DOI: https://doi.org/10.15276/aait.08.2025.26
UDC 004.89:004.415.5

Vision-based GUI testing: a systematic review of computer vision advances in software quality assurance

Maksym A. Chaikovskyi[1)]
ORCID: https://orcid.org/0000-0002-7854-7036; chaikovskyi.maksym@stud.onu.edu.ua
Eugene V. Malakhov[1)]
ORCID: https://orcid.org/0000-0002-9314-6062; eugene.malakhov@onu.edu.ua. Scopus Author ID: 56905389000
[1)] Odesa I. I. Mechnikov National University, 2, Dvoryanska Str. Odesa, 65082, Ukraine

ABSTRACT

Ensuring the reliability and adaptability of graphical user interfaces (GUIs) has become a central challenge in contemporary software quality assurance, as modern applications increasingly rely on visually dense, dynamic, and cross-platform interaction patterns. Traditional script-based and DOM-dependent testing approaches are brittle and expensive to maintain, since even minor layout shifts can invalidate large portions of test logic. These limitations have accelerated interest in computer vision and artificial intelligence techniques that evaluate GUIs directly from rendered visuals, enabling more robust, flexible, and human-like validation. This study aims to systematize current research on vision-based GUI testing and to identify how emerging multimodal and language-guided methods reshape quality-assurance practices. Conducted as a structured review of academic and industrial work published between 2010 and 2025, the analysis synthesizes findings across five methodological families: classical image-processing and template-matching approaches, deep neural detectors, generative models for synthetic augmentation, reinforcement-learning agents for autonomous exploration, and transformer-based multimodal systems integrating large language models. The review incorporates an additional empirical-synthesis chapter that consolidates reported evaluation results for vision transformer-based perception, large language models - guided test reasoning, and multimodal vision–language pipelines. The results reveal a consistent technological trajectory: from early perceptual matching to high-fidelity object detection, interactive workflow exploration, and, most recently, semantic interpretation of GUI tasks through natural-language grounding. Experimental evidence shows that visual transformers improve structural perception, reinforcement learning agents enhance interaction coverage, and multimodal systems introduce reasoning capabilities that approach human-like test generation. At the same time, the review identifies persistent challenges, including dataset scarcity – particularly of multimodal screenshot-instruction-execution corpora – fragmented evaluation practices, cross-platform variability, and the computational overhead of multimodal inference. The study concludes that vision-based GUI testing is evolving into a cognitively informed quality-assurance paradigm that integrates perception, behavior, and semantic reasoning. Its novelty lies in providing the unified taxonomy of methods, a consolidated synthesis of empirical verification results, and a set of actionable future research directions. These contributions offer practical value for advancing reproducibility, robustness, and methodological coherence in next-generation GUI-testing systems.

Keywords: Computer vision; GUI testing; multimodal artificial intelligence; software quality assurance; deep learning; large language models; reinforcement learning, automation

For citation: **Chaikovskyi M. A., Malakhov E. V. "Vision-based GUI testing: a systematic review of computer vision advances in software quality assurance". *Applied Aspects of Information Technology*. 2025; Vol.8 No.4: 397–423. DOI: https://doi.org/10.15276/aait.08.2025.26**

INTRODUCTION

Software quality assurance (QA) comprises systematic processes designed to ensure that software systems satisfy functional requirements, perform reliably, and deliver an acceptable user experience. It ensures that applications meet functional requirements and provide reliable user experiences. Quality assurance activities typically include both verification – ensuring that the product is built correctly, and validation – ensuring that the correct product is built. As digital products mediate critical activities – from finance to healthcare – failures can have serious impacts.

Conventional QA practices rely on a combination of manual testing and automated test scripts. Manual testing provides flexibility and human intuition but is resource-intensive and error-prone. Automated testing, implemented through frameworks such as Selenium or Appium, offers improved efficiency but suffers from fragility: even minor changes in the Document Object Model (DOM), widget identifiers, or layout can invalidate large parts of the test suite [1].

Computer vision (CV) offers a paradigm shift in this domain. Unlike DOM-based or code-centric testing, CV-driven approaches enable machines to perceive and interpret software interfaces much like human testers do – through visual recognition and reasoning [2]. This opens opportunities to automate tasks that were previously resistant to scripting, such as identifying misaligned buttons, detecting rendering errors, or verifying the presence of interactive elements. By operating directly on

screenshots, layouts, or rendered UI states, CV methods provide a layer of abstraction that is resilient to underlying code changes, thereby improving maintainability and adaptability. Moreover, the integration of deep learning and reinforcement learning (RL) techniques allows for advanced behaviors, such as exploratory testing agents capable of navigating applications without explicit scripts [3]. These capabilities highlight why computer vision has emerged as one of the most promising technologies for the future of QA automation.

Graphical User Interfaces (GUIs) present an even greater challenge. GUIs are inherently visual, and many of their defects are perceptual, such as misaligned components, overlapping elements, or rendering inconsistencies [4]. Such issues frequently escape DOM-level inspection. Moreover, the diversity of platforms (desktop, mobile, web) complicates the development of reusable test strategies, further reducing maintainability and scalability. These shortcomings have motivated the exploration of CV as a complementary technology for software testing.

Graphical User Interfaces serve as the primary point of interaction between end users and software systems. Errors in GUI design or behavior directly affect usability, accessibility, and user satisfaction. CV-based methods are uniquely suited to this problem space: they can detect discrepancies invisible to code-level testing, compare rendered interfaces against design specifications, and even generate automated scenarios for test validation [5]. From early tools like Sikuli [2] to industrial solutions such as UI X-RAY [6] and modern YOLO-based detectors [7], [8], CV evolution in QA reflects both the rapid progress of AI technologies and the growing demands of industry-scale testing.

While a number of surveys and reviews have addressed the application of artificial intelligence to software testing more broadly [9], [10], there remains a gap in the research with respect to computer vision specifically. Existing systematic studies focus either on general AI-based testing, such as broad reviews of learning-based or automation-oriented QA approaches [9], [11], or on GUI testing unrelated to visual perception, including pattern-based GUI-testing reviews and systematic mapping studies of event- or model-driven techniques [12], [13]. General surveys on the use of computer vision in software engineering also dedicate only limited attention to GUI analysis and do not synthesize CV-based GUI-testing methods into a coherent framework [14]. As a result, current surveys do not integrate findings across Convolutional Neural Network (CNN)-based GUI detection, Generative Adversarial Network (GAN)-based data augmentation, reinforcement-learning agents, or emerging multimodal/transformer-based GUI-testing approaches, which makes it difficult for researchers and practitioners to understand the trajectory of vision-based GUI testing and identify open research directions.

To address this gap, the present work conducts a structured review of vision-based GUI testing technologies from 2010 to 2025. Building on prior surveys while extending beyond their scope, this study aims to provide a unified, methodologically grounded understanding of how computer vision has been applied to GUI testing and where critical research gaps remain.

The contributions of this paper can be summarized as follows:

– First, it traces the historical development of computer vision in software QA, from foundational image-based tools [2], [15] to contemporary approaches that leverage deep learning and multimodal AI [16], [17];

– Second, it proposes a classification of methods and applications, classifying research by technique (e.g., CNNs, GANs, RL, VLMs), task (e.g., element detection, test generation, defect identification), and domain (e.g., mobile apps, web applications, games, enterprise systems);

– Third, it critically analyzes the strengths, limitations, and practical impact of these methods, emphasizing the role of emerging benchmark datasets and evaluation frameworks in consolidating empirical practices within vision-based GUI testing.

Overall, this review positions vision-based GUI testing as an emerging paradigm that extends QA beyond pixel-level verification toward structural understanding and semantic interpretation of interface behavior. By synthesizing developments across multiple methodological generations, the study provides a foundation for advancing robust, scalable, and cognitively informed GUI-testing systems.

1. LITERATURE REVIEW AND PROBLEM STATEMENT

1.1. Conceptual Background and Definitions

GUI testing refers to the verification of visual layouts, component presence, rendering correctness, and interactive behavior of software interfaces. Traditional GUI-testing approaches operate on internal representations such as event models, DOM

structures, widgets, or model-based abstractions. These techniques validate interactions and logic but often struggle with visual defects, layout inconsistencies, and rendering issues that appear only at the perceptual level.

Vision-based GUI testing addresses these limitations by analyzing screenshots, rendered states, or visual traces using CV and machine-learning techniques. Instead of depending on accessibility identifiers or internal widget metadata, vision-based methods interpret GUI states directly from pixels, enabling more resilient detection of visual irregularities, theme variations, cross-platform discrepancies, and layout changes [4], [7], [17]. This perspective aligns GUI testing with the broader shift toward perceptual software analysis documented in CV-for-SE literature [14]. General-purpose QA automation strategies unrelated to vision (e.g., test-case generation from natural language or predictive defect analytics) are not addressed in depth, except where they intersect with vision-based approaches.

Over more than a decade of development, computer-vision-based GUI testing has expanded into a diverse research domain encompassing visual detection, structural layout understanding, visual regression analysis, visual interaction modeling, and more recently, semantic reasoning over GUI states using multimodal AI. The breadth of these methodological directions motivates the need for a unified conceptual and empirical synthesis.

1.2. Related work

Several surveys and mapping studies provide important context for understanding the broader space of GUI testing and AI-driven software quality assurance. Although none focus specifically on vision-based GUI testing, each contributes conceptual or methodological insights relevant to this review.

Surveys on AI-assisted software testing outline major applications of machine learning and intelligent automation across the testing lifecycle, describing trends in data-driven test generation, predictive defect models, and maintenance challenges in automated test suites [9], [11]. These observations inform the methodological discussion of evaluation practices and data requirements later synthesized in this review.

Industry- and practitioner-focused analyses further highlight real-world barriers to test automation at scale, emphasizing the brittleness of GUI scripts, high maintenance costs, and the need for more resilient automation tools capable of tolerating interface changes [10]. These insights underscore the practical motivation for perception-based GUI testing, which seeks to improve robustness by grounding analysis in rendered visual states rather than internal identifiers.

Surveys devoted specifically to GUI-testing methodologies classify existing techniques into event-driven, model-based, and pattern-based approaches, offering taxonomic principles, terminology, and a historical overview of GUI-testing research [12], [13]. While these works do not examine computer-vision-based methods, their conceptual frameworks inform the classification structure developed in this review.

A broader software-engineering perspective is provided by surveys on computer vision in SE workflows, which describe how perception-based techniques are applied to tasks such as documentation analysis, traceability, and code visualization [14]. Although GUI analysis is addressed only briefly, this body of work highlights the increasing integration of perception into software-analysis pipelines and the absence of domain-specific syntheses centered on GUI testing.

Taken together, these surveys contribute an understanding of methodological trends, industrial challenges, and conceptual foundations in both GUI testing and AI-driven QA. However, they treat GUI testing, computer vision, and multimodal AI as separate domains, leaving a gap in the literature regarding an integrated, CV-focused synthesis.

1.3. Problem Statement and Research Gap

Based on the reviewed surveys and background literature, several gaps in existing research can be articulated.

Lack of a unified synthesis of vision-based GUI testing methods**.** Existing reviews classify GUI testing or AI testing broadly but do not integrate work across classical CV, deep learning, GAN-based augmentation, RL-based exploration, and multimodal reasoning;

Absence of a comprehensive taxonomy. No prior survey categorizes vision-based GUI testing methods by technique, task, and application domain;

Inconsistent evaluation methodologies. Studies employ heterogeneous datasets, metrics, and experimental designs, limiting comparability and reproducibility;

Limited coverage of multimodal and transformer-based approaches. Recent developments in vision–language models (VLMs) and Large

Language Model (LLM)-guided testing have not been synthesized in prior surveys;

Separation of research communities. Contributions from mobile testing, web automation, game testing, and enterprise GUI validation remain fragmented, hindering cumulative progress;

Need for a historical mapping of research. Existing surveys do not trace how vision-based GUI testing evolved from early template-based tools to modern multimodal models.

2. RESEARCH AIM AND OBJECTIVES

The aim of this review is to provide a comprehensive, methodologically grounded synthesis of vision-based GUI-testing research. By integrating findings from computer vision, software engineering, reinforcement learning, and multimodal AI, the review seeks to address the conceptual fragmentation, inconsistent evaluation practices, and lack of unified methodological frameworks identified in prior work.

