user behavior specification. It also offers advanced self-healing maintenance strategies that leverage intelligent element

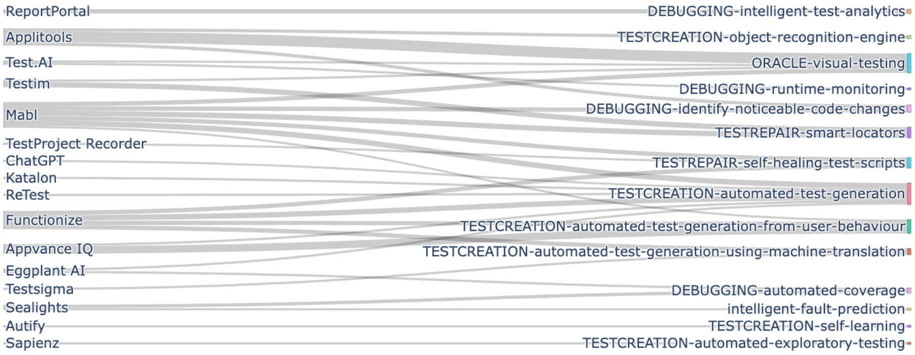
ReportPortal
Applitools
Test.AI
Testim
Mabl
TestProject Recorder
ChatGPT
Katalon
ReTest
Functionize
Appvance IQ
Eggplant AI
Testsigma
Sealights
Autify
Sapienz
DEBUGGING-intelligent-test-analytics
TESTCREATION-object-recognition-engine
ORACLE-visual-testing
DEBUGGING-runtime-monitoring
DEBUGGING-identify-noticeable-code-changes
TESTREPAIR-smart-locators
TESTREPAIR-self-healing-test-scripts
TESTCREATION-automated-test-generation
TESTCREATION-automated-test-generation-from-user-behaviour
TESTCREATION-automated-test-generation-using-machine-translation
DEBUGGING-automated-coverage
intelligent-fault-prediction
TESTCREATION-self-learning
TESTCREATION-automated-exploratory-testing

Fig. 4. Tools vs Solutions

selection. Additionally, Functionize provides intelligent visual differences recognition, allowing developers to quickly identify changes to the user interface that may impact the functionality of the AUT.

In contrast, other tools seem to cater to more specific solutions. For instance, ReportPortal focuses on providing intelligent test analytics, while TestSigma is used for automated test generation with machine translation tools.

Overall, about the link relationship between AI-based solutions and existing tools (RQ_2), we can observe that in our analysis, we identified tools that are more flexible and seem to support more solutions in contrast to more specialized tools.

4 Discussion

This section discusses the achieved results in terms of observations and evidences, as well as the open issues and threats to validity affecting our study.

Observations and Evidences. By analyzing the results collected with our retrospective analysis, we derived the following observations.

O1: We can identify TA problems that could be faced by AI-enhanced existing solutions implemented in AI-based tools. For instance, a well-known challenge in GUI testing is the creation of effective oracles [20]. The use of oracles based on visual testing, using AI and computer vision approaches, is suggested in the grey literature as one of the possible ways to face this challenge. Furthermore, 11 tools that support oracle visual testing have been identified, e.g., Applitools, AI testbot, Mabl, Sealights, Testim and Test.ai. **O1** can be of interest, in particular, for practitioners for quickly selecting the most appropriate solutions for their TA problems and the most adequate tools that support the solutions to their TA problems.

O2: We can identify TA problems that require solutions involving several phases of TA. For instance, in the test planning phase, we defined two problems such as (i) planning what to test and (ii) identifying critical paths in the application under test, among the others. We observe that the latter problem can be mainly faced with automatic test creation solutions, ranging from crawling the application under test, up to the creation test scripts using user behaviors. The former problem instead can be faced by solutions involving test scripts creation (e.g., automatic test creations solutions, test creation by focusing on application areas that are predicted as more buggy), test selection (e.g., based on fault prediction), test execution (e.g., adoption of intelligent test re-execution strategies), and, finally, test debugging approaches (e.g., adoption of test analytics). **O2** can be of interest, in particular, for researchers that can better highlight as some TA problems are addressed from different perspectives, i.e., for some problems, specific ad-hoc solutions can be adequate while, for other problems, more complex solutions need to be studied.

