arXiv:2606.05405v4 [cs.AI] 25 Oct 2026

Agents' Last Exam

Website | GitHub | HuggingFace | Leaderboard

Organization & Execution Team

Yiyou Sun*, Xinyang Han*, Weichen Zhang*, Yuanbo Pang*, Tianyu Wang*, Yuhan Cao*, Yixiao Huang*, Xiang Liu, Zonglin Di, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, Weishu Zhang, Tyler Zeng, Ying Yan, Bo Liu, Hanson Wen, Mingyang Xu, Xiaoyuan Liu, Zimeng Chen, Zhaowei Zhang, Weiyan Shi, Amanda Dsouza, Vincent Sunn Chen, Jingxu Xie, Dewen Lei, Hanxuan Zhang, Yinhe Wang, Xinran Zhang, Chao Zhou, Sheldon Huang, Dawn Song

* Core contributors.

Advisory Committee

Patrick Bryant, Carl Boettiger, Yamini Rangan, Bradley Rothenberg, Kyle Steinfeld, Arvind Rao, Tapio Schneider, Georgios Yannakakis, Laure Zanna, Kaan Ozbay, Ida Sim, Tarek Zohdi, George Em Karniadakis, Jack Gallant, Teresa Head-Gordon

Data Contributors

Chris Yuhao Liu, Rongwang Hu, Yafei Cheng, Aras Bacho, Zengyi Qin, Alan Yuille, Yixiong Chen, Hao Liu, Lin Zeng, Bo Chen, Zhendong Li, Litian Gong, Shashank Muralidhar Bharadwaj, Thorben Jansen, HAIXIANG TANG, Jianhong Tu, Jixian Liu, Maojia Song, Ruheng Wang, Shengcao Cao, Yanxin Zhang, Yingxuan Yang, Zongzheng Zhang, Honglin Bao, Hyunjun Kim, Peiran Xu, Shuo Lu, Zheng Zhang, Zhonghua Wang, Aditya Prakash, Bohan Lyu, Chang Ma, Haoxuan Chen, Hengduan Fan, Molei Tao, Roman Sitelew, Russell Alan Poldrack, Yang Liu, Yang Liu, Yichen Wu, Yuanze Lin, Zhenglu Li, Zibu Wei, Zijiao Chen, Abderrahim Bendahi, Alexander Schubert, Chase Broderick, Chengzhi Liu, Haoran Sun, Haoyang Li, Haoyue Hua, Jiayuan Ding, Jing Huang, Shangding Gu, Tuan Vinh, Xiatao Sun, Yanqiong Jiang, Yiran Hu, Zechen Xu, Ziao Yang, Chenguang Wang, Costas Spanos, Dongrong Yang, Haocheng Wang, Jiayi Cheng, JICHENG WANG, Junbo Peng, Kangqi Zhang, Keying Kuang, Lichao Sun, Ming Yin, Orson Xu, Vincent Siu, Wanzhe Liao, Wenqi Shi, Wenxi Deng, Xiaopan Zhang, Yangjunfeng Zhu, Yifan Shen, Yi Jiang, Yinsheng Li, Youbang Pan, Yuan Li, Yuchen Zhu, Yunshi Zhang, Zhenning Yang, Ziyan Rao, Alireza Rafiei, Andrew Glover, Bixuan Ren, Boqin Yuan, Chonghao Cai, Deren Lei, Hanwen Xing, Haoran Jin, Hector Gomez, Ishaan Singh Chandok, Jiaming Hu, Jianxin Sun, Justin Purnomo, Kunyang Sun, Lei Ding, Longtai Liao, Lukas Kim, Ming Liu, Qingpo Wuwu, Ran Jin, Ruiqi Zhang, Ting Sun, Wenbo Huang, Xuandong Zhao, Peter Chen, Chuanyang Jin, Qiming Bao, Xuhuan Shen, Yifan Zhang, Yi Li, Yingze Wang, Yixin Wang, Yushi Yang, Zhen Dong, Zhiling Yan, Zhongxing Xu, Zibo Zhao, Zixin Jiang, Ali Emami, Andy Zeyi Liu, Angela Wu, Aylin Caliskan, Baihe Huang, Baixiang Huang, Benjamin Yang, Benran Zhang, Brian Zhao, Carlo Bosio, Charlie Guo, Che Liu, Chenchen Du, Chen Dai, Chenlong Zhang, Chenyu Li, Chenyu Zhu, Chi Zhang, Core Francisco Park, Daofeng Li, Dehao Wu, Dowling Wong, Erin Kasson, Fangyu Wu, Fan Xu, Fiona Shen, Haimin Hu, Haitao Li, Hang Zhou, Hannah Szlyk, Hanning Zhang, Hanwen Zhang, Hao Fan, Haoran Ye, HaoYang Shang, Haoyue Bai, Hengyu Fu, Hongwei Li, Huiqi Wang, Jiachen Li, Jialu Wang, Jiaming Liu, Jiawei Shen, Jierui Zuo, Jingxuan Fan, Jingyan Li, Jirong Yang, Jonah So, Jorin Overwiening, Joseph Cavanagh, Joseph Lee, Junge Zhang, Kaiyuan Liu, Kaustubh Dhole, Kevin Qinghong Lin, Kunpeng Zhang, Kyle Montgomery, Layna Paraboschi, Liana, Lihui Liu, Liwei Zhou, Lu Lu, Mengyuan "Millie" Wu, Mikko Tolonen, Mingxuan Yan, Minqi Shi, Murphy Niu, Pan Lu, Patricia Cavazos-Rehg, Qijian Zhao, Qiuyang Mang, Rimika Jaiswal, Ruiling Xu, Run-Ze Fan, Ruoxi Ning, Santanu Bosu Antu, Scott Rankin, Sen Yang, Shi Gu, Shi Qiu, Sichen Zhu, Sihan Xu, Siqi Zhu, Song Zhang, Sophia Sun, Stefano De Castro, Swati Rajwal, Tianneng Shi, Tianxing Chen, Tianyue Zhou, Ting Wang, Ventus Shaw, Vladislav Susoy, Weihang Xiao, Wei Wang, Wenbo Lv, Wenjin Zhang, Wenshuo Zhang, Wenyu Liang, Xiaohu Yu, Xiao Li, Xiaoli Yang, Xin Lan, Xinyu Wang, Xuan Liu, Xuewen Chen, Xu Pan, Yan Zhu, Yicheng Lyu, Yifan Gu, Yimin Zhao, Ying Zhang, Yin Ren, Yipu Chen, Yiren Rong, Yi Shao, Yishu Zhu, Yizhi Zhang, Yongyi Xiong, Yuangang Li, Yuchen Wu, Yuhaohua Zheng, Yuhao Wu, Yunfu Deng, Yushan Li, Yu Wu, Yuzheng Xu, Yuzhou Nie, Zehao Wang, Zhehang Du, Zhengming Yu, Zhihan Jiang, Zhi Li, Zhixu Li, Zhuoran Yi, Tao Sun, Zhun Wang, Justin Xu, Shi Bo, Helena Casademunt, Wenjia Chen, Cheng Cheng, Nawen Deng, Rao Fu, Tianfu Fu, Yifan Han, Zhenyu He, Qiao Jin, Langlang Li, Yuetai Li, Sylvia Liu, Subhabrata Mukherjee, Yunqi Ouyang, He Ren, Dawei Shi, Haoran Wu, Zhiyue Wu, Hannah Yao, Jenny Yu, Rhea Zhan, Luqing Zhou, Blake Zhu, Junfan Zhu, Sanjiv Das, Yanjun Gao, Wenbo Guo, Erika J Schneider, Bing Liu, Zhiyong Lu, Yian Ma, Mark Mueller, Radha Poovendran, Somayeh Sojoudi, Huaxiu Yao, Yinglun Zhu, Károly Zsolnai-Fehér, Yegor Denisov-Blanch, Changran Hu, Vincent Koc, Hezheng Yin, Naomi Xia, Xinyi Bai, Arsh Shah Dilbagi, Jing Gu, Yutong Liu, Lin Shi, Qizheng Zhang, Yuexing Hao, Xiaoming Li, Xiangyi Li, Bingran You, Wenjun Ke, Daoan Zhang, Ari Nemet, Tianyi Zhang, Xingda Lyu, Pranit Chawla, Genta Indra Winata, Maosen Tang, Purva Chiniya, Qiuyi Chen, Steven Dillmann, Wanghan Xu, Wenqi Yang, Xiaojun Wu, Xin Luo, Yue Huang

Leading institution: University of California, Berkeley. Corresponding to {sunyiyou,dawnsong}@berkeley.edu.
Full affiliations in Appendix A.

Abstract

Recent AI systems have achieved strong results on a wide range of benchmarks, yet these gains have not translated into economically meaningful deployment across many professional domains. We argue that this gap is largely an evaluation problem: widely used benchmarks lack sustained performance measurement on real and economically valuable workflows. This paper introduces **Agents' Last Exam (ALE)**, a benchmark designed to evaluate AI agents on long horizon, economically valuable, real world tasks with verifiable outcomes. Developed in collaboration with 250+ industry experts, ALE covers non-physical industries defined with reference to O*NET / SOC 2018 (the U.S. federal occupational taxonomy). It is organized around a task taxonomy with 55 sub fields grouped into 13 industry clusters covering 1K+ tasks. Current results show that the hardest tier remains far from saturated: across mainstream harness and backbone configurations, the average full pass rate is below 1%. ALE is designed as a living benchmark: its task pool grows continuously as new workflows and industries are onboarded. More broadly, ALE is intended not merely as another leaderboard, but as an instrument for closing the gap between benchmark success and GDP relevant impact.



Figure 1: **Agents' Last Exam** spans a broad taxonomy of professional tasks and realistic workflows.

1 Introduction

Over the past few years, AI systems have cleared one celebrated benchmark after another: world-champion games [43], olympiad mathematics [18], and competitive programming [16]. Yet by the metric that ultimately matters, economic output, the broader impact has remained surprisingly muted; benchmark victories have accumulated faster than measurable transformation in core industries. This gap, which we view as a utility problem for AI, suggests that the field now needs evaluations that measure not only abstract competence, but also the ability to carry out long-horizon, economically valuable work in real professional environments.

This gap matters because AI progress is remarkably shaped by the benchmarks the field chooses to optimize. Benchmarks do not merely record capability; they focus research attention, define engineering targets, and often determine which domains become tractable for rapid improvement. The recent history of AI makes this pattern clear: once a domain is captured by a verifiable and widely used evaluation, progress in that domain tends to accelerate, and deployment often follows, like ImageNet [13] played this role in computer vision. For economically central sectors such as finance, law, electrical engineering, and manufacturing, however, comparable evaluations remain underdeveloped. If such benchmarks can be built and eventually saturated, that outcome would signify more than success on a test: it would indicate that AI systems have become capable of performing the underlying professional workflows at a level sufficient for real industrial adoption.

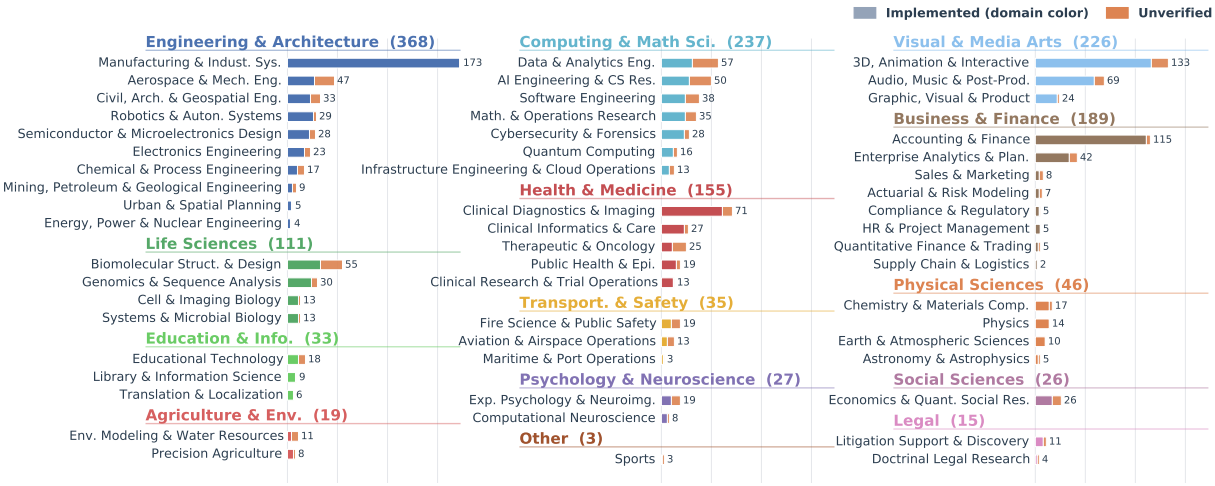


Figure 2: **Distribution of 1,490 task instances across the ALE taxonomy.** Each row is one of the 55 subdomains, grouped under the 13 top-level domains (parenthetical numbers give domain totals). Stacked bars decompose each subdomain into fully-implemented instances (domain color) and expert submissions awaiting Quality Control (QC) Process (orange). All 55 subdomains receive non-zero coverage. Current runnable task instances target either Linux or Windows virtual machines.

Building such evaluations is difficult for structural reasons. First, long-horizon authentic workflows are expensive to collect because they must be sourced from real software and organizational contexts. Prior work often adopts task units that are easier to collect, whether shorter computer-use tasks [51], synthetic environment construction [1], or purely question-answering setups [53]. Second, broad industry coverage with authentic, economically valuable workflows is also hard. It requires sustained access to experts across domains and deep insight into the industrial landscape. Existing benchmarks often evaluate on a limited set of domains [3]. Third, verification is intrinsically hard for real workflows because the output space is heterogeneous. A correct deliverable may be a file, spreadsheet, media artifact, report, design, or model. As a result, benchmarks that measure economically valuable work often rely on human judgment, as in GDPval [37] and the Remote Labor Index [23]. These constraints help explain why existing benchmarks often trade away one of realism, breadth, or verifiability. They jointly motivate **Agents' Last Exam (ALE)**.

> **Why "Last Exam"?** The name carries a dual aspiration. *Last as competence threshold*: an agent that passes these industry exams demonstrates readiness to carry out sustained, economically valuable work in that profession, not merely to answer questions about it. *Last as difficulty frontier*: by grounding evaluation in authentic long-horizon workflows that require professional judgment, ALE sits at the boundary of what current systems can reliably accomplish.

ALE is a benchmark of 1K+ task instances spanning 55 subfields and 13 industry clusters, developed in collaboration with 250+ domain experts. To ensure broad and representative industry coverage, expert advisory committees map each domain's workflow landscape and identify economically meaningful workflow families, anchored in the O*NET / SOC 2018 occupational taxonomy [38, 46]. Its task workflows are sourced from real professional practice: rather than inventing synthetic scenarios, experts contribute projects they have already completed, which then undergo multi-round quality control, including first-pass review, engineer dry-runs, and final peer review by expert committees, before admission. Most tasks demand computer use that interleaves GUI interaction (desktop applications, browsers, domain-specific software) with CLI operations (shell scripting, code execution, file manipulation), requiring the union of capabilities that existing benchmarks test in isolation.

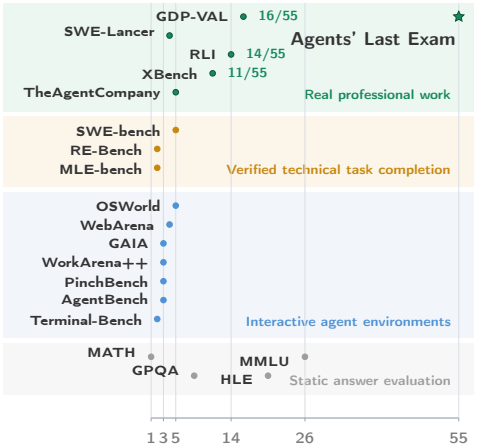


Figure 3: **Benchmark positioning map.** Prior benchmarks are placed by mapping their published domains onto the ALE domain taxonomy.

To make heterogeneous real-world outputs verifiable without human judges, ALE standardizes evaluation around structured deliverable-based or milestone-based checks against expert-provided references and rubrics.

ALE's target evaluation subject is the *Generalist Computer-Use Agent* (GCUA), such as Claude Code [4] or Codex [33], that combines visual perception, code execution, tool use, and long-horizon planning within a single action loop. ALE's task surface is, by construction, a superset of GUI-only benchmarks like OSWorld [51] and CLI-only benchmarks like Terminal-Bench [24]. For coverage comparisons, we use the 55 ALE subdomains as a common coordinate system and map each prior benchmark's published subjects, applications, repositories, or occupations onto this taxonomy (Figure 3). Current results confirm that ALE is far from saturated: the strongest configuration (Codex with GPT-5.5), which already achieves 82% on Terminal-Bench, scores below 50% even on ALE's easiest tier and under 10% on the hardest; most mainstream agents, including Claude Code, record near-zero pass rates at that difficulty level.

More broadly, ALE is intended not merely as another leaderboard, but as an instrument for closing the gap between benchmark success and GDP-relevant impact: **if frontier AI agents can pass this last exam, then progress on the benchmark may begin to register as real economic transformation.**

2 Benchmark Design and Dataset Construction

2.1 Benchmark Design Principles: What Tasks are We Looking for?

The benchmark is defined by three high-level requirements. They determine which workflows are admitted into the dataset and which are rejected in the public submission portal:

• **Representativeness.** The workflow should match real professional practice and use the software that domain experts would actually use. For example, architectural experts would typically use SolidWorks or Rhino rather than AutoCAD to convert a 2D blueprint into a 3D model.

• **Complexity.** A task should be an end-to-end deliverable that would take an expert substantial time, rather than only a few UI operations. The key distinction is between a workflow and an action.

Undesired example. "Apply a color filter in DaVinci" is too narrow because it's a single local edit.

Better example. "Move a running cheetah into another race video" is suitable because it requires tracking, rotoscoping, compositing, and color matching within one coupled workflow.

• **Verifiability.** The output should admit deterministic checking or an unambiguous rubric tied to observable artifacts. The strongest case is a deterministic deliverable that can be compared directly against a reference output. When exact matching is impossible, the judgment should still reduce to a measurable artifact.

Undesired example. "Design an RPG game with monsters" provides no objectively checkable target.

Better example. "Reproduce the game *mota.exe* using RPGMaker XP" is verifiable because the resulting map geometry, character attributes, and event states can be automatically compared against a reference version under identical trajectories of user operations.

2.2 Benchmark Scope and Taxonomy: Which Domains Do We Cover?

Rather than selecting industries ad hoc or by economic ranking, we ground the ALE taxonomy in SOC 2018 [46] and O*NET [38]: we cluster occupations with similar software-mediated workflows into ALE industries, exclude sectors whose core work is not meaningfully digital, and group the result into 13 domains spanning 55 subdomains (Figure 1; full derivation in Appendix B.1). To enable fair cross-benchmark comparison, we map each prior benchmark's published categories (subjects, applications, repositories, or occupations) onto the same 55-subdomain taxonomy via an LLM-assisted classifier. The result exposes a coverage gap that no existing benchmark closes: even the union of 16 major prior benchmarks leaves **13 of 55 subdomains** entirely uncovered (Table 2).

2.3 Task Construction Pipeline: How are the Tasks Created?

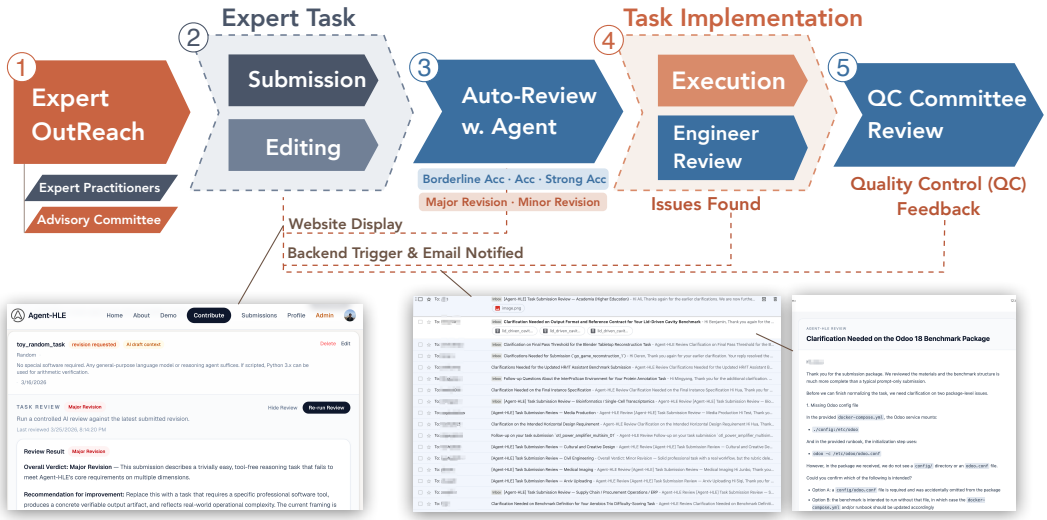


Figure 4: **Task construction pipeline.** Tasks proceed from expert sourcing through submission, first-pass review, engineering implementation, and final quality control.

Tasks in ALE cannot be crowdsourced from lay workers; they must arise from the actual routines of domain professionals and undergo rigorous screening to guarantee authenticity, complexity, and technical executability. We therefore employ a staged construction protocol (Figure 4) with five gates. *Expert sourcing* recruits domain specialists through an advisory committee of industry practitioners, ensuring coverage across the taxonomy. *Task submission* routes proposals through a dedicated web portal (`https://agents-last-exam.org/submit/new/form`), where experts upload past projects that took them days or weeks of professional work and AI-assisted tools help refine each proposal until five core components are fully specified: a natural-language description, input files, target software, expected deliverable, and evaluation specification. *First-pass review* screens submissions with conference-style decisions (`major / minor revision`, `borderline accept`, `accept`, `strong accept`); revisions loop back to the expert. *Task implementation* converts accepted specifications into runnable assets, provisioned software containers, and codified evaluation logic, with engineer dry-runs and automatic routing back to the expert when gaps are found. *Final QC* is a peer review by the expert committee that verifies reference-output correctness, calibration of evaluation bounds (neither impossibly narrow nor spuriously permissive), and sufficiency of context before the task is admitted. More details are deferred to Appendix B.2.

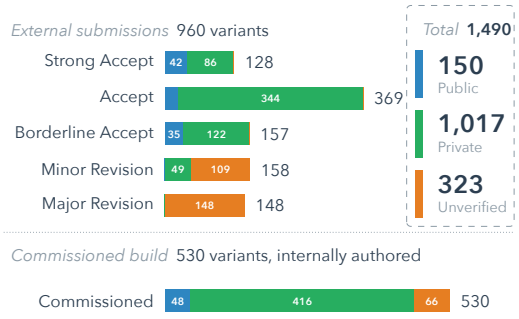


Figure 5: **Provenance and review yield.** The 1,490 task instances split into 960 external submissions (top, by first-pass review verdict) and 530 commissioned tasks (bottom). Each bar is segmented by release state: 150 public, 1,017 private, and 323 unverified pending QC.

Public/private release strategy. Benchmark contamination, whether through pre-training data overlap or task-specific optimization, is a central threat to the long-term validity of any public evaluation. ALE addresses this by releasing only 150 of the 1,490 task instances (∼10%) publicly, with the remainder held in a private pool (Figure 5). ALE is further designed for *rolling evaluation*: private task instances will periodically rotate into the public set while retired public tasks are replaced, maintaining an uncontaminated evaluation surface over successive model generations. Appendix D.1 verifies empirically that the public subset is representative of the full pool.

3 Evaluation Pipeline

The previous section describes how expert submissions are collected and converted into verified benchmark instances. This section specifies what happens once a task exists: how it is executed, how

the agent interacts with the environment, and how the outcome is scored. The pipeline is organized around an **uncoupling** of three components (the *task specification*, the *agent*, and the *environment*) so that they can all be easily interchanged.

Terminology. Throughout the paper we use *task workflow* for an end-to-end professional procedure and *task instance* for one runnable case of a task workflow (one concrete (input, output) pair, sharing the same `evaluate()` but differing in input and output data). The rare appearance single "task" refers to the runnable instance level.

3.1 Pipeline Architecture

Figure 6 illustrates the end-to-end evaluation pipeline. A benchmark instance is realized through three decoupled components that interact via well-defined interfaces. The *task specification* is an executable `main.py` encoding the five elements supplied during construction (description, input assets, target software, reference assets, evaluation criteria) and exposing three lifecycle functions: `load()` declares the task and compute requirements, `start()` provisions the VM into a deterministic starting state, and `evaluate()` scores the agent's output in $[0, 1]$. The *agent* (a harness orchestrating a foundation model) receives only the task description and metadata, then runs an action loop over screenshots, shell output, mouse and keyboard, file edits, and API calls until it terminates. The *environment* is a remote virtual machine with a canonical four-directory layout: `input/` (read-only assets), `software/` (pre-installed applications), `output/` (the agent's sole writable target), and `reference/` (ground-truth artifacts hidden from the agent and used only for scoring). This decoupling lets any agent that conforms to the action interface be evaluated on any task, and a single specification be deployed across cloud VMs or local containers without modification. Per-component interfaces and the directory contract are detailed in Appendix C.1, with the lifecycle protocol in Appendix C.2.

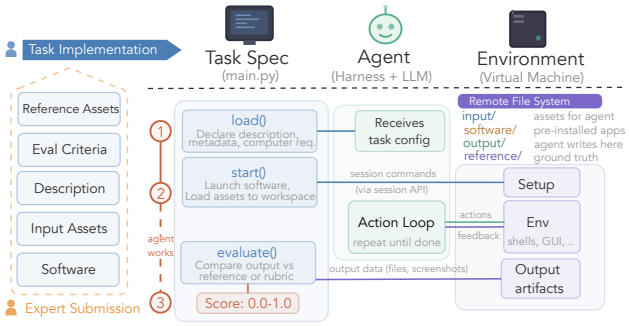


Figure 6: **Evaluation pipeline architecture.** Each benchmark instance is defined by a Task Specification (`main.py`) that orchestrates a three-phase lifecycle (`load()`, `start()`, `evaluate()`) over a remote virtual-machine environment. The agent (harness + model) receives only the task description and metadata, interacts with the environment through an action loop, and produces output artifacts that the specification scores against references or rubrics.

3.2 Agent Architecture: From CLI/GUI-agents to Generalist CUA

The tasks in ALE require agents that can read GUI screens, type in dialog boxes, execute shell commands, write and debug code, invoke APIs, and manage long-running sessions, often within a single task workflow. No single existing agent family covers this surface natively, so the benchmark targets a broader agent class that we make explicit here, together with how the prevailing harness architecture is extended to reach it.

We decompose an agent's operational capabilities into five functional layers (Figure 7): **Brain** (LLM reasoning and planning), **Eyes** (GUI perception via screenshots), **Body** (orchestration and control flow), **Hands** (structured tool invocation), and **Feet** (the runtime substrate on which actions take effect). This decomposition exposes a clean split among existing families. Traditional **CLI-agents** such as SWE-agent [52] and ForgeCode [45] have full Brain, Body, Hands, and Feet but lack Eyes by construction; framework-style agents such as OpenClaw are not strictly CLI-only, yet they ship without a native GUI module. **GUI-agents** built on vision-language action models cover Brain and Eyes but expose only shallow Body, narrow Hands (mostly mouse and keyboard), and restricted Feet, leaving them unable to write code, manage files, or sustain long workflows. ALE's task workflows demand the union of both surfaces, so the agent class the benchmark evaluates is the **Generalist CUA-agent** (GCUA): an agent with full capability across all five layers. **We adopt the "Generalist" qualifier deliberately, because the industry often equates CUA-agents with GUI-agents; this conflation is incomplete.**

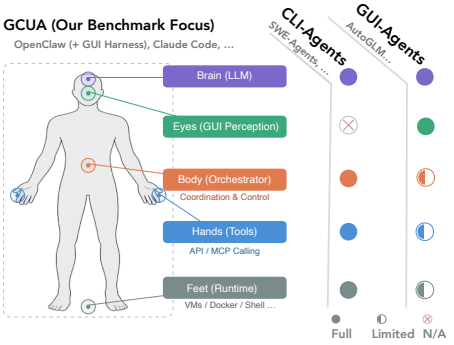


Figure 7: **Agent capability taxonomy.** Five functional layers define an agent's operational surface. Generalist CUA-agents (GCUA) possess full capability across all layers; CLI-agents lack visual perception (Eyes); GUI-agents have limited orchestration, tool use, and runtime access (Body, Hands, Feet).

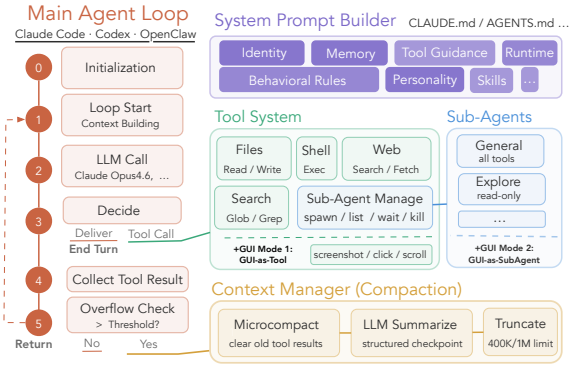


Figure 8: **Typical GCUA harness architecture.** The main agent loop (left) cycles through context building, LLM inference, action decision, tool execution, and overflow management. The system prompt builder, tool system (including GUI harness via MCP), sub-agents, and context compaction manager are shared across mainstream harness implementations.

The harness layer that mediates model and environment has converged toward a structure rich enough to support GCUA. Early agents revolved around a thin reasoning loop in the style of ReAct [54] (interleaved *Think* and *Action* steps); contemporary harnesses such as Claude Code [4], Codex [33], and OpenClaw share a richer macro-architecture that we treat as representative of the modern agent-harness layer (Figure 8): a main agent loop, a modular system prompt builder, a unified tool system, sub-agent dispatch, and a context compaction manager for long-horizon runs. Because these harnesses are CLI-native, lifting them to GCUA reduces to adding GUI capability. We use two modes: **GUI-as-Tool** exposes GUI operations as ordinary tools in the main loop, while **GUI-as-SubAgent** delegates GUI interaction to a specialized vision-language sub-agent. We currently reserve GUI-as-SubAgent for models without native vision input, such as DeepSeek V4 [12]. Our primary benchmark evaluation uses GUI-as-Tool to measure integrated visual reasoning and action over the full task. Component-level harness internals are deferred to Appendix C.4.

3.3 Evaluation Modes

The deliverables that `evaluate()` must score are highly heterogeneous (CAM toolpaths, financial workbooks, 3D meshes, game world states, rendered screenshots, structured filings, free-text reports). Rather than impose a single scoring metric, ALE composes every task's evaluation along two orthogonal axes. **(i) Comparison form**: task authors pick from a small palette of artifact modes: exact / hashed values, structured numeric or tabular fields with manifest-driven tolerances, geometric surface or point-cloud distances, visual appearance (vision-LLM judge), behavioral world state under a fixed input trajectory, and free-text rubric scoring. **(ii) Composition**: the per-artifact signals are combined either by a *gate-and-score* pattern or by averaging a binary checklist or per-file scores. Gate-and-score is the most common pattern: a binary precondition (e.g., "no toolpath collision," "file parses without error") must pass before a continuous quality metric (surface deviation, dimensional accuracy, etc.) is evaluated; failure on the gate forces the task score to 0 regardless of partial progress on other criteria. The full mode taxonomy with per-task-workflow assignments, helper APIs, and worked examples are given in Appendix C.3.

ALE **deliberately avoids LLM-as-judge** wherever a deterministic alternative exists; a task workflow whose only proposed scoring path is "ask a model whether the result looks correct" is rejected at QC and re-engineered to expose a checkable artifact. The minority of task workflows that genuinely require an LLM judge (video clip, game screenshot, rendered scene, etc) are scored not by general-purpose holistic prompts but by narrow, evidence-anchored yes/no probes whose answers code aggregates into the score. Each task workflow exposes a list of task instances (e.g., the 18 workpiece instances in `manufacturing/gcode`) that share a single `evaluate()` but differ in input and reference.

Table 1: **Main results on ALE.** Each difficulty level reports the full-pass rate (**Pass**, %), the mean score (**Score**, %), total API cost (💵), total wall-clock time (🕒), and total token use (**Tok.**). The final **Overall Pass Rate** column reports the full-pass rate over all distinct tasks in the three difficulty levels. "–" cost data not available. Superscript ± values denote score standard deviations estimated from three independent runs of the same task instance; due to compute budget constraints, only a subset of configurations include repeated runs. †Model uses an additional visual sub-agent for visual perception. The lower panel reports **ALE-CLI**, the Linux-only subset, comparing CLI agents alongside GCUA references (*).

	Near-Term (67 tasks)					**Full-Spectrum** (55 tasks)					**Last-Exam** (38 tasks)					**Overall**
	Pass (%)	Score	💵	🕒	Tok.	Pass (%)	Score	💵	🕒	Tok.	Pass (%)	Score	💵	🕒	Tok.	**Pass Rate**
Mainstream Agent Harnesses (paired LLM + GUI-as-Tool)																
Codex [33] (GPT-5.5 [35])	**38.1**	64.7$^{\pm 2}$	$243	154h	550M	22.7	36.0$^{\pm 3}$	$157	125h	389M	0.0	11.2$^{\pm 2}$	$170	111h	754M	**24.0**
ALE-Claw (GPT-5.5 [35])	32.8	**67.4**	$148	20h	168M	**23.6**	**41.1**	$71	14h	56M	**2.6**	**12.8**	$107	18h	127M	23.0
Claude Code [4] (Fable 5 [5])	34.3	63.4	$953	76h	348M	20.9	34.1$^{\pm 1}$	$609	67h	333M	0.0	5.2	$840	70h	236M	22.0
Cursor [10] (GPT-5.5 [35])	32.1	60.8$^{\pm 1}$	$69	49h	58M	20.0	32.7	$50	21h	37M	**2.6**	10.7	$61	21h	64M	20.7
Cursor [10] (Composer 2.5 [11])	34.3	61.1	$23	158h	95M	18.2	30.8	$30	40h	131M	0.0	8.8	$34	62h	152M	20.4
Cursor [10] (Opus 4.7 [7])	29.9	61.2	$558	38h	112M	20.0	37.6	$866	19h	225M	**2.6**	11.4	$588	21h	122M	20.4
Droid [14] (GPT-5.5 [35])	29.9	58.2	$106	32h	99M	16.4	33.4	$71	22h	60M	**2.6**	11.3	$74	38h	92M	19.1
ALE-Claw (Opus 4.7 [7])	28.4	60.5	$312	26h	361M	18.2	36.6	$258	19h	302M	0.0	7.9	$594	46h	710M	18.4
Gemini CLI [15] (Gemini 3.1 Pro [17])	26.9	53.5	$342	113h	240M	12.7	26.4	$962	94h	483M	0.0	0.9	$733	74h	498M	15.8
Claude Code [4] (Opus 4.8 [8])	26.9	60.2	$806	289h	192M	10.9	27.5	$486	79h	119M	0.0	5.2	$602	99h	158M	15.8
Claude Code [4] (Opus 4.7 [7])	20.9	54.3	$496	20h	121M	12.7	29.1	$747	15h	204M	0.0	10.0	$685	20h	171M	13.2
Droid [14] (Opus 4.7 [7])	27.6	60.2$^{\pm 1}$	$737	21h	182M	3.6	10.9	$167	5h	40M	0.0	8.0	$496	13h	147M	12.8
ALE-Claw (GPT-5.4 [34])	20.9	44.3	$66	22h	179M	9.1	22.9	$179	26h	597M	0.0	8.1	$100	21h	322M	11.8
Codex [33] (GPT-5.4 [34])	13.4	22.8	$56	28h	87M	3.6	7.7	$42	10h	67M	0.0	0.0	$46	14h	69M	7.2
Grok CLI [50] (Grok 4.3 [50])	9.0	30.4$^{\pm 1}$	$126	28h	104M	7.3	17.0	$97	21h	78M	0.0	2.3	$73	18h	60M	6.6
LLM Model Comparison (fixed OpenClaw [36] + GUI-as-Tool)																
GPT-5.5 [35]	**35.8**	**65.7**	$218	48h	224M	18.2	32.1	$104	27h	102M	0.0	**10.9**	$155	27h	179M	**21.1**
GPT-5.4 [34]	33.6	57.8$^{\pm 1}$	$70	54h	104M	**19.4**	**34.3**$^{\pm 1}$	$130	73h	285M	0.0	7.3$^{\pm 1}$	$110	47h	218M	20.5
Claude Opus 4.7 [7]	26.9	56.5	$493	57h	218M	10.9	27.5	$393	37h	175M	0.0	4.3	$842	55h	454M	15.1
Gemini 3.1 Pro [17]	26.1	48.3$^{\pm 1}$	$570	61h	626M	10.9	23.6	$1387	74h	1854M	0.0	3.1	$1176	52h	1417M	14.1
Claude Opus 4.6 [6]	22.4	51.2$^{\pm 1}$	$242	58h	238M	13.6	28.5$^{\pm 1}$	$115	67h	95M	0.0	5.6$^{\pm 1}$	$160	52h	119M	14.1
DeepSeek V4 Pro†[12]	19.9	43.8$^{\pm 2}$	$131	106h	356M	10.9	23.8$^{\pm 2}$	$73	66h	201M	0.0	2.5$^{\pm 1}$	$131	73h	384M	12.4
Qwen3.7 Max [41]	17.9	46.9	$268	112h	509M	10.9	27.2	$308	43h	592M	0.0	6.4	$196	44h	382M	11.8
GLM 5.1†[55]	20.1	45.6$^{\pm 2}$	$112	124h	329M	9.1	21.8$^{\pm 1}$	$237	116h	580M	0.0	6.2$^{\pm 1}$	$188	100h	575M	11.5
Claude Sonnet 4.6 [9]	16.4	40.6	$120	36h	154M	7.3	19.6	$67	18h	101M	0.0	1.3	$48	13h	79M	9.9
Kimi K2.6 [28]	15.7	35.1$^{\pm 2}$	$47	119h	208M	6.4	18.2$^{\pm 1}$	$40	111h	140M	0.0	1.6	$36	85h	121M	9.2
MIMO v2.5 [26]	11.9	35.1$^{\pm 2}$	$15	79h	206M	9.1	20.8$^{\pm 1}$	$23	66h	321M	0.0	3.4$^{\pm 1}$	$13	62h	221M	8.6
Qwen3.6 Plus [2]	12.7	35.8$^{\pm 2}$	$100	117h	466M	8.2	22.9$^{\pm 1}$	$120	89h	480M	0.0	5.0$^{\pm 1}$	$89	67h	376M	8.6
MiniMax M2.7†[27]	10.4	24.5$^{\pm 1}$	$9	85h	131M	3.6	8.4$^{\pm 1}$	$10	64h	146M	0.0	1.3	$10	56h	105M	5.9
Grok 4.3 [50]	6.7	24.4$^{\pm 1}$	$38	78h	143M	3.6	12.9$^{\pm 2}$	$28	56h	97M	0.0	2.3$^{\pm 1}$	$25	51h	84M	4.3
Agent Harness Comparison (fixed GPT-5.5 [35] + GUI-as-Tool)																
Codex [33]	**38.1**	64.7$^{\pm 2}$	$243	154h	550M	22.7	36.0$^{\pm 3}$	$157	125h	389M	0.0	11.2$^{\pm 2}$	$170	111h	754M	**24.0**
ALE-Claw	32.8	**67.4**	$148	20h	168M	**23.6**	**41.1**	$71	14h	56M	**2.6**	**12.8**	$107	18h	127M	23.0
OpenClaw [36]	35.8	65.7	$218	48h	224M	18.2	32.1	$104	27h	102M	0.0	10.9	$155	27h	179M	21.1
Cursor [10]	32.1	60.8$^{\pm 1}$	$69	49h	58M	20.0	32.7	$50	21h	37M	**2.6**	10.7	$61	21h	64M	20.7
Droid [14]	29.9	58.2	$106	32h	99M	16.4	33.4	$71	22h	60M	**2.6**	11.3	$74	38h	92M	19.1
Agent Harness Comparison (fixed Claude Opus 4.7 [7] + GUI-as-Tool)																
Cursor [10]	**29.9**	**61.2**	$558	38h	112M	**20.0**	**37.6**	$866	19h	225M	**2.6**	**11.4**	$588	21h	122M	**20.4**
ALE-Claw	28.4	60.5	$312	26h	361M	18.2	36.6	$258	19h	302M	0.0	7.9	$594	46h	710M	18.4
Claude Code [4]	20.9	54.3	$496	20h	121M	12.7	29.1	$747	15h	204M	0.0	10.0	$685	20h	171M	13.2

ALE-CLI (Linux-only subset, 105 tasks).

	Near-Term (47 tasks)					**Full-Spectrum** (42 tasks)					**Last-Exam** (20 tasks)					**Overall**
	Pass (%)	Score	💵	🕒	Tok.	Pass (%)	Score	💵	🕒	Tok.	Pass (%)	Score	💵	🕒	Tok.	**Pass Rate**
ALE-CLI: CLI Agents + GUI-as-Tool																
Codex [33] (GPT-5.5 [35])*	**37.2**	**68.3**$^{\pm 3}$	$142	86h	258M	**19.0**	**32.2**$^{\pm 2}$	$107	93h	213M	0.0	**13.1**$^{\pm 3}$	$101	62h	394M	**23.3**
Claude Code [4] (Sonnet 4.6 [9])*	24.5	54.2$^{\pm 2}$	$38	23h	84M	14.3	28.3$^{\pm 1}$	$54	26h	112M	0.0	8.9$^{\pm 2}$	$63	38h	118M	16.7
ForgeCode [45] (Sonnet 4.6 [9])	22.0	48.1$^{\pm 3}$	$44	41h	170M	8.7	17.6$^{\pm 2}$	$39	32h	112M	0.0	2.8	$20	52h	83M	13.3
Hermes [32] (Sonnet 4.6 [9])	21.6	52.4$^{\pm 3}$	$219	31h	216M	8.7	22.5$^{\pm 1}$	$132	23h	121M	0.0	4.1$^{\pm 2}$	$123	24h	192M	13.2
Terminus [24] (Sonnet 4.6 [9])	19.1	50.0$^{\pm 2}$	$75	42h	208M	8.3	22.7	$55	32h	513M	0.0	4.1	$148	53h	630M	11.9
OpenHands [49] (Sonnet 4.6 [9])	13.8	31.9$^{\pm 5}$	$74	36h	213M	7.1	13.6$^{\pm 3}$	$21	26h	191M	0.0	3.7$^{\pm 2}$	$81	49h	288M	9.0

4 Experiment

ALE's task instances are drawn from authentic professional workflows that experts carry out on real computer environments, routinely interleaving shell commands, GUI applications, file manipulation, and web research within a single task. As argued in Section 3.2, this operational surface requires **Generalist CUA-agents** (GCUA) with full capability across all five functional layers (Brain, Eyes, Body, Hands, Feet). We therefore evaluate all agent systems in GCUA configuration.

4.1 Main Results

To bring every agent into GCUA configuration, each system is extended with **GUI-as-Tool** mode: a unified CUA MCP bridge exposes 14 desktop-action tools (Table 5) as standard entries in the agent's tool system, so that a single foundation model reasons over both shell output and visual feedback within one action loop. Table 1 reports mean scores, pass rates, costs, and wall-clock times for agent systems and foundation models. Specifically, mean score is the mean fine-grained task score, full pass

rate is the share of tasks receiving full credit. Each run is capped at five hours; the overall timeout rate is 3.8%, ranging from ∼1% for lightweight harnesses to 5.7% for OpenClaw (see Appendix D.2). Rows are grouped into five blocks: (a) mainstream harness–backbone configurations; (b) model sweeps with the harness fixed to OpenClaw+GUI-as-Tool; (c) harness sweeps with the backbone fixed to GPT-5.5; (d) harness sweeps with the backbone fixed to Claude Opus 4.7; and (e) **ALE-CLI**, the 105 Linux-only task instances that can be attempted by CLI-only agents (ForgeCode, Hermes, Terminus) without GUI desktop access, reported in the lower panel of Table 1 with Codex and Claude Code as GCUA references.

ALE-CLI: a harder CLI-focused sub-benchmark. ALE-CLI is a natural comparison point to Terminal-Bench [24], which contains ∼100 terminal-centric tasks at a similar scale. However, ALE-CLI tasks are substantially harder and require longer agent sessions: Codex [33] with GPT-5.5 [35], the strongest configuration that achieves 82% on Terminal-Bench, reaches only a 23.3% overall pass rate on ALE-CLI (37.2% on Near-Term, 19.0% on Full-Spectrum, 0.0% on Last-Exam).

Three difficulty tiers. A single run of a frontier agent on one ALE task costs $3–10 on average and takes tens of minutes to hours. Evaluating the full 152-task public set is therefore expensive, so ALE organizes tasks into three difficulty tiers that let the community choose a subset matching its evaluation budget and goal. **Near-Term** (67 tasks) contains workflows that current frontier agents can partially complete, with top pass rates approaching 40%; these tasks are the most cost-effective target for short-term leaderboard competition and rapid iteration. **Full-Spectrum** (55 tasks) covers, by design, each of ALE's 55 subdomains with at least one task instance, ensuring broad domain coverage for comprehensive evaluation. **Last-Exam** (38 tasks) comprises the hardest workflows, on which most agents achieve a 0% pass rate; these tasks anchor the benchmark's long-term headroom and are best reserved for milestone evaluations rather than routine testing.

ALE-Claw: a Self-implemented GCUA reference. We implement **ALE-Claw** to test whether the basic GCUA components in Section 3.2, namely a single action loop, modular tools, GUI-as-Tool, and context compaction, suffice for achieving a comparable performance to frontier harnesses. ALE-Claw is a simplified implementation derived from OpenClaw [36] and scoped to isolated benchmark runs. It omits product features such as long-term memory management and user preference customization, which are useful for interactive agents such as Claude Code [4] but not required for single-task evaluation. Holding the foundation model fixed, ALE-Claw has comparable performance to default OpenClaw on ALE. Appendix C.4 documents the implementation differences.



Figure 9: **Experiment analysis overview.** (a) Domain-level mean scores for Claude Fable 5 and GPT-5.5, each averaged over harnesses with completed runs on the selected public task set; the sparse transportation domain is omitted. (b) Tool-call mix for the best available table-backed configuration per harness. (c) Tool-call mix for backbone models under a fixed OpenClaw harness. (d) Failure root-cause taxonomy for failed Claude Code + Opus 4.7 public-task runs.

4.2 Analysis

Domain-level performance. Figure 9(a) shows mean scores by domain for Claude Fable 5 and GPT-5.5, each averaged over harnesses. The two frontier models exhibit similar domain profiles: computational mathematics and agriculture/environment score highest (∼55–85%), followed by business and legal (∼50–55%), while education remains lowest (below 25%). This shared ranking likely reflects both an imbalance in intrinsic model capability across domains and uneven exposure to tool-use tasks during training, where code-adjacent domains receive far more coverage than specialized professional workflows.

Table 2: **Positioning of ALE against representative benchmarks.** Industry counts use the mapping in Appendix B.1; "–" indicates that the benchmark does not declare a domain taxonomy and is not directly comparable.

Benchmark	Task form	Size	Breadth	Domain grounding	Task source	Horizon	Verification
Knowledge & exam-style QA							
MMLU [20]	Knowledge QA	∼16K	26 / 55	Academic subjects	Mined (exams)	Seconds	Auto (exact match)
GPQA [42]	Knowledge QA	∼500	8 / 55	Hard sciences	Domain experts	Minutes	Auto (exact match)
HLE [39]	Knowledge QA	∼2.5K	20 / 55	Academic subjects	Domain experts	Minutes	Auto (exact / LLM-judge)
Agentic & computer-use benchmarks							
GAIA [25]	Tool-use QA	∼500	–	General assistant	Curators	Minutes	Auto (exact match)
SWE-bench [21]	Code patch	∼2K	5 / 55	OSS Python repos	Mined (GitHub)	Minutes–Hours	Auto (unit tests)
OSWorld [51]	GUI Operation	∼400	5 / 55	Desktop apps	Curators	Minutes	Auto (state checks)
WebArena [56]	GUI Operation	∼800	4 / 55	Web apps (6 sites)	Curators	Minutes	Auto (state checks)
Terminal-Bench 2 [24]	CLI Operation	∼100	6 / 55	Terminal workflows	Curators	Minutes–Hours	Auto (state checks)
PinchBench [22]	Assistant tasks	∼50	3 / 55	Productivity + coding	Curators	Minutes	Auto (tests / LLM-judge)
Economically grounded / project-style benchmarks							
GDPval [37]	Project deliverable	∼200	16 / 55	Top GDP occupations	Industry experts	Hours–Days	Human (expert)
RLI [23]	Project deliverable	∼250	14 / 55	Upwork's categories	Mined (Upwork)	Hours–Days	Human (expert)
ALE (ours)	**Project deliverable**	**∼1.5K**	**55 / 55**	**SOC 2018 + O*NET**	**Industry experts**	**Hours–Weeks**	**Auto (deterministic scripts)**

Tool usage. The tool traces, normalized using the taxonomy in Appendix C.4.1, reveal that both harness and model shape the tool-call mix (Figure 9(b,c)). GUI usage remains below task demand: 34% of public task instances designate graphical software as the primary tool, yet the GUI share stays small across most configurations, as agents execute GUI tasks through Bash/CLI substitutes.

Failure taxonomy. We classified the failed tasks of Claude Code + Opus 4.7 into a two-level taxonomy (Figure 9(d); details in Appendix D.3). Understanding and Approach failures together account for roughly three quarters of cases, indicating that the dominant bottleneck is domain knowledge rather than execution capability. Lacking specialized knowledge, agents default to ad-hoc scripts instead of the intended domain software, reinforcing the GUI-underutilization pattern above.

Additional analysis. Appendix D.4 further decomposes the performance variation into model and harness effects, finding that the choice of foundation model accounts for roughly $3\times$ the spread of the choice of agent harness among well-engineered systems. Appendix D.5 examines the cost, time, and token efficiency of each configuration (Figure 13), showing that higher resource consumption does not reliably translate to better performance. Appendix D.6 provides per-task instance score heatmaps (Figures 14–16) that visualize every task–agent combination across the three tiers.

5 Related Work

Table 2 positions ALE against prominent benchmarks; the "Breadth" column reports industry coverage by mapping each benchmark's published categories onto our 55-industry taxonomy. Knowledge and exam-style benchmarks such as MMLU [20], GPQA [42], and HLE [39] are topically broad but test what a model *knows*, not what it can *do*. Agentic benchmarks including SWE-bench [21], OSWorld [51], WebArena [56], and GAIA [25] add multi-step interaction and tool use but cover only a handful of software-centric domains and typically rely on curator-authored tasks rather than real professional workflows. The closest contemporaries, GDPval [37] and RLI [23], target economically grounded, project-scale evaluation but still leave large portions of the labor market untested (16 and 14 of 55 industries, respectively) and depend on expensive human grading. ALE closes these gaps: it is the first benchmark to cover all 55 SOC/O*NET industries, draws every task from a real project completed by one of 300+ practitioners, and replaces human evaluation with deterministic, rubric-based automated verification.

6 Conclusion

We introduced **ALE**, a benchmark of 960 expert-authored task workflows (1,490 task instances) across 55 digital industries, sourced from work experts have already shipped, anchored in the SOC/O*NET taxonomy, and scored through deterministic checks and structured rubrics rather than open-ended LLM judging. Frontier agents clear only a small fraction today; we release ALE as an instrument for closing the gap between benchmark success and GDP-relevant impact, where saturation would signal that agents can sustain the long-horizon, tool-intensive work professional practice actually requires.

7 Broader Impact

The premise of ALE is to quantitatively measure the usefulness of agents on increasingly complex, multi-step tasks that reflect real professional work requiring planning, perception, and creative problem-solving. Although agents currently pass only a small fraction of ALE tasks, continued advances in data and training protocols in response to more sophisticated benchmarks are likely to improve agentic capabilities. As a potentially disruptive technology, agentic AI may reshape labor markets, particularly entry-level roles held by workers with bachelor's or doctoral degrees, in ways that mirror earlier disruptions to blue-collar work caused by the automation of physical labor. By providing early, measurable signals of these changes, ALE can help policymakers, educators, employers, and workers anticipate workforce transitions and create new roles that keep pace with potential job displacement as agents become more capable. Finally, by democratizing benchmark development across domain experts and making governance, access policies, and evaluation procedures explicit and transparent, benchmarks such as ALE may support more informed regulation and the responsible development of increasingly powerful agentic technologies.

Acknowledgments and Disclosure of Funding

We gratefully acknowledge the Tianqiao & Chrissy Chen Institute (TCCI), Snorkel AI, and UniPat AI for their financial and credit support.

References

[1] Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-anything: Turn any software into an agent environment, 2026.

[2] Alibaba Cloud. Alibaba unveils qwen3.6-plus to accelerate agentic ai deployment for enterprises and alibaba's ai applications. https://www.alibabacloud.com/en/press-room/alibaba-unveils-qwen3-6-plus-to-accelerate-agentic, 2026. Accessed: 2026-05-05.

[3] AlphaEval Team. Alphaeval: Real-world agent benchmark. https://alphaeval.ai/, 2026. Accessed: 2026-04-09.

[4] Anthropic. Claude code: An agentic coding tool. https://docs.anthropic.com/en/docs/claude-code, 2025. Accessed: 2026-04-09.

[5] Anthropic. Claude fable 5 and claude mythos 5. https://www.anthropic.com/news/claude-fable-5-mythos-5, 2026. Accessed: 2026-06-11.

[6] Anthropic. Claude opus 4.6 system card. https://www-cdn.anthropic.com/14e4fb01875d2a69f646fa5e574dea2b1c0ff7b5.pdf, 2026. Accessed: 2026-05-07.

[7] Anthropic. Introducing claude opus 4.7. https://www.anthropic.com/news/claude-opus-4-7, 2026. Accessed: 2026-05-05.

[8] Anthropic. Introducing claude opus 4.8. https://www.anthropic.com/news/claude-opus-4-8, 2026. Accessed: 2026-06-11.

[9] Anthropic. Claude sonnet 4.6. https://www.anthropic.com/claude/sonnet, 2026. Accessed: 2026-05-05.

[10] Cursor. Cursor cli. https://cursor.com/en-US/cli, 2025. Accessed: 2026-05-05.

[11] Cursor. Introducing composer 2.5. https://cursor.com/blog/composer-2-5, 2026. Accessed: 2026-06-11.

[12] DeepSeek. Models & pricing. https://api-docs.deepseek.com/quick_start/pricing, 2026. Accessed: 2026-05-05.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[14] Docker. Droid. https://docs.docker.com/ai/sandboxes/agents/droid/, 2026. Accessed: 2026-05-05.

[15] Google. Gemini cli. https://google-gemini.github.io/gemini-cli/, 2025. Accessed: 2026-05-05.

[16] Google. Gemini wins international collegiate programming contest gold. https://blog.google/technology/google-deepmind/gemini-gold-icpc/, 2025. Accessed: 2026-04-09.

[17] Google. Gemini 3.1 pro: A smarter model for your most complex tasks. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/, 2026. Accessed: 2026-05-05.

[18] Google DeepMind. Advanced version of gemini with deep think officially achieves gold-medal standard at the international mathematical olympiad. https://deepmind.google/discover/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/, 2025. Published 2025-07-21.

[19] Yu Guo, Ryan Wen Liu, Jingxiang Qu, Yuxu Lu, Fenghua Zhu, and Yisheng Lv. Asynchronous trajectory matching-based multimodal maritime data fusion for vessel traffic surveillance in inland waterways. *IEEE Transactions on Intelligent Transportation Systems*, 24(11):12779–12792, 2023. doi: 10.1109/TITS.2023.3285415.

[20] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[21] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[22] Kilo AI. PinchBench: An OpenClaw agent benchmark leaderboard. https://pinchbench.com/, 2026. Snapshot of 2026-04-13.

[23] Mantas Mazeika, Alice Gatti, Cristina Menghini, Udari Madhushani Sehwag, Shivam Singhal, Yury Orlovskiy, Steven Basart, Manasi Sharma, Denis Peskoff, Elaine Lau, Jaehyuk Lim, Lachlan Carroll, Alice Blair, Vinaya Sivakumar, Sumana Basu, et al. Remote labor index: Measuring ai automation of remote work, 2025.

[24] Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations, ICLR 2026*, 2026.

[25] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf 0008, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[26] MiMo-V2.5. Mimo-v2.5. https://huggingface.co/collections/XiaomiMiMo/mimo-v25, 2026.

[27] MiniMax. Minimax-m2.7. https://huggingface.co/MiniMaxAI/MiniMax-M2.7, 2026. Accessed: 2026-05-05.

[28] Moonshot AI. Model list. https://platform.kimi.ai/docs/models, 2026. Accessed: 2026-05-05.

[29] National Center for O*NET Development. O*NET 30.2 Database, 2026. URL https://www.onetcenter.org/database.html. Sponsored by the U.S. Department of Labor, Employment and Training Administration.

[30] National Institutes of Health. Nih-wide strategic plan for fiscal years 2021–2025. Technical report, