Architecting a High-Performance Computer Automation System: Surpassing General Agents' ACE

1. Executive Summary

This report presents an exhaustive technical analysis and architectural blueprint for a next-generation computer automation system, designed to surpass the capabilities and performance of General Agents' ACE (Autonomous Computer Expert) application. The primary objective is to architect a system capable of achieving superhuman-speed automation with an action prediction latency significantly below ACE's claimed 324ms. This document delves into a comprehensive analysis of ACE, explores state-of-the-art AI models, computer vision techniques, action execution frameworks, and performance optimization strategies. It recommends a robust technology stack, outlines a detailed system architecture, and provides an implementation roadmap, risk analysis, and identifies key innovation opportunities. The ultimate goal is to lay the groundwork for a billion-dollar application that will revolutionize human-computer interaction.

2. Introduction

The field of computer automation is on the cusp of a paradigm shift, moving from rule-based scripts to intelligent, context-aware agents capable of understanding and interacting with graphical user interfaces (GUIs) at near-human or even superhuman speeds. General Agents' ACE application has emerged as a notable player, demonstrating impressive capabilities in real-time desktop automation.[1] This report outlines the research and architectural design for a system engineered not only to compete with ACE but to establish a new benchmark in speed, accuracy, and multi-application orchestration. The core mission is to develop an AI-driven automation system that can perceive, reason, and act upon computer interfaces with unprecedented efficiency, targeting an action prediction latency well below 324ms.

2.1. Project Goals and Objectives

The primary goal is to architect a computer automation system that delivers demonstrably superior performance and broader capabilities than General Agents' ACE. Key objectives include:

- Achieving an end-to-end action prediction and execution latency significantly lower than ACE's 324ms benchmark.
- Developing robust multi-application orchestration across diverse desktop environments (Windows, macOS, Linux).

- Ensuring high accuracy in UI understanding and action execution through advanced Vision-Language Models (VLMs) and computer vision techniques.
- Designing a scalable and resilient architecture capable of supporting millions of concurrent users.
- Identifying novel approaches and patent-worthy innovations to create a distinct competitive advantage.

2.2. Scope and Methodology

This report covers a comprehensive analysis of existing technologies, architectural design considerations, technology stack recommendations, performance projections, and a phased implementation roadmap. The methodology involves:

1. **Comprehensive ACE Analysis:** Detailed study of publicly available information on ACE, including its functionalities, performance claims, and underlying technology hints.
2. **Technology Stack Research:** Investigation into core AI models (VLMs), computer vision libraries, OCR engines, UI element detection methods, accessibility APIs, and action execution frameworks.
3. **Performance Optimization Research:** Exploration of GPU acceleration, model serving optimizations, caching strategies, and real-time inference pipelines.
4. **Programming Language and Framework Analysis:** Evaluation of C++, Rust, Go, and Python for core application development, alongside frontend and backend technologies.
5. **Latency Optimization Strategy:** Focused research on techniques to minimize action prediction latency.
6. **Scalability and Infrastructure Planning:** Analysis of cloud and edge computing strategies for large-scale deployment.

The research draws upon academic papers, open-source projects, patent filings, technical blogs, developer forums, and competitor analyses.

3. Comprehensive ACE Analysis

General Agents' ACE is marketed as a "computer autopilot" that controls the user's mouse and keyboard to perform tasks on their desktop at "superhuman speed".[1] It interacts with any installed software based on screen content and user prompts.

3.1. Technical Details from Generalagents.com/ace

- **Functionality:** ACE operates by performing mouse clicks and keystrokes based on screen understanding and natural language prompts.[2] It is designed to work

with existing software on a user's desktop.[1]

- **Training Paradigm:** ACE is trained using a "behavioral training paradigm".[4] This involves learning from screen recordings of real people performing tasks, logging every click and keystroke.[1] General Agents claims this method allows Ace to understand the "why and how" behind actions, not just "what buttons to click".[1] They state it has been trained on over a million tasks by software specialists and domain experts.[2] This approach is contrasted with models trained solely on text and images, with the claim that behavioral training generalizes better.[4]
- **Performance Claims:**
  - ace-control-small model achieves an action prediction latency of 324ms.[2]
  - ace-control-medium model has a latency of 533ms.[2]
  - General task processing is cited around 500ms.[1]
  - This is significantly faster than competitors like OpenAI's Operator (6385ms) and Claude 3.7 Sonnet (9656ms) on General Agents' internal benchmarks.[1]
- **Accuracy:**
  - A "Model Accuracy Comparison" chart on their website shows "Correct Left-Click Predictions" for ace-control-medium and ace-control-small outperforming Operator, Molmo-72B-0924, Claude 3.7 Sonnet, UI-TARS-72B-SFT, OmniParser V2 + GPT-4o, Gemini 2.0 Flash, and Qwen2.5-VL-72B-Instruct.[2] Specific percentages from the chart indicate ace-control-medium achieves approximately 77% click accuracy, and ace-control-small achieves around 70%.[2]
  - Accuracy improves with more training resources, reportedly jumping from 35% to 77% with increased resources, following a typical AI scaling curve.[1]
- **Underlying Models:** General Agents states they are developing "foundation video-language-action models" trained on internet-scale data.[4] The models ace-control-small and ace-control-medium are mentioned specifically.[2]
- **Availability:** Ace is available as a research preview, with access via signup.[1] The ace-control models are available to selected partners through their developer platform.[2]
- **Limitations:** ACE is still learning and can make mistakes, sometimes stumbling on complex tasks or unfamiliar interfaces.[1] An example failure shown is summarizing information from a Wikipedia page and writing it to another tab.[2]

3.2. Demonstration Video Analysis

Demonstration videos showcase ACE performing tasks such as:

- Booking an Airbnb: Opening Chrome, navigating to airbnb.com, entering details (Cairo, May 1st), selecting dates, finding, selecting the first option, and

proceeding to confirm/pay.[8]
- Developer Task: Selecting a traceback error message, opening Chrome, navigating to a specific GitHub repo, and creating an issue with title and description.[8]
- System Control: Switching macOS to dark mode and enabling Game Center via System Settings.[8]
- Other examples include finding LinkedIn candidates and iMessaging them, video editing in Premiere (changing speed, cutting, exporting), and zipping files, uploading to Drive, and texting the link.[1]

Observations from Demonstrations:

- **UI Interaction Patterns:** ACE appears to interact with standard UI elements (buttons, text fields, menus) directly. Interactions are rapid and sequential. The system seems to identify elements visually and then simulate clicks or keystrokes.
- **Response Times:** The videos emphasize speed, with actions completed in "milliseconds, not minutes".[1] The Airbnb booking, GitHub issue creation, and macOS settings changes are shown at 1x speed and appear very fast, consistent with the sub-second latency claims.[8]
- **Computer Vision Implementation:** The system must be performing robust screen understanding to identify and locate target UI elements for interaction. The "behavioral training" on screen recordings suggests a strong reliance on visual input paired with action sequences.[1]
- **Action Prediction Mechanisms:** The 324ms latency for ace-control-small [2] suggests a highly optimized model capable of rapidly predicting the next mouse/keyboard action based on the current screen state and the user's prompt. This likely involves a VLM that outputs discrete actions (e.g., click coordinates, key presses).
- **Multi-application Orchestration:** Demonstrations show ACE seamlessly switching between and operating multiple applications (Chrome, System Settings, iMessage, Premiere, GitHub, Finder) to complete tasks.[1]

3.3. Claimed Superiority Analysis

ACE is claimed to be significantly faster and more accurate than:

- **OpenAI's Operator:** ACE processes tasks in ~500ms, Operator in >6,300ms.[1] ACE ace-control-small has 324ms latency vs. Operator's 6385ms.[2] ACE models show higher click accuracy.[2] Operator is noted to use a Computer-Using Agent (CUA) model and achieves 38.1% on OSWorld benchmark, while a regular human gets 72.4%.[9] Operator is in public preview for ChatGPT Pro subscribers

($200/month) and operates a PC in the cloud, kicking control to the user for logins/CAPTCHAs.[9]

- **Claude Computer Use:** ACE processes tasks in ~500ms, Claude 3.7 Sonnet in ~9,600ms.[1] ACE ace-control-small has 324ms latency vs. Claude 3.7 Sonnet's 9656ms.[2] ACE models show higher click accuracy.[2] Claude's computer use achieved 22% on OSWorld and is available via API.[9] It controls the user's own PC in real-time.[10]
- **OmniParser V2 + GPT-4o:** ACE ace-control-small has 324ms latency vs. OmniParser V2 + GPT-4o's 12642ms.[2] ACE models show higher click accuracy.[2]
- **Owen (presumed to be Molmo-72B-0924 or similar):** ACE ace-control-small has 324ms latency vs. Molmo-72B-0924's 6599ms.[2] ACE models show higher click accuracy.[2]
- **Gemini:** ACE is stated to leave Gemini "in the digital dust".[1] Gemini 2.0 Flash has a latency of 3069ms on ACE's benchmark.[2]

The core of ACE's claimed superiority lies in its speed, attributed to its specialized training and model architecture.[1] The behavioral training paradigm, learning from human computer interactions (screen recordings, mouse/keyboard logs), is presented as a key differentiator.[1] This approach contrasts with models primarily trained on text and static images, suggesting ACE develops a more intuitive understanding of UI interaction flows.

3.4. Underlying AI Models, Frameworks, and Architectural Patterns

- **AI Models:** "Foundation video-language-action models" are being developed.[4] These models are trained on "internet-scale data".[4] The specific models mentioned are ace-control-small and ace-control-medium.[2] The term "video-language-action" implies the model processes visual screen input (like a video stream of screen changes), understands language prompts, and outputs sequences of actions (mouse/keyboard operations). There's a mention that ACE can generalize to use video game UIs despite not being in training data, and plans to incorporate video game playtime data.[11] This suggests a VLM architecture capable of learning complex UI interactions from diverse visual data.
- **Frameworks:** The exact software frameworks are not detailed. However, given the focus on desktop automation, it would likely involve native OS APIs for screen capture, input simulation, and potentially accessibility services. The team has expertise in machine learning and reinforcement learning.[4] The mention of reinforcement learning could imply that, beyond behavioral cloning from demonstrations, the agent might refine its policies through trial and error or reward-based learning in simulated or real environments. The "ACE" acronym is

also used by NVIDIA for their Avatar Cloud Engine, which includes the ACE Controller SDK based on the Python Pipecat framework for multimodal conversational AI.[12] While this is a different "ACE," the underlying concepts of managing multimodal interactions are relevant. The term "ACE model" is also used in an academic paper (unrelated to General Agents) describing a cognitive architecture framework with layers for aspiration, strategy, agent model, executive function, cognitive control, and task prosecution.[13] This hierarchical approach to agent design could be an inspiration, though not directly confirmed for General Agents' ACE.

- **Architectural Patterns:** The system operates directly on the user's desktop.[1] This suggests a client-side application. The architecture likely follows a perception-reasoning-action loop:
  1. **Perception:** Capture screen content, potentially identify UI elements and text.
  2. **Reasoning:** The VLM processes the visual input and user prompt to decide the next action.
  3. **Action:** Execute the predicted mouse/keyboard action. This loop repeats until the task is complete.[9] Given the low latency, this loop must be highly optimized. The behavioral training paradigm suggests a form of imitation learning or behavioral cloning.[14]

3.5. API Structure, SDKs, and Integration Capabilities

- **Developer Platform:** General Agents mentions a developer platform where ace-control models are available to selected partners.[2] However, public details about this platform, its API structure, SDKs, or specific integration capabilities are minimal. The platform URL (platform.generalagents.com) leads to a login page with no further public information.[25]
- **Comparison to Other Agent APIs/SDKs:**
  - **Cloudflare Agents API/SDK:** Provides server-side Agent classes and client-side AgentClient classes for building and connecting to agent instances. Supports state management, WebSockets, and React hooks. Agents are globally unique instances addressed by ID, suitable for scaling.[26] This is a general AI agent framework, not specifically for UI automation like ACE.
  - **NVIDIA ACE Controller SDK:** Allows building controller services for multimodal voice bots and avatars using the Pipecat framework. It integrates NVIDIA services like Riva (speech AI) and Audio2Face (facial animation).[12] This is focused on conversational and embodied AI, different from ACE's desktop automation.
  - **Zapier MCP with Claude:** Zapier offers Model Context Protocol (MCP)

integration with Claude, allowing Claude to interact with other apps via Zapier's automation capabilities.[9] This is an example of enabling an LLM to act through existing automation platforms.

The lack of public API/SDK information for General Agents' ACE [25] suggests a currently closed ecosystem or a focus on direct enterprise solutions rather than a broad developer platform. This presents an opportunity for a competing system to offer superior integration capabilities.

4. Technology Stack Research and Recommendations

To surpass ACE, a meticulously selected and optimized technology stack is paramount. This section evaluates options for core AI models, computer vision, action execution, performance optimization, and programming languages.

4.1. Core AI Models Investigation: Vision-Language Models (VLMs)

The heart of the system will be a Vision-Language Model (VLM) or a Video-Language-Action (VLA) model, capable of understanding screen content (vision), user instructions (language), and predicting appropriate computer control actions. General Agents' ACE uses "foundation video-language-action models" trained on behavior data.[4] Our system must leverage or develop models with superior UI understanding, reasoning, and action generation capabilities, optimized for extremely low latency.

