Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Why Clicking Buttons Is Harder Than Calling APIs: A Comparative Study of Agent Control Surfaces

Deepak Babu Piskala
Principal Applied Scientist
Independent Researcher, Seattle, WA, USA

Kumaresan Durvas Jayaraman
Bharathidasan University
Thiruchirapalli, Tamil Nadu, India

***Abstract* - Two paradigms dominate how AI agents execute real-world digital actions such as purchasing, approving, editing, and filing. UI-grounded agents perceive human-designed interfaces via screenshots, DOM structures, or accessibility trees and act through clicks, keystrokes, and scrolls, achieving maximal surface coverage without requiring custom integrations. API-grounded agents invoke formally defined endpoints with structured schemas, explicit authorization, and deterministic validation, achieving stronger contracts but limited to what providers choose to expose. We compare these paradigms across eight dimensions (reliability, robustness, security, latency, auditability, coverage, cost, and human-in-the-loop design) using a running case study of an enterprise expense-approval workflow spanning Slack, SAP, and a legacy intranet portal to ground every dimension in concrete operational reality. We introduce a taxonomy of Control Surface Contracts that reframes the binary UI-vs-API choice as a spectrum along three axes: contract strength, semantic completeness, and adversarial exposure. Drawing on published benchmarks (WebArena: 14% agent vs. 78% human success; WebShop: 29% vs. 59%), enterprise automation evidence, and security threat models, we find that neither paradigm alone is sufficient. We propose a hybrid decision policy that prefers APIs for high-stakes and regulated actions, uses UI automation as a principled fallback for long-tail coverage, and identify open research problems in grounding, tool governance, and agentic interface engineering.**

***Keywords* - AI agents; tool use; web agents; computer use; API automation; UI automation; RPA; agentic actions**

I. INTRODUCTION

Consider a mundane but consequential enterprise workflow: an employee submits a travel expense report for $4,280 via a Slack message to their manager. An AI agent acting on the manager's behalf must (1) read the Slack message, (2) look up the expense in the company's SAP system, (3) validate it against corporate travel policy, (4) approve or flag it, (5) notify the employee of the decision, and (6) log every step for SOX audit compliance.

This single workflow exposes a fundamental tension in how agents interact with digital systems. Slack offers a rich REST API with OAuth scopes, structured payloads, and deterministic responses: an ideal surface for programmatic action. But the company's SAP deployment is a decade-old on-premises installation whose approval screen is accessible only through a Java-based GUI with no exposed API. The agent faces a choice of control surfaces: use a formal API where one exists, or perceive and manipulate the graphical interface as a human would.

This choice, between what we call UI-grounded and API-grounded agent paradigms, is the subject of this paper. UI-grounded agents, exemplified by OpenAI's Operator [1] and Anthropic's computer use [2], perceive interfaces through screenshots and DOM snapshots, then execute clicks, keystrokes, and scrolls. Their premise is powerful: if a human can use it, the agent can too. API-grounded agents, built on tool-calling protocols from OpenAI [3] and Anthropic [4], invoke formally defined endpoints with typed schemas and explicit authorization. Their premise is equally compelling: if there is a contract, the agent should use it.

Neither premise is universally correct. UI agents offer unmatched breadth (they can operate on arbitrary websites and legacy applications), but current benchmarks reveal stark reliability gaps: 14% task success on WebArena versus 78% for humans [5], and 29% on WebShop versus 59% for experts [6]. API agents offer strong contracts, auditability, and performance, but they are constrained to surfaces that providers choose to expose and maintain. In the expense-approval scenario, the agent must use both paradigms within a single workflow.

Contributions. This paper makes four contributions:

1) A systematic, eight-dimensional comparison of UI-grounded and API-grounded agent paradigms, grounded in a running enterprise case study (Section IV).
2) A novel taxonomy of Control Surface Contracts that reframes UI-vs-API as a spectrum along three axes: contract strength, semantic completeness, and adversarial exposure (Section III).
3) A hybrid decision framework with a concrete per-subtask decision flow for selecting the appropriate control surface (Section V).
4) An identification of open research problems at the intersection of HCI, systems, and ML (Section VI).