O3: We identified solutions presented in the grey literature that are not supported by existing available tools. However, it is important to keep in mind that the absence of a tool does not necessarily indicate a lack of existing solutions for a particular problem. It could simply mean that the specific tooling solutions were not mentioned in the literature due to the incompleteness of our analysis. For instance, decoupling the test framework from the host environment is referenced, in the grey literature, as one possible solution for facilitating cross-platform testing. However, it seems that this solution is not adequately supported by the existing AI-enhanced tools, that mainly provide approaches that allow the identification of different environmental configurations, to face cross-platform testing. Another solution that seems to be not adequately supported by tools concerns the execution of test cases with mock responses: no tools support the construction of mock objects that can be used in TA. Concerning the test selection and optimization, solutions aiming at prioritizing test cases, removing unnecessary test cases and GUI-based testing seem to be not adequately supported by existing AI-enhanced tools. **O3** can be of interest to both professionals and researchers with the aim of developing innovative tools and technologies capable of supporting the identified solutions.

O4: We identified tools that support solutions related to different TA phases while other tools are specific for a given TA phase. For instance, tools such as Applitools, EggplantAI, Functionize, Mabl, and Test.ai, are able to support different TA phases, e.g., test creation, maintenance, execution. Conversely, tools such as Katalon, Retest, Sapienz, and Testsigma seem to be more specific, thus mainly supporting a given testing phase, e.g., test creation. **O4** can be of interest for practitioners for selecting the most appropriate tools to use in their business, by taking into account the problems they have to face and also other aspects such as specificity and flexibility of tools.

Open Issues. Our analysis covers three years of grey literature concerning the usage of artificial intelligence for test automation. We do not claim that this work captures all relevant grey literature but we are confident that the included documents cover the most important tools up to February 2023. On the other hand, the grey literature is the one that reacts faster to rapidly evolving technological advancements, e.g., the recent release of large language models (LLMs) to the public. While these tools were only partially included in our analysis, we expect to see wider adoption of LLMs in the context of test automation in the future. Further research is necessary to fully understand the complex relationships between artificial intelligence and test automation. Nonetheless, our analysis provides a valuable starting point for understanding the current state of the practice.

Threats to Validity. Threats to the internal validity concern biases and errors during the selection of documents and classification of the considered items (i.e., problems, solutions, and AI-based test automation tools). In particular, the classification task is very difficult in the context of grey literature because the web documents are often informative and non-technical and the terminology is vague and sometimes ambiguous. We relied on an existing taxonomy of works and on an existing procedure, thus we also inherit the threats of the previous work. Our search may have missed relevant documents that are not captured by the search queries. To minimize classification errors, we followed a systematic and structured procedure with multiple interactions. Each doubt concerning creating a new category or classifying an item was discussed among the authors. Concerning the external validity, we considered only Google/arXiv documents in a specific time frame, and our findings may not generalize to other documents or other search engines and repositories. Concerning reproducibility, all our results, in terms of data, plots and references are available in our replication package [24].

5 Related Work

5.1 Test Automation and AI/ML

In the software testing community, there is an increasing adoption of AI/ML solutions to automate the different phases of testing and to deal with problematic issues (e.g., test-suite maintenance and test case prioritization). Test generation is one of the most relevant areas in which the adoption of AI/ML has been investigated. For instance, Zhang et al. [39] and Walia et al. [37] propose the adoption of Computer vision approaches to automate the GUI test generation, with the goal of reducing the required human effort. Qian et al. [23] adopt an evolved OCR-based technique for localizing GUI elements for test generation. Test maintenance is another well-known testing phase that traditionally requires a huge human effort (e.g., for page object generation [29,31,30]) and so in which AI/ML techniques can be beneficial. In fact, computer vision approaches [1] have been also widely used for web test migration [15,14,28] and test repair [33]. Neural embedding of web pages are used to automated the web page similarity for automated model inference [32]. Code-less functional test automation is investigated by Vos et al. [36] and by Phuc Nguyen et al. [22] for test maintenance. The latter study combines Selenium and a ML technique for reducing the time spent by testers changing and modifying the test code. Among other testing issues, Camara et al. [2] propose an ML-based approach for using test code smells as predictors of flaky tests. Mahajan et al. [18] use a computer vision approach for the detection of cross-browser incompatibilities. Feng et al. [4] adopt a computer vision approach for prioritizing test cases for mobile applications. Yadav et al. [38] use ML to check the new code and identify areas of the code in which the test coverage can be increased. Differently from these works, we do not aim at investigating a specific AI/ML technique for TA but rather we aim at going in-depth in the grey literature about TA and AI/ML techniques by realizing a retrospective analysis for capturing the state-of-the-art in terms of TA problems, proposed solutions, and existing tools.

5.2 Secondary Studies

In the literature there are several reviews and surveys in the context of TA via AI/ML. For instance, Trudova et al. [35] report on a systematic literature review (SLR) conducted to study the role of AI/ML in TA. The result of the review confirms that most of the literature studies investigate the use of ML and computer vision techniques for reducing manual intervention in software testing and improving both the effectiveness and reusability of test suites. Lima et al. [16] report on a SLR in which they show that fuzzing and regression testing are the most studied types of testing that adopt ML techniques such as, in particular, neural networks. Leger et al. [12] report on a literature review about the adoption of AI in software testing, especially, by focusing on challenges' identification. They show that the most relevant ones are, e.g., the domain knowledge gap problem, the training data availability, the oracle problem, the computational, cost size and quality of the dataset, test case design, and test result interpretation.

More related to the work presented in this paper are the following two papers. Jha et al. [10] present a preliminary SLR about E2E test automation tools by focusing on testing phases where AI techniques can be adopted: test script generation, test data generation, test execution, test maintenance, and root cause analysis. They also briefly present existing AI-enhanced tools such as Katalon Studio, Applitools, Testim, TestCraft, Parasoft SOAtest, Mabl, AccelQ, and Functionize.

We believe that our work could help practitioners better comprehend the state-of-the-art of AI/ML for test automation, for instance, select the most appropriate tools for their testing purposes of problems. However, our work could help researchers in capturing issues that could require more investigation and new research directions.

6 Conclusions and Future Work

This paper focuses on the analysis of the grey literature about how practitioners perceive the adoption of AI to improve TA. We presented the results of a retrospective analysis conducted by starting from the data collected by Ricca et al. [25] which was extended with additional sources and analyses.

Our investigation include several interesting results, for instance, about the identification of: (i) TA problems faced by existing AI-enhanced solutions implemented in provided tools; (ii) TA problems that requires solutions involving several TA phases (e.g., test creation, execution and maintenance); (iii) solutions investigated but not supported by existing available tools; and, finally, (iv) tools supporting multiple TA phases.

Future research directions consist in conducting a multi-vocal literature review [6] by integrating the findings gathered from the grey literature with those of the white literature. It would be also interesting to conduct controlled experiments with existing AI-enhanced tools, to quantify the benefits they provide and to validate the observed connections with the TA problems and the investigated solutions.

References

1. Bajammal, M., Stocco, A., Mazinanian, D., Mesbah, A.: A Survey on the Use of Computer Vision to Improve Software Engineering Tasks. TSE (2020)

2. Camara, B., Silva, M., Endo, A., Vergilio, S.: On the use of test smells for prediction of flaky tests. p. 46–54. SAST '21, Association for Computing Machinery (2021)
3. Choudhary, S.R., Zhao, D., Versee, H., Orso, A.: WATER: Web Application TEst Repair. In: Proc. of 1st International Workshop on End-to-End Test Script Engineering. pp. 24–29. ETSE '11, ACM (2011)
4. Feng, Y., Jones, J.A., Chen, Z., Fang, C.: Multi-objective test report prioritization using image understanding. In: Proceedings of 31st IEEE/ACM International Conference on Automated Software Engineering. pp. 202–213. ASE 2016, ACM, New York, NY, USA (2016)
5. García, B., Gallego, M., Gortázar, F., Munoz-Organero, M.: A survey of the selenium ecosystem. Electronics **9**, 1067 (06 2020)
6. Garousi, V., Felderer, M., Mäntylä, M.V.: Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. IST **106**, 101–121 (2019)
7. Gyimesi, P., Vancsics, B., Stocco, A., Mazinanian, D., Árpád Beszédes, Ferenc, R., Mesbah, A.: BugJS: A benchmark of javascript bugs. In: Proc. of 12th IEEE International Conference on Software Testing, Verification and Validation. p. 12 pages. ICST '19, IEEE (2019)
8. Gyimesi, P., Vancsics, B., Stocco, A., Mazinanian, D., Árpád Beszédes, Ferenc, R., Mesbah, A.: BugJS: A benchmark and taxonomy of javascript bugs. Software Testing, Verification And Reliability (2020)
9. Inc., P.T.: Collaborative data science (2015), `https://plot.ly`
10. Jha, N., Popli, R.: Artificial intelligence for software testing-perspectives and practices. In: CCICT '21. pp. 377–382 (2021)
11. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering (2007)
12. Leger, G., Barragan, M.J.: Mixed-signal test automation: Are we there yet? In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–5 (2018)
13. Leotta, M., Clerissi, D., Ricca, F., Tonella, P.: Approaches and tools for automated end-to-end web testing. Advances in Computers **101**, 193–237 (01 2016)
14. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: Automated migration of DOM-based to visual web tests. In: Proceedings of 30th Symposium on Applied Computing. pp. 775–782. SAC 2015, ACM (2015)
15. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: PESTO: Automated migration of DOM-based web tests towards the visual approach. Software Testing, Verification And Reliability **28**(4) (2018)
16. Lima, R., da Cruz, A.M.R., Ribeiro, J.: Artificial intelligence applied to software testing: A literature review. In: 2020 15th Iberian Conference on Information Systems and Technologies (CISTI). pp. 1–6 (2020)
17. Luo, Q., Hariri, F., Eloussi, L., Marinov, D.: An empirical analysis of flaky tests. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. p. 643–653. FSE 2014, Association for Computing Machinery, New York, NY, USA (2014)
18. Mahajan, S., Halfond, W.G.J.: Detection and localization of HTML presentation failures using computer vision-based techniques. In: Proceedings of 8th IEEE International Conference on Software Testing, Verification and Validation. pp. 1–10. ICST '15 (2015)
19. McKinney, W., et al.: Data structures for statistical computing in python. In: Proceedings of the 9th Python in Science Conference. vol. 445, pp. 51–56. Austin, TX (2010)
20. Memon, A., Banerjee, I., Nagarajan, A.: What test oracle should i use for effective gui testing? In: 18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings. pp. 164–173 (2003)
21. Mesbah, A., van Deursen, A., Lenselink, S.: Crawling ajax-based web applications through dynamic analysis of user interface state changes. ACM Trans. Web **6**(1) (mar 2012)
22. Phuc Nguyen, D., Maag, S.: Codeless web testing using Selenium and machine learning. In: ICSOFT 2020: 15th International Conference on Software Technologies. pp. 51–60. ICSOFT '20, ScitePress, Online, France (Jul 2020)