- **4.1.1. State-of-the-Art Commercial VLMs**
  - **GPT-4V / GPT-4o (OpenAI):**
    - **Capabilities:** GPT-4V and the newer GPT-4o possess strong multimodal understanding. GPT-4.1 (an evolution of GPT-4o) shows exceptional performance in coding, instruction following, and long context understanding (Video-MME benchmark score of 72.0%).[27] It also excels in frontend coding, with human graders preferring its websites over GPT-4o's 80% of the time.[27] Vision fine-tuning is available for GPT-4o, allowing customization with images and text to improve vision capabilities for tasks like object detection and UI element localization.[28] Automat, an enterprise automation company, used vision fine-tuning on GPT-4o with screenshots to improve UI element location success rate from 16.60% to 61.67% (a 272% uplift).[28] Coframe improved website generation consistency by 26% with vision fine-tuning.[28] GPT-4 with rich linguistic descriptions improved zero-shot recognition by an average of 7% across 16 datasets.[29]

        - **Relevance:** Strong general visual reasoning. Fine-tuning capabilities are crucial for specializing in UI understanding and action prediction. The success of Automat in improving UI element localization via fine-tuning is highly relevant.[28]
        - **Concerns:** Latency for real-time control needs careful evaluation. Cost at scale. Dependence on OpenAI APIs.
    - **Gemini Ultra/Pro Vision (Google):**
        - **Capabilities:** Gemini models are built to be multimodal from the ground up.[30] Gemini 2.5 Pro is being extended to become a "world model" for planning and simulation.[31] It supports image captioning, Q&A, PDF transcription/reasoning (up to 2M tokens), object detection with bounding boxes, and image segmentation.[30] "Project Mariner" (now part of Gemini) focuses on computer use capabilities, including browser interaction, and these are being integrated into the Gemini API.[31] Gemini 2.5 Pro shows strong performance on benchmarks like MMMU (79.6% visual reasoning), LiveCodeBench (75.6% code generation), and SWE-bench (63.2% agentic coding).[32] Time to first token is ~0.7s, full response (1k tokens) ~2.8s.[33]
        - **Relevance:** Native multimodality and Google's focus on agentic capabilities (Project Mariner) are promising. Object detection and segmentation are directly applicable to UI element identification.
        - **Concerns:** Latency for interactive control needs to be significantly lower than general API response times. Cost and API limitations.
    - **Claude 3 / Claude 4 Vision (Anthropic):**
        - **Capabilities:** Claude 3 and 4 models feature vision capabilities, understanding and analyzing images.[34] They claim "best-in-class vision capabilities" compared to other leading models, accurately transcribing text from imperfect images and processing web UI, photos, charts, graphs, and technical diagrams.[35] Claude can process multiple images per request (up to 20 on claude.ai, 100 via API).[34] Images should ideally be placed before text in prompts.[34] Users report Claude excels in longer response lengths, fewer refusals, natural language generation, and faster response times compared to GPT-4 in some coding and template generation tasks.[36] A YouTube test showed Claude 3.5 Sonnet performing well in web UI automation tasks, making only one mistake in a tricky situation and being consistent in its errors.[37]
        - **Relevance:** Explicit mention of "processing web UI" is highly relevant. Strong performance in text transcription from images is also beneficial for UI understanding.
        - **Concerns:** Latency for real-time control. Cost. The OSWorld benchmark

for Claude Computer Use was 22%, lower than Operator's 38.1%.[9]

- **4.1.2. Open-Source VLMs**
  - **LLaVA (Large Language and Vision Assistant):**
    - **Capabilities:** Open-source VLM fine-tuned on GPT-generated multimodal instruction-following data.[38] LLaVA-1.5 was used in early experiments for the Self-Operating Computer Framework to estimate pixel clicks, though reliability was an issue before switching to GPT-4-vision-preview.[39] LLaVA-o1 introduces structured reasoning stages (summary, caption, reasoning, conclusion).[41]
    - **Relevance:** Proven applicability in computer control experiments. Open-source nature allows for deep customization and fine-tuning.
    - **Concerns:** Initial versions may lack the robustness and fine-grained UI understanding of commercial models without significant further development. Performance on UI-specific benchmarks needs to be established.
  - **CogVLM / CogAgent:**
    - **Capabilities:** CogVLM is an open-source VLM with 10B visual and 7B language parameters, known for detailed image descriptions and visual grounding.[42] CogAgent is an 18B parameter VLM specializing in GUI understanding and navigation, supporting 1120x1120 resolution input for recognizing tiny elements.[43] CogAgent outperforms LLM-based methods using extracted HTML on Mind2Web (PC GUI) and AITW (Android GUI) benchmarks using only screenshots.[43] Falcon-UI is a GUI agent model pretrained for GUI comprehension and then fine-tuned on Android and Web GUI datasets.[47]
    - **Relevance:** CogAgent's specific focus on GUI understanding, high-resolution input, and strong benchmark performance on UI tasks make it a very strong open-source candidate.
    - **Concerns:** 18B parameters is still large for client-side deployment without significant optimization.
  - **Qwen-VL (Alibaba):**
    - **Capabilities:** Qwen2.5-VL series (3B, 7B, 72B parameters) is designed for AI agents, finance, and commerce. Excels in visual recognition, reasoning, long video analysis, object localization (bounding boxes, points, JSON output), and structured data extraction.[48] It features enhanced agent functionality for computer and mobile use, leveraging advanced grounding, reasoning, and decision-making.[48] A "Computer-Use Agent" cookbook is provided.[48]
    - **Relevance:** Strong focus on agentic capabilities and visual grounding. The

smaller 3B/7B models might be suitable for optimization.
        - **Concerns:** Performance of smaller variants needs to match larger commercial models for complex reasoning.
    - **InternVL:**
        - **Capabilities:** InternVL3 features a native multimodal pre-training paradigm, jointly acquiring multimodal and linguistic capabilities.[50] Incorporates V2PE for extended multimodal contexts and advanced post-training (SFT, MPO). Shows marked improvements in tool usage, GUI agents, and spatial reasoning due to expanded domain-specific datasets.[50] InternVL3-78B achieves 72.2 on MMMU.[50] It shows strong performance on GUI grounding benchmarks like ScreenSpot.[52]
        - **Relevance:** Native multimodal pre-training and focus on GUI agents and tool use are highly aligned with project goals.
        - **Concerns:** Larger models (78B) are not suitable for client-side deployment without extreme optimization. Smaller variants' performance needs to be competitive.
- **4.1.3. Specialized UI Models**
    - **UIBert:** BERT-like model trained to understand application elements (buttons, menus, text inputs) and interaction operations (click, scroll).[54] Requires GUI widgets labeled with type and interaction.
    - **ScreenAI (Google):** Based on PaLI architecture with pix2struct flexible patching. Trained on a mixture of datasets including a novel "Screen Annotation task" (identify UI element type, location, description).[55] Achieves SOTA on UI/infographics tasks like WebSRC and MoTIF.[55]
    - **Pix2Struct (Google):** Pretrained image-to-text model for visual language understanding, learning to parse masked screenshots of web pages into simplified HTML.[56] Achieves SOTA on UI tasks like RefExp, Widget Captioning, Screen2Words.[56] Uses variable-resolution inputs for ViT.
    - **Fuyu-8B (Adept AI):** Decoder-only multimodal transformer supporting arbitrary image resolutions. Designed for digital agents, responds in <100ms for large images. Performs visual Q&A, captioning, and fine-grained localization on screen images.[57]
    - **Relevance:** These models are specifically designed or pretrained for UI understanding tasks, potentially offering better performance and efficiency for element detection, localization, and functional understanding compared to general VLMs if fine-tuned or integrated appropriately. ScreenAI's screen annotation task and Pix2Struct's HTML parsing from screenshots are particularly interesting pretraining objectives. Fuyu-8B's speed is a key attribute.

- **Concerns:** May lack the general reasoning capabilities of larger VLMs. Integration with a broader reasoning engine might be needed. Availability and ease of fine-tuning for open-source specialized models.

- **4.1.4. Model Comparison: Inference Speed, Accuracy, Deployment Costs**
  - **ACE Benchmarks:** ace-control-small (324ms), ace-control-medium (533ms). Competitors: UI-TARS-72B-SFT (1977ms), Gemini 2.0 Flash (3069ms), Qwen2.5-VL-72B-Instruct (3790ms), Operator (6385ms), Molmo-72B-0924 (6599ms), Claude 3.7 Sonnet (9656ms), OmniParser V2 + GPT-4o (12642ms).[2]
  - **Click Accuracy (ACE Benchmarks):** ace-control-medium (~77%), ace-control-small (~70%), Operator (~58%), Molmo-72B-0924 (~55%), Claude 3.7 Sonnet (~52%), UI-TARS-72B-SFT (~50%), OmniParser V2 + GPT-4o (~45%), Gemini 2.0 Flash (~43%), Qwen2.5-VL-72B-Instruct (~40%).[2]
  - **Other Benchmarks:**
    - GPT-4.1: 54.6% on SWE-bench Verified, 72.0% on Video-MME.[27]
    - Gemini 2.5 Pro: 79.6% on MMMU, 75.6% on LiveCodeBench, 63.2% on SWE-bench Verified. TTFT ~0.7s.[32]
    - Claude 3.5 Sonnet: Reported good for web UI automation.[37]
    - InternVL2-8B with CV preprocessing showed 49% improvement in UI test case generation accuracy.[59]
    - CogAgent (18B) SOTA on Mind2Web and AITW using only screenshots.[43]
    - InternVL3-78B: 72.2 on MMMU. ScreenSpot accuracy: InternVL3-72B (88.7%), UI-TARS-72B (88.4%), Qwen2.5-VL-72B (87.1%).[50]
    - Fuyu-8B: <100ms response for large images.[57] VQAv2 74.2, OKVQA 60.6.[57]
  - **Deployment Costs:** Commercial models (GPT-4o, Gemini, Claude) have per-token/per-image API costs.[28] Open-source models require infrastructure for hosting and inference, but no per-call cost.
  - **Table: VLM Comparison for UI Automation**

Model Family	Key UI Strengths	Reported Latency (Relevant)	Accuracy (Relevant Benchmarks)	Cost Factor	Open Source
ACE (Baseline)	Behavioral training, fast action prediction	ace-control-small: 324ms [2]	ace-control-small: ~70% click acc. [2]	N/A (Proprietary)	No

GPT-4V/GPT-4o	Strong general vision, fine-tunable for UI [28]	Higher than ACE (Operator: 6385ms)	Automat: 272% UI loc. uplift [28]; GPT-4.1: 54.6% SWE-bench [27]	High (API)	No
Gemini Ultra/Pro	Native multimodal, Project Mariner (computer use) [31], obj. detect/seg [30]	3069ms (Gemini 2.0 Flash) [2]	~43% click acc. (Gemini 2.0 Flash) [2]; Gemini 2.5 Pro: 79.6% MMMU, 63.2% SWE-bench [32]	High (API)	No
Claude 3/4 Vision	"Best-in-class vision," web UI processing [35]	9656ms (Claude 3.7 Sonnet) [2]	~52% click acc. (Claude 3.7 Sonnet) [2]; Good web UI test performance [37]	High (API)	No
LLaVA	Open, computer control experiments [39]	Untested for sub-300ms UI tasks	Needs UI-specific fine-tuning	Medium (Infra)	Yes
CogAgent	GUI specialization, 1120px input, SOTA Mind2Web/AITW [43]	Untested for sub-300ms UI tasks	SOTA on Mind2Web/AITW [43]	Medium (Infra)	Yes
Qwen2.5-VL	Agent focus, grounding, "Computer-Use Agent"	3790ms (72B-Instruct) [2]	~40% click acc. (72B-Instruct	Medium (Infra)	Yes

	cookbook [48]		) [2]		
InternVL3	Native multimodal pretrain, GUI agent focus, SOTA MMMU [50]	Untested for sub-300ms UI tasks	ScreenSpot: 8B (81.4%), 38B (88.3%), 72B (88.7-90.9%) [52]	Medium (Infra)	Yes
ScreenAI	PaLI+Pix2Struct, Screen Annotation task, SOTA WebSRC/MoTIF [55]	Untested for sub-300ms UI tasks	SOTA on WebSRC/MoTIF [55]	Medium (Infra)	Yes (Google)
Pix2Struct	Screenshot to HTML pretrain, SOTA UI RefExp/Captioning [56]	Untested for sub-300ms UI tasks	SOTA RefExp (94.2%), Widget Captioning (136.7 CIDEr) [56]	Medium (Infra)	Yes (Google)
Fuyu-8B	<100ms for large images, UI Q&A, fine-grained localization [57]	<100ms [57]	VQAv2 74.2 [57]	Medium (Infra)	Yes (Adept)

* The selection of a core VLM is a critical decision. While commercial models like GPT-4o offer powerful general capabilities and the option for vision fine-tuning (as demonstrated by Automat's 272% uplift in UI element localization [28]), they come with API costs and potential latency that might be challenging to optimize below the 324ms target for every interaction. Open-source models, particularly those specialized for GUI tasks like CogAgent (with its high-resolution input and strong Mind2Web/AITW performance [43, 44, 45, 46]) or InternVL3 (with its native multimodal pre-training and GUI agent focus [50, 51, 52]), offer greater control for deep

optimization and on-device deployment. Fuyu-8B's reported <100ms inference is also highly attractive.[57] A strategy involving a highly optimized, possibly distilled, smaller open-source model for common, fast interactions, potentially augmented by a larger model (either open-source hosted or commercial via API) for more complex reasoning or fallback, seems promising. The key will be rigorous benchmarking of these models on UI-specific tasks, focusing on both accuracy (like click prediction and task completion) and end-to-end latency.

- **4.1.5. Model Optimization Techniques**
  - **Quantization:** Reducing model precision (e.g., FP16 to INT8 or INT4) to decrease size and speed up inference. Techniques like AWQ (Activation-aware Weight Quantization) [60] and GPTQ [60] are common. AWQ is noted for its hardware-friendliness and good performance on LLMs and VLMs, even for multi-modal models for the first time, without backpropagation.[61] Integer-only arithmetic can further reduce computational resources.[60] Quantization can yield significant speedups (e.g., RTN up to 37% faster for 4-bit).[66]
  - **Distillation:** Training a smaller "student" model to mimic a larger "teacher" model. This can transfer capabilities while reducing size and latency by 40-60% with modest accuracy drops (2-4%).[68] Knowledge distillation is a core component for modern generative models.[60] SPORT uses iterative tool usage exploration and preference optimization, which can be seen as a form of self-distillation or learning from AI feedback.[72] MIND is a modality-informed KD framework for multimodal tasks.[71]
  - **Pruning:** Removing redundant weights or structures (neurons, attention heads). Structured pruning is generally better for hardware acceleration.[60] OSSCAR is a one-shot structured pruning framework scalable to tens of billions of parameters.[73] Pruning can achieve significant FLOPs reduction (e.g., Diff-Pruning ~50%).[76]
  - These techniques are crucial for deploying capable VLMs on client devices or edge servers to meet low-latency requirements. A combination of these (e.g., pruning followed by quantization) often yields the best results. The choice depends on the specific VLM architecture and target hardware.
- **4.1.6. Edge Deployment Possibilities**
  - Deploying smaller, optimized VLMs directly on user devices (edge) can significantly reduce latency by eliminating network roundtrips.[64]
  - Challenges include limited processing power, memory, and energy on edge devices.[64] Model compression techniques (quantization, pruning, distillation) are essential.[64]
  - Specialized hardware (NPUs, TPUs on edge devices) can accelerate VLM

inference.[64]
    - A hybrid approach, with a smaller model on the edge for common/fast tasks and a larger model in the cloud for complex reasoning, is a viable strategy.[82]
- **4.1.7. Recommended VLM Strategy**
    - **Core Model:** Prioritize a highly optimized open-source VLM, such as a custom-trained/fine-tuned version of CogAgent or InternVL3, due to their strong GUI understanding capabilities and potential for deep optimization. Fuyu-8B is also a strong candidate if its general reasoning can be sufficiently enhanced for complex tasks or if used in a dual-model setup.
    - **Behavioral Cloning/Imitation Learning:** Adopt a "behavioral training paradigm" similar to ACE.[1] This involves collecting extensive data of human-computer interactions (screen recordings, mouse/keyboard logs, window hierarchy changes) across various applications and platforms. This data will be used for supervised fine-tuning of the chosen VLM to predict action sequences (e.g., click coordinates, text to type, key presses). The EBC-LLMAgent approach for mobile apps, which includes demonstration encoding, code generation, and UI mapping, offers relevant concepts.[14]
    - **Continuous Learning:** Implement a data flywheel. User interactions (with consent and anonymization) should be collected to continuously fine-tune and improve the model's accuracy and coverage of new applications and UI patterns.
    - **Optimization for Latency:** Aggressively apply quantization (e.g., 4-bit AWQ or GPTQ), structured pruning, and knowledge distillation to create a highly efficient model deployable on client devices or low-latency edge servers.
    - **Dual-Model Architecture (Potential):** Explore a primary lightweight, client-side VLM for common, fast actions, with the ability to call a more powerful backend VLM for complex planning or novel situations. The client-side model could be a distilled version. This aligns with the need for sub-324ms responses for most interactions while retaining strong reasoning for complex tasks.