II. BACKGROUND AND RELATED WORK

A. Agentic Actions

An agentic action is an interaction that alters external digital state (submitting a form, purchasing an item, approving a request, modifying a repository) rather than purely generating text or performing internal reasoning. This aligns

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

with the ReAct paradigm [7], where models iteratively interleave reasoning traces with actions that receive environment feedback. Our scope is digital enterprise and web workflows; we exclude robotics and physical-world manipulation.

B. UI-Grounded Agents

A UI-grounded agent perceives an interface and issues actions at the UI level. The architecture follows an observe–plan–act–verify loop [7, 8]: (1) capture a screenshot and optionally a DOM or accessibility-tree snapshot; (2) compress the observation into actionable candidates (element IDs, bounding boxes, selectors); (3) plan the next step via chain-of-thought or task decomposition; (4) execute a primitive action (click, type, scroll); (5) verify success or request human takeover.

The most general instantiation of this loop uses Vision-Language-Action (VLA) models: large multimodal models that jointly perceive a screenshot (vision), reason about the task in natural language (language), and output a concrete action such as pixel coordinates for a click or a text string to type (action) [9, 10, 11]. The VLA pipeline operates as follows. At each step, the model receives the current screen image (and, optionally, a textual goal or conversation history). A vision encoder (often a ViT backbone or a high-resolution patch embedding) produces spatial features that preserve the layout of UI elements [11, 10]. These visual features are fused with the language instruction through cross-attention or interleaved token sequences, allowing the model to ground the task description in the visual context. The model then decodes an action token sequence that resolves to a specific output: a pixel coordinate pair (x, y) for a click, a typed string, or a scroll direction [9, 8]. This perceive–reason–act cycle repeats in a closed loop until the task goal is satisfied or a termination condition is met.

Pix2Act [9] demonstrates that a Pix2Struct-based model, trained end-to-end on screenshot–action pairs, can learn to control web interfaces and MiniWoB++ tasks purely from pixels, without access to DOM or accessibility metadata. CogAgent [11] scales this approach to a high-resolution (1120×1120) dual-encoder architecture that achieves state-of-the-art performance on both web and mobile GUI benchmarks. ShowUI [10] introduces a visual token selection mechanism that reduces the computational cost of processing full-resolution screenshots by learning which image patches are relevant for UI grounding, cutting token consumption by up to 33% while maintaining grounding accuracy.

Architecturally, UI agents span a spectrum of formality. At one end, pixel-first systems like OpenAI's Computer-Using Agent [1] and Anthropic's computer use [2] operate from screenshots and mouse/keyboard primitives alone. In the middle, hybrid-observation systems like BrowserGym [12] combine pixels with accessibility trees and DOM snapshots. At the structured end, benchmark environments like WebArena [13] abstract actions to element-ID references (click [id]), reducing perceptual ambiguity while remaining realistic.

The community has developed increasingly realistic evaluation environments: MiniWoB++ [14] (toy UI tasks), WebShop [6] (shopping, 29% best model vs. 59% human), Mind2Web [15] (2,000+ tasks across 137 websites), WebArena [13] (812 tasks with execution-based evaluation), VisualWebArena [16] (multimodal web tasks), WorkArena [5] (enterprise ServiceNow tasks via BrowserGym), and Online-Mind2Web [17] (300 tasks on 136 live websites, arguing that offline evaluations overstate progress). TheAgentCompany [18] simulates a full enterprise environment and reports only 24% autonomous task completion for the best baseline.

C. API-Grounded Agents

An API-grounded agent accomplishes tasks by invoking provider APIs through a structured tool-calling protocol. The architecture follows an intent–select–call–validate loop: (1) parse the user goal and relevant context; (2) select a tool and construct structured arguments (JSON-schema-constrained); (3) execute the API call with retries and rate-limit handling; (4) validate the response via server-side schema checks and business-rule enforcement; (5) log the call with a unique identifier for traceability.

OpenAI's function-calling protocol [3] and Anthropic's tool-use protocol [4] formalize this loop. Research systems like Toolformer [19] train models to decide which APIs to call, when, and with what arguments, while Gorilla [20] specifically targets reducing hallucinated API invocations through retrieval-augmented generation.

Integration standards aim to reduce the N × M problem of connecting N agents to M tools. The Model Context Protocol (MCP) [21] introduces an open standard for agent–tool connectivity. Microsoft's Copilot Studio [22] provides prebuilt and custom connectors. OAuth 2.0 [23] underpins least-privilege authorization patterns.

D. Hybrid and Enterprise Automation

In practice, deployments are rarely pure. Microsoft Power Automate [24] explicitly distinguishes API-based "cloud flows" from UI-based "desktop flows," framing UI automation as a bridge for "old apps without APIs." UiPath [25] combines both in unified workflows. ECLAIR [26] reports that traditional RPA adoption is inhibited by 12–18 month setup times, 60% initial accuracy, and multi-FTE maintenance burdens, motivating hybrid approaches.

Fig. 1. Side-by-side reference architectures applied to the expense-approval workflow. Left: the UI-grounded observe–plan–act–verify loop. Right: the API-grounded intent–select–call–validate loop. Dashed arrows indicate the iterative loop.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

III. CONTROL SURFACE CONTRACTS

Most comparisons treat UI and API as a binary choice. We propose a more predictive framing: model the agent's action mechanism as choosing among control surface contracts, i.e., interfaces with varying degrees of formal guarantees, observability, and adversarial exposure.

A. Three Axes

Axis 1: Contract Strength (CS). How formally specified and stable is the interface? High CS: typed API schema with versioning, authentication scopes, and server-side validation (e.g., OpenAI tool calling [3], REST APIs with OpenAPI specs). Medium CS: DOM or accessibility-tree representations with element IDs (e.g., WebArena [13], BrowserGym [12]). Semi-structured but subject to UI drift. Low CS: raw pixels with coordinate-based actions (e.g., Operator [1], Anthropic computer use [2]). Minimal formal constraints.

Axis 2: Semantic Completeness (SC). What fraction of the user-facing task space can the surface express? APIs can be deep but partial, providing only what providers choose to expose. UIs are often complete for end-user tasks but can be blocked by CAPTCHAs, login walls, or hidden flows.

Axis 3: Adversarial Exposure (AE). How much untrusted content can influence the agent's perception and actions? UI agents operate in a high-AE environment: webpage content (including advertisements, user-generated text, and potentially injected instructions) appears directly in the observation channel [27, 28]. API agents face lower but non-zero AE: malicious tool descriptions, compromised MCP servers [21], or poisoned API documentation can mislead tool selection and argument construction [20].

B. Mapping Systems to the Taxonomy

TABLE I

CONTROL SURFACE CONTRACT POSITIONS FOR REPRESENTATIVE SYSTEMS AND EXPENSE-WORKFLOW STEPS

System / Step	CS	SC	AE
Representative systems			
OpenAI tool calling	H	L–M	L
Anthropic tool use	H	L–M	L
MCP ecosystem	H	M	L–M
BrowserGym / WorkArena	M	H	M
WebArena (element-ID)	M	H	M
Operator (pixel-first)	L	H	H
Anthropic computer use	L	H	H
Expense-workflow steps			
Read Slack message (API)	H	H	L
Look up expense SAP (API)	H	M	L
Look up expense SAP (GUI)	L	H	M–H
Approve expense (API)	H	M	L
Approve expense (GUI)	L	H	H
Notify via Slack (API)	H	H	L
Audit log (structured)	H	H	L

CS = Contract Strength, SC = Semantic Completeness, AE = Adversarial Exposure. H = High, M = Medium, L = Low.

The taxonomy reveals a key insight: the optimal paradigm choice is per-subtask, not per-workflow. In the expense scenario, the Slack steps occupy the high-CS, low-AE quadrant (prefer API), while the legacy SAP steps occupy the low-CS, high-AE quadrant (UI is the only option, but demands stronger safeguards). The agent's decision policy should select the lowest-AE, highest-CS surface that remains semantically complete for the subtask.

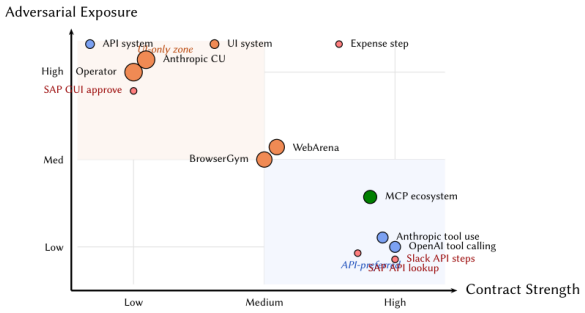

Fig. 2. Control Surface Contracts: systems and expense steps plotted by Contract Strength vs. Adversarial Exposure. Marker size encodes Semantic Completeness. Shaded regions mark the API-preferred and UI-only zones.

IV. MULTI-DIMENSIONAL COMPARISON

We now compare the two paradigms across eight dimensions, using the expense-approval workflow to anchor each analysis. Table II summarizes the findings; the subsections below provide detail.

TABLE II

SUMMARY COMPARISON ACROSS EIGHT DIMENSIONS

Dimension	UI-Grounded	API-Grounded
Reliability	WebShop 29%, WebArena 14%	Tool-selection errors; server-validated [API]
Robustness	Brittle to redesigns, A/B tests	Drift via deprecation; version-pinnable [API]
Security	High AE (prompt injection)	Lower AE; schema-constrained [API]
Latency	Rendering overhead; sequential	One call replaces many steps [API]
Auditability	Screenshots; heavy, privacy risk	Structured call records [API]
Coverage	Any human-usable interface [UI]	Limited to exposed endpoints
Dev Cost	Low upfront, high ongoing	Higher upfront, low ongoing [Depends]
Human-in-Loop	Natural shared control [UI]	Requires custom UX

A. Reliability and Task Success

UI-grounded agents remain far from human-level reliability on realistic tasks. In WebShop, the best model achieves 29% task success versus 59% for expert humans [6]. In WebArena, a GPT-4-based agent achieves 14% versus 78% for humans [5]. SeeAct [8] isolates the bottleneck: with oracle grounding (the correct element is provided), GPT-4V reaches 51.1%; without it, performance drops sharply, confirming that grounding, i.e., mapping high-level intent to the correct UI element, is the dominant failure mode.

API-grounded agents face a different failure profile: tool-selection errors (choosing the wrong endpoint) and argument hallucination (fabricating parameter values). Gorilla [20] demonstrates that retrieval-augmented generation can reduce hallucinated API calls, but the problem persists at scale as tool

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

catalogs grow. Critically, API failures are often explicit: a malformed call returns an error code with a diagnostic message, enabling retry or escalation. UI failures are often silent: the agent clicks the wrong button and proceeds without detecting the error.

B. Robustness and Drift

UI agents drift when interfaces change. Enterprise RPA vendors report that UI redesigns, A/B tests, and localization changes are the primary causes of automation failure [29]. Microsoft Power Automate implements selector fallback and image-based fallback mechanisms [29] precisely because selector breakage is expected, not exceptional. ECLAIR [26] reports that traditional RPA workflows require multiple full-time engineers for ongoing maintenance.

API agents drift when endpoints are deprecated or schemas evolve. However, this drift is typically announced (deprecation notices, changelog entries, version headers) and testable (contract tests in CI/CD pipelines). Version pinning and semantic versioning provide predictable migration paths.

C. Security and Adversarial Threats

Security is the dimension where the paradigms diverge most sharply. UI agents operate in a high-adversarial-exposure environment. The observation channel (screenshots, DOM) contains untrusted content: advertisements, user-generated text, and, critically, potential prompt injections. OWASP ranks prompt injection as the top risk for LLM applications [27]. Greshake et al. [28] demonstrate that indirect prompt injections embedded in web content can compromise real-world LLM-integrated applications. OpenAI's own guidance for Operator recommends sandboxing, domain allowlists, and mandatory human takeover for logins, payment details, and CAPTCHAs [1, 30].

API agents face lower adversarial exposure because the action channel is mediated by schemas and server-side validation. However, they are not immune: malicious tool descriptions in MCP servers can embed hidden instructions [21], and compromised API documentation can mislead argument construction [20]. OpenAI's safety guidance stresses validating tool inputs server-side even when generated by the model [30].

D. Latency and Cost

API-grounded agents dominate on performance. A single API call can replace dozens of UI micro-actions (navigate, scroll, find element, click, wait for render, verify). OpenAI supports parallel tool calls [3], and Anthropic enables programmatic tool orchestration [4], reducing round trips. UiPath explicitly states that API automations "run a lot faster" than UI-based automations once implemented [25]. UI agents incur overhead from browser rendering, sequential interaction steps, and repeated perception cycles. WebArena Verified reports a median evaluation time of 13.2 hours for a full benchmark run [31].

E. Observability and Auditability

API calls naturally produce structured audit records: endpoint, parameters, response payload, HTTP status, timestamp, and unique call ID. These integrate seamlessly with distributed tracing standards like OpenTelemetry [32] and satisfy the evidence requirements of SOC 2 [33], HIPAA [34], and PCI DSS [35]. UI actions require screen recordings, click-coordinate logs, and before/after screenshots for auditability. These records are heavier, harder to query, and create privacy exposure because screenshots may capture sensitive data unrelated to the audited action.

F. Coverage and Long-Tail Reach

This is the dimension where UI-grounded agents dominate. Their premise ("if a human can use it, the agent can too") provides coverage of arbitrary websites, legacy applications, and internal tools that offer no API. OpenAI positions Operator explicitly as working "without requiring custom API integrations" [1]. Microsoft frames UI-based desktop flows as a bridge for "old apps without APIs" [24]. API-grounded agents are limited to surfaces that providers choose to expose. Many enterprise systems, especially legacy on-premises deployments, niche SaaS tools, and internal portals, offer no public API, incomplete APIs, or APIs gated behind enterprise contracts.

G. Development and Maintenance Cost

UI-grounded automation has low initial integration cost: no API contract, no OAuth setup, no endpoint mapping. But maintenance is expensive: selector breakage from UI updates requires continual repair. ECLAIR [26] reports that traditional RPA requires 12–18 months of setup and 60% initial accuracy, with multiple FTEs for ongoing maintenance. API-grounded automation has higher upfront cost: per-provider endpoint mapping, authentication setup, schema alignment, and data-governance configuration. But once built, stable APIs require minimal ongoing maintenance. MCP [21] and connector ecosystems (Copilot Studio [22]) aim to reduce marginal integration costs by standardizing the tool surface.

H. Human-in-the-Loop and User Experience

UI-grounded agents have a natural advantage for shared control: the user interface is the same surface the user understands. A finance manager can watch the agent navigate SAP, intervene if it selects the wrong cost center, and take over for sensitive steps. Operator [1] operationalizes this with a takeover mechanism for logins, payments, and CAPTCHAs. WorkArena [5] explicitly notes that UI assistants are "more amenable to human oversight" than API-based alternatives. API-grounded agents can feel opaque without custom UX to surface their decisions. A user receiving a Slack notification "Expense #4280 approved by agent" has no visibility into the reasoning process unless the system explicitly surfaces a decision trace (e.g., a confirmation card with "Approve / Reject" buttons before execution).

V. HYBRID DECISION FRAMEWORK

Neither paradigm dominates across all dimensions. The comparison in Section IV reveals that API-grounded agents are preferable on six of eight dimensions (reliability, robustness, security, latency, auditability, cost over time), while UI-grounded agents are essential for coverage and offer

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

advantages for human-in-the-loop transparency. This motivates a hybrid-first policy: prefer APIs where available and adequate; use UI automation as a principled, safeguarded fallback.

A. Decision Matrix

TABLE III

DECISION MATRIX: WHEN TO PREFER EACH PARADIGM

Situation	API	UI
High-stakes irreversible actions	Strongly prefer	Only with human approval
Regulated data (HIPAA, PCI, SOX)	Strongly prefer	High risk (screenshot PII)
Long-tail / legacy apps	Not available	Only option
High-throughput workflows	Strongly prefer	Too slow
User transparency needed	Requires custom UX	Natural fit
Prompt-injection threat	Prefer + validation	Sandboxed only

B. Per-Subtask Decision Flow

We propose a concrete decision flow that an agent (or its orchestrator) executes for each subtask. The flow implements the "lowest-AE, highest-CS" principle from Section III. When a subtask arrives, the orchestrator first checks whether a trusted API exists. If yes, it verifies least-privilege authentication is available, then determines whether the action is high-stakes. High-stakes API subtasks route through a human approval gate; others proceed with automatic validation. If no API exists, the orchestrator checks UI policy compliance, sandbox availability, and sensitivity (login/payment/CAPTCHA). Sensitive UI subtasks require user takeover; others proceed with step monitoring. Subtasks that cannot meet safety requirements are escalated to a human.

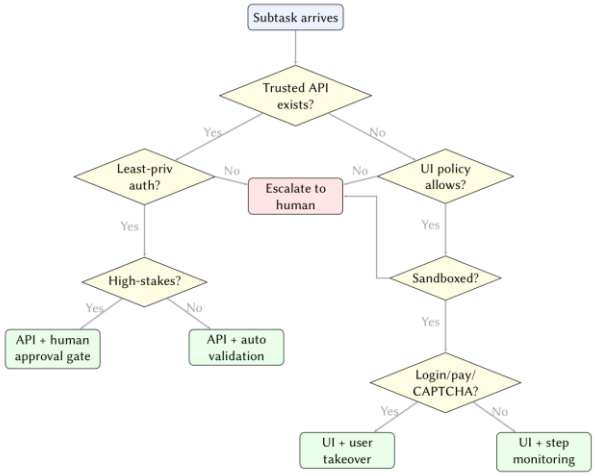

Fig. 3. Per-subtask decision flow for selecting the control surface. The left branch handles API-available subtasks; the right branch handles UI-only subtasks with progressive safeguards. Subtasks that cannot meet safety requirements are escalated to the human.

C. Applying the Framework to the Expense Workflow

We walk the expense-approval workflow through the decision flow:

1. Read Slack message: Trusted API exists (Slack Web API) → least-privilege auth (bot token, channels:history) → not high-stakes → API path with automatic validation.
2. Look up expense in SAP: Two scenarios. Cloud SAP Concur: Trusted API → auth available → not high-stakes (read-only) → API path. Legacy SAP GUI: No API → UI policy allows → sandboxed (isolated VM) → UI path with step monitoring.
3. Approve expense: High-stakes (financial, irreversible). Cloud: API + human approval gate (Slack confirmation card). Legacy GUI: UI + user takeover (manager watches and confirms).
4. Notify employee: Slack API → API path with automatic validation.
5. Audit logging: Structured API logs for API steps; screenshot + action trace for UI steps; both aggregated into a unified audit record.

This walkthrough demonstrates that the framework produces different paradigm selections for different steps within a single workflow, which matches operational reality in hybrid enterprise environments.

VI. OPEN PROBLEMS AND RESEARCH DIRECTIONS

The comparison and framework reveal several open problems at the intersection of machine learning, systems, and human-computer interaction.

1. Grounding as the central UI-agent bottleneck. SeeAct's oracle-grounding experiment [8] shows that if grounding were solved, UI agents would be dramatically more capable. Research on stable element representations, pixel–DOM alignment, and intent-to-action mapping is likely the highest-leverage path for UI agents.

2. Live-web evaluation as the standard. Online-Mind2Web [17] argues convincingly that offline cached evaluations overstate UI-agent progress. The community should adopt live-website evaluation as the standard for "generalist" claims, with explicit measurement of drift sensitivity over time.

3. Tool governance at scale. As tool catalogs grow (MCP servers, connector marketplaces, community plugins), agents face a routing-and-trust problem: selecting the right tool from hundreds of candidates while ensuring the tool itself is not malicious or stale. OpenAI warns about malicious MCP servers and unexpected behavior changes [30]; formalizing tool review, versioning, and trust attestation is an open challenge.

4. Semantic validation beyond syntax. Gorilla [20] addresses syntactic correctness of API calls, but agents must also satisfy semantic constraints: business rules, invariants, and domain-specific preconditions. Future benchmarks should evaluate not just "did the call parse?" but "did the action comply with policy?"

5. Defense in depth for prompt injection. OWASP [27] and Greshake et al. [28] establish that no single mitigation is sufficient against prompt injection. Research should pursue layered defenses: strict data-flow separation between

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

instructions and observations, least-privilege action schemas, runtime anomaly detection, and formal verification of agent decision boundaries.

6. Standardized audit primitives for UI actions. API calls produce naturally structured logs. UI actions need equivalent primitives: standardized records of action intent, UI evidence (element identity, before/after state), and outcome verification, all privacy-preserving and compliance-ready for SOC 2, HIPAA, and PCI DSS.

7. Agentic interface engineering as a discipline. The evidence suggests that UI-vs-API is fundamentally an interface engineering problem. UI agents need better interfaces to UIs (stable intermediate representations, grounded element identity, safe execution sandboxes). API agents need better interfaces to APIs (standard schemas, tool discovery, version governance, security review). We advocate for "agentic interface engineering" as a first-class research agenda that treats action surfaces as engineered products with explicit contracts, safety invariants, observability, and lifecycle management, analogous to how MLOps treats model deployment as an engineering discipline.

VII. CONCLUSION

We have compared two paradigms for autonomous digital actions, UI-grounded and API-grounded agents, across eight dimensions, using an enterprise expense-approval workflow to make abstract trade-offs concrete.

The core tension is reach versus contract. UI-grounded agents offer unmatched coverage: they can operate on any interface a human can use, including legacy systems with no API. But they pay for this reach with brittle grounding, vulnerability to prompt injection, high latency, and weak auditability. API-grounded agents offer strong contracts, deterministic validation, structured audit trails, and superior performance, but only where providers choose to expose and maintain endpoints.

Our Control Surface Contracts taxonomy reframes this binary as a spectrum, and our hybrid decision framework provides concrete, per-subtask guidance: prefer APIs for high-stakes, regulated, and high-throughput actions; use UI automation as a principled fallback for long-tail coverage, with mandatory sandboxing and human oversight for sensitive boundaries.

The expense-approval workflow demonstrates that real enterprise tasks require both paradigms within a single process. The path forward is not choosing one over the other but engineering better action surfaces for both, treating "if a human can use it" and "if there is a contract" as complementary design principles for the agentic era.

REFERENCES

[1] OpenAI, "Introducing Operator," Technical report, OpenAI, January 2025.

[2] Anthropic, "Computer Use (Public Beta)," Technical report, Anthropic, 2024.

[3] OpenAI, "Function Calling and Tool Use," API documentation, 2024.

[4] Anthropic, "Tool Use with Claude," API documentation, 2024.

[5] A. Drouin, M. Gasse, M. Caccia, I. H. Laradji, M. Del Verme, L. Boisvert, M. Thakkar, T. Marty, D. Vazquez, N. Chapados, and A. Lacoste, "WorkArena: How Capable Are Web Agents at Solving Common Knowledge Work Tasks?" in Proc. ICML, 2024.

[6] S. Yao, H. Chen, J. Yang, and K. Narasimhan, "WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents," in Proc. NeurIPS, 2022.

[7] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in Proc. ICLR, 2023.

[8] B. Zheng, B. Gou, J. Kil, H. Sun, and Y. Su, "GPT-4V(ision) Is a Generalist Web Agent, If Grounded," in Proc. ICML, 2024.