To achieve this aim, the study pursues the following objectives:

1. Trace the historical development of computer vision in software QA, from foundational image-based approaches [2], [15] to contemporary deep-learning-driven and multimodal architectures [16], [17];

Develop a unified taxonomy that classifies vision-based GUI-testing methods by technique (e.g., CNNs, GANs, RL agents, vision–language models), task (e.g., element detection, defect identification, test generation), and application domain (e.g., mobile, web, games, enterprise systems);

Analyze strengths, limitations, and practical implications of existing methods, highlighting recurring challenges such as dataset scarcity, scenario generation, cross-platform generalization, and robustness to real-world UI variability;

Review experimental methodologies, including datasets, evaluation metrics, benchmark suites such as GUI Testing Arena [18], and validation strategies used across the literature;

Identify methodological gaps and future research opportunities, highlighting challenges related to dataset scarcity, cross-platform robustness, semantic correctness, and the integration of multimodal reasoning into practical GUI-testing workflows.

These objectives structure the remainder of the paper, guiding the review methodology, the synthesis of findings, and the development of a comprehensive taxonomy in the Results section.

3. MATERIALS AND METHODS

The review adopts a structured qualitative synthesis approach intended to capture the breadth of computer-vision-based GUI testing research published between 2010 and 2025. The methodology emphasizes identifying methodological families, categorizing tasks and application domains, and analyzing evaluation practices.

3.1. Search strategy and data sources

The corpus for this review consists of peer-reviewed scientific papers and academically recognized sources related to computer-vision-based GUI testing.

Because GUI testing constitutes a specialized research area within software engineering, relevant papers were collected through:

– screening reference lists in existing surveys on AI-based testing and GUI testing [9], [10], [11], [12], [13], 14];

– identifying key contributions across major software engineering and AI venues;

– including representative works on deep-learning-based detection, GAN-based augmentation, reinforcement-learning agents, and multimodal GUI-testing systems;

– incorporating seminal foundational tools (e.g., Sikuli [2]) and contemporary multimodal approaches published between 2020-2025 ([16], [19], [20], [21], [22]).

The resulting corpus covers classical image-based tools, CNN/YOLO detectors, hybrid structural-CV models, generative augmentation methods, RL-based GUI exploration systems, and multimodal LLM-guided frameworks. These works collectively represent the progression of computer-vision-based GUI testing and provide adequate breadth for a comprehensive synthesis.

3.2. Inclusion and Exclusion Criteria

A study was included if it met at least one of the following criteria:

– Primary relevance: It proposed, evaluated, or benchmarked a vision-based GUI-testing approach;

– Methodological relevance: It introduced GUI-grounded applications of deep learning, Vision Transformers (ViTs), reinforcement learning, generative augmentation, or multimodal AI;

– Empirical relevance: It presented datasets, benchmarks, evaluation protocols, or annotation schemes used for GUI analysis;

– Supportive relevance: It contributed methodological insights necessary to understand

GUI-testing components (e.g., CV detection models, RL interaction frameworks, VLM architectures).

Exclusion criteria:

– Studies focused exclusively on non-visual GUI testing (purely event-driven, DOM-based, or model-based) without a perceptual component;

– Papers lacking methodological detail or evaluation (e.g., conceptual proposals without implementation);

– Duplicate publications, short abstracts, workshops without substantive technical contribution, or non-English works;

– General computer-vision papers with no GUI relevance unless used to contextualize a technique employed in GUI-testing research.

3.3. Use of Supporting Non-GUI Sources

While the core synthesis includes only studies that satisfy the inclusion criteria, certain papers that fall outside the GUI domain were retained as supporting sources when they provide essential conceptual, methodological, or contextual value.

These non-primary sources were used in the following cases.

1. Foundational computer-vision models (e.g., YOLO, ViT, GAN frameworks) that underpin GUI-specific methods and inform their technical capabilities.

Reinforcement-learning papers originally targeting non-GUI environments when their architectures or training practices inform GUI exploration methods.

Computer vision for software engineering surveys [14], which contextualize how visual analysis is integrated into broader SE tasks.

AI-testing and automation reviews [9], [10], [11], which inform methodological trends, evaluation practices, and industrial challenges relevant to GUI QA.

Domain-adjacent studies (e.g., game or mobile visual analytics) that provide parallel insights into dataset scarcity, robustness, or cross-platform variability.

These supporting papers were **not** analyzed as part of the primary evidence set in Section 4 but were referenced when necessary to clarify definitions, justify methodological choices, or contextualize the evolution of techniques. This practice follows accepted standards in software-engineering systematic reviews where domain-adjacent research strengthens conceptual and methodological grounding.

3.4. Data Extraction

A structured data-extraction form was used to capture methodological and empirical details.

For each study, the following data categories were recorded:

– publication metadata (authors, year, venue);

– technique type (e.g., classical CV, CNN/YOLO, GAN, RL, multimodal VLM);

– primary task (element detection, defect detection, layout analysis, scenario generation, RL navigation);

– application domain (mobile, web, games, enterprise);

– datasets used (public, custom, synthetic/GAN-augmented);

– evaluation metrics (mAP, IoU, precision/recall, RL rewards, qualitative assessment);

– experimental setup (training configuration, test environment, baselines);

– reported strengths, limitations, and applicability.

This schema ensured consistency across studies despite differences in methodology or task focus.

3.5. Data Synthesis

The review uses thematic synthesis to integrate findings across extracted dimensions. Themes emerged through iterative comparison of technique families, task categories, evaluation practices, and verification strategies. Quantitative patterns (e.g., publication year trends, domain prevalence) were used to support the qualitative narrative.

3.6. Threads to Validity

Potential threats to validity include:

– Coverage bias: Although representative, some niche or emerging studies may be excluded;

– Heterogeneity of evaluation methods: Diverse datasets and metrics limit the comparability of results;

– Publication bias: Positive results are more prevalent in published works, potentially underrepresenting failures;

– Domain variability: Variations between mobile, web, desktop, and game GUIs complicate cross-domain generalization.

To mitigate these threats, the review incorporates studies across multiple methodological families, triangulates findings using existing surveys, and employs transparent selection, extraction, and synthesis strategies.

4. RESEARCH RESULTS

4.1. Historical Development of Vision-Based GUI Testing (2010–2025)

The evolution of vision-based GUI testing spans more than a decade and reflects a continuous shift from simple visual matching toward deep perceptual understanding and multimodal reasoning. The trajectory can be divided into several major phases, each marked by characteristic methods, toolkits, and research priorities.

4.1.1. Phase 1 (2010–2013): Early Image-Based Testing and Foundational Tools

The foundations of vision-based GUI testing lie in the recognition that software quality assurance cannot rely solely on traditional code- and DOM-based techniques. Graphical user interfaces often fail in ways invisible to code-level inspection: misaligned widgets, overlapping buttons, blurred or missing icons, or subtle visual regressions that directly affect user experience. To address these issues, researchers began applying CV methods that enabled testing systems to perceive software interfaces in a manner closer to human inspection [2], [6]. Such approaches introduced the possibility of resilience to code changes, cross-platform generalization, and scalable automation of perceptual verification. Early surveys on computer vision in software engineering also contextualized these developments within a broader trend toward integrating perceptual analysis into software-quality tasks [14].

One of the earliest demonstrations that GUIs could be tested directly from rendered pixel data, without reliance on DOM structure or program instrumentation, was introduced by Chang, Yeh, and Miller in their vision-based GUI testing framework [23]. Tools such as Sikuli (2010) demonstrated the potential of template matching, allowing testers to interact directly with screenshots instead of internal identifiers [2]. Edge detection and key-point-based descriptors (e.g., SIFT, SURF, ORB) were also applied to identify GUI components, while pixel-wise comparison provided a baseline for visual regression testing. Although pioneering, these techniques were fragile to resolution differences, layout shifts, and rendering artifacts. Their brittleness revealed the need for more robust and adaptive approaches, setting the stage for the subsequent transition toward hybrid and learning-based methods.

Taken together, this initial phase established the core insight that perceptual properties of GUIs must be validated visually, not only structurally. It introduced the fundamental shift from DOM-oriented verification to image-based testing, laying the groundwork for the deep-learning, generative, reinforcement-learning, and multimodal advances that followed in later years.

4.1.2. Phase 2 (2014–2017): Structural-Visual Hybrids, Invariants, and Early Industrial Adoption

As vision-based GUI testing matured, it began transitioning from academic exploration to practical deployment. This stage marked the consolidation of computer-vision techniques into usable testing frameworks and commercial solutions, bridging the gap between research prototypes and real-world software-engineering environments.

Early adoption of computer-vision testing in industry demonstrated its capacity to detect interface inconsistencies that traditional DOM- or code-based methods overlooked. A notable example is UI X-Ray [6], introduced by IBM researchers, which applied visual analysis to identify layout defects, misalignments, and rendering issues across enterprise applications. In parallel, commercial tools such as Applitools began offering visual-regression testing powered by perceptual comparison models. These systems integrated directly into CI/CD pipelines and frameworks such as Selenium and Appium, making visual QA automation accessible to enterprise teams. The success of these platforms confirmed that computer vision could deliver measurable gains in accuracy, maintainability, and cost efficiency.

Academic research played a complementary role by developing hybrid methodologies and open resources that strengthened practical adoption. Early structural-visual approaches laid the groundwork for later deep-learning hybrids, combining edge detection, layout parsing, and heuristic grouping to improve robustness. These efforts also helped standardize evaluation practices by introducing reproducible datasets, annotated examples, and baseline models – making it easier for both academia and industry to evaluate, compare, and extend visual testing tools.

Despite these promising developments, early systems revealed limitations that shaped subsequent research directions. Most could detect visual anomalies but lacked semantic understanding – the ability to determine whether an interface behaved according to user intent. Dependence on baseline screenshots limited scalability and robustness under UI variation. Moreover, these techniques remained largely static, focusing on isolated screenshots rather

than sequential user interactions. These limitations foreshadowed later advances in learning-based perception, synthetic data generation, interactive reinforcement-learning agents, and multimodal reasoning models.

The industrial adoption of computer-vision-based QA validated the technology's potential beyond research settings. It demonstrated that visual analysis could be systematically integrated into software-testing workflows, encouraged standardization across tools and frameworks, and highlighted the need for more intelligent, adaptable methods. Most importantly, this phase established the foundation for the next generation of testing systems-those capable of dynamic interaction, richer perception, and eventually semantic reasoning-positioning vision-based QA as a cornerstone of modern software-quality practice.

4.1.3. Phase 3 (2016–2020): Deep Learning and GUI Object Detection

The introduction of deep learning marked a transformative period in vision-based GUI testing. CNNs enabled automatic extraction of discriminative visual features, replacing brittle handcrafted descriptors and significantly improving robustness in GUI-component detection [7]. During this phase, computer-vision-driven GUI testing evolved from experimental prototypes to scalable, data-driven systems capable of supporting real-world applications.

CNN-based detectors such as Faster R-CNN, SSD, and the YOLO family (YOLOv3–YOLOv5) were successfully adapted to GUI-component detection tasks, enabling real-time localization of buttons, icons, text fields, and other interface elements [4], [7], [8], [17]. These models provided substantial gains in accuracy, speed, and cross-application generalization compared to template-based or hybrid structural approaches. Their success encouraged the development of larger annotated GUI datasets and improved training pipelines, making data-driven perceptual analysis increasingly practical.

Hybrid frameworks also advanced during this period. UIED (2020) combined traditional edge detection with CNN-based classification to reconstruct GUI layouts, integrating structural parsing with high-level semantic recognition [5]. This architecture illustrated the power of blending classical CV signals with learned representations, laying the groundwork for later transformer-based and multimodal models.

Deep-learning models were also adapted to non-standard GUI environments. For example, studies on visual bug detection in HTML5 Canvas games demonstrated the applicability of CNN-based models to dynamically rendered, non-DOM environments [24]. Similarly, widget detection for industrial mobile games introduced one of the largest GUI datasets in the field, benchmarking YOLO, SSD, and other detectors across varied UI styles and interaction patterns [25]. These domain-specific investigations showed that deep-learning-based GUI detection could scale beyond mobile and web apps to gaming and high-variability interactive systems.

Several works highlighted the importance of robustness to diverse resolutions, themes, and UI design languages. Multi-scale CNN architectures and cascaded feature extractors demonstrated improved precision under real-world variability [4]. These enhancements reflected a broader methodological shift toward architectures that can handle heterogeneous layouts, small objects (e.g., icons), and dense visual regions.

This period represents the breakthrough point where GUI testing embraced scalable computer vision. Deep-learning detectors transformed perception quality, hybrid systems integrated structure and semantics, and domain-expanding studies validated applicability across mobile, web, and game interfaces. This phase established the technical foundations on which subsequent advances – synthetic data generation, reinforcement learning, and multimodal reasoning – were built.