23. Qian, J., Ma, Y., Lin, C., Chen, L.: Accelerating OCR-Based Widget Localization for Test Automation of GUI Applications. Association for Computing Machinery (2023)
24. Replication Package. https://github.com/riccaF/quatic2023-replication-package-material/ (2023)
25. Ricca, F., Marchetto, A., Stocco, A.: AI-based Test Automation: A Grey Literature Analysis. In: Proceedings of 14th IEEE International Conference on Software Testing, Verification and Validation Workshops. ICSTW 2021, Springer (2021)
26. Ricca, F., Stocco, A.: Web test automation: Insights from the grey literature. In: Proceedings of 47th International Conference on Current Trends in Theory and Practice of Computer Science. SOFSEM 2021, Springer (2021)
27. Schmidt, M.: The sankey diagram in energy and material flow management. Journal of Industrial Ecology **12**(2), 173–185 (2008)
28. Stocco, A., Leotta, M., Ricca, F., Tonella, P.: PESTO: A tool for migrating DOM-based to visual web tests. In: Proceedings of 14th International Working Conference on Source Code Analysis and Manipulation. pp. 65–70. SCAM '14, IEEE Computer Society (2014)
29. Stocco, A., Leotta, M., Ricca, F., Tonella, P.: Why creating web page objects manually if it can be done automatically? In: Proceedings of 10th International Workshop on Automation of Software Test. pp. 70–74. AST 2015, IEEE/ACM (2015)
30. Stocco, A., Leotta, M., Ricca, F., Tonella, P.: Clustering-aided page object generation for web testing. In: Proceedings of 16th International Conference on Web Engineering. pp. 132–151. ICWE 2016, Springer (2016)
31. Stocco, A., Leotta, M., Ricca, F., Tonella, P.: APOGEN: Automatic Page Object Generator for Web Testing. Software Quality Journal **25**(3), 1007–1039 (Sep 2017)
32. Stocco, A., Willi, A., Starace, L.L.L., Biagiola, M., Tonella, P.: Neural embeddings for web testing. arXiv:2306.07400 (2023)
33. Stocco, A., Yandrapally, R., Mesbah, A.: Visual web test repair. In: Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE '18, ACM (2018)
34. Triou, E., Abbas, Z., Kothapalle, S.: Declarative testing: A paradigm for testing software applications. In: 2009 Sixth International Conference on Information Technology: New Generations. pp. 769–773 (2009)
35. Trudova., A., Dolezel., M., Buchalcevova., A.: Artificial intelligence in software test automation: A systematic literature review. In: Proceedings of the ENASE. pp. 181–192. INSTICC, SciTePress (2020)
36. Vos, T.E.J., Aho, P., Pastor Ricos, F., Rodriguez-Valdes, O., Mulders, A.: Testar – scriptless testing through graphical user interface. Software Testing, Verification and Reliability **31**(3), e1771 (2021)
37. Walia, R.: Application of machine learning for gui test automation. In: 2022 XXVIII International Conference on Information, Communication and Automation Technologies (ICAT). pp. 1–6 (2022)
38. Yadav, V., Botchway, R.K., Senkerik, R., Kominkova, Z.O.: Robot testing from a machine learning perspective. In: 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET). pp. 1–4 (2021)
39. Zhang, C., Cheng, H., Tang, E., Chen, X., Bu, L., Li, X.: Sketch-guided GUI Test Generation for Mobile Applications. In: Proc. of ASE '17. pp. 38–43 (2017)