4.2. Computer Vision and Screen Understanding

Accurate and real-time screen understanding is fundamental. This involves capturing the screen, identifying UI elements, extracting text, and understanding their spatial relationships and functionalities.

- **4.2.1. Real-Time Screen Capture Technologies**
    - **Cross-Platform Libraries:** Libraries like scap (Rust) leverage native OS APIs (ScreenCaptureKit for macOS, Windows.Graphics.Capture for Windows,

PipeWire for Linux) for optimal performance and offer features like target selection and exclusion.[86] windows-capture is a high-performance Rust/Python library for Windows using Graphics Capture API.[87]
    - **Performance:** Native OS APIs generally offer the lowest latency and resource usage. Efficient compression and frame handling are key.[88] scap aims for high quality and optimal performance.[86] windows-capture only updates frames when required, enhancing performance.[87]
    - **Recommendation:** Utilize a Rust-based library like scap for its cross-platform native API integration and performance focus. This core component should provide raw frame data and potentially diffs between frames to the perception module.
- **4.2.2. OCR Engines**
    - **Open Source:**
        - **Tesseract OCR:** Widely used, supports multiple languages. Version 4+ uses LSTM. Accuracy varies by language and image quality; good for English (92%), Hindi (93%), Malayalam (93%), but lower for Tamil (78%), Arabic (74%) in one study.[90] Requires preprocessing for noisy images.[91]
        - **EasyOCR:** Python library, deep learning-based, supports 80+ languages. Simpler API. Can outperform Tesseract on noisy images but may struggle with severe blur or low resolution.[90] Accuracy reported lower than Tesseract in one study for several languages (e.g., English 71%).[90]
        - **PaddleOCR:** Developed by Baidu, efficient for high-volume, real-time tasks. Good performance in Arabic (91%) and Malayalam (91%) in one study.[90]
        - **MMOCR, Keras-OCR:** Other open-source options with varying strengths.[90]
    - **Commercial Alternatives:**
        - **Google Cloud Vision API:** AI-powered, recognizes printed/handwritten text. Pricing: Text Detection and Document Text Detection are free for the first 1000 units/month, then $1.50/1000 units (1001-5M), then $0.60/1000 units (>5M).[92] PDF processing: 5 pages = 1 unit.[93]
        - **Azure AI Vision (Microsoft):** OCR features for image analysis. Pricing for "Read" (OCR): Standard (S1) tier: $1.50/1000 transactions (0-1M), $0.60/1000 (1M+). Commitment tiers offer discounts (e.g., $375/month for 500k transactions).[95]
        - **Adobe Acrobat Pro DC, ABBYY FineReader PDF:** Enterprise-level PDF management with advanced OCR, but typically for document conversion not real-time UI text extraction.[94]
    - **Recommendation:** Implement a hybrid OCR strategy.

1. **Primary:** A locally run, optimized Tesseract or EasyOCR for speed and cost-effectiveness on clear UI text.
2. **Secondary/Fallback:** Utilize Google Cloud Vision API or Azure AI Vision for complex cases, handwritten text within UI, or when local OCR confidence is low. This provides a balance of speed, cost, and accuracy. The specific choice between Google and Azure can be based on broader cloud strategy and detailed cost analysis for expected volumes. Google's Document Text Detection is particularly suited for dense text areas.[92]

- **4.2.3. UI Element Detection**
  - **Object Detection Models:**
    - **YOLO (You Only Look Once) variants:** YOLOv8 achieves high precision (95%) and mAP (97%) for UI components in some studies.[96] Efficient for real-time detection. Can struggle with very small or clustered objects.[97]
    - **Detectron2 (Facebook AI Research):** Platform for object detection and segmentation. Supports models like Faster R-CNN, Mask R-CNN, RetinaNet, Panoptic Segmentation.[98] Modular and high-performance.
    - **Custom CNN Architectures:** Can be trained for specific UI element types.
  - **Specialized UI Models with Detection Capabilities:**
    - **ScreenAI:** Uses a layout annotator (DETR-based) to identify UI elements (image, pictogram, button, text) and their spatial relationships.[55]
    - **Pix2Struct:** Parses screenshots into HTML-like structures, implicitly identifying elements.[56]
    - **CogAgent:** High-resolution input (1120x1120) specifically for recognizing tiny page elements and text.[43]
  - **Recommendation:** Rely primarily on Accessibility APIs for element detection due to their accuracy and semantic information. For elements not exposed via accessibility (e.g., in custom-rendered UIs, games), the core VLM (fine-tuned with UI data, potentially incorporating techniques from CogAgent or ScreenAI) should handle visual detection and localization. A dedicated lightweight object detector (like an optimized YOLO variant) could serve as a faster, preliminary element proposer if the VLM is too slow for initial broad detection, but this adds complexity. The VLM's visual grounding capability will be key here.
- **4.2.4. Accessibility APIs Integration**
  - **Windows UI Automation (UIA):** Unified object model for UI frameworks. Provides programmatic access to UI elements, properties, and control patterns.[100] Can be used for automated testing and by AI agents.[101] Libraries like pywinauto (supports UIA backend) [103], uiautomation (Python wrapper) [105],

and Rust crate uiautomation [105] provide access. C# examples also exist.[107]
    - **macOS Accessibility API (NSAccessibility):** Provides information about UI elements, their hierarchy, roles, and actions.[111] Can be controlled via AppleScript [113] or programmatically (e.g., libuic [117], Rust bindings from cacao, core-foundation [118]). AccessKit offers a Rust abstraction layer.[121] The CUA framework utilizes macOS accessibility trees in VMs.[123]
    - **Linux AT-SPI (Assistive Technology Service Provider Interface):** D-Bus based framework for communication between assistive technologies and applications. Exposes UI element tree.[124] Client libraries include libatspi (C) and pyatspi2 (Python).[124] AccessKit also wraps AT-SPI.[121]
    - **Cross-Platform Abstraction (AccessKit):** AccessKit is a Rust-based library providing a cross-platform abstraction over native accessibility APIs (UIA, NSAccessibility, AT-SPI).[121] It uses a "push model" for UI tree updates. AI agents can use it for precise perception and reliable execution.[121]
    - **Recommendation:** Leverage accessibility APIs as the primary source for UI element information (type, name, state, bounding box, actions). Implement this via a cross-platform Rust-based abstraction layer, either by heavily utilizing/extending AccessKit or building a similar custom library. This provides structured, semantic data directly from the OS, which is more reliable and efficient than purely visual methods for many standard UI elements. This structured data can then be fed to the VLM along with screenshots to provide rich context.
- **4.2.5. Visual Grounding Techniques**
    - **Definition:** Mapping natural language descriptions or intentions to specific image regions or UI elements. Crucial for resolving ambiguity (e.g., "click the green button" when there are multiple).
    - **LVLM Native Capabilities:** Some VLMs have inherent visual grounding capabilities through attention mechanisms. A few attention heads in frozen LVLMs can demonstrate strong grounding ("localization heads") without fine-tuning.[127]
    - **Specialized Models/Frameworks:**
        - **ScreenSpot-Pro:** Benchmark for GUI grounding in high-resolution professional settings. Highlights challenges with small targets and complex environments. Proposes ScreenSeekeR, a visual search method using a planner to guide cascaded search, achieving 48.1% accuracy without retraining.[128]
        - **Aria-UI:** LMM for GUI grounding using a pure-vision approach, avoiding HTML/AXTree. Uses a scalable data pipeline for instruction synthesis and incorporates action histories for context-aware reasoning.[130]

        - **UGround:** Universal visual grounding model trained on 10M synthetic GUI elements over 1.3M screenshots. Outperforms existing models by up to 20%.[130] Relies on an external LLM planner. Input image size up to 36 grids of CLIP@224 (9216 tokens).[134]
        - **Iris:** Visual agent with Information-Sensitive Cropping (ISC) using edge detection to focus on dense regions, and Self-Refining Dual Learning (SRDL).[130]
        - **Set-of-Mark (SoM) Prompting:** Enhances visual grounding by overlaying images with marks for fine-grained object recognition.[130]
    - **Recommendation:** The core VLM should be trained/fine-tuned with a strong emphasis on visual grounding for UI elements. This can be achieved by:
        1. Incorporating large-scale GUI grounding datasets (like those used for UGround or generated via Aria-UI's pipeline) into the fine-tuning process.
        2. Exploring architectures or fine-tuning techniques that enhance localization head capabilities within the VLM.
        3. Potentially integrating a lightweight, specialized grounding module (inspired by ScreenSeekeR or SoM if the core VLM struggles with precision) that works in tandem with the VLM's broader understanding. The VLM identifies the likely target element and its properties, and the grounding module refines the exact coordinates or confirms the selection in ambiguous cases.
- **4.2.6. Recommended CV Pipeline**
    1. **Screen Capture:** Use a high-performance, low-latency cross-platform library (e.g., scap [86]) to capture the current screen state. Capture should be triggered by user prompt or completion of the previous action.
    2. **Accessibility Data Extraction:** Simultaneously, query the OS accessibility API (via the Rust-based abstraction layer) for the current application's UI tree, focused element, and properties of visible elements (name, role, state, bounding box).
    3. **OCR:** If necessary (e.g., for text not available via accessibility API or for visual verification), run fast local OCR (e.g., optimized Tesseract) on relevant screen regions identified by accessibility data or previous VLM focus. Use cloud OCR as a fallback for high-accuracy needs on difficult text.
    4. **VLM Input Assembly:** Combine the screenshot, structured data from accessibility APIs (e.g., list of key elements with their properties and bounding boxes), and OCRed text into a multimodal prompt for the VLM.
    5. **VLM Processing (Perception & Reasoning):** The VLM processes this input to understand the current UI state in the context of the user's goal and predicts the next action (e.g., "click button 'Submit' at [x,y]" or "type 'hello'

into text field 'Username'").

6. **Visual Grounding & Disambiguation:** If the VLM's predicted action involves an ambiguous element (e.g., "the green button"), or if precise coordinates are needed for an element identified semantically, the visual grounding component (either native to the VLM or a separate module) refines the target to an exact bounding box or click point.
7. **Action Formulation:** Convert the VLM's decision and grounded target into a specific command for the Action Execution Framework.

- This pipeline prioritizes structured data from accessibility APIs for reliability and efficiency, falling back to visual processing by the VLM when necessary. The VLM's role is not just to "see" but to "understand" the UI in context, guided by both visual and semantic information.

4.3. Action Execution Framework

This framework translates the VLM's decisions into actual mouse/keyboard operations and application interactions.

- **4.3.1. Native OS Automation APIs**
  - **Windows:** UI Automation (UIA) is the modern framework for programmatic UI access.[100] It allows discovering controls, navigating the visual tree, accessing state, and performing actions via Automation Patterns (e.g., InvokePattern for clicks, ValuePattern for text input).[107] Python libraries like pywinauto (with "uia" backend) [103] and uiautomation [105] wrap these APIs. The Rust uiautomation crate also provides bindings.[105] Win32 API calls can also be used for lower-level control.
  - **macOS:** The Accessibility API (NSAccessibility) is the primary interface.[111] Actions can be scripted via AppleScript (which uses accessibility features) [113] or triggered programmatically. libuic is an older C++ library for this.[117] Rust bindings exist for Cocoa and Core Foundation, enabling interaction with these APIs.[118] AccessKit provides a Rust abstraction.[121]
  - **Linux:** AT-SPI over D-Bus is the standard accessibility framework.[124] pyatspi2 provides Python bindings. For direct input simulation, xdotool (for X11) [136] and ydotool or wtype (for Wayland) [140] are common. RustAutoGUI is a cross-platform library that likely wraps these or similar mechanisms.[141]
  - **Table: Native OS Automation API Comparison**

API/Tool	Platform(s)	Primary Access	Control Depth	Reliability	Performance	Ease of Integratio

		Method(s)				**n (Rust/C++)**
Windows UI Automation (UIA)	Windows	COM API,.NET wrappers	High (elements, patterns, properties) [100]	Good for UIA-compliant apps	Moderate to High	Good (Rust uiautomation [105], C++ via COM)
macOS Accessibility API (NSAccessibility)	macOS	Objective-C API, AppleScript bridge	High (elements, attributes, actions) [111]	Good for Cocoa apps	Moderate to High	Good (Rust via objc crate, Swift)
Linux AT-SPI	Linux	D-Bus API, libatspi (C), pyatspi2 (Python)	High (elements, interfaces, events) [124]	Varies by toolkit/app support	Moderate	Moderate (D-Bus bindings)
xdotool	Linux (X11)	Command-line, X11 events	Medium (window/input sim.) [136]	Good for X11	High (for direct input)	Via process execution
ydotool/wtype	Linux (Wayland)	Command-line, uinput/Wayland protocols	Medium (input sim.) [140]	Emerging, compositor dependent	High (for direct input)	Via process execution
Win32 API (SendInput, etc.)	Windows	C API	Low to High (depends on API used)	High (low-level)	Very High	Excellent (direct C/C++ calls)
AppleScript (GUI Scripting)	macOS	Scripting language	Medium to High (UI element	Good for scriptable apps	Low to Moderate	Via osascript process

			scripting) [113]			execution

* The native APIs provide the most granular control and reliability when correctly implemented. Windows UIA and macOS Accessibility API are mature and well-documented. Linux AT-SPI is powerful but its consistency can vary depending on the application toolkit's ATK/AT-SPI bridge implementation. Direct input simulation tools like xdotool are fast but less robust to UI changes. A combination of accessibility API for targeting and native input simulation for action execution is often optimal.