4.1.4. Phase 4 (2018–2022): Synthetic Data, GAN-Based Augmentation, and Robustness

As deep-learning adoption grew, the scarcity of labeled GUI datasets became a critical obstacle. GANs emerged as a practical solution for creating synthetic training data and simulating edge cases.

GANs emerged as a powerful tool for overcoming data scarcity. GAN-based approaches synthesized realistic GUI screens and introduced controlled visual distortions – occlusions, misalignments, corrupted assets, lighting variations, missing icons, or inconsistent color schemes. These techniques augmented training corpora, enriched distribution diversity, and reduced overfitting to limited real-world samples [26], [27].

By generating targeted defect types, researchers could stress-test object detectors under conditions that were previously rare or missing in real datasets. This also improved robustness to noise, layout diversity, and theme variations, significantly strengthening downstream detection models.

In addition to full-screen synthesis, several works focused on intraclass image augmentation and

fine-grained defect generation. GAN-based methods were used to simulate subtle anomalies – blur, pixel corruption, minor shifts in spacing – that commonly appear in visual regressions. These synthetic variations made detectors more sensitive to small but user-visible issues, expanding coverage beyond the coarse failures detectable by classical CV.

Complementary work in adjacent domains reinforced this trend. Studies demonstrated that multi-stage and cascaded deep-learning pipelines exhibit improved resilience under real-world variability [33], while neuro-fuzzy hybrid architectures showed promise for lightweight, data-efficient learning [28]. Although not GUI-focused, these findings provided methodological support for generative and hybrid strategies in GUI testing.

Generative augmentation also supported domain transfer across heterogeneous GUI environments. For example, game-focused GUI-detection work employed synthetic generation to diversify icon appearance and UI widget styles, demonstrating improved cross-skin and cross-resolution generalization [25]. Similar techniques were applied in Canvas-based GUI-bug detection to simulate dynamic rendering artifacts and visually complex backgrounds [24].

These cross-domain applications validated the broader usefulness of generative frameworks in GUI-centric perception tasks and highlighted the need for domain-randomization techniques tailored to UI semantics.

The generative modeling era also coincided with the first community attempts to standardize GUI datasets. Works that combined real screenshots with synthetically generated ones offered more diverse training sets and paved the way for later benchmarks such as GUI Testing Arena [18]. These hybrid datasets allowed detectors to be evaluated across a wider range of visual environments and improved reproducibility in experimental studies.

The period from 2018 to 2022 can be characterized as the synthetic-data expansion era in vision-based GUI testing. Generative models addressed long-standing data shortages, enabled systematic defect simulation, improved cross-platform generalization, and influenced dataset design and evaluation. These advances strengthened the robustness of GUI detectors and laid the groundwork for the next major transition – interactive, agent-based exploration and multimodal reasoning – by establishing the perceptual resilience required for more complex testing architectures.

4.1.5. Phase 5 (2018–2023): Reinforcement Learning and Autonomous GUI Exploration

While deep-learning-based perception significantly improved GUI element detection, static screenshot analysis remained insufficient for capturing multi-step interactions, dynamic UI flows, and user-driven behavior. This limitation motivated the emergence of RL approaches, in which autonomous agents learn to navigate GUIs visually, exploring applications through sequential decision-making. By rewarding state coverage, RL-based testers reproduce user workflows, generate test cases dynamically, and adapt to changes without fragile scripts – an approach consistent with broader deep-learning evidence showing that hierarchical and cascaded architectures can enhance detection precision and temporal stability in dynamic environments [29].

Initial breakthroughs included Adamo et al. (2018), one of the earliest demonstrations of deep reinforcement learning (DRL) for Android GUI testing. Their work showed that agents could learn navigation policies directly from screenshots, identifying clickable elements and exploring application states without predefined scripts. Shortly thereafter, Eskonen et al. (2020) extended these ideas by automating GUI exploration using image-based DRL, addressing challenges such as sparse rewards and visually similar states.

These foundational studies established the premise that RL agents could approximate human exploratory behavior, enabling the discovery of navigation paths that would be difficult to encode manually.

Subsequent work refined RL architectures for mobile and web testing. Cai et al. (2021) introduced a ResNet-enhanced deep RL model for Android app testing, demonstrating improvements in state representation and action selection. Systems such as DinoDroid further explored state-space coverage, rewarding agents for reaching diverse UI states and uncovering hidden navigation paths [3], [30].

These methods represented a shift from verifying isolated screens to examining full interaction sequences.

By interpreting screenshots as observations and using rewards to guide exploration, RL-based systems could:

– autonomously traverse interfaces;

– discover multi-step workflows;

– generate test cases dynamically;

– and adapt to UI changes without brittle scripted logic.

Although fully semantic multimodal agents emerged later, this period also saw the introduction of hierarchical and attention-based RL models, which improved state abstraction and reduced perceptual redundancy during navigation. Research on multi-agent coordination and path optimization further suggested that distributed exploration strategies could enhance coverage and efficiency in complex apps [34], [35].

These advances foreshadowed the next phase, in which agents would begin combining perception with linguistic reasoning.

Toward the end of this period, early studies explored LLM-guided RL, in which large language models interpret high-level goals and provide action guidance during visual exploration. For example, works such as LLM-Based Deep Reinforcement Learning Agents for Bug Detection demonstrated that language-informed reward shaping and reasoning could significantly improve GUI navigation efficiency and reduce exploration dead-ends [31]. These systems represent a hybrid category: still rooted in RL-based exploration, but beginning to incorporate semantic priors provided by large language models.

While multimodal and full VLM-based systems are discussed in Phase 6, these transitional LLM-guided RL models show how reinforcement learning evolved toward increasingly human-like testing behaviors.

The 2018–2023 periods marked the rise of interactive, agent-based GUI testing, expanding the field beyond static perception.

Reinforcement learning enabled tools to:

– navigate interfaces autonomously;

– uncover complex workflows;

– dynamically generate test sequences;

– and adapt to visual and structural changes.

This methodological shift brought the field closer to human exploratory testing and laid the groundwork for the multimodal reasoning systems that emerged beginning in 2023.

4.1.6 Phase 6 (2021–2025): Vision Transformers, Multimodal Models, and LLM-Guided Testing

The years 2021–2025 represent the most recent and rapidly evolving stage of vision-based GUI testing, characterized by a transition from perceptual accuracy toward semantic understanding and reasoning. Building upon the methodological maturity established through CNNs, generative augmentation, and reinforcement learning, the field began adopting transformer-based architectures and multimodal vision – language systems. These approaches aimed to interpret GUIs not only as visual arrangements of components, but as meaningful interfaces whose behavior must be aligned with user intent, textual requirements, and application semantics.

The introduction of ViTs into GUI-analysis pipelines demonstrated substantial improvements in cross-layout generalization and structural interpretation. Unlike CNNs, which rely on localized receptive fields, ViTs employ global self-attention, allowing them to capture long-range dependencies and relational patterns across an interface – such as alignment between buttons, spacing between UI components, or consistency between text and icons. Early studies showed ViTs outperforming CNNs when evaluating unseen applications or GUI styles, laying the technical foundation for multimodal reasoning systems that integrate vision with language [16].

A transformative development in this period was the rise of multimodal models that combine screenshot analysis with the semantic capabilities of LLMs. These systems process visual information through a vision encoder (CNN or ViT) while using an LLM to interpret natural-language specifications, reason about workflows, or generate test procedures.

Research demonstrated that multimodal architectures could support capabilities such as:

– extracting GUI semantics from screenshots;

– identifying discrepancies between textual requirements and visual interface states;

– performing specification-grounded validation;

– generating scenario-based or behavior-driven test cases [20], [21].

Systems such as ScenGen (LLM-guided scenario-based GUI testing) showed that LLMs can translate high-level requirements (“verify that checkout requires authentication”) into actionable, step-by-step test sequences grounded in GUI screenshots [20]. This evolution marked the shift from perceptual correctness (e.g., verifying that elements appear correctly) to semantic correctness (e.g., verifying that the interface behaves as intended).

As a bridge between RL-based testing and multimodal reasoning, researchers introduced LLM-guided deep reinforcement learning agents. These systems use visual observations to navigate interfaces, but rely on an LLM to interpret goals, summarize state histories, or propose high-level exploration strategies. Works such as LLM-Based Deep Reinforcement Learning Agents to Detect Bugs demonstrated that LLM-generated reasoning

can significantly improve exploration efficiency, reduce dead-end actions, and guide agents toward semantically meaningful interface states [31].

These transitional systems blur the boundary between perception, action, and language reasoning, serving as a precursor to fully multimodal GUI-test agents.

Although not GUI-specific, broader work on LLM-based software testing provided essential context for this phase. Papers such as Software Testing with Large Language Models highlighted how LLMs can transform test-generation workflows, requirement interpretation, debugging assistance, and scenario-based reasoning across software engineering tasks [22]. These insights accelerated interest in applying LLMs to GUI testing, where similar challenges – semantic ambiguity, multi-step workflows, and specification grounding – are especially pronounced.

The shift toward transformer-based and multimodal reasoning represents a redefinition of what automated GUI testing can achieve. Earlier approaches focused on visual fidelity – whether an interface looked correct. In contrast, multimodal systems address semantic fidelity – whether the interface's behavior aligns with user requirements, design specifications, and expected workflows. By enabling scenario generation, semantic validation, and language-guided exploration, these methods move automated GUI testing toward human-like comprehension, setting the stage for the next generation of testing frameworks.

4.1.7. Summary

Across these six phases, vision-based GUI testing has evolved from simple screenshot matching to integrated perception–reasoning systems capable of semantic interpretation and autonomous exploration. Each methodological transition was driven by the limitations of earlier approaches and enabled by advances in computer vision, machine learning, and multimodal AI. This historical progression motivates the unified taxonomies and an evaluation analysis presented in the following subsections and is visually summarized in Figure.

4.2. Taxonomy of Vision-Based GUI Testing Methods

Vision-based GUI testing encompasses several distinct methodological families that differ in how they represent visual information, extract structure, model interactions, and incorporate semantic reasoning. Unlike the historical analysis in Section 4.1, which describes the temporal evolution of these techniques, the purpose of this taxonomy is to present a structural organization of the field based solely on computational principles. Each category below outlines the defining characteristics, representative methodological patterns, and conceptual boundaries that distinguish one family from another.

4.2.1. Classical Image-Processing and Template-Matching Methods

Classical image-processing approaches form the most direct and deterministic family of vision-based GUI testing techniques. These methods operate purely on rendered pixels using template matching, edge detection, keypoint descriptors, region growing, color histograms, or threshold-based comparisons, without any learned representations. Such techniques treat GUI analysis as a pattern-matching task rather than a statistical learning problem, relying on predefined visual heuristics to identify components or detect visual changes.

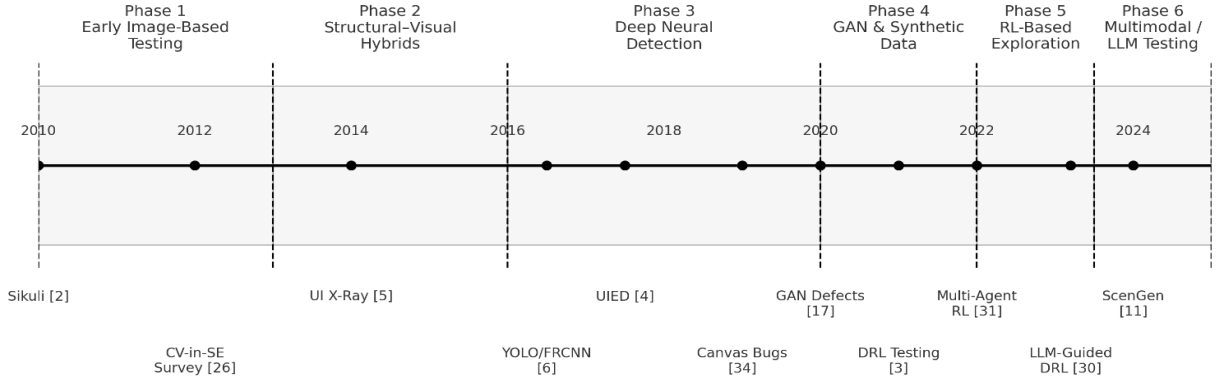

Figure. **Historical Development of Vision-Based GUI Testing (2010–2025)**
***Source*: compiled by the authors**

This family is exemplified by early GUI-testing tools that match screenshot snippets to on-screen regions [2], pixel-difference and perceptual-contrast techniques used for detecting visual inconsistencies in HTML5 canvas applications [24], and heuristic GUI-analysis frameworks that employ edge maps or connected components to extract interface structure [2], [32]. Pattern-oriented GUI-testing approaches also fall into this category when they use rule-based visual recognition to match layout motifs or structural configurations [12]. Broader surveys of computer vision in software engineering further document how such classical techniques – contour detection, Hough transforms, connected-component labeling, and related heuristics – have historically been applied to GUI parsing and screen-flow reconstruction tasks [14], and systematic mapping studies confirm that many pre–deep-learning GUI-testing approaches relied heavily on deterministic visual features rather than learned models [13].

Classical image-processing methods are distinguished by their explicit feature engineering and high interpretability: decisions follow directly from handcrafted descriptors or pixel relations rather than latent learned features. Because they operate on raw screenshots, they are inherently cross-platform and do not depend on DOM structures or accessibility metadata [2]. At the same time, their deterministic nature makes them sensitive to minor visual variations – changes in resolution, theme, icon style, rendering quality, or font smoothing can break template matches or disrupt edge-based segmentation [24]. These methods also lack semantic understanding, as they can compare pixels or shapes but cannot infer component roles or behavioral expectations. As a result, classical techniques are effective for straightforward regression comparison and controlled-environment detection tasks but do not generalize reliably to stylistically diverse, densely populated, or dynamic modern interfaces [14].

4.2.2. Deep Neural Networks for GUI Perception

Deep neural models form a second major family of techniques characterized by learned visual representations. CNNs and modern object detectors (e.g., Faster R-CNN, SSD, YOLO) learn discriminative visual features directly from data, enabling more reliable detection of buttons, text fields, menus, icons, and other interface components across heterogeneous GUI styles [7], [8], [17], [4]. Such models allow for flexible element categorization and bounding-box localization, substantially improving recognition stability relative to handcrafted approaches.

Hybrid deep-learning pipelines also belong to this category when traditional segmentation is combined with learned classification. A prominent example is UIED, which merges edge-based structural partitioning with CNN-based component recognition [5]. Additional deep detectors target specific domains such as mobile GUIs [17], game UIs [25], and display-error localization [4].

Deep neural models are distinguished by their ability to generalize across visual variability and learn GUI-specific features automatically. Unlike classical methods, they are not constrained to handcrafted descriptors and perform consistently even when themes, resolutions, or graphical styles vary. They remain limited, however, by their reliance on annotated training data and challenges with extremely dense or visually cluttered screens, which motivates complementary approaches such as generative augmentation.

4.2.3. Transformer-Based Vision Models

ViTs extend deep perception with global self-attention, enabling models to capture long-range relationships and higher-order structural dependencies across entire interface layouts. Whereas CNNs focus primarily on local receptive fields, ViTs jointly model interactions among spatially distant elements, making them well suited for tasks that depend on layout semantics, component alignment, and text–icon relationships.

In GUI-testing contexts, ViTs have been applied for robust component recognition and as visual encoders in multimodal pipelines, particularly when interfaces vary widely in design style or screen composition [19], [20], [21]. Their distinguishing feature is the representation of GUI images as sequences of patches processed through attention layers, supporting more flexible global reasoning than convolutional architectures.

ViTs thereby provide a conceptual bridge between pure visual perception and multimodal semantic interpretation, functioning either as standalone detectors or as upstream encoders for language-guided systems.

4.2.4. Generative and Synthetic-Data Models

Generative models, particularly GAN-based architectures, constitute a technique family focused on data creation and augmentation rather than perception or interaction. They address the frequent scarcity of labeled GUI datasets by synthesizing visually plausible screen variations and defect conditions.

GANs have been used to generate misaligned components, corrupted assets, occlusions, or other GUI anomalies, improving the robustness of downstream classifiers and detectors under distribution shifts [26], [27]. Generative augmentation is especially relevant in domains such as mobile apps and games, where visual diversity is high and annotation is costly [4], [25]. Broader hybrid generative pipelines also assist with noise robustness and domain adaptation [28], [33].

This family differs from other technique groups in that generative models do not interpret GUI content directly; instead, they serve as auxiliary mechanisms that enhance the generalization capacity of perception models.

4.2.5. Reinforcement-Learning and Autonomous Navigation Agents

RL approaches define a separate technique family that models GUI testing as a sequential decision-making process. Unlike perception-focused models, RL agents generate interaction sequences by observing screenshots, selecting actions, and receiving rewards tied to state coverage or task completion.

Representative RL approaches include DQN-based agents for web and Android navigation [3], [30], semantic representation – enhanced RL for more meaningful state abstraction [35], and platform-independent frameworks that use image embeddings to generalize across applications [1].

The distinguishing feature of RL methods is their ability to coordinate perception and action over multiple steps. They consume visual encodings produced by CNNs or ViTs but operate at the interaction level rather than the recognition level. This positions RL as a complement to perception techniques, enabling end-to-end testing workflows rather than static screen analysis.

4.2.6. Vision–Language and Multimodal Reasoning Models

VLMs and LLMs integrated with visual encoders form an emerging methodological family that supports semantic interpretation, specification grounding, and scenario generation. These systems combine vision modules (CNNs or ViTs) to encode screenshots, and language models to interpret requirements, reason about interface behavior, or generate test sequences.

Recent work demonstrates multimodal test-generation frameworks that align natural-language specifications with GUI states [19], [20], as well as LLM-driven agents that derive exploration strategies or identify semantic inconsistencies [21], [22], [31], [36].

What distinguishes this family is its ability to operate at a conceptual level, mapping textual intent to visual states. Multimodal systems are therefore situated above perception and RL pipelines, extending GUI testing from visual correctness to semantic correctness.

4.2.7. Hybrid, Structural, and Invariant-Based Techniques

Several approaches do not fall cleanly into a single computational paradigm and instead integrate structural heuristics, symbolic rules, or invariant extraction with perceptual analysis.

Hybrid methods such as UIED combine classical heuristics (layout segmentation, edge detection) with deep classifiers [5], while invariant-based techniques infer stable visual or structural constraints to serve as test oracles [37]. Pattern-based GUI testing methods also belong here when they use structural templates to assess layout conformance [12]. Broader CV-in-SE techniques that reconstruct navigation flows or screen hierarchies provide additional cross-disciplinary contributions [14].

These approaches are distinguished by their emphasis on structural relationships rather than pixel-level features or fully learned representations, forming a complementary category that intersects with perception, layout analysis, and semantic validation.

4.2.8. Summary

The technique families summarized above differ not only in computational foundations but also in how they contribute to GUI testing – ranging from deterministic pixel-level methods [2], [12] to learned visual perception [5], [7], semantic multimodal reasoning [21], [22], [31], [36], interaction-driven exploration [1], [3], [30], and structurally guided hybrid approaches [5], [37].

To consolidate these distinctions and clarify how each family aligns with specific testing needs, Table provides a concise comparison of their core capabilities, inherent limitations, and typical application domains. This table serves as a bridge between the taxonomy and the subsequent analysis of strengths, constraints, and methodological implications.

Table. **Comparison of Technique Families in Vision-Based GUI Testing**

Technique Family	Core Capabilities	Core Limitations
Classical Image Processing & Template Matching	– Very interpretable; decisions follow directly from pixel or shape matching. – No labeled data required; works immediately across platforms. – Effective for simple regression checks, canvas-rule validation, or pixel-fidelity inspection. – Robust in controlled environments with stable layouts.	– Extremely sensitive to visual variation (resolution, theme, anti-aliasing) [24]. – No semantic understanding (detects pixels, not component roles) [14]. – Poor generalization to dense or stylistically diverse UIs [13]. – Cannot model state transitions or interactions.
Deep Neural Networks	– Learn discriminative features directly from GUI screenshots. – High accuracy for element detection, classification, and visual parsing [7, 8]. – More robust to layout/theme variations than handcrafted methods. – Scales well to large, visually diverse datasets.	– Require labeled GUI datasets, which remain scarce [26]. – Performance drops sharply in dense screens with overlapping elements. – Limited capacity for semantic interpretation. – Training/finetuning costs can be high.
Vision Transformers	– Capture global relationships across entire GUI layouts through self-attention. – Superior generalization across unseen design styles. – Stronger layout-level reasoning (alignment, spacing, structural consistency). – Can serve as encoders for multimodal pipelines.	– Require large, diverse datasets to unlock full capabilities. – Computationally expensive relative to CNNs. – Still early in GUI-specific adoption – benchmarks remain limited. – Pure ViT models may still lack deep semantic grounding without language conditioning.
Generative & Synthetic-Data Models	– Expand datasets without manual labeling – critical for GUI scarcity. – Create realistic visual defects (misalignment, occlusion, corrupted assets). – Improve robustness and generalization of CNN/ViT detectors through synthetic diversity. – Enable stress testing under rare or extreme visual conditions	– Do not perform perception or testing by themselves – only support other models. – Risk of low-quality or unrealistic synthetic samples if training is unstable. – Mode collapse can reduce coverage of defect space [26]. – Hard to guarantee semantic fidelity of synthetically generated UIs.
Reinforcement Learning and Autonomous Agents	– Model GUI testing as multi-step interaction, not just screen analysis. – Discover workflows, hidden states, and dynamic transitions [3]. – Reduce reliance on brittle scripted test cases. – Can adapt exploration based on reward structure (coverage, failures).	– Visual state representations can be brittle without strong encoders (CNN/ViT). – Sparse rewards and large state spaces make training difficult [30]. – Struggle to interpret semantic intent without language models. – High computational and configuration cost; difficult to integrate into CI/CD.
Vision–Language Models & Multimodal LLM-Guided Systems	– Align GUI screenshots with natural-language requirements, enabling semantic testing. – Generate test scenarios and interaction plans from specifications. – Identify semantic mismatches (e.g., button exists but contradicts expected role). – Enable human-like reasoning about GUI behaviors.	– Susceptible to hallucination – may invent states or tasks [22]. – Interpretability and reliability remain challenging. – Require multimodal datasets that remain limited in size and coverage. – High computational cost; CI/CD integration remains experimental.
Hybrid, Structural, and Invariant-Based Approaches	– Combine strengths of classical structure analysis with learned semantic perception. – Capture spatial hierarchies, layout rules, or invariants difficult for pure DL models. – Often require less training data due to structural priors. – Useful for oracle generation and layout-validation tasks.	– More complex to design – requires domain heuristics or structural templates. – Limited generalization outside assumed layout patterns. – Performance depends on correctness of structural segmentation or invariant extraction. – Not suitable for end-to-end workflow testing without additional modules.

Source: **compiled by the authors**

4.3. Task-Based Taxonomy of Vision-Based GUI Testing

Vision-based GUI testing spans a diverse set of computational tasks, each addressing a different aspect of how interfaces are perceived, interpreted, or interacted with. Unlike Section 4.2, which groups methods by technique family, this section organizes prior research according to the functional objectives that vision-based systems pursue. These tasks frequently overlap – e.g., detection may support layout parsing or workflow exploration – but they represent distinct problem formulations that structure how datasets, architectures, and evaluation metrics are chosen.

4.3.1. Element Detection and Classification

The most fundamental task in vision-based GUI testing is the identification and categorization of interface components such as buttons, text fields, icons, menus, images, or composite widgets. Deep-learning approaches dominate this task: YOLO, SSD, and Faster R-CNN have been adapted to recognize GUI elements across mobile, web, and desktop interfaces [4], [7], [8], [17]. Hybrid systems like UIED combine structural segmentation with CNN-based classifiers to improve detection accuracy in visually dense layouts [5]. Earlier template-matching approaches [2], [24] also fall into this task category but lack the robustness needed for diverse modern GUIs. Element detection forms the computational basis for many downstream tasks, supplying the primitives required for layout analysis, interaction modeling, and semantic reasoning.

4.3.2. Layout Parsing and Structural Understanding

Beyond locating atomic elements, many GUI-testing workflows require extracting the layout hierarchy, spatial relationships, and structural grouping of interface components. Approaches range from classical edge-based or connected-component segmentation [5], [23] to deep models that infer higher-level structure from learned visual features [5]. Recent vision-transformer encoders support more global reasoning, capturing alignment patterns, spacing regularities, and hierarchical block structure [19], [20], [21]. Structural understanding is essential for validating layout conformance, detecting misalignment defects, reconstructing GUI hierarchies, and generating invariants that serve as oracles [37].

4.3.3. Visual Regression and Rendering Verification

A separate family of tasks focuses on validating visual fidelity across builds or platforms. This includes pixel differencing, perceptual contrast analysis, and learned visual regression checks used to identify missing assets, style inconsistencies, rendering artifacts, and perturbations introduced during deployment [24]. Deep-learning detectors [4] and generative augmentation methods [26], [27] further support this task by improving robustness to variations in theme, lighting, resolution, or graphical noise. Visual regression tasks are primarily comparative: they determine whether the interface still *looks* as expected.

4.3.4. Workflow Exploration and Action Sequencing

GUI testing frequently requires multi-step interaction rather than static screen analysis. RL approaches model GUI usage as a sequential decision process, enabling agents to explore interfaces, discover reachable states, and reproduce workflows without predefined scripts [1], [3], [30]. Semantic-enhanced RL variants incorporate additional representation layers to improve state abstraction [35]. Vision-only RL compensates for environments without accessibility metadata, making it suitable for web apps, mobile apps, and enterprise systems where screen transitions are dynamic and context-dependent.

4.3.5. Semantic Testing and Specification

Recent multimodal research extends GUI testing from perceptual correctness to semantic correctness – ensuring the interface behaves in a way consistent with natural-language requirements. VLMs and large LLMs paired with vision encoders can interpret GUI screenshots, correlate visual content with textual descriptions, and detect mismatches between interface state and expected behavior [19], [20], [21], [22], [31], [36]. Tasks in this category include natural-language test generation, semantic defect detection, workflow validation against specifications, and question-answering about GUI functionality. Unlike visual regression or detection tasks, semantic testing evaluates *what the GUI means* and whether it fulfills its functional intent.

4.3.6. Data Augmentation and Synthetic Scenario Generation

Generative models define a task category centered on enriching datasets with synthetic screens, defects, perturbations, or layout variations.

GAN-based augmentation pipelines produce realistic misalignments, occlusions, and visual distortions that improve model robustness under distribution shifts [26], [27], [28], [33]. Synthetic data generation is particularly valuable for mobile and game GUIs, where manual annotation is costly and highly variable designs require broader representation coverage. In several pipelines, augmentation is a prerequisite task that improves downstream detection, classification, and semantic-reasoning performance.

4.3.7. Oracle Construction and Rule-Based Validation

A final task category involves constructing oracles – formal or heuristic criteria used to determine whether a GUI behaves correctly. Some approaches derive invariants from visual structure [37], others employ rule-based layout checks as in pattern-oriented testing [12], while hybrid visuo-structural models combine edge maps or block segmentation with learned classifiers to validate interface conformance [5], [14]. Oracle construction differs from detection or regression in that it focuses on correctness criteria themselves, not only on identifying mismatches.

4.4. Domain-Based Taxonomy of Vision-Based GUI Testing

Vision-based GUI testing techniques are applied across a range of software domains that differ in interface structure, interaction patterns, device constraints, and visual complexity. This section groups prior work according to the application domains in which GUI-testing approaches have been evaluated. Domain characteristics often determine which technique families are feasible, which tasks are prioritized, and which datasets or evaluation metrics are appropriate.

4.4.1. Mobile Applications

Mobile interfaces present constraints of limited screen space, high density of interactive elements, gesture-based interaction, and device fragmentation. These characteristics make them a frequent target for visual element detection, layout analysis, and RL-based workflow exploration.

Deep-learning detectors have been widely adapted for mobile UI element recognition [4], [17], enabling bounding-box and widget-type classification from screenshots. Reinforcement-learning agents such as DQN-based Android testers operate directly on mobile screen states, discovering navigation flows and implicit interactions [30], [35]. Synthetic augmentation has also been used to increase representation diversity and stress-test detectors under device-dependent rendering variations [25], [27]. Vision–language models begin to appear in this space as well, enabling requirement-grounded reasoning on mobile UIs [20].

4.4.2. Web Applications

Web interfaces vary widely in structure, from static informational layouts to highly dynamic single-page applications. Their variability makes them well suited for vision-based perception approaches as well as RL-driven interaction models.

Classical template-based techniques are used for stable or highly structured web components [2], whereas deep-learning detectors support consistent cross-browser element identification [7], [8]. RL agents have been applied to web navigation tasks, exploring DOM-less or canvas-heavy environments through screenshot-based state representations [3]. Vision–language models further support specification-grounded testing based on natural-language descriptions of web requirements [19], [21], [36].

4.4.3. Desktop and Enterprise Systems

Enterprise applications often feature dense, multi-panel layouts, high-precision alignment requirements, and long-lived legacy interfaces. These properties make them a strong fit for structural layout parsing, invariant extraction, and transformer-based recognition.

Hybrid structural methods that combine segmentation and learned classification (e.g., UIED) have been applied to dense enterprise UIs where layout hierarchy is critical [5], and invariant-based oracles help validate consistent spacing, alignment, and functional roles [37]. Transformer-based encoders capture global relationships in enterprise dashboards and form-heavy desktop systems [19], [21], while multimodal testing approaches support semantic validation aligned with textual documentation or workflow rules [20], [22].

4.4.4. Game Interfaces and Graphics-Driven Environments

Game interfaces and other graphics-driven environments – whether rendered via HTML5 canvas, custom game engines, or native graphical frameworks – lack traditional DOM structures and often employ highly stylized or dynamically changing visual elements. These characteristics make them especially suitable for pixel-based regression techniques, deep visual detection under

non-standard rendering conditions, and generative augmentation.

Classical computer-vision methods such as pixel differencing, perceptual contrast analysis and handcrafted visual heuristics are used to detect rendering artifacts, missing assets, or visual anomalies in game UIs and graphical overlays [24]. Generative models support this domain by synthesizing diverse visual distortions and defect cases that stress-test detectors under variable lighting, animation, and texture conditions [26], [27]. Deep-learning pipelines – including hybrid detectors that combine structural cues with learned features – are applied to identify UI components embedded within complex or visually dense game scenes [4], [25].

4.5. Strengths, Limitations, and Benchmarks

The diversity of computer-vision techniques, interaction models, and application domains in vision-based GUI testing necessitates a unified synthesis of their shared strengths, recurring limitations, and underlying evaluation practices. Consolidating these insights provides a clearer picture of the methodological maturity of the field and clarifies where empirical evidence supports (or constrains) current approaches. This subsection integrates findings across the reviewed literature – spanning classical CV, deep learning, GAN-based augmentation, RL-driven exploration, and multimodal reasoning–to articulate the cross-cutting characteristics of current methods and the benchmark resources used to evaluate them.

4.5.1. Strengths of Vision-Based GUI Testing Approaches

High perceptual fidelity and resilience to UI changes

By operating directly on rendered pixels rather than DOM trees or accessibility identifiers, vision-based systems remain robust under markup modifications, framework transitions, or cross-platform styling differences. Deep-learning detectors such as YOLO, Faster R-CNN, and SSD provide accurate, real-time localization of GUI widgets – including buttons, text fields, icons, and layout blocks – across diverse environments [4], [7], [8], [17]. Classical image-matching approaches remain valuable for pixel-level regression in highly dynamic graphical environments [2], [24].

Structural parsing and layout robustness

Hybrid and invariant-based approaches, such as UIED [5] and constraint-checking methods [37], provide layout-level reasoning that complements pure detection. They infer grouping, alignment, and hierarchical organization, enabling testers to identify structural defects that pixel-based diffing cannot reliably capture.

Autonomous exploration of multi-state workflows

Reinforcement-learning agents – applied to mobile (DQN-based), web navigation, and hybrid interaction models – enable test scripts to emerge organically from exploration rather than manual specification [1], [3], [30], [35]. These agents are particularly effective in uncovering hidden paths, deep state spaces, and interaction bugs that deterministic pipelines overlook.

Semantic reasoning and alignment with specifications

Multimodal and vision–language systems extend GUI testing beyond perception. ViTs capture global layout dependencies, while LLM-guided and VLM-based agents interpret natural-language requirements, generate scenario-based test steps, and validate semantic correctness [19], [20], [21], [22], [36]. This introduces specification-aware testing unavailable in classical or CNN-based models.

Synthetic data generation for improved robustness

Generative models – including GAN-augmented training corpora and synthetic defect generation – address the long-standing scarcity of labeled GUI data. These techniques improve resilience under distribution shifts, noise, occlusion, and rare defect cases [26], [27], [28], [33].

4.5.2. Limitations and Open Challenges

Scarcity of large, diverse, and standardized GUI datasets

Most studies rely on small project-specific datasets or screenshots collected from limited applications. GUI diversity – across resolutions, themes, OS styles, and design systems – makes generalization difficult. While GANs help mitigate scarcity, they cannot fully substitute for broad real-world training data.

Limited interpretability and explainability of model decisions

CNNs, ViTs, RL agents, and multimodal systems often behave as black boxes. Error analysis is hindered by opaque decision boundaries, especially in VLM-based reasoning pipelines where hallucinated actions and misinterpreted cues remain common failure modes [20], [21], [22].

Cross-application and cross-platform generalization challenges

Even strong detectors degrade when deployed on unseen applications with different iconography, typography, or interaction styles. Transfer learning is helpful but not yet sufficient to guarantee broad coverage.

Scalability constraints and CI/CD integration difficulties

Large vision models and LLM-guided workflows are computationally expensive. RL-based testing requires prolonged state exploration, and multimodal pipelines introduce inference overhead unsuitable for high-frequency CI builds.

Absence of unified, reliable oracles for GUI correctness

Pixel-diffing is noisy; handcrafted invariant rules do not generalize; semantic oracles generated by VLMs remain unreliable. Oracle construction thus remains an open problem across all technique families.

4.5.3. Benchmark Datasets and Evaluation Frameworks

Benchmarking practices in vision-based GUI testing remain highly fragmented, and this fragmentation directly affects how findings across papers can be interpreted and compared. While some datasets support reliable cross-model comparison, many evaluation setups are custom, incompatible, or insufficiently documented. This subsection clarifies the current landscape of benchmarks, the kinds of comparisons that are feasible, and the key areas where standardization is still missing.

GUI Detection and Layout Benchmarks

Several works build on large-scale GUI corpora for detection and layout parsing. UIED trains a ResNet-50 classifier on approximately 90,000 GUI element crops extracted from the Rico mobile app dataset, demonstrating state-of-the-art performance on GUI-element detection and segmentation relative to earlier rule- or heuristic-based methods [5]. Rico itself contains around 66,000 Android screens, providing a broad base for mobile interface analysis and enabling derived datasets for clickable vs. non-clickable components and icon categories [17].

These resources allow direct comparison among models that evaluate on the same Rico-derived detection tasks – for example, different CNN or hybrid pipelines reporting precision/recall/F1 on shared splits [4], [5], [17]. However, many papers still use customized subsets, private annotation schemes, or different label taxonomies, which restrict cross-paper comparison even when the underlying corpus is nominally the same.

Graphics-Driven and Canvas-Based Benchmarks

In graphics-intensive and game-like environments, benchmarks are built around pixel-level regression and visual bug detection. The HTML5 canvas study on visual bugs in games uses a dedicated set of game screens and rendered frames to evaluate pixel-diff and perceptual metrics for detecting rendering anomalies and missing assets [24]. Within this domain, comparisons are meaningful across approaches that reuse the same canvas/game dataset, but such datasets are rarely adopted outside their original studies. As a result, these benchmarks primarily support within-domain comparison and offer limited leverage for evaluating general-purpose GUI-testing pipelines.

Synthetic and GAN-Augmented Benchmarks

GAN-based augmentation studies focus on constructing more diverse defect datasets from limited real data [26], [27]. For instance, intraclass augmentation is used to inflate the size and variability of small defect datasets, demonstrating improved robustness and generalization of defect detectors compared to training on the original data alone [27]. These synthetic benchmarks are effective for within-paper ablation and sensitivity studies (e.g., “with vs. without GAN-augmented data”), but comparability across papers is weak because each work typically defines its own generation process, visual style, and evaluation split.

Reinforcement Learning Interaction Environments

RL-based GUI testing is evaluated using bespoke interaction environments and application sets. DinoDroid, for example, is tested on 64 open-source Android applications, allowing the authors to zcompare coverage and bug-finding effectiveness against random-testing baselines such as Monkey within a consistent mobile benchmark suite [30]. WebRLED evaluates deep RL exploration on a set of real-world web applications with repeated experimental runs and statistical comparison against competing tools [3].

These environments support solid intra-study comparison (e.g., “our DRL agent vs. three baseline tools on the same apps”), but cross-study comparison is difficult because each work chooses different applications, reward structures, and state encodings. Even when benchmarks are publicly released, differences in configuration and experimental protocol limit the direct comparability of reported coverage or defect-detection rates.

Multimodal and VLM-Based Benchmarks

Recent multimodal works construct datasets that pair GUI screenshots with textual instructions, scenarios, or requirements to evaluate semantic grounding and test-generation correctness [19], [20], [21], [22], [36]. Some benchmarks quantify success in following natural-language tasks (e.g., "navigate to the settings screen and enable Wi-Fi") or in generating GUI-level test cases that match specification constraints. These setups enable comparison among models that share the same screenshot–instruction pairs, but current datasets are small, heterogeneous, and often tailored to a single tool or architecture. There is, at present, no widely adopted multimodal GUI benchmark analogous to Rico for mobile screens.

Unified Evaluation Frameworks

GUI Testing Arena represents one of the first attempts to consolidate evaluation across multiple GUI-testing tasks [18]. Its benchmark dataset combines several sources of GUI defects and interactions, including: 53 real-world applications with recorded display and interaction issues, 79 artificially injected display defects and 26 interaction defects, and 6,421 display / 1,871 interaction defect instances derived from the AitW corpus [18]. This composition allows unified evaluation of detection, defect classification, and exploration behavior under a shared protocol. Within the Arena, models and agents can be compared directly on common metrics and task definitions, making it a key step toward standardized empirical assessment.

However, even GUI Testing Arena does not yet cover all modern paradigms – such as instruction-following VLMs, LLM-guided DRL agents, or cross-platform test orchestration – and thus cannot currently serve as a complete unifying benchmark for the field.

Across the reviewed studies, three conclusions emerge:

– where comparison *is* possible: direct comparison is feasible when models share the same corpus and task definition (e.g., GUI-element detection on Rico-derived datasets, or multiple techniques evaluated under the GUI Testing Arena protocol) [4], [5], [17], [18];

– where comparison is *not* possible: comparisons are unreliable when datasets are custom, synthetic, or partially overlapping, as in many GAN-based augmentation studies, RL environments, and multimodal setups with bespoke GUI–text pairings [3], [21], [22], [26], [27], [30], [35], [36];

– where standardization is missing: the field lacks a benchmark that jointly spans perception, workflow exploration, and semantic specification alignment across mobile, web, desktop, and game interfaces; no canonical multimodal GUI dataset exists, and evaluation protocols for LLM-guided testing agents vary widely; this lack of standardization is a major barrier to cumulative progress and is itself a central finding of the present review.

4.6. Experimental Verification of ViT-, LLM-, and Multimodal Models

Recent advances in Vision Transformers, large language models, and multimodal architectures have introduced new forms of experimental validation that differ substantially from those used for CNN-based detection or RL systems. This subsection synthesizes how these models are evaluated in contemporary research, grounding each methodological pattern in the empirical evidence reported in the reviewed papers.

4.6.1. Verification of Vision Transformers

Vision Transformers have been evaluated primarily on Rico-derived datasets and other annotated mobile-interface corpora. Experiments typically assess mean Average Precision, precision/recall, and IoU-based segmentation accuracy to compare ViTs with CNN baselines such as YOLOv3, YOLOv5, SSD, or Faster R-CNN. Studies consistently report that ViTs achieve higher robustness under global layout variation and long-range spatial dependencies, particularly when small widgets or thin UI elements are involved. Generalization experiments – such as testing on previously unseen applications or interface themes – show that self-attention architectures suffer less degradation under resolution shifts or non-uniform padding than convolution-only models. This aligns with broader findings that ViTs provide superior structural understanding of interface composition, although their adoption remains largely perception-oriented rather than interactive or semantic.

4.6.2. Verification of LLM-Based GUI Testing

Evaluation of LLM-guided GUI-testing systems centers on instruction following, visual grounding, and execution accuracy. ScenGen, for example, reports mean GUI-element localization accuracy improving from 80.79 % to 97.76 %, and in several scenarios reaching 100 % after self-correction mechanisms refine ambiguous element predictions.

These corrections help resolve misidentifications caused by visual ambiguity or icon similarity.

LLM-based reasoning is assessed by comparing the model's decomposed steps against gold-standard test paths. In ScenGen's evaluation, the system successfully completes 84 of 99 multi-step scenario tasks and shows substantially fewer abnormal terminations than earlier LLM-based tools such as ASOT or GPTDroid. This improvement is attributed to multi-agent collaboration and iterative feedback loops, which help the system recover from unexpected interface transitions and heterogeneous screen content.

Execution-grounded evaluation – running the generated steps within a real app or web environment – provides critical insights beyond static reasoning tests. While LLMs may generate plausible chains of thought offline, real execution surfaces additional errors, such as misjudging task completion when visual cues are ambiguous. For example, ScenGen incorrectly detects login completion when success indicators are implicit or weakly expressed in the UI. This highlights the gap between textual reasoning and grounded perception.

Even when successful, LLM-based systems incur measurable computational overhead. ASOT and GPTDroid frequently exceed 11 minutes on complex workflows, whereas ScenGen completes comparable tasks in 4–5 minutes, albeit still slower than classical tools due to multimodal inference costs.

4.6.3. Verification of Multimodal Vision–Language Models

Multimodal systems integrate screenshot analysis with natural-language reasoning, and thus require evaluation protocols that link visual grounding to semantic correctness. Studies test GUI-understanding by pairing screenshots with textual goals or requirements and measuring whether the model selects the correct visual target, interprets task constraints, and maintains internal consistency in its reasoning [19], [20], [21], [22], [36], [38]. These evaluations reveal a common pattern: perception and reasoning are interdependent, and errors in either modality propagate.

Scenario-based test generation experiments show that multimodal models generate coherent workflows only when the visual encoder provides reliable element representations. When visual ambiguity is present – such as icon similarity or clutter – models may hallucinate steps or skip mandatory transitions, similar to the failure cases observed in ScenGen's mislocalized-icon examples. Multimodal systems also struggle with implicit state changes that lack clear visual feedback, mirroring the challenges observed in LLM-only evaluations.

Oracle construction is another emerging evaluation dimension: models are asked to predict expected outcomes based on a screenshot–instruction pair. Early results indicate that such semantic oracles remain brittle, as models may overgeneralize or misinterpret UI context, especially when dealing with atypical layouts or non-standard design systems.

Across studies, perceptual–semantic consistency tests suggest that multimodal systems perform well when inputs are clean and conventional but degrade under stylistically diverse or highly interactive UIs.

4.6.4 Verification of Multimodal Vision–Language Models

Quantitative evidence across papers reveals distinct strengths and constraints. ViTs consistently outperform CNNs in structural perception, especially for small components and complex layouts [17], [4]. RL systems provide measurable improvements in workflow exploration: in DinoDroid, the DQN agent selects expected events 85.2 % of the time, compared with 51.6 % for random exploration, while leveraging child-state features increases optimal path selection to 81.2 %, significantly surpassing random baselines. Coverage gains also demonstrate practical benefits: adding login-handling raises coverage on k9-mail from 7 % to 40 % within one hour.

Meanwhile, multimodal LLM-guided systems demonstrate strong semantic reasoning but remain vulnerable to visual ambiguity, incomplete feedback, and computational overhead. They improve task-completion accuracy dramatically – as illustrated by ScenGen's 84/99 success rate – but still require robust visual cues to avoid hallucinations or premature termination.

4.6.5. Summary

Experimental verification of modern GUI testing models shows a clear methodological divergence. ViTs excel in structural perception; RL agents in exploratory interaction; and multimodal systems in semantic task reasoning. However, each class faces characteristic limitations: ViTs require large annotated corpora, RL agents struggle with heterogeneous app designs and large state spaces, and multimodal models remain sensitive to ambiguous UI signals and incur higher computational costs. Across all categories, execution-grounded evaluation proves essential, as offline metrics consistently overestimate real

performance. These findings reveal both the potential and the current constraints of advanced GUI-testing methods and provide a foundation for the future research directions outlined in the next subsection.

4.7. Planned Applications in Author's Future Work

The insights obtained from this review directly inform the next steps of the author's research agenda. The planned work focuses on contributions that are both scientifically meaningful and feasible for an individual researcher, with an emphasis on incremental benchmark development, lightweight hybrid prototyping, and systematic evaluation protocols.

4.7.1. Contribution to a Standardized Multimodal GUI Benchmark

Rather than constructing a large, fully comprehensive benchmark – which requires coordinated multi-institutional effort – the author aims to contribute modular benchmark components that can expand existing resources.

Future work will focus on:

– creating a small but carefully annotated multimodal subset (e.g., 200–500 screenshots);

– labeling GUI components, clickable targets, and layout relations;

– pairing each screenshot with a set of natural-language task descriptions;

– providing ground-truth action sequences for a small number of representative workflows.

Such a contribution can serve as:

– a *benchmark extension* to Rico- or UIED-derived datasets;

– an initial seed for future large-scale multimodal datasets;

– a fully reproducible reference subset for evaluating vision–language alignment.

Even a small well-annotated multimodal dataset is valuable because no public GUI dataset yet contains aligned screenshots + instructions + execution traces.

4.7.2. Prototyping a Lightweight Hybrid ViT–LLM Pipeline

Future research will focus on building a lightweight prototype that integrates:

– an off-the-shelf Vision Transformer (for element detection or layout embedding);

– a small instruction-following LLM (e.g., 7B range) to generate test steps conditioned on visual features.

The planned prototype will:

– use frozen pretrained models to avoid heavy training workloads;

– explore simple concatenation or projection-based fusion strategies;

– evaluate whether hybridization improves semantic grounding compared to LLM-only baselines.

Expected contributions:

– evidence on whether ViT embeddings reduce hallucinated steps;

– initial guidelines for simple and computationally tractable fusion of perception and reasoning;

– a reproducible reference implementation for the community.

4.7.3. Creation of an Execution-Grounded Evaluation Protocol

Developing a standardized protocol for evaluating GUI-testing models in execution environments.

This protocol will define:

– a set of common interaction **tasks** (e.g., login, open settings, navigate menus);

– expected sequences of GUI actions for each task;

– rules for scoring success, failure, partial correctness, and step deviations;

– an open-source evaluation harness reusable by other researchers.

Such a protocol addresses a major gap identified in this review: the lack of shared evaluation conventions for LLM-, ViT-, and multimodal GUI-testing pipelines.

Even a minimal protocol improves comparability across studies and provides a foundation for future standardization efforts.

4.7.4. Expected Outcomes and Research Contributions

Through the incremental work described above, the author expects to achieve several valuable contributions:

– a publicly shared multimodal mini-benchmark, enabling reproducible experiments;

– a baseline hybrid ViT–LLM prototype, demonstrating measurable improvements in visual grounding;

– a standardized execution-based evaluation protocol, reducing fragmentation across GUI-testing studies;

– comparative empirical analysis of existing and prototype models using the same evaluation pipeline;

– open-source scripts and annotation tools to facilitate community adoption.

These contributions directly address the methodological gaps identified previous sections – especially the scarcity of multimodal datasets, lack of standardized evaluation, and the need for integrated perception–reasoning.

5. DISCUSSION OF RESULTS

The results of this systematic review reveal a rapidly evolving field in which computer vision, deep learning, reinforcement learning, and multimodal reasoning converge to reshape automated GUI testing. While earlier sections documented historical development, taxonomies, empirical strengths, limitations, and emerging verification practices, the present discussion interprets these results from a broader methodological and conceptual perspective. Several themes emerge that clarify both the technological trajectory and the underlying scientific challenges of vision-based GUI quality assurance.

5.1. Convergence toward Human-Like Perceptual and Semantic Testing

One of the most prominent patterns across the reviewed literature is the gradual shift from low-level perceptual checks toward increasingly **human-like analysis of interface semantics**. The field has moved from template-based matching and handcrafted CV features to CNNs, then to ViTs, and finally to VLMs and LLM-guided testing.

Each transition reflects an expansion in representational depth:

– CNNs improved perceptual fidelity but lacked semantic abstraction;

– ViTs enabled global structural understanding of layout and component relationships;

– LLM-guided and multimodal agents introduced reasoning about goals, workflows, and specifications.

This trajectory suggests that GUI-testing research is now defined less by the ability to *see* interface elements and more by the ability to *interpret* them. The integration of textual instructions, scenario descriptions, and domain knowledge into testing flows marks a conceptual shift from static artifact-based QA to cognitive testing systems capable of aligning observed behavior with intended system functionality.

5.2. Fragmentation in Evaluation Practices and Its Implications

The Results section demonstrated that evaluation setups across studies are highly heterogeneous, especially for RL-based and multimodal models. From a methodological standpoint, this fragmentation implies that the field currently lacks a stable empirical foundation: detection models can be compared when they share datasets, but RL agents use divergent environments and reward functions, and VLM/LLM-based systems rely on custom instruction sets and private benchmarks.

This heterogeneity creates a dual challenge: it limits the reproducibility of published findings and complicates the accumulation of evidence across studies. The absence of standardized multimodal datasets – particularly those pairing GUI screens with textual goals and execution traces – further restricts objective evaluation of semantic groundedness. As a result, even promising models cannot yet be meaningfully compared across independent research groups.

The implications are clear: methodological standardization is a prerequisite for scientific progress. The recent introduction of GUI Testing Arena is encouraging, but its scope remains limited to perception-oriented tasks. Broader multimodal benchmarks will be essential for evaluating the next generation of GUI-testing systems.

5.3. Interdependence of Perception, Exploration, and Reasoning

The results also reveal strong interdependencies across the three major capability dimensions of GUI testing:

1) perception (e.g., element detection, layout understanding);

2) exploration (e.g., RL-based navigation, state-space traversal);

3) semantic reasoning (e.g., instruction following, scenario consistency).

Failures in any one layer propagate to the others. Multimodal systems hallucinate steps when visual grounding is weak; RL agents stall when perception fails to detect interactive components; ViT-based detectors correctly parse layout but cannot reason about user intent. This interdependence suggests that incremental improvements in isolated components will bring diminishing returns unless paired with integrated perception–reasoning architectures.

Consequently, the most promising research direction indicated by the review is not the

refinement of single-method families but the design of hybrid pipelines that combine complementary strengths – e.g., ViT-based perception with LLM-based reasoning – while mitigating their shared limitations through closed-loop execution feedback.

5.4. Practical Feasibility and Emerging Trends in Real-World Adoption

The results highlight an emerging tension between scientific advances and practical usability. ViTs and multimodal models offer unprecedented accuracy and semantic richness yet introduce substantial computational and engineering overhead. RL-based systems achieve meaningful coverage improvements but require extensive training time, environment setup, and reward tuning. Meanwhile, classical tools such as Applitools or pixel-differencing remain dominant in industry due to their simplicity and reliability, even though they lack semantic depth.

This divergence suggests that real-world adoption of advanced GUI-testing pipelines depends not only on accuracy or reasoning capability but also on pragmatic constraints such as:

– inference latency;

– integration into CI/CD pipelines [39];

– reproducibility of results;

– ease of configuring test environments.

The field's future growth will therefore require models that balance cognitive capabilities with operational practicality – a challenge well suited to lightweight hybrid architectures and modular benchmark contributions.

5.5. Broader Implications for the Research Community

The synthesis of results indicates several broader implications for GUI-testing research:

– the field is transitioning from meaningfully contribution, where models are expected not only to detect defects but also to explain, reason, and validate behavior;

– standardization gaps impede progress, making community-driven benchmarks an urgent priority;

– integration of perception, reasoning, and execution appears essential for achieving robust, real-world performance;

– cross-platform generalization remains an open scientific challenge due to variability in visual design systems, interaction patterns, and layout conventions;

– the increasing use of LLM-based models aligns GUI testing with broader trends in explainable AI, multimodal learning, and user-centric software verification.

Together, these implications position vision-based GUI testing as a multidisciplinary research domain at the intersection of computer vision, natural-language processing, software engineering, and human–computer interaction.

Overall, the results demonstrate that vision-based GUI testing has achieved substantial progress in perceptual accuracy, exploratory testing, and semantic interpretation, yet still faces critical barriers to standardization, reproducibility, and cross-platform robustness. The field is technologically advanced but methodologically fragmented. Addressing these issues through hybrid architectures and systematic evaluation frameworks will be essential for establishing a coherent, scalable, and scientifically grounded pathway forward.

CONCLUSIONS

This systematic review traced the evolution of vision-based GUI testing from early image-matching systems such as Sikuli toward contemporary approaches based on deep convolutional detectors, generative augmentation, reinforcement-learning agents, Vision Transformers, and multimodal vision–language models. The historical mapping synthesized in this work highlights a coherent technological trajectory: from brittle perceptual techniques to increasingly integrated pipelines capable of structural understanding, exploratory behavior, and semantic alignment with natural-language specifications.

A structured taxonomy was developed that organizes existing methods along three dimensions – technique, task, and application domain. This taxonomy articulates how classical CV, CNN-based detectors, GAN-driven augmentation, RL-based interaction models, ViTs, and multimodal architectures address distinct categories of GUI-testing problems. It also clarifies the boundaries and complementary strengths of these approaches, showing that no single family of techniques is sufficient for comprehensive GUI quality assurance. Instead, effective systems emerge from combining perceptual robustness, dynamic interaction capabilities, and high-level reasoning.

The review further identified several recurring challenges. Dataset scarcity remains a significant obstacle, particularly the absence of standardized multimodal datasets that pair screenshots with natural-language instructions and action sequences. Evaluation practices are fragmented across studies, limiting comparability and reproducibility. Cross-

platform generalization continues to be difficult due to substantial visual and structural variability in mobile, web, desktop, and game interfaces. These limitations were examined in the chapter *4.6 Experimental Verification*, which consolidated empirical findings from studies on ViTs, RL agents, and multimodal architectures.

The evidence shows that:

– ViTs outperform convolutional models in capturing layout structure and small interactive components, especially under resolution shifts and non-uniform padding;

– RL agents demonstrate measurable gains in workflow exploration, outperforming random baselines and improving coverage when they incorporate child-state features or environment-specific heuristics;

– LLM-based and multimodal systems achieve high task-completion rates in scenario-based testing but struggle when visual feedback is implicit, iconography is ambiguous, or screen transitions are unconventional.

Across all categories, execution-grounded evaluation proved essential: systems that perform well in static or offline analyses often degrade in real application contexts, underscoring the need for standardized, interaction-level benchmarks.

Despite these challenges, the analysis also reveals consistent advances. Deep-learning detectors provide robust high-fidelity perception; generative models improve data diversity; RL agents achieve meaningful gains in workflow exploration; and multimodal reasoning introduces semantic interpretability and specification alignment. These capabilities collectively suggest that vision-based GUI testing is transitioning toward more "human-like" forms of analysis, where models integrate perception, behavior, and semantic understanding.

As outlined in chapter *4.7 Planned Applications*, several targeted research directions are both feasible and strategically aligned with the gaps identified in this review. These include contributing modular multimodal benchmark subsets, developing a lightweight hybrid ViT–LLM prototype to examine perception–language coupling, and designing an execution-grounded evaluation protocol to improve cross-study comparability. Such contributions address the core limitations highlighted across the surveyed literature and support the field's movement toward methodological consolidation.

In summary, vision-based GUI testing has progressed into a mature yet still rapidly evolving research domain. Its future development will likely depend on hybrid perceptual–semantic architectures, richer multimodal benchmarks, and unified evaluation practices. The insights synthesized in this review provide a consolidated reference point for these efforts and chart a clear research trajectory toward more robust, interpretable, and generalizable GUI-testing systems.

REFERENCES

1. Yu, S., Fang, C., Li, X., Ling, Y., Chen, Z. & Su, Z. "Effective, platform-independent GUI testing via image embedding and reinforcement learning". *ACM Trans. Softw. Eng. Methodol*. 2024; 33 (7): 175, https://www.scopus.com/record/display.uri?eid=2-s2.0-85206217027. DOI: https://doi.org/10.1145/3674728.

2. Yeh, T., Chang, T.-H. & Miller, R. C. "Sikuli: using GUI screenshots for search and automation". In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology (UIST '09). Association for Computing Machinery*. New York, NY, USA. 2009. p. 183–192, https://www.scopus.com/record/display.uri?eid=2-s2.0-70450179821.
DOI: https://doi.org/10.1145/1622176.1622213.

3. Gu, Z., Liu, C., Wu, G., Zhang, Y., Yang, C., Liang, Z., Chen, W. & Wei, J. "Deep reinforcement learning for automated web GUI testing". *arXiv*. 2025. DOI: https://doi.org/10.48550/arXiv.2504.19237.

4. Pan, X., Huan, Z., Li, Y. & Cao, Y. "Enhancement of GUI display error detection using improved faster R-CNN and multi-scale attention mechanism". *Applied Sciences*. 2024; 1 (3): 1144, https://www.scopus.com/record/display.uri?eid=2-s2.0-85196321740.
DOI: https://doi.org/10.3390/app14031144.

5. Xie, M., Feng, S., Xing, Z., Chen, J. & Chen, C. "UIED: A hybrid tool for GUI element detection". In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020). Association for Computing Machinery*. New York, NY, USA. 2020. p. 1655–1659, https://www.scopus.com/record/display.uri?eid=2-s2.0-85097192553. DOI: https://doi.org/10.1145/3368089.3417940.

6. Chen, C., Pistoia, M., Shi, C., Girolami, P., Ligman, J. W. & Wang, Y. “UI X-Ray: interactive mobile UI testing based on computer vision”. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces (IUI '17). Association for Computing Machinery*. New York, NY, USA, 2017. p. 245–255, https://www.scopus.com/record/display.uri?eid=2-s2.0-85016479948.
DOI: https://doi.org/10.1145/3025171.3025190.

7. Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. “You only look once: unified, real-time object detection”. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA. 2016. p. 779–788, https://www.scopus.com/record/display.uri?eid=2-s2.0-84986308404.
DOI: https://doi.org/10.1109/CVPR.2016.91.

8. Chen, J., Xie, M., Xing, Z., Chen, C., Xu, X. & Zhu, L. “object detection for graphical user interface: old fashioned or deep learning or a combination?” In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020). Association for Computing Machinery*. New York, NY, USA. 2020. p. 1202–1214, https://www.scopus.com/record/display.uri?eid=2-s2.0-85094213888.
DOI: https://doi.org/10.1145/3368089.3409691.

9. Khaliq, Z., Farooq, S. U. & Khan, D. “Artificial Intelligence in Software Testing: Impact, Problems, Challenges and Prospect”. *arXiv*. 2022. DOI: https://doi.org/10.48550/arXiv.2201.05371.

10. Ricca, F., Marchetto, A. & Stocco, A. “A multi-year grey literature review on ai-assisted test automation”. *arXiv*. 2024. DOI: https://doi.org/10.48550/arXiv.2408.06224.

11. Escalante-Viteri, A. & Mauricio, D. “Artificial intelligence in software testing: a systematic review of a decade of evolution and taxonomy”. *Algorithms*. 2025; 18 (11): 717. DOI: https://doi.org/10.3390/a18110717.

12. Kousar, A., Khan, S. U. R., Mashkoor, A. & Iqbal, J. “A systematic literature review on graphical user interface testing through software patterns”. *IET Software*. 2025; 2025 (1): 9140693, https://www.scopus.com/record/display.uri?eid=2-s2.0-105002586107.
DOI: https://doi.org/10.1049/sfw2/9140693.

13. Nie, L., Said, K.S., Ma, L., Zheng, Y. & Zhao, Y. “A systematic mapping study for graphical user interface testing on mobile apps”. *IET Software*. 2023; 17 (3): 249–267, https://www.scopus.com/record/display.uri?eid=2-s2.0-85149362528.
DOI: https://doi.org/10.1049/sfw2.12123.

14. Bajammal, M., Stocco, A., Mazinanian, D. & Mesbah, A. “A survey on the use of computer vision to improve software engineering tasks”. In *IEEE Transactions on Software Engineering*. 2022; 48 (5): 1722–1742, https://www.scopus.com/record/display.uri?eid=2-s2.0-85096090338.
DOI: https://doi.org/10.1109/TSE.2020.3032986.

15. Dwarakanath, A., Dubash, N. & Podder, S. “Machines that test software like humans”. *arXiv*. 2018. DOI: https://doi.org/10.48550/arXiv.1809.09455.

16. Chaikovskyi, M. & Malakhov, E. “Improving software development quality assurance process with computer vision techniques”. *Proceedings of Scientific Conference on Modern Achievements of Science and Education*. 2025. – Available from: https://iftomm.ho.ua/pages/mase-2025.php.

17. Degaki, R., Colonna, J., Lopez, Y., Carvalho, J. & Silva, E. “Real-time detection of mobile GUI elements using convolutional neural networks”. In *Proceedings of the Brazilian Symposium on Multimedia and the Web (WebMedia '22). Association for Computing Machinery*. New York, NY, USA. 2022. p. 159–167, https://www.scopus.com/record/display.uri?eid=2-s2.0-85139782490.
DOI: https://doi.org/10.1145/3539637.3558044.

18. Zhao, K., Song, J., Sha, L., Shen, H., Chen, Z., Zhao, T., Liang, X. & Yin, J. “GUI Testing Arena: A unified benchmark for GUI element detection”. *arXiv*. 2024. https://doi.org/10.48550/arXiv.2412.18426.

19. Wang, S., Wang, S., Fan,Y., Li, X. & Liu, Y. “Leveraging large vision-language model for better automatic web GUI testing”. *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2024, https://www.scopus.com/record/display.uri?eid=2-s2.0-85215514657.
DOI: https://doi.org/10.48550/arXiv.2410.12157.