[9] P. Shaw, M. Joshi, J. Cohan, J. Berant, P. Pasupat, H. Hu, U. Khandelwal, K. Lee, and K. Toutanova, "From Pixels to UI Actions: Learning to Follow Instructions via Graphical User Interfaces," in Proc. NeurIPS, 2023.

[10] K. Q. Lin, L. Li, D. Gao, Z. Yang, S. Wu, Z. Bai, W. Lei, L. Wang, and M. Z. Shou, "ShowUI: One Vision-Language-Action Model for GUI Visual Agent," arXiv preprint arXiv:2411.17465, 2024.

[11] W. Hong, W. Wang, Q. Lv, J. Xu, W. Yu, J. Ji, Y. Zhang, Y. Wang, Z. Wang, Y. Dong, M. Ding, and J. Tang, "CogAgent: A Visual Language Model for GUI Agents," in Proc. CVPR, 2024.

[12] T. Le Sellier De Chezelles, M. Gasse, A. Drouin, et al., "The BrowserGym Ecosystem for Web Agent Research," arXiv preprint arXiv:2412.05467, 2024.

[13] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, and G. Neubig, "WebArena: A Realistic Web Environment for Building Autonomous Agents," in Proc. ICLR, 2024.

[14] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, "Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration," in Proc. ICLR, 2018.

[15] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2Web: Towards a Generalist Agent for the Web," in Proc. NeurIPS, 2023.

[16] J. Y. Koh, R. Lo, L. Jang, V. Duvvur, M. C. Lim, P.-Y. Huang, G. Neubig, and S. Zhou, "VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks," in Proc. ACL, 2024.

[17] T. Xue, W. Qi, T. Shi, C. H. Song, B. Gou, D. Song, H. Sun, and Y. Su, "An Illusion of Progress? Assessing the Current State of Web Agents," in Proc. COLM, 2025.

[18] F. F. Xu et al., "TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks," arXiv preprint arXiv:2412.14161, 2024.

[19] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," in Proc. NeurIPS, 2023.

[20] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large Language Model Connected with Massive APIs," arXiv preprint arXiv:2305.15334, 2023.

[21] Anthropic, "Model Context Protocol Specification," Open standard, https://modelcontextprotocol.io, 2024.

[22] Microsoft, "Add Tools to Agents with Connectors," Copilot Studio documentation, 2025.

[23] D. Hardt, "The OAuth 2.0 Authorization Framework," RFC 6749, IETF, October 2012.

[24] Microsoft, "Cloud Flows and Desktop Flows Overview," Power Automate documentation, 2025.

[25] UiPath, "API Automation vs. UI Automation: Integration Service Overview," UiPath documentation, 2024.

[26] M. Wornow, A. Narayan, K. Opsahl-Ong, Q. McIntyre, N. H. Shah, and C. Ré, "Automating the Enterprise with Foundation Models," arXiv preprint arXiv:2405.03710, 2024.

[27] OWASP Foundation, "OWASP Top 10 for LLM Applications," Version 1.1, 2024.

[28] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in Proc. AISec, 2023.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

[29] Microsoft, “Build Resilient Desktop Flows: Selectors, Fallback Strategies, and Image-Based Automation,” Power Automate documentation, 2025.

[30] OpenAI, “A Practical Guide to Building Agents” Technical documentation, 2025.

[31] Amine El Hattami et al., “WebArena Verified: Towards Reproducible Web Agent Evaluation,” Preprint, 2025.

[32] OpenTelemetry Authors, “OpenTelemetry Specification: Traces,” https://opentelemetry.io/docs/specs/otel/trace/, 2024.

[33] AICPA, “SOC 2: Trust Services Criteria,” Standard, 2022.

[34] U.S. Department of Health and Human Services, “HIPAA Security Rule,” 45 CFR Part 164, Subparts A and C, 2003.

[35] PCI Security Standards Council, “PCI DSS v4.0,” Standard, March 2022.