- **4.3.2. Cross-Platform Frameworks (for controlling other apps)**
  - Libraries like PyAutoGUI [142] or RustAutoGUI [141] attempt to provide a unified API for input simulation and basic window control. However, for deep UI element interaction (beyond coordinate-based clicks or image recognition), they often fall back to platform-specific mechanisms or have limited capabilities.
  - True cross-platform control at the level of UI element semantics (like ACE requires) is challenging without a deep abstraction layer over native accessibility APIs. AccessKit [121] is a step in this direction for *providing* accessibility, which could be leveraged by an automation tool for *consuming* it.
  - These are generally less suitable for the core, high-performance, deep UI interaction needed to surpass ACE, compared to direct native API utilization or a highly optimized custom abstraction layer.
- **4.3.3. Input Simulation**
  - **Mouse/Keyboard Events:** Essential for all platforms. Must support precise coordinate clicking, dragging, scrolling, typing individual characters, pressing modifier keys, and complex key combinations. Native OS functions (e.g., SendInput on Windows, CGEventPost on macOS, xdo_send_keysequence_window_down/up via libxdo for X11) provide the most reliable and low-level control.
  - **Touch Gestures:** If mobile device interaction or touch-enabled desktop apps are in scope, this will require platform-specific APIs (e.g., Android ADB shell input, iOS XCUITest framework). Not a primary focus for desktop automation but a future consideration.
  - **Multi-Monitor Support:** Crucial. The system must correctly map global screen coordinates to specific monitor coordinates and ensure actions are

performed on the intended display, especially when dealing with applications spanning multiple monitors or when the agent needs to move windows between monitors. OS APIs provide functions to enumerate displays and get their geometries.
    - The reliability of input simulation is paramount. Issues like incorrect focus, timing problems, or misinterpretation of coordinates can lead to task failure. The action execution module must ensure the target application window is active and the specific UI element is ready to receive input before simulation.
- **4.3.4. Browser Automation (WebDriver Alternatives)**
    - While general desktop automation techniques can control browser UIs, dedicated browser automation tools offer more robust interaction with web elements (DOM inspection, JavaScript execution, network interception).
    - **Playwright:** Strong cross-browser support (Chromium, Firefox, WebKit), multi-language APIs (Python, Java, JS, C#), auto-waiting, multi-context browsing, device emulation.[144] Generally faster and more reliable than Selenium. Its architecture supports parallel execution well.[145]
    - **Puppeteer:** Primarily for Chromium-based browsers (experimental Firefox). JavaScript/TypeScript only. Strong integration with Chrome DevTools Protocol.[144]
    - **Selenium:** Older, widely adopted, but often slower and can be less reliable for modern web apps.
    - **Recommendation:** For dedicated browser automation tasks (e.g., complex web scraping, filling intricate web forms, interacting with SPAs), integrate **Playwright**. Its feature set, cross-browser/language support, and performance make it the most suitable choice. For simple browser interactions (e.g., opening a URL, clicking a visible button), the general desktop automation capabilities might suffice.
- **4.3.5. Recommended Action Execution Strategy**
    1. **Core Language:** Rust for the action execution engine due to its performance, safety, and excellent FFI capabilities for interacting with native C/Objective-C APIs.
    2. **Abstraction Layer:** Develop a unified action execution API in Rust (e.g., execute_click(element_id, coordinates), execute_type(element_id, text)).
    3. **Platform-Specific Modules:** Underneath this Rust API, implement platform-specific modules:
        - **Windows:** Utilize the uiautomation Rust crate [105] or direct COM interop with UIA, and SendInput (via winapi crate) for input simulation.
        - **macOS:** Use Rust's objc crate to call Accessibility APIs (NSAccessibility) and core_graphics for input simulation (CGEventPost).

    - **Linux:** Interface with AT-SPI via D-Bus bindings in Rust (e.g., dbus crate). For input simulation, use libraries that wrap libxdo for X11 or explore Wayland-specific protocols/tools like wtype or ydotool (potentially via process execution if direct Rust bindings are immature).
4. **Input Simulation Primitives:** Implement highly reliable and precise mouse (move, click, drag, scroll) and keyboard (press, release, type string, shortcuts) primitives. These must handle window focus, coordinate systems (screen vs. window, DPI scaling), and multi-monitor environments.
5. **Browser Interaction:** For tasks explicitly identified as "web browser automation," the Rust core can invoke Playwright scripts (e.g., run a separate Playwright process or use language bindings if a Rust Playwright binding matures).
6. **Error Handling and Retries:** Implement robust error handling (e.g., element not found, action failed) and configurable retry mechanisms.
7. **Synchronization:** Ensure actions are synchronized with the UI state. For example, wait for an element to be enabled or visible before attempting to interact with it, using information from the Accessibility API.

This strategy prioritizes direct native control for performance and reliability, abstracted by a Rust core for cross-platform consistency, and leverages specialized tools like Playwright where appropriate.

4.4. Performance Optimization

Achieving sub-324ms latency requires aggressive optimization across the entire pipeline.

- **4.4.1. GPU Acceleration Frameworks**
    - **NVIDIA:** CUDA is the dominant platform for NVIDIA GPUs. Libraries like cuDNN, cuBLAS, and TensorRT are essential for optimizing deep learning model inference. The NVIDIA ACE Controller SDK leverages Pipecat, which integrates NVIDIA services, indicating a CUDA-centric approach for their hardware.[12]
    - **AMD:** ROCm is AMD's open-source platform for GPU computing, offering an alternative to CUDA.
    - **Apple:** Metal is Apple's low-level graphics and compute API for its GPUs, providing high performance on macOS and iOS devices.
    - **Cross-Platform:** Vulkan is a low-level, cross-platform graphics and compute API. OpenCL is another cross-platform standard, though often less performant than vendor-specific APIs for deep learning.
    - **Recommendation:** The VLM inference engine should primarily target CUDA

for NVIDIA GPUs due to the mature ecosystem and widespread adoption. For macOS, Metal support is crucial for on-device performance. For broader GPU support on Linux/Windows (especially AMD), exploring Vulkan compute shaders or ROCm (if model frameworks support it well) is necessary. The choice will be heavily influenced by the selected VLM's underlying deep learning framework (e.g., PyTorch, TensorFlow) and its backend support.

- **4.4.2. Model Serving Optimizations**
  - **TensorRT (NVIDIA):** Optimizes neural network models for inference on NVIDIA GPUs, significantly reducing latency and increasing throughput. It performs graph optimizations, layer fusion, and precision calibration.
  - **ONNX Runtime:** A cross-platform inference engine for models in the Open Neural Network Exchange (ONNX) format. It supports hardware acceleration across various platforms (CPUs, GPUs from NVIDIA, AMD, Intel) and execution providers (CUDA, TensorRT, DirectML, OpenVINO, ROCm, CoreML). This provides flexibility in deployment.
  - **OpenVINO (Intel):** Optimizes deep learning models for Intel hardware (CPUs, iGPUs, VPUs).
  - **Recommendation:** Convert the fine-tuned VLM to ONNX format. Use **ONNX Runtime** as the primary inference engine due to its cross-platform and cross-hardware capabilities. For NVIDIA-specific deployments where maximum performance is critical, leverage the TensorRT execution provider within ONNX Runtime or use TensorRT directly. For Apple Silicon, explore Core ML execution provider via ONNX Runtime.
- **4.4.3. Caching Strategies for UI States and Actions**
  - **UI State Cache:** Cache the recognized structure of frequently encountered application windows or UI views (e.g., element hierarchies, properties, locations obtained from accessibility APIs or VLM perception). This avoids re-processing unchanged UI regions. A Least Recently Used (LRU) cache with a mechanism to invalidate entries when UI changes are detected (e.g., via accessibility events) would be suitable.
  - **Action Sequence Cache:** For repetitive multi-step tasks identified by the VLM (e.g., "login to X app," "save current file"), cache the sequence of predicted actions. If the initial state matches a cached entry, the sequence can be replayed, bypassing VLM reasoning for those steps. This is a form of learned macro.
  - **VLM Prediction Cache:** Cache VLM outputs (predicted actions) for specific (screen state, prompt) inputs if they are highly deterministic and frequently recurring. This is more complex due to the high dimensionality of screen states but could be beneficial for very common interactions.

    - Effective caching requires robust mechanisms for cache key generation (e.g., hashing relevant parts of the UI state) and cache invalidation to ensure stale data isn't used.
- **4.4.4. Real-Time Inference Pipelines and Streaming Architectures**
    - The entire process from screen capture to action execution must be designed as a low-latency stream.
    - **Pipelining Stages:** Overlap execution of different stages. For example:
        - Stage 1: Screen Capture + Accessibility Data Fetch.
        - Stage 2: OCR (on demand) + VLM Input Preparation.
        - Stage 3: VLM Inference (Perception + Reasoning).
        - Stage 4: Action Execution + Next State Capture Trigger.
    - **Asynchronous Operations:** Use asynchronous programming extensively (e.g., Rust's async/await, Go's goroutines, C++ with coroutines or thread pools) to prevent blocking operations from stalling the pipeline. Screen capture, VLM inference, and action execution can often run in parallel to some extent.
    - **Frame Buffering/Queueing:** Use efficient, bounded queues between stages to manage data flow and absorb minor timing variations.
    - The goal is to ensure that a new cycle of perception-reasoning-action can begin as quickly as possible, ideally processing screen updates at a high frame rate if necessary for highly dynamic UIs (though for most desktop apps, changes are event-driven).
- **4.4.5. Distributed Computing for Complex Automations**
    - While core interaction loops must be low-latency and likely local, very complex, multi-step tasks or tasks requiring extensive knowledge retrieval could benefit from distributed components.
    - **Hybrid Edge-Cloud Model:** A lightweight agent on the user's device handles most interactions. For tasks requiring, for instance, deep research, summarization of large documents, or complex planning beyond the local VLM's capability, the local agent could delegate sub-problems to a more powerful backend VLM in the cloud. The cloud VLM returns a plan or intermediate result, and the local agent executes the UI interactions. This aligns with strategies for deploying effective AI agents by integrating with various systems and potentially offloading intensive computations.[83]
    - This requires a robust communication protocol (e.g., gRPC) between the local agent and backend services.

4.5. Programming Languages and Development Frameworks

The choice of programming languages and frameworks is critical for achieving

performance, safety, and developer productivity.

- **4.5.1. Core Application Development**
  - **C++:**
    - **Pros:** Unmatched performance potential, direct system-level access, extensive libraries, mature tooling for real-time systems.[147]
    - **Cons:** Manual memory management (prone to errors like leaks, buffer overflows, dangling pointers), complex, steeper learning curve for modern C++, build times can be long.[147]
  - **Rust:**
    - **Pros:** Performance comparable to C++ but with compile-time memory safety (ownership and borrowing) eliminating garbage collectors and many common C++ bugs. Fearless concurrency (data race prevention at compile time). Modern tooling (Cargo, crates.io). Excellent FFI for C/C++ interop. Growing ecosystem for system programming, including UI automation (e.g., uiautomation crate for Windows [105], AccessKit [121], scap [86]).
    - **Cons:** Steeper learning curve due to ownership model. Ecosystem still younger than C++ in some specialized areas, though rapidly maturing.[147] Compilation times can be longer than Go or Python.
  - **Go (Golang):**
    - **Pros:** Excellent built-in concurrency (goroutines, channels), simple syntax, fast compilation, easy deployment (static binaries). Good for networked services and backend systems.[149] Garbage collected, simplifying memory management compared to C++.
    - **Cons:** Performance for raw CPU-bound tasks generally lower than C++/Rust. GC pauses can be an issue for strict real-time guarantees. AI/ML ecosystem significantly less mature than Python's.[151] FFI capabilities are less straightforward than Rust's.
  - **Python:**
    - **Pros:** Unparalleled AI/ML ecosystem (TensorFlow, PyTorch, Hugging Face Transformers, LangChain, etc. [151]). Rapid prototyping and development. Large number of UI automation libraries (PyAutoGUI, pywinauto).
    - **Cons:** Global Interpreter Lock (GIL) limits true parallelism for CPU-bound tasks. Higher memory consumption. Slower execution speed for performance-critical sections compared to C++/Rust/Go. Not ideal for low-level system components requiring minimal overhead.
  - **Table: Programming Language Suitability Matrix**

Feature	C++	Rust	Go	Python
Raw Performance (CPU)	Very High	Very High	High	Low to Medium
Memory Safety	Manual (Low)	High (Compile-time)	Medium (GC)	Medium (GC)
Concurrency Model	Complex (Manual threads)	High (Fearless, async/await, ownership)	Very High (Goroutines, channels)	Medium (Asyncio, GIL limitations)
AI/ML Ecosystem	Medium (libs like dlib)	Growing (e.g., tch-rs, Candle)	Low	Very High (PyTorch, TensorFlow, Hugging Face) [151]
System-Level Access	Very High	Very High	High	Medium (via ctypes, etc.)
Cross-Platform Dev/Deploy	Medium (build systems)	High (Cargo)	Very High (static binaries)	Medium (venv, dependencies)
Developer Productivity	Medium	Medium to High (steep initial curve)	High	Very High (for scripting/AI)
Community & Talent Pool	Very Large	Growing Rapidly	Large	Very Large
Suitability for Core Engine	High	**Very High (Recommended)**	Medium	Low
Suitability for AI Orchestration	Low	Medium	Medium	**Very High (Recommended)**

* A hybrid language approach is strongly recommended. **Rust** is the prime candidate for the performance-critical core agent (perception, action execution, native API interactions) due to its blend of C++-level performance and memory safety without GC overhead.[147, 148] Its strong FFI capabilities allow seamless integration with Python. **Python** should be used for the higher-level AI orchestration, VLM interaction, task management, and scripting, leveraging its rich AI/ML ecosystem.[151, 153, 154] This allows for rapid iteration on AI logic while maintaining a high-performance, safe core. C++ could be an alternative to Rust for the core if the team has deep existing expertise and robust practices for managing memory safety, but Rust offers compelling advantages for a new system. Go is excellent for backend microservices if that architectural path is chosen.