20. Yu, S., Ling, Y., Fang, C., Zhou, Q., Chen, C., Zhu, S. & Chen, Z. “LLM-guided scenario-based GUI testing”. *arXiv*. 2025. DOI: https://doi.org/10.48550/arXiv.2506.05079.

21. Liu, Z., Chen, C., Wang, J., Chen, M., Wu, B., Che, X., Wang, D. & Wang, Q. “Make LLM a testing expert: bringing human-like interaction to mobile GUI testing via functionality-aware decisions”. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24). Association for Computing Machinery*. New York, NY, USA. 2024. p. 1–13. DOI: https://doi.org/10.1145/3597503.3639180.

22. Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S. & Wang, Q. “Software Testing wth Large Language Models: survey, landscape, and vision”. *IEEE Trans. Softw. Eng*. 2024; 50 (4): 911–936. DOI: https://doi.org/10.1109/TSE.2024.3368208.

23. Chang, T.-H., Yeh, T. & Miller, R. C. “GUI testing using computer vision”. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI '10). ACM*. New York, NY, USA. 2010. p. 1535–1544. https://doi.org/10.1145/1753326.1753555.

24. Macklon, F., Taesiri, M. R., Viggiato, M., Antoszko, S., Romanova, N., Paas, D. & Bezemer, C.-P. “Automatically detecting visual bugs in HTML5 canvas games”. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22). Association for Computing Machinery*. New York, NY, USA. 2023. p. 1–11, https://www.scopus.com/record/display.uri?eid=2-s2.0-85139900652. DOI: https://doi.org/10.1145/3551349.3556913.

25. Wu, X., Jaming, Y., Ke, C., Xiaofei, X., Yujing, H., Ruochen, H., Lei, M. & Jianjun, Z. “Widget detection-based testing for industrial mobile games”. *IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Melbourne, Australia. 2023. p. 173–184, https://www.scopus.com/record/display.uri?eid=2-s2.0-85171776069.
DOI: https://doi.org/10.1109/ICSE-SEIP58684.2023.00021.

26. Sivakumar, V. M., Musham, N. K., Ganesan, S. & Ranganathan, P. “CNN-based image analysis for detecting UI defects in mobile apps: Advancing automated GUI testing with GANs”. *International Journal of Advanced Multidisciplinary Research and Studies*. 2024; 4: 2439–2446. DOI: https://doi.org/10.62225/2583049X.2024.4.6.4416.

27. Sampath, V., Maurtua, I., Martín, J. J., Iriondo, A., Lluvia, I. & Aizpurua, G. “Intraclass image augmentation for defect detection using generative adversarial neural network..” *Sensors*. 2023; 23 (4): 1861, https://www.scopus.com/record/display.uri?eid=2-s2.0-85148972522.
DOI: https://doi.org/10.3390/s23041861.

28. Bodyanskiy, Y., Chala, O., Filatov, V. & Pliss, I. “Neo-Fuzzy Radial-Basis function neural network and its combined learning”. *Studies in Computational Intelligence, Springer*. 2023, https://www.scopus.com/record/display.uri?eid=2-s2.0-85169779711.
DOI: https://doi.org/10.1007/978-3-031-37450-0_19.

29. Shpinareva, I. M., Yakushina, A. A., Voloshchuk, L. A. & Rudnichenko, N. D. “Detection and classification of network attacks using the deep neural network cascade”. *Herald of Advanced Information Technology*. 2021; 4 (3): 244–255. DOI: https://doi.org/10.15276/hait.03.2021.4.

30. Zhao, Y., Harrison, B. & Yu, T. “DinoDroid: Testing Android Apps Using Deep Q-Networks”. *ACM Trans. Softw. Eng. Methodol*. 2024; 33 (5): 122, https://www.scopus.com/record/display.uri?eid=2-s2.0-85195583752. DOI: https://doi.org/10.1145/3652150.

31. Sakai, Y., Tahara, Y., Ohsuga, A. & Sei, Y. “Using LLM-based deep reinforcement learning agents to detect bugs in web applications”. In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence*. 2025; 3: 1001–1008, https://www.scopus.com/record/display.uri?eid=2-s2.0-105001979448. DOI: https://doi.org/10.5220/0013248800003890.

32. Zimmermann, D. & Koziolek, A. “GUI-based software testing: an automated approach using GPT-4 and Selenium WebDrive”. *38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. Luxembourg, Luxembourg. 2023. p. 171–174, https://www.scopus.com/record/display.uri?eid=2-s2.0-85178510464.
DOI: https://doi.org/10.1109/ASEW60602.2023.00028.

33. Rudnichenko, N., Vychuzhanin, V., Otradskya, T., Petrov, I. & Shpinareva, I. “hybrid intelligent system for recognizing biometric personal data”. *Computational & Information Technologies for Risk-Informed Systems: the 3rd International Workshop on Computational & Information Technologies for Risk-Informed Systems (CITRisk 2022) co-located with XXII International scientific and technical conference on Information*

Technologies in Education and Management (ITEM 2022). 2023. p. 74–85, https://www.scopus.com/record/display.uri?eid=2-s2.0-85163828807.

34. Penko, O., Pienko, V. & Malakhov, E. "Multiagent path finding using Dijkstra Algorithm". *2024 IEEE 19th International Conference on Computer Science and Information Technologies (CSIT)*. Lviv, Ukraine. 2024. p. 1–4, https://www.scopus.com/record/display.uri?eid=2-s2.0-105005833136. DOI: https://doi.org/10.1109/CSIT65290.2024.10982587.

35. Vuong, T. & Takada, S. "Semantic analysis for deep Q-network in android GUI testing". *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*. 2019; 2019-July: 123–128, https://www.scopus.com/record/display.uri?eid=2-s2.0-85071395982.
DOI: https://doi.org/10.18293/SEKE2019-080.

36. Shah, V. & Yadav, P. "The future of software testing automation: innovations, challenges, and emerging alternatives". *ICICI*. 2025. p. 1588–1593, https://www.scopus.com/record/display.uri?eid=2-s2.0-105012203459. DOI: https://doi.org/10.1109/ICICI65870.2025.11069964.

37. Yarifard, A. A., Araban, S., Paydar, S., Garousi, V., Morisio, M. & Coppola, R. "Extraction and empirical evaluation of GUI-level invariants as GUI Oracles in mobile app testing". *Inf. Softw. Technol*. 2025; 177 (C), https://www.scopus.com/record/display.uri?eid=2-s2.0-85203545379.
DOI: https://doi.org/10.1016/j.infsof.2024.107531.

38. Poth, A., Rrjolli, O. & Wang, H. "Benchmarking AI-Facilitated UI test-script generation: a reproducible evaluation framework". In *Yilmaz, M., Clarke, P., Riel, A., Messnarz, R., Zelmenis, M., Buce, I.A. (eds) Systems, Software and Services Process Improvement. EuroSPI 2025. Communications in Computer and Information Science*. 2026; 2657, https://www.scopus.com/record/display.uri?eid=2-s2.0-105014497590. DOI: https://doi.org/10.1007/978-3-032-04288-0_9.

39. Zhao, X., Wang, H., Ding, J., Hu, Z., Tian, Q. & Wang, Y. "Augmenting software quality assurance with AI and automation using PyTest-BDD". 2025; 33 (1): 2026, https://www.scopus.com/record/display.uri?eid=2-s2.0-105019502407. DOI: https://doi.org/10.1007/s10515-025-00566-w.

Conflicts of Interest: The authors declare that they have no conflict of interest regarding this study, including financial, personal, authorship or other, which could influence the research and its results presented in this article

Received 06.10.2025
Received after revision 26.11.2025
Accepted 04.12.2025

DOI: https://doi.org/10.15276/aait.08.2025.26
УДК 004.89:004.415.5

Тестування графічних інтерфейсів на основі зору: систематичний огляд досягнень технології комп'ютерного зору у сфері забезпечення якості програмного забезпечення

Чайковський Максим Андрійович[1)]
ORCID: https://orcid.org/0000-0002-7854-7036; chaikovskyi.maksym@stud.onu.edu.ua
Малахов Євгеній Валерійович[1)]
ORCID: https://orcid.org/0000-0002-9314-6062; eugene.malakhov@onu.edu.ua. Scopus Author ID: 56905389000
[1)] Одеський національний університет імені І. І. Мечникова, вул. Дворянська, 2. Одеса, 65082, Україна

АНОТАЦІЯ

Забезпечення надійності та адаптивності графічних інтерфейсів користувача (GUI) стало одним із ключових викликів сучасного контролю якості програмного забезпечення, оскільки сучасні застосунки дедалі більше спираються на візуально насичені, динамічні та кросплатформенні способи взаємодії. Традиційні скриптові та DOM-залежні методи тестування є крихкими та затратними в обслуговуванні, адже навіть незначні зміни в макеті можуть знецінювати значну частину тестових сценаріїв. Ці обмеження стимулювали застосування методів комп'ютерного зору та штучного інтелекту, які дозволяють

оцінювати GUI безпосередньо за зображеннями, забезпечуючи більш гнучку, стійку та “людиноподібну” перевірку інтерфейсів.

Метою цього дослідження є систематизація сучасних підходів до візуального тестування GUI та визначення того, як мультимодальні й мовно-орієнтовані моделі змінюють підходи до забезпечення якості. Огляд виконано як структуроване вивчення академічних і промислових джерел за період 2010–2025 рр. Аналіз узагальнює результати у межах п’яти технологічних напрямів: класичні методи обробки зображень, глибокі нейронні детектори, генеративні моделі для синтетичного збагачення даних, агенти з підкріпленим навчанням для автономного дослідження інтерфейсів та трансформерні мультимодальні системи з інтегрованими великими мовними моделями. Окремо узагальнено експериментальні результати щодо візуальних трансформерів, великих мовних моделе-керованих систем і мультимодальних моделей.

Отримані результати демонструють сталу технологічну еволюцію: від раннього зіставлення зображень – до високоточного виявлення елементів, інтерактивної навігації та семантичного тлумачення GUI через природну мову. Емпіричні дані засвідчують, що візуальних трансформери підсилюють структурне сприйняття, агенти на основі навчання з підкріпленням розширюють покриття сценаріїв, а мультимодальні моделі наближають генерацію тестів до людських принципів мислення. Водночас виявлено низку проблем: нестачу датасетів, зокрема мультимодальних; фрагментованість критеріїв оцінювання; кросплатформенну мінливість та значні обчислювальні витрати мультимодальних моделей.

Загалом візуальне тестування GUI формується як когнітивно орієнтована парадигма забезпечення якості, що поєднує сприйняття, поведінкову взаємодію та семантичне міркування. Новизна дослідження полягає у запропонованій уніфікованій таксономії методів, систематизованому підсумку емпіричних верифікацій та окресленні практичних напрямів майбутніх досліджень. Ці внески мають практичну цінність для підвищення відтворюваності, надійності та методологічної узгодженості в системах тестування графічного інтерфейсу наступного покоління.

Ключові слова: комп’ютерний зір; тестування графічних інтерфейсів; мультимодальний штучний інтелект; забезпечення якості програмного забезпечення; глибоке навчання; великі мовні моделі; навчання з підкріпленням; автоматизація

ABOUT THE AUTHORS

Maksym A. Chaikovskyi - Postgraduate, Department of Mathematical Support of Computer Systems. Odesa I. I. Mechnikov National University, 2, Dvoryanska Str.Odesa, 65082, Ukraine
ORCID: https://orcid.org/0000-0002-7854-7036; chaikovskyi.maksym@stud.onu.edu.ua
Research field: Computer vision; machine learning; software engineering

Чайковський Максим Андрійович - аспірант кафедри Математичного забезпечення комп’ютерних систем. Одеський національний університет імені І. І. Мечникова, вул. Дворянська, 2. Одеса, 65082, Україна

Eugene V. Malakhov - Doctor of Engineering Sciences, Professor, Head of Department of Mathematical Support of Computer Systems. Odesa I. I. Mechnikov National University, 2, Dvoryanska Str. Odesa, 65082, Ukraine
ORCID: https://orcid.org/0000-0002-9314-6062; eugene.malakhov@onu.edu.ua. Scopus Author ID: 56905389000
Research field: Databases theory; metamodeling, the methods of data mining and other data structuring, data processing methods

Малахов Євгеній Валерійович - доктор технічних наук, професор, завідувач кафедри Математичного забезпечення комп’ютерних систем. Одеський національний університет імені І. І. Мечникова, вул. Дворянська, 2. Одеса, 65082, Україна