- **4.5.2. Frontend Technologies (Agent's Own UI)**
  - If the agent requires a GUI for configuration, monitoring, or user interaction:
    - **Dear ImGui:** Lightweight, immediate-mode GUI library. Excellent for developer tools, debug interfaces, and simple UIs embedded within a C++/Rust application. Very performant but not for highly complex, polished end-user applications.[155] Hello ImGui simplifies multiplatform Dear ImGui development.[155]
    - **Qt:** Mature, feature-rich, cross-platform framework for complex desktop applications. Steeper learning curve and licensing considerations (LGPL/Commercial).[156] Rust bindings exist (e.g., CXX-Qt [120]).
    - **Tauri:** Builds desktop applications using web technologies (HTML, CSS, JS) for the frontend and a Rust backend. Uses system WebViews, resulting in smaller binaries and lower memory usage compared to Electron.[159] A good option if a modern web-based UI is desired and Rust is already in the stack. Startup time is fast, comparable to Electron, but initial build times can be slower due to Rust compilation.[159] Memory usage is significantly lower (e.g., 172 MB vs 409 MB for Electron in one benchmark with 6 windows).[159]
    - **WebAssembly (WASM):** Can run C++/Rust code in a browser or WebView at near-native speed. Good for performance-critical UI components within a web-based frontend.[163]
    - **Neutralinojs:** Another lightweight Electron alternative, allows using any backend language that can output to a socket. Focuses on small memory/CPU footprint.[161]
  - **Table: Frontend Technology Comparison**

Technology	Rendering Performance	Resource Usage	Cross-Platform	Dev Complexity	UI Richness	Integration (Rust Core)
Dear ImGui	Very High	Very Low	Yes	Low-Medium	Basic-Medium	Excellent
Qt (via CXX-Qt)	High	Medium-High	Yes	High	Very High	Good
Tauri (WebView)	Good	Low-Medium	Yes	Medium	High (Web)	Excellent (Rust backend)
Native SDKs	Very High	Low-Medium	No (Per OS)	High	Very High	Good (via FFI)
WASM + Web UI	Variable (High for WASM parts)	Medium	Yes (Browser)	High	High (Web)	Good

* **Recommendation:** For the agent's own UI (if needed beyond a simple CLI or tray icon), **Tauri** with a Rust backend offers the best balance of modern UI capabilities (using web tech), performance, resource efficiency, and seamless integration with a Rust core. For internal debugging or highly performance-sensitive overlay UIs, **Dear ImGui** is an excellent choice.

- **4.5.3. Backend Architecture**
  - **Microservices vs. Monolith:** For a system aiming for 100M+ users, a **microservices architecture** is highly recommended for scalability, maintainability, independent deployment of components (e.g., VLM serving, data ingestion, user management), and fault isolation.[165] While initial development might be more complex than a monolith, the long-term benefits for a large-scale AI system are substantial. Each microservice can be scaled independently based on demand.[165]

- **Message Queue Systems:**
  - **Apache Kafka:** Ideal for high-throughput, durable event streaming. Suitable for collecting behavioral data from millions of agents, log aggregation, and feeding real-time data to training pipelines and analytics systems.[169] Kafka's log-structured persistence and replay capabilities are valuable for training data and auditing.
  - **RabbitMQ:** Good for complex routing scenarios and when strong message delivery guarantees are needed for inter-service communication, though Kafka can also provide strong guarantees.[171]
  - **NATS:** Lightweight, very high performance, and low latency. Excellent for command-and-control messages, service-to-service RPCs, or telemetry where extreme speed is prioritized over Kafka's stronger durability/replay features for every message.[171] NATS JetStream adds persistence.[172]
- **Real-time Communication (Agent-Backend / Inter-Service):**
  - **gRPC:** Built on HTTP/2, uses Protocol Buffers for efficient binary serialization. Supports bidirectional streaming, multiplexing (reducing latency), and is strongly typed. Excellent for low-latency, high-performance inter-service communication within a microservices architecture and potentially for agent-to-backend communication if the agent is not browser-based.[173]
  - **WebSockets:** Establishes a persistent, full-duplex connection. Good for real-time bidirectional communication between a client (e.g., agent UI if web-based) and backend. More textual and potentially higher overhead than gRPC if using JSON payloads.[173]
- **Database Choices:**
  - **Action History/User Data:** A distributed SQL database like PostgreSQL with Citus, or a NoSQL document database like MongoDB, depending on query patterns and consistency needs.
  - **Behavioral Data/Telemetry:** A data lake solution (e.g., S3 + Spark/Presto/Trino) for raw data, feeding into a data warehouse (e.g., Snowflake, BigQuery, Redshift) for analytics and ML training.
  - **VLM Embeddings/Vector Search:** Specialized vector databases (e.g., Pinecone, Weaviate, Milvus) or extensions for existing databases (e.g., pgvector for PostgreSQL).
  - **UI State/Element Cache:** A fast in-memory cache like Redis or Memcached.
- **Table: Backend Technology Comparison**

Category	Technology	Throughput	Latency	Reliability/Persistence	Scalability	Use Case for Agent System
Architecture	Microservices	High	Low (indiv. services)	High (fault isolation)	Very High	Core backend structure for all components [165]
	Monolith	Medium	Medium	Low (single PoF)	Medium	Not recommended for target scale
Message Queues	Kafka	Very High	Low-Medium	Very High (durable log)	Very High	Behavioral data ingestion, event sourcing, async task queue [169]
	RabbitMQ	High	Low-Medium	High (flexible routing)	High	Complex inter-service messaging, task queues [171]
	NATS	Very High	Very Low	Medium (Core) / High (JetStream)	Very High	Low-latency commands, telemetry, service discovery [171]
Real-time Comm.	gRPC	High	Very Low	High	High	Inter-service RPCs,

						potentially agent-backend control channel [173]
	WebSockets	Medium	Low	Medium	High	Agent UI to backend (if web UI), real-time status updates [173]

* **Recommendation:** A **microservices architecture** is essential. Use **Kafka** for high-volume asynchronous data streams (telemetry, training data). Employ **gRPC** for low-latency synchronous communication between backend services. If the agent has a web-based UI or needs a persistent connection for real-time updates from the backend, **WebSockets** can be used for that specific path. NATS could be considered for specific ultra-low-latency internal messaging if Kafka/gRPC proves insufficient for certain hot paths. The database choices will be heterogeneous, tailored to the needs of each service.

5. System Architecture Design

Based on the comprehensive analysis, the proposed system architecture aims for high performance, scalability, and modularity, leveraging a hybrid language approach and a microservices backend.

5.1. Overall System Architecture

The system will consist of three main parts: the **Client-Side Agent Application**, optional **Edge Services**, and the **Backend Cloud Platform**.

Kodavsnitt

```
graph TD
    A[User] -- Prompts/Interactions --> B(Client-Side Agent Application);
    B -- Screen Data/Local VLM Inference --> C{VLM Processing Core (Rust)};
    C -- Low-Level Actions --> D(Action Execution Engine (Rust));
    D -- Native OS Calls --> E;
    E -- UI Changes --> F(Screen Capture (Rust));
    F --> C;
    B -- Telemetry/Complex Tasks --> G(Backend Cloud Platform - Microservices);
    G -- Model Updates/Guidance --> B;
    G -- Scalable VLM Inference/Data Storage/Training --> G;

    subgraph Client-Side Agent Application
        C; D; F;
        H[Perception Module (CV+Accessibility)];
        I[Local VLM/Action Predictor];
        J;
        K[Caching Module];
        C --> H; H --> I; I --> C; C --> J; J --> G; C --> K; K --> C;
    end

    subgraph Backend Cloud Platform - Microservices
        L[API Gateway];
        M;
        N;
        O;
        P;
        Q;
        R;
        S;
        L --> M; L --> N; N --> O; O --> N;
        B --> P; P --> Q; Q --> O; P --> R;
        O --> S; Q --> S; R --> S;
    end
```

Data Flows:

1. **User Interaction:** User provides a prompt or interacts with an application.

2. **Client Agent - Perception:** The Client-Side Agent captures the screen, extracts accessibility data, and performs OCR if needed. This data, along with the user prompt, is fed to the Local VLM/Action Predictor.
3. **Client Agent - Local Reasoning/Delegation:**
    - For simple/common actions, the Local VLM predicts the action directly (target <324ms).
    - For complex tasks, the request is sent to the Backend Cloud Platform via the Communication Module.
4. **Backend - Complex Reasoning:** The Task Orchestration Service in the backend may use the Core VLM Reasoning Service to generate a plan or a specific action. This is then sent back to the client.
5. **Client Agent - Action Execution:** The VLM Processing Core (local or guided by backend) sends a concrete action (e.g., click coordinates, text to type) to the Action Execution Engine.
6. **Client Agent - OS Interaction:** The Action Execution Engine uses native OS APIs to perform the action on the target application.
7. **Feedback Loop:** UI changes resulting from the action are captured by the Screen Capture module, restarting the perception cycle.
8. **Telemetry & Training Data:** Anonymized interaction data (screenshots, actions, outcomes, latency metrics) is sent to the Behavioral Data Ingestion Service in the backend for model improvement and analytics.

5.2. Component-Level Design

- **5.2.1. Client-Side Agent Application (Rust Core with Python AI Orchestration)**
    - **Perception Module (Rust):**
        - Real-time screen capture (e.g., using scap library [86]).
        - Cross-platform Accessibility API interface (Rust, leveraging AccessKit principles [121] for Windows UIA, macOS NSAccessibility, Linux AT-SPI).
        - Local OCR engine (e.g., Tesseract bindings for Rust) for fast text extraction from specific regions.
        - Data assembler to prepare multimodal input for the VLM.
    - **VLM Processing Core (Rust with Python FFI for VLM):**
        - Manages interaction with the local VLM.
        - If Python is used for VLM, Rust calls Python functions via FFI (e.g. PyO3).
        - Handles visual grounding logic, possibly calling specialized Rust functions for performance.
    - **Local VLM/Action Predictor (Python/ONNX Runtime):**
        - A highly optimized, quantized, and potentially distilled version of the main

VLM (e.g., optimized CogAgent, InternVL, or Fuyu-8B variant).
        - Served by ONNX Runtime with GPU acceleration (Metal on macOS, DirectML/OpenVINO on Windows, Vulkan/ROCm on Linux where available, CUDA as primary).
        - Focuses on predicting common, single-step actions with very low latency.
    - **Action Execution Module (Rust):**
        - Translates VLM action predictions into native OS calls.
        - Direct input simulation (mouse clicks, keystrokes, scrolling) using platform-specific APIs (e.g., SendInput on Windows, CGEventPost on macOS, xdotool/ydotool equivalents via Rust).
        - Window management (focus, move, resize).
        - Interface for Playwright for complex browser tasks.
    - **Communication Module (Rust):**
        - Secure gRPC or HTTPS/WebSockets for communication with the backend (task delegation, telemetry, model updates).
    - **Caching Module (Rust):**
        - LRU caches for UI element data (from accessibility APIs), OCR results, and potentially common VLM action predictions for specific UI states.
- **5.2.2. Edge Services (Optional)**
    - Deployed in regional data centers or near-user locations.
    - Could host regional instances of the Core VLM Reasoning Service for lower latency access for users in that region.
    - May perform localized data aggregation before forwarding to the central backend.
    - Leverage CDN capabilities for faster model delivery/updates to clients.[68]
- **5.2.3. Backend Services (Microservices - Go or Rust for performance, Python for ML/Data)**
    - **API Gateway (e.g., Kong, Spring Cloud Gateway, or custom Go/Rust):** Single entry point for all client requests. Handles routing, authentication, rate limiting.
    - **User Authentication & Management Service (Go/Rust):** Manages user accounts, authentication tokens, permissions.
    - **Task Intake & Orchestration Service (Go/Python):** Receives complex tasks from clients, breaks them down into sub-tasks, and coordinates with other services (especially Core VLM).
    - **Core VLM Reasoning Service (Python with C++/Rust bindings for inference):** Hosts the large, powerful VLM (e.g., full CogAgent, InternVL3, or a powerful commercial model if API-based). Uses GPU-accelerated serving (e.g., Triton Inference Server, TorchServe, or custom ONNX Runtime serving).

    - **Behavioral Data Ingestion & Processing Service (Kafka + Go/Python consumers):** Receives anonymized interaction data from clients. Validates, preprocesses, and stores data for training and analytics. Kafka provides the scalable, durable message queue.[169]
    - **Model Training & Fine-tuning Service (Python + MLOps tools):** Manages the ML training lifecycle. Uses frameworks like Kubeflow, MLflow. Periodically retrains/fine-tunes VLMs based on new data.
    - **Telemetry & Analytics Service (e.g., Prometheus, Grafana, ELK stack, custom Python/Go service):** Collects and analyzes system performance metrics, user engagement, error rates.
    - **Databases:**
        - **User/Metadata DB:** PostgreSQL or CockroachDB for user profiles, task history.
        - **Behavioral Data Lake:** Cloud storage (S3, GCS, Azure Blob).
        - **Vector DB:** For similarity search on VLM embeddings (e.g., UI states, prompts).
        - **Cache:** Redis for session data, frequently accessed metadata.

5.3. API Design and SDK Strategy

- **Internal APIs (gRPC):** All inter-service communication within the backend will use gRPC for its performance, strong typing (Protobufs), and streaming capabilities.[173] Client-to-backend communication for task delegation and telemetry can also use gRPC.
- **Developer API/SDK (External):**
    - A public RESTful API (with OpenAPI specification) or gRPC API for developers to programmatically control the agent, submit tasks, and retrieve results.
    - Client SDKs in key languages (Python, JavaScript/TypeScript initially, potentially C# and Java later) to simplify integration.
    - The SDK should allow:
        - Starting/stopping automation tasks.
        - Sending natural language prompts.
        - Subscribing to task status updates and results.
        - Potentially defining custom tools or knowledge bases for the agent.
        - Access to (anonymized and aggregated) performance metrics.
    - This contrasts with ACE's current limited partner access [2] and aims to foster an ecosystem.

5.4. Data Management

- **Action History:** Store user commands, VLM-predicted actions, screenshots (or

hashes/embeddings), element identifiers, and outcomes. Stored in a scalable database (e.g., Cassandra or a distributed SQL DB) for debugging, user history, and fine-tuning.
- **User Profiles:** Store user preferences, custom shortcuts, application-specific configurations, and consent settings. Use a relational or document database.
- **Behavioral Data Lake:** Collect vast amounts of anonymized interaction data (sequences of (screenshot, prompt, accessibility_tree_snapshot, action, outcome, latency)) for continuous VLM training. Store in a cost-effective object store (S3/GCS/Azure Blob) with a data catalog.
- **VLM Models:** Store VLM checkpoints and optimized versions (ONNX, TensorRT plans) in a model repository (e.g., MLflow Model Registry, S3 versioned buckets).
- **UI Element Cache:** Client-side or edge-side cache for frequently seen UI element properties (from accessibility or VLM perception) to speed up re-identification.

5.5. Modularity and Extensibility

- **Microservices Architecture:** Inherently modular. Each service can be developed, deployed, and scaled independently.
- **Client-Side Agent Plugins:** Design the client agent with a plugin architecture. This would allow:
  - Adding support for new applications with custom interaction logic.
  - Integrating new perception modules (e.g., specialized OCR for a specific font).
  - Allowing third-party developers to create and share plugins.
- **Tool Use Framework:** Implement a robust tool use framework within the VLM, allowing it to call external APIs or run local scripts as part of its action plan. This greatly enhances extensibility. InternVL3 shows improvements in tool usage.[50]

The adoption of a dual-VLM strategy—a lean, fast client-side model for common interactions and a powerful backend model for complex reasoning—is a cornerstone of this architecture. This addresses the dual requirements of ultra-low latency for typical operations and high cognitive ability for challenging tasks. The entire system is designed around a continuous data flywheel: user interactions generate rich behavioral data, which is anonymized and used to iteratively refine the VLMs. This feedback loop is critical for achieving and maintaining a competitive edge in accuracy and capability, as highlighted by ACE's own statements about accuracy improving with more training resources.[1] Furthermore, a strong API-first approach for external developers, unlike ACE's current closed model [25], will foster an ecosystem, driving broader adoption and innovation.

6. Latency Optimization Strategy (Target: <324ms)

Achieving an action prediction and execution latency below ACE's 324ms [2] is a critical objective. This requires a multi-faceted strategy focusing on every stage of the automation pipeline.

6.1. Bottleneck Analysis in Current Systems

ACE's ace-control-small model reportedly achieves 324ms latency, while their ace-control-medium is at 533ms. Competing systems like Operator (6385ms) and Claude 3.7 Sonnet (9656ms) are significantly slower on ACE's reported benchmarks.[2] The primary bottlenecks in such systems typically include:

1. **VLM Inference Time:** Large VLMs, even optimized, can take hundreds of milliseconds to seconds for inference.
2. **Screen Capture and Preprocessing:** Capturing high-resolution screens and preparing them for the VLM (resizing, normalization) adds latency.
3. **Computer Vision Processing:** If separate CV models are used for OCR or element detection before VLM input, their inference time adds to the total.
4. **Action Planning/Decision Logic:** The VLM's reasoning process to decide the next action.
5. **Action Execution Overhead:** The time taken by the OS or automation framework to physically simulate the mouse/keyboard input.
6. **Network Latency:** If any part of the reasoning loop involves cloud communication.

General Agents' ACE likely achieves its speed through a highly specialized VLM trained end-to-end on behavioral data, minimizing discrete processing steps, and potentially running a very optimized model locally.[1]

6.2. End-to-End Latency Budget for Proposed System

To consistently beat 324ms, a target of **<250ms** for the entire perception-to-action loop is desirable for common interactions. This can be broken down:

- **Screen Capture & Accessibility Data Fetch:** < 20ms (native OS calls are fast)
- **Input Assembly & Local VLM Preprocessing:** < 30ms
- **Local VLM Inference (Perception & Action Prediction):** < 150ms
- **Action Execution (Native OS Call):** < 30ms
- **Internal Overheads & Communication:** < 20ms
    - **Total Target:** < 250ms

This budget necessitates a highly optimized local VLM for most interactions.

6.3. Advanced Inference Optimization Techniques

- **Model Quantization:**
  - Employ techniques like **AWQ (Activation-aware Weight Quantization)** or **GPTQ (Generalized Post-Training Quantization)** to reduce model precision to INT4 or INT8.[60]
  - AWQ is particularly promising as it's hardware-friendly, doesn't require backpropagation, and has shown good results for VLMs.[61] TinyChat, an inference framework for AWQ, offers >3x speedup for 4-bit models.[61]
  - These techniques can reduce model size by 4-8x and inference latency significantly, often by 30-75% or more.[61]
- **Model Pruning:**
  - Utilize **structured pruning** to remove entire neurons, channels, or attention heads, which yields actual speedups on standard hardware.[60]
  - Frameworks like **OSSCAR** offer one-shot structured pruning for very large models, scalable to tens of billions of parameters.[73]
  - Pruning can reduce FLOPs and latency, often by 30-50% or more depending on sparsity.[68]
- **Knowledge Distillation:**
  - Train a smaller, faster "student" VLM to mimic the behavior of a larger, more capable "teacher" VLM using the collected behavioral data.[60]
  - This is key for the proposed dual-VLM architecture, where the local VLM is a distilled version optimized for speed. MIND [71] and SPORT [72] offer insights into distillation for multimodal agents.
- **Optimized Runtimes:**
  - Use ONNX Runtime with hardware-specific execution providers (TensorRT for NVIDIA, CoreML for Apple, DirectML for Windows, OpenVINO for Intel).[177]
  - Techniques like kernel fusion (combining multiple operations into a single kernel) and efficient weight packing (as in TinyChat [61]) reduce memory bandwidth bottlenecks and kernel launch overhead.

5.4. Hardware Acceleration Strategies

- **Client-Side GPUs:** Leverage integrated and discrete GPUs available on user desktops.
  - NVIDIA: CUDA.
  - AMD: ROCm (Linux/Windows), Vulkan.
  - Apple: Metal.

    - Intel: OpenVINO for iGPUs, Vulkan. The client application must detect available hardware and select the optimal execution path.
- **NPUs (Neural Processing Units):** Modern CPUs and SoCs (especially in laptops and future desktops) include NPUs designed for AI acceleration. Utilizing these for parts of the VLM or specific CV tasks can offload the GPU and reduce power consumption. Windows DirectML and Apple's Neural Engine are key APIs.
- **Cloud/Edge TPUs/GPUs:** For backend VLM processing or delegated complex tasks, leverage high-performance server-grade GPUs (NVIDIA A100/H100, AMD MI-series) or TPUs (Google Cloud). Edge servers can be equipped with inference accelerators like NVIDIA Jetson or Google Edge TPUs.[76]
- **Custom Silicon (Long-term):** For ultimate performance and efficiency, developing custom silicon (ASIC or FPGA-based accelerator) tailored to the VLM architecture could be a long-term strategy, though this involves significant investment and development time.

5.5. Predictive Caching and Speculative Execution

- **Predictive Caching:**
    - Based on user behavior patterns and current application context, proactively fetch and cache UI information (accessibility trees, screenshots of anticipated next screens) or pre-load parts of the VLM relevant to likely next tasks.
    - If the VLM predicts a sequence of actions, the UI state resulting from the first few actions can be speculatively fetched or rendered to warm caches for subsequent steps.
- **Speculative Execution:**
    - If the VLM identifies a high-probability next action (e.g., clicking "OK" on a common dialog), the system could speculatively *begin* executing this action (e.g., moving the mouse towards the button) while the VLM confirms or refines subsequent steps. This is risky and requires careful confidence thresholding and rollback mechanisms.
    - A safer form is speculative *perception*: if a click action is predicted, start capturing and processing the screen area around the click target immediately, anticipating the UI change.
    - This can shave off crucial milliseconds by overlapping decision-making with action preparation or early execution stages.

5.6. Network Optimization (for Hybrid Cloud/Edge Deployments)

- If parts of the VLM reasoning are offloaded to the cloud/edge:
    - Use efficient communication protocols like gRPC with Protocol Buffers.[173]
    - Compress data payloads (e.g., screen data, VLM inputs/outputs). Techniques

like sending screen diffs instead of full screenshots can reduce bandwidth.
    - Utilize CDNs or edge servers to cache model parameters or serve inference requests closer to the user, reducing network latency.[68] Cloudflare Workers AI and Vectorize are examples of CDN-based AI services.[176]

The combination of aggressive model optimization (quantization, pruning, distillation), deployment on local hardware accelerators (GPU/NPU), intelligent caching, a streaming architecture, and potentially a dual-VLM approach (fast local model + powerful backend model) is essential to consistently achieve the sub-324ms latency target. Each millisecond saved in every component of the perception-reasoning-action loop contributes to the overall goal.

6. Scalability and Infrastructure

Supporting potentially 100M+ users requires a highly scalable and resilient infrastructure, likely involving a hybrid cloud-edge architecture.

6.1. Cloud Provider Comparison for AI Workloads (AWS vs. Azure vs. GCP)

- **AWS:**
    - **Services:** Broad and deep service portfolio. Amazon SageMaker for ML model building/training/deployment. Amazon Bedrock for accessing foundation models (including Anthropic's Claude).[178] EC2 instances with various GPUs (NVIDIA). AWS RoboMaker for robotics (relevant for agent concepts).
    - **Networking:** VPC, Direct Connect, CloudFront CDN (over 600 PoPs).[178]
    - **Pricing:** Per-second billing for many services (min 60s for EC2). Spot Instances offer up to 90% discount but can be interrupted. Spot prices are highly dynamic.[179]
- **Azure:**
    - **Services:** Strong enterprise focus. Azure Machine Learning for MLOps. Azure AI Foundry for accessing AI models. Azure OpenAI Service for GPT models.[178] GPU instances available.
    - **Networking:** Virtual Network, ExpressRoute, Azure CDN.[178]
    - **Pricing:** Per-second billing for some container instances. Reserved Savings for 1-year commitments. Spot pricing is more predictable than AWS.[179] Arm CPUs on Azure offer significant cost savings.[179]
- **GCP (Google Cloud Platform):**
    - **Services:** Leader in AI/ML. Vertex AI for end-to-end MLOps. Access to Google's Gemini models. TensorFlow and TPUs (Tensor Processing Units) for high-performance training and inference.[178] Cloud Functions, Cloud Run for serverless.

    - **Networking:** Superior global network infrastructure.
    - **Pricing:** Per-second billing. Committed Use Discounts. Preemptible VMs (similar to Spot Instances) offer up to 80% discount. Spot pricing is more predictable than AWS.[179]
- **Table: Cloud Provider AI/ML Service Comparison**

Feature	AWS	Azure	GCP
Primary ML Platform	SageMaker	Azure Machine Learning	Vertex AI [178]
Foundation Model Access	Bedrock (Claude, Meta, etc.) [178]	Azure AI Foundry, Azure OpenAI Service [178]	Vertex AI Model Garden (Gemini, etc.)
Specialized AI Hardware	NVIDIA GPUs, Inferentia, Trainium	NVIDIA GPUs, FPGAs	NVIDIA GPUs, Google TPUs [178]
Spot/Preemptible Instances	Spot Instances (up to 90% off, dynamic pricing) [179]	Spot Virtual Machines (less dynamic pricing)	Preemptible VMs (up to 80% off, more predictable pricing) [179]
Strengths for VLMs	Broad GPU availability, mature MLOps (SageMaker)	Strong OpenAI integration, enterprise focus	Cutting-edge AI/ML research, TPUs for large-scale training/inference, strong network

- **Recommendation:** GCP is particularly strong for AI/ML workloads due to TPUs, Vertex AI, and its advanced network. However, all three major providers offer competitive GPU instances and MLOps platforms. The choice may depend on existing enterprise relationships, specific pricing models for the required scale, and availability of specialized hardware like latest-generation TPUs or GPUs in desired regions. A multi-cloud strategy, while complex, could also be considered for resilience and cost optimization. For serving VLMs, factors like inference-optimized instances and managed model serving endpoints (e.g., SageMaker Endpoints, Vertex AI Endpoints, Azure ML Endpoints) are crucial.

6.2. Edge Computing Strategies for Reduced Latency

- **Purpose:** Process data closer to the user to reduce latency, conserve bandwidth, and enhance privacy/reliability.[64] Edge AI can reduce round-trip latency by up to 80% in some scenarios.[80]
- **Client-Side (On-Device) Processing:** Run a highly optimized, smaller VLM directly on the user's device (PC).
  - **Pros:** Lowest possible latency for common actions, offline capability, enhanced privacy.
  - **Cons:** Limited by user's hardware (CPU, GPU, RAM, NPU), model size/complexity constraints, battery consumption on laptops.
- **Near-User Edge Servers (e.g., Telco Edge, Local Data Centers):** Deploy VLM inference servers in locations geographically close to user concentrations.
  - **Pros:** Lower latency than central cloud, more compute power than client devices. Telco infrastructure (regional data centers, CDN nodes, near-RAN sites) can be leveraged as hierarchical AI edges for caching and partial/full inference.[68]
  - **Cons:** Infrastructure cost, management complexity of distributed servers.
- **Hardware:** Utilize edge-specific AI accelerators (NVIDIA Jetson, Google Edge TPU, Intel Movidius).[80]
- **Model Optimization for Edge:** Essential. Quantization, pruning, distillation are critical to fit models onto resource-constrained edge devices.[64]
- **Red Hat Solutions for Edge:** Red Hat Device Edge (RHEL + MicroShift) for constrained environments, Single Node OpenShift for more powerful edge locations.[82]

6.3. Hybrid Cloud-Edge Architectures

- **Concept:** Combine the strengths of local/edge processing for low-latency tasks and cloud processing for complex, resource-intensive tasks or centralized data management/training.[82]
- **Strategy for Computer Automation Agent:**
  1. **Local Agent Dominance:** The client-side agent with its optimized local VLM handles the majority of UI interactions to achieve sub-324ms latency.
  2. **Edge Fallback/Augmentation:** If the local VLM lacks confidence or the task requires slightly more resources, it can query a nearby edge server hosting a slightly larger/more capable VLM.
  3. **Cloud for Complexity & Learning:** For very complex planning, tasks requiring extensive external knowledge, or initial processing of novel UIs, the agent can query the powerful Core VLM in the central cloud. The cloud also handles aggregation of behavioral data and model retraining.
- **Data Flow:** User interactions primarily processed locally. Complex queries or data

for retraining are sent to the cloud. Model updates are pushed from the cloud to edge servers and clients.

- **Benefits:** Balances latency, capability, cost, and privacy. Allows for graceful degradation if cloud connectivity is lost (local agent can still perform many tasks). Crowe LLP emphasizes iterative deployment, starting small and integrating with existing systems, which aligns with a hybrid approach.[83]

6.4. Content Delivery Networks (CDNs) for Global Distribution

- **Purpose:** CDNs traditionally cache static content closer to users. For AI applications, they can be evolved to:
  - Deliver VLM model files and updates efficiently to globally distributed clients and edge servers.[68]
  - Host AI inference tasks at the edge (CDN edge compute). Cloudflare Workers AI and Vercel AI SDK are examples of CDNs offering AI inference capabilities and tools.[176]
  - Cloudflare uses its global network of NVIDIA GPUs for edge inference and offers Vectorize (vector DB) and AI Gateway.[176]
- **Recommendation:** Utilize a CDN with edge compute capabilities (like Cloudflare or Akamai with Linode GPU instances [176]) for:
  - Distributing client application installers and updates.
  - Distributing VLM model files (especially updates to local/edge VLMs).
  - Potentially hosting lightweight inference tasks or API gateways at the CDN edge for ultra-low latency in specific regions.

6.5. Load Balancing Strategies for Millions of Concurrent Users

- **Necessity:** Essential for distributing traffic across backend services and edge inference nodes to prevent overload and ensure high availability.[180]
- **Traditional Techniques:** Round-robin, least connections, least response time [[181],

Citerade verk

1. Ace, the Real-Time AI Agent That Uses Your Computer for You ..., hämtad maj 23, 2025, https://digialps.com/ace-the-real-time-ai-agent-that-uses-your-computer-for-you/
2. Introducing Ace - General Agents, hämtad maj 23, 2025, https://generalagents.com/ace/
3. Compare Ace vs. Open Computer Agent in 2025 - Slashdot, hämtad maj 23, 2025, https://slashdot.org/software/comparison/Ace-AI-Agent-vs-Open-Computer-Agent/

4. About - General Agents, hämtad maj 23, 2025, https://generalagents.com/about/
5. Ace Realtime Copilot : Watch Your Computer Tasks Completed at ..., hämtad maj 23, 2025, https://www.youtube.com/watch?v=ze_46cn0A5g&vl=hi
6. Introducing Ace - General Agents, hämtad maj 23, 2025, https://www.generalagents.com/ace
7. Software Engineer | General Agents Careers, hämtad maj 23, 2025, https://generalagents.com/careers/software-engineer/
8. The Fastest "Computer Control" Agent I've Ever Seen - YouTube, hämtad maj 23, 2025, https://www.youtube.com/watch?v=08UfljELhH4
9. What are Claude computer use and ChatGPT Operator? - Zapier, hämtad maj 23, 2025, https://zapier.com/blog/claude-computer-use-openai-operator/
10. Claude Computer Use vs OpenAI Operator : r/ClaudeAI - Reddit, hämtad maj 23, 2025, https://www.reddit.com/r/ClaudeAI/comments/1ibo5kh/claude_computer_use_vs_openai_operator/
11. Ace is an in-progress computer use model and the devs recently learned that it can generalize to use any video game's UI despite it not being in the training data : r/singularity - Reddit, hämtad maj 23, 2025, https://www.reddit.com/r/singularity/comments/1kio4z5/ace_is_an_inprogress_computer_use_model_and_the/
12. NVIDIA/ace-controller: Pipecat framework based orchestrator for building real-time, voice-enabled, and multimodal conversational AI agents - GitHub, hämtad maj 23, 2025, https://github.com/NVIDIA/ace-controller
13. Conceptual Framework for Autonomous Cognitive Entities - arXiv, hämtad maj 23, 2025, https://arxiv.org/pdf/2310.06775
14. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/html/2410.22916v1
15. Integrating Behavioral Cloning into a Reinforcement Learning pipeline 1. Introduction 2. Related works - POLITesi, hämtad maj 23, 2025, https://www.politesi.polimi.it/retrieve/f1762a5c-9ea2-4b84-88a2-1218a08938b5/Executive_Summary_Andrea_DSilva.pdf
16. LLM-based Interactive Imitation Learning for Robotic Manipulation - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2504.21769v1
17. (PDF) Imitation Learning for Robotics: Progress, Challenges, and Applications in Manipulation and Teleoperation - ResearchGate, hämtad maj 23, 2025, https://www.researchgate.net/publication/385858967_Imitation_Learning_for_Robotics_Progress_Challenges_and_Applications_in_Manipulation_and_Teleoperation
18. arXiv:2503.23434v1 [cs.LG] 30 Mar 2025, hämtad maj 23, 2025, https://arxiv.org/pdf/2503.23434
19. Characterizing Unintended Consequences in Human-GUI Agent Collaboration for Web Browsing - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2505.09875v1
20. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/html/2504.03515v3
21. Imitation Learning via Focused Satisficing - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2505.14820v1
22. Behavior Cloning and Replay of Humanoid Robot via a Depth Camera - MDPI, hämtad maj 23, 2025, https://www.mdpi.com/2227-7390/11/3/678

23. (PDF) Imitation Learning: A Survey of Learning Methods (2017 ..., hämtad maj 23, 2025, https://scispace.com/papers/imitation-learning-a-survey-of-learning-methods-1bfrt7qge0
24. A Survey of Imitation Learning Methods, Environments and Metrics - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2404.19456v2
25. Platform | General Agents, hämtad maj 23, 2025, https://platform.generalagents.com
26. Agents API - Cloudflare Docs, hämtad maj 23, 2025, https://developers.cloudflare.com/agents/api-reference/agents-api/
27. Introducing GPT-4.1 in the API - OpenAI, hämtad maj 23, 2025, https://openai.com/index/gpt-4-1/
28. Introducing vision to the fine-tuning API | OpenAI, hämtad maj 23, 2025, https://openai.com/index/introducing-vision-to-the-fine-tuning-api/
29. GPT4Vis: What Can GPT-4 Do for Zero-shot Visual Recognition? - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2311.15732v2
30. Image understanding | Gemini API | Google AI for Developers, hämtad maj 23, 2025, https://ai.google.dev/gemini-api/docs/image-understanding
31. Google I/O 2025: Gemini as a universal AI assistant - Google Blog, hämtad maj 23, 2025, https://blog.google/technology/google-deepmind/gemini-universal-ai-assistant/
32. Gemini Pro - Google DeepMind, hämtad maj 23, 2025, https://deepmind.google/models/gemini/pro/
33. Gemini 2.5 Cost and Quality Comparison | Pricing & Performance, hämtad maj 23, 2025, https://www.leanware.co/insights/gemini-2-5-cost-quality-comparison
34. Vision - Anthropic, hämtad maj 23, 2025, https://docs.anthropic.com/en/docs/build-with-claude/vision
35. Anthropic's Claude in Amazon Bedrock - AWS, hämtad maj 23, 2025, https://aws.amazon.com/bedrock/anthropic/
36. How To Use Claude 3 To Automate Your Low-Code AI Workflows - BuildShip, hämtad maj 23, 2025, https://buildship.com/blog/how-to-use-claude-3-to-automate-your-low-code-ai-workflows
37. Image recognition test: Claude 3.5 Sonnet. Can it be used to ..., hämtad maj 23, 2025, https://www.youtube.com/watch?v=94dXPIOMIFY
38. LLaVa - Hugging Face, hämtad maj 23, 2025, https://huggingface.co/docs/transformers/v4.43.4/en/model_doc/llava
39. The Self-Operating Computer Framework: A Year in Review, hämtad maj 23, 2025, https://www.hyperwriteai.com/blog/the-self-operating-computer-framework-a-year-in-review
40. www.hyperwriteai.com, hämtad maj 23, 2025, https://www.hyperwriteai.com/blog/the-self-operating-computer-framework-a-year-in-review/
41. LLaVA-o1: Let Vision Language Models Reason Step-by-Step - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2411.10440v1

42. cogvlm | AI Model Details - AIModels.fyi, hämtad maj 23, 2025, https://www.aimodels.fyi/models/replicate/cogvlm-naklecha
43. CogAgent: A Visual Language Model for GUI Agents - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2312.08914v2
44. arXiv:2312.08914v3 [cs.CV] 27 Dec 2024, hämtad maj 23, 2025, http://arxiv.org/pdf/2312.08914
45. CogAgent: A Visual Language Model for GUI Agents - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2312.08914v1
46. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/abs/2312.08914
47. Falcon-UI: Understanding GUI Before Following User Instructions - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2412.09362v1
48. GitHub - QwenLM/Qwen2.5-VL, hämtad maj 23, 2025, https://github.com/QwenLM/Qwen2.5-VL
49. Qwen2_5-VL-7B-Instruct model | Clarifai - The World's AI, hämtad maj 23, 2025, https://clarifai.com/qwen/qwen-VL/models/Qwen2_5-VL-7B-Instruct
50. InternVL3: Exploring Advanced Training and Test-Time Recipes for Open-Source Multimodal Models - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2504.10479v1
51. InternVL3: Exploring Advanced Training and Test-Time Recipes for Open-Source Multimodal Models - arXiv, hämtad maj 23, 2025, https://arxiv.org/pdf/2504.10479?
52. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/abs/2504.10479
53. InternVL3: Exploring Advanced Training and Test-Time Recipes for Open-Source Multimodal Models - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2504.10479v3
54. dsa23.techconf.org, hämtad maj 23, 2025, https://dsa23.techconf.org/download/webpub2023/pdfs/DSA2023-5EN72RInktS0slKydpKoRu/047700a860/047700a860.pdf
55. ScreenAI: A visual language model for UI and visually-situated ..., hämtad maj 23, 2025, https://research.google/blog/screenai-a-visual-language-model-for-ui-and-visually-situated-language-understanding/
56. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/html/2210.03347
57. Fuyu 8b · Models · Dataloop, hämtad maj 23, 2025, https://dataloop.ai/library/model/adept_fuyu-8b/
58. fuyu-8b model | Clarifai - The World's AI, hämtad maj 23, 2025, https://clarifai.com/adept/fuyu/models/fuyu-8b
59. The Developer's Guide to UI Testing Automation with Llama 3.2 ..., hämtad maj 23, 2025, https://www.ionio.ai/blog/how-we-automate-ui-testing-with-multimodal-llms-llama-3-2-and-gemini-api
60. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/html/2409.01990v1
61. arxiv.org, hämtad maj 23, 2025, http://arxiv.org/pdf/2306.00978
62. Activation-Informed Merging of Large Language Models - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2502.02421v1

63. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/html/2409.01990v3
64. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/html/2502.07855v1
65. [2505.14052] Improved Methods for Model Pruning and Knowledge Distillation - arXiv, hämtad maj 23, 2025, https://arxiv.org/abs/2505.14052
66. www.arxiv.org, hämtad maj 23, 2025, https://www.arxiv.org/pdf/2505.15909
67. Resource-Efficient Language Models: Quantization for Fast and Accessible Inference - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2505.08620v1
68. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/html/2504.03708v1
69. [2505.13111] Why Knowledge Distillation Works in Generative Models: A Minimal Working Explanation - arXiv, hämtad maj 23, 2025, https://arxiv.org/abs/2505.13111
70. [2502.19545] Winning Big with Small Models: Knowledge Distillation vs. Self-Training for Reducing Hallucination in QA Agents - arXiv, hämtad maj 23, 2025, https://arxiv.org/abs/2502.19545
71. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/abs/2502.01158
72. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/html/2504.21561v3
73. OSSCAR: One-Shot Structured Pruning in Vision and Language Models with Combinatorial Optimization - arXiv, hämtad maj 23, 2025, https://arxiv.org/pdf/2403.12983?
74. A Comprehensive Study of Structural Pruning for Vision Models - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2406.12315v5
75. arxiv.org, hämtad maj 23, 2025, https://arxiv.org/pdf/2403.12983
76. GenAI at the Edge: Comprehensive Survey on Empowering Edge Devices - Qeios, hämtad maj 23, 2025, https://www.qeios.com/read/JEU3U0
77. [2502.07855] Vision-Language Models for Edge Networks: A Comprehensive Survey - arXiv, hämtad maj 23, 2025, https://arxiv.org/abs/2502.07855
78. Vision-Language Models for Edge Networks: A Comprehensive Survey - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2502.07855
79. Distributed inference with collaborative AI agents for Telco-powered Smart-X - AWS, hämtad maj 23, 2025, https://aws.amazon.com/blogs/industries/distributed-inference-with-collaborative-ai-agents-for-telco-powered-smart-x/
80. (PDF) Edge Computing Architectures for Real-Time Robotic Control ..., hämtad maj 23, 2025, https://www.researchgate.net/publication/390897926_Edge_Computing_Architectures_for_Real-Time_Robotic_Control_and_Decision-Making
81. Understand Edge Computing: A Quick Overview - Lyzr AI, hämtad maj 23, 2025, https://www.lyzr.ai/glossaries/edge-computing/
82. Moving AI to the edge: Benefits, challenges and solutions - Red Hat, hämtad maj 23, 2025, https://www.redhat.com/en/blog/moving-ai-edge-benefits-challenges-and-solutions
83. 7 Strategies To Get an Effective AI Agent Up and Running | Crowe LLP, hämtad maj 23, 2025, https://www.crowe.com/insights/7-strategies-to-get-an-effective-ai-agent-up-and-running

84. [2410.22916] Explainable Behavior Cloning: Teaching Large Language Model Agents through Learning by Demonstration - arXiv, hämtad maj 23, 2025, https://arxiv.org/abs/2410.22916
85. [Revue de papier] Explainable Behavior Cloning: Teaching Large Language Model Agents through Learning by Demonstration - Moonlight, hämtad maj 23, 2025, https://www.themoonlight.io/fr/review/explainable-behavior-cloning-teaching-large-language-model-agents-through-learning-by-demonstration
86. CapSoftware/scap: High-performance, cross-platform ... - GitHub, hämtad maj 23, 2025, https://github.com/CapSoftware/scap
87. NiiightmareXD/windows-capture: Fastest Windows Screen ... - GitHub, hämtad maj 23, 2025, https://github.com/NiiightmareXD/windows-capture
88. Best Screen Recording Software: Top Solutions in 2025, hämtad maj 23, 2025, https://www.appypieautomate.ai/blog/best-screen-recording-software
89. Screen Recording Software Comparison: Top Picks for 2025 ..., hämtad maj 23, 2025, https://blog.screendesk.io/screen-recording-software-comparison/
90. aclanthology.org, hämtad maj 23, 2025, https://aclanthology.org/2024.icon-1.48.pdf
91. High-Performance OCR Applications for Low-Quality PDF ... - HackMD, hämtad maj 23, 2025, https://hackmd.io/@Hamze/rytf5yQ0ke
92. Pricing | Cloud Vision API, hämtad maj 23, 2025, https://cloud.google.com/vision/pricing
93. Building a document OCR tool using GCP OCR and Node.js - Transloadit, hämtad maj 23, 2025, https://transloadit.com/devtips/building-a-document-ocr-tool-using-gcp-ocr-and-node-js/
94. Best OCR Software for Data Extraction and Automation in 2025, hämtad maj 23, 2025, https://clickup.com/blog/ocr-software/
95. Pricing - Computer Vision API | Microsoft Azure, hämtad maj 23, 2025, https://azure.microsoft.com/en-us/pricing/details/cognitive-services/computer-vision/
96. thesai.org, hämtad maj 23, 2025, https://thesai.org/Downloads/Volume16No4/Paper_103-Deep_Learning_Based_UI_Design_Analysis.pdf
97. Detect, Track, and Query: A YOLO-based Vision Application – longjie.yang's Site - GW Blogs, hämtad maj 23, 2025, https://blogs.gwu.edu/longjie-yang/2024/12/09/detect-track-and-query-a-yolo-based-vision-application/
98. Detectron2: A Rundown of Meta's Computer Vision Framework - viso ..., hämtad maj 23, 2025, https://viso.ai/deep-learning/detectron2/
99. Detectron2 is a platform for object detection, segmentation and other visual recognition tasks. - GitHub, hämtad maj 23, 2025, https://github.com/facebookresearch/detectron2
100. Using UI Automation for Automated Testing - Win32 apps | Microsoft Learn, hämtad maj 23, 2025, https://learn.microsoft.com/en-us/windows/win32/winauto/uiauto-usefortesting

101. What is UI automation? | UiPath, hämtad maj 23, 2025, https://www.uipath.com/automation/ui-automation
102. UI automation? LLM-based automation? You need both. | UiPath, hämtad maj 23, 2025, https://www.uipath.com/blog/automation/both-ui-automation-and-ai-based-automation
103. pywinauto/pywinauto: Windows GUI Automation with Python (based on text properties) - GitHub, hämtad maj 23, 2025, https://github.com/pywinauto/pywinauto
104. Easy Way for Beginners to Automate Desktop GUI Applications || Swapy Tool || Python|| pywinauto - YouTube, hämtad maj 23, 2025, https://m.youtube.com/watch?v=QtJXqF7rf54&pp=ygUMI3N3YXB5X3dvcmxk
105. uiautomation - crates.io: Rust Package Registry, hämtad maj 23, 2025, https://crates.io/crates/uiautomation
106. uiautomation - PyPI, hämtad maj 23, 2025, https://pypi.org/project/uiautomation/
107. Automate your UI using Microsoft Automation Framework - CodeProject, hämtad maj 23, 2025, https://www.codeproject.com/Articles/141842/Automate-your-UI-using-Microsoft-Automation-Framew
108. Introduction to UIA: Microsoft's Accessibility API - YouTube, hämtad maj 23, 2025, https://www.youtube.com/watch?v=6b0K2883rXA
109. Create UI Automation tests for a DevExpress-powered WinForms application. - GitHub, hämtad maj 23, 2025, https://github.com/DevExpress-Examples/winforms-app-ui-automation-testing
110. WinAppDriver/Samples/C#/NotepadAndCalculatorTest/README.md at master - GitHub, hämtad maj 23, 2025, https://github.com/microsoft/WinAppDriver/blob/master/Samples/C%23/NotepadAndCalculatorTest/README.md
111. Accessibility Programming Guide for OS X - Apple Developer, hämtad maj 23, 2025, https://developer.apple.com/library/archive/documentation/Accessibility/Conceptual/AccessibilityMacOSX/
112. Integrating accessibility into your app | Apple Developer Documentation, hämtad maj 23, 2025, https://developer.apple.com/documentation/accessibility/integrating_accessibility_into_your_app
113. Mac Automation Scripting Guide: Automating the User Interface, hämtad maj 23, 2025, https://developer.apple.com/library/archive/documentation/LanguagesUtilities/Conceptual/MacAutomationScriptingGuide/AutomatetheUserInterface.html
114. Automating Split View on Mac - AppleScript, hämtad maj 23, 2025, https://www.macscripter.net/t/automating-split-view-on-mac/74222
115. Scripting the unscriptable: GUI Scripts in macOS - Ross Matsuda - YouTube, hämtad maj 23, 2025, https://www.youtube.com/watch?v=UKhLLvuxbxU

116. 24.3. GUI Scripting Examples - AppleScript: The Definitive Guide, 2nd Edition [Book], hämtad maj 23, 2025, https://www.oreilly.com/library/view/applescript-the-definitive/0596102119/ch24s03.html
117. The libuic allows to programmatically control UI elements of OS-X applications. - GitHub, hämtad maj 23, 2025, https://github.com/polym0rph/libuic
118. Are we GUI yet?, hämtad maj 23, 2025, https://areweguiyet.com/
119. macOS and iOS APIs — list of Rust libraries/crates // Lib.rs, hämtad maj 23, 2025, https://lib.rs/os/macos-apis
120. A 2025 Survey of Rust GUI Libraries | boringcactus, hämtad maj 23, 2025, https://www.boringcactus.com/2025/04/13/2025-survey-of-rust-gui-libraries.html
121. AgentKit: A Technical Vision for Building Universal AI Automation for ..., hämtad maj 23, 2025, https://dev.to/zhanghandong/agentkit-a-technical-vision-for-building-universal-ai-automation-for-human-computer-interaction-2523
122. AgentKit: A Technical Vision for Building Universal AI Automation for Human-Computer Interaction Based on Rust - Reddit, hämtad maj 23, 2025, https://www.reddit.com/r/rust/comments/1k1x5ak/agentkit_a_technical_vision_for_building/
123. Install and Test with Cua: the Docker Integration for Computer-Use ..., hämtad maj 23, 2025, https://huggingface.co/blog/lynn-mikami/cua
124. AT-SPI2 - Freedesktop.org, hämtad maj 23, 2025, https://www.freedesktop.org/wiki/Accessibility/AT-SPI2/
125. Assistive Technology Service Provider Interface - Wikipedia, hämtad maj 23, 2025, https://en.wikipedia.org/wiki/Assistive_Technology_Service_Provider_Interface
126. Accessibility/ATK/AT-SPI/AT-SPI on D-Bus - Wiki, hämtad maj 23, 2025, https://wiki.linuxfoundation.org/accessibility/d-bus
127. [2503.06287] Your Large Vision-Language Model Only Needs A Few Attention Heads For Visual Grounding - arXiv, hämtad maj 23, 2025, https://arxiv.org/abs/2503.06287
128. ScreenSpot-Pro: GUI Grounding for Professional High-Resolution Computer Use - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2504.07981v1
129. [2504.07981] ScreenSpot-Pro: GUI Grounding for Professional High-Resolution Computer Use - arXiv, hämtad maj 23, 2025, https://arxiv.org/abs/2504.07981
130. GUI-Agents-Paper-List/paper_by_key/paper_grounding.md at main ..., hämtad maj 23, 2025, https://github.com/OSU-NLP-Group/GUI-Agents-Paper-List/blob/main/paper_by_key/paper_grounding.md
131. Aria-UI: Visual Grounding for GUI Instructions - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2412.16256v1
132. Scaling Computer-Use Grounding via User Interface Decomposition and Synthesis - arXiv, hämtad maj 23, 2025, https://arxiv.org/html/2505.13227v1

133. Navigating the Digital World as Humans Do: Universal Visual Grounding for GUI Agents, hämtad maj 23, 2025, https://arxiv.org/html/2410.05243v2
134. Navigating the Digital World as Humans Do: Universal Visual Grounding for GUI Agents, hämtad maj 23, 2025, https://openreview.net/forum?id=kxnoqaisCT
135. Self-Operating Computer download | SourceForge.net, hämtad maj 23, 2025, https://sourceforge.net/projects/self-operating-computer.mirror/
136. xdotool man | Linux Command Library, hämtad maj 23, 2025, https://linuxcommandlibrary.com/man/xdotool
137. jordansissel/xdotool: fake keyboard/mouse input, window management, and more - GitHub, hämtad maj 23, 2025, https://github.com/jordansissel/xdotool
138. Automate ANYTHING on Linux (using xdotool) - YouTube, hämtad maj 23, 2025, https://www.youtube.com/watch?v=feLbkm5aV_0
139. Using at, xdotool, and a Bash script to automate Netflix watching (Ubuntu 19.10) - YouTube, hämtad maj 23, 2025, https://www.youtube.com/watch?v=YOk387rWktY
140. Wayland - ArchWiki, hämtad maj 23, 2025, https://wiki.archlinux.org/title/Wayland
141. RustAutoGUI 2.5.0 - Optimized Cross-Platform GUI Automation library, now with OpenCL GPU Acceleration : r/rust - Reddit, hämtad maj 23, 2025, https://www.reddit.com/r/rust/comments/1k93red/rustautogui_250_optimized_crossplatform_gui/
142. Best pywinauto Alternatives & Competitors - SourceForge, hämtad maj 23, 2025, https://sourceforge.net/software/product/pywinauto/alternatives
143. Top pywinauto Alternatives in 2025 - Slashdot, hämtad maj 23, 2025, https://slashdot.org/software/p/pywinauto/alternatives
144. Playwright vs Puppeteer: What's the Difference? - Autify, hämtad maj 23, 2025, https://autify.com/blog/playwright-vs-puppeteer
145. Puppeteer vs. Playwright: Automated testing tools compared ..., hämtad maj 23, 2025, https://www.contentful.com/blog/puppeteer-vs-playwright/
146. Playwright vs Puppeteer: Best Choice for Web Scraping? - BrowserCat, hämtad maj 23, 2025, https://www.browsercat.com/post/playwright-vs-puppeteer-web-scraping-comparison
147. Rust in Systems Programming: Why Devs Are Choosing Rust Over ..., hämtad maj 23, 2025, https://dev.to/arjun98k/rust-in-systems-programming-why-devs-are-choosing-rust-over-c-and-c-6e5
148. The Rise of Rust: A Safer Alternative to C++ in System Development - DEV Community, hämtad maj 23, 2025, https://dev.to/hexadecimalsoftware/the-rise-of-rust-a-safer-alternative-to-c-in-system-development-2poi
149. Rust vs Go Comparison: Performance, Popularity, & More - Relia Software, hämtad maj 23, 2025, https://reliasoftware.com/blog/rust-vs-go-comparison
150. Rust And Go Performance Comparison - Proxify, hämtad maj 23, 2025, https://proxify.io/articles/rust-and-go-performance-comparison

151. Python vs Go: Which is the Best Language for Your Project?, hämtad maj 23, 2025, https://www.ongraph.com/python-vs-go-which-to-choose/
152. Golang vs Python for AI & Machine Learning: Which One is Better in 2025? - Rubyroid Labs, hämtad maj 23, 2025, https://rubyroidlabs.com/blog/2025/05/golang-vs-python-ai-machine-learning/
153. Building Agentic AI Systems in Python A Beginner's Guide - Codewave, hämtad maj 23, 2025, https://codewave.com/insights/agentic-ai-systems-python-guide/
154. Top 7 Python Frameworks for AI Agents - KDnuggets, hämtad maj 23, 2025, https://www.kdnuggets.com/top-7-python-frameworks-for-ai-agents
155. Hello ImGui — Hello, Dear ImGui - Cross-platform Gui apps with ..., hämtad maj 23, 2025, https://pthom.github.io/hello_imgui/
156. Why Use Dear ImGui for Game Engines if It's Not for Complex Apps? - Reddit, hämtad maj 23, 2025, https://www.reddit.com/r/cpp/comments/1erq2vx/why_use_dear_imgui_for_game_engines_if_its_not/
157. Qt seems to be the most popular library for cross-platform GUI apps written in C... | Hacker News, hämtad maj 23, 2025, https://news.ycombinator.com/item?id=24987574
158. QT vs Dear ImGui vs ___? For cross platform GUI? : r/cpp - Reddit, hämtad maj 23, 2025, https://www.reddit.com/r/cpp/comments/1dc9jgh/qt_vs_dear_imgui_vs_for_cross_platform_gui/
159. Tauri vs. Electron: performance, bundle size, and the real trade-offs, hämtad maj 23, 2025, https://gethopp.app/blog/tauri-vs-electron
160. Electron vs Tauri - Coditation, hämtad maj 23, 2025, https://www.coditation.com/blog/electron-vs-tauri
161. My experience with Tauri vs Neutralino - Reddit, hämtad maj 23, 2025, https://www.reddit.com/r/tauri/comments/1ik9ha5/my_experience_with_tauri_vs_neutralino/
162. Proton Native vs Tauri vs Neutralino - YouTube, hämtad maj 23, 2025, https://www.youtube.com/watch?v=M8breg0t2QI
163. WebAssembly Performance: How Fast Is WASM? - Clover Dynamics, hämtad maj 23, 2025, https://www.cloverdynamics.com/blogs/web-assembly-performance-how-fast-is-wasm
164. The Rise of WebAssembly in High-Performance Web Apps - A WP Life, hämtad maj 23, 2025, https://awplife.com/the-rise-of-webassembly-in-high-performance-web-apps/
165. AI and Microservices Architecture - SayOne Technologies, hämtad maj 23, 2025, https://www.sayonetech.com/blog/ai-and-microservices-architecture/
166. Designing Microservices Using AI: A Systematic Literature Review - MDPI, hämtad maj 23, 2025, https://www.mdpi.com/2674-113X/4/1/6
167. From Monoliths to Microservices to Packaged Business Capabilities (PBCs) - Mia-Platform, hämtad maj 23, 2025,

https://mia-platform.eu/blog/from-monoliths-to-microservices-to-pbcs/
168. How SaaS Companies Can Migrate from Monolithic to Microservices using Generative AI, hämtad maj 23, 2025, https://www.optisolbusiness.com/insight/how-saas-companies-migrate-from-monolithic-to-microservices-using-generative-ai
169. How Kafka + Flink Power Real-Time AI Agents - Conduktor, hämtad maj 23, 2025, https://conduktor.io/blog/ai-agents-at-scale-the-critical-role-of-kafka-and-flink
170. How Apache Kafka and Flink Power Event-Driven Agentic AI in Real Time - Kai Waehner, hämtad maj 23, 2025, https://www.kai-waehner.de/blog/2025/04/14/how-apache-kafka-and-flink-power-event-driven-agentic-ai-in-real-time/
171. Comparing RabbitMQ, Kafka, and NATS | bugfree.ai, hämtad maj 23, 2025, https://www.bugfree.ai/knowledge-hub/comparing-rabbitmq-kafka-nats
172. Cloud Native Kafka Alternatives: Apache Kafka vs. NATS - GitHub, hämtad maj 23, 2025, https://github.com/AutoMQ/automq/wiki/Cloud-Native-Kafka-Alternatives:-Apache-Kafka-vs.-NATS
173. Why gRPC is a Great Choice - DEV Community, hämtad maj 23, 2025, https://dev.to/emiroberti/why-grpc-is-a-great-choice-2coc
174. gRPC vs WebSocket | When Is It Better To Use? - Wallarm, hämtad maj 23, 2025, https://www.wallarm.com/what/grpc-vs-websocket-when-is-it-better-to-use
175. Can gRPC replace REST and WebSockets for Web Application Communication?, hämtad maj 23, 2025, https://grpc.io/blog/postman-grpcweb/
176. CDN Ecosystem Embraces AI - Tech News - Bizety, hämtad maj 23, 2025, https://bizety.com/2025/01/19/cdn-ecosystem-embraces-ai/
177. Benchmarking LLM Speed, hämtad maj 23, 2025, https://llm-tracker.info/howto/Benchmarking-LLM-Speed
178. AWS vs. Azure vs. Google Cloud: A Complete Comparison ..., hämtad maj 23, 2025, https://www.datacamp.com/blog/aws-vs-azure-vs-gcp
179. Cloud Pricing Comparison: AWS vs. Azure vs. Google in 2025, hämtad maj 23, 2025, https://cast.ai/blog/cloud-pricing-comparison/
180. Optimizing Server Load Balancing with AI Agents | AI Agent, hämtad maj 23, 2025, https://aiagent.app/usecases/ai-agents-for-server-load-balancing
181. (PDF) AI-Driven Load Balancing for Energy-Efficient Data Centers, hämtad maj 23, 2025, https://www.researchgate.net/publication/384129889_AI-Driven_Load_Balancing_for_Energy-Efficient_Data_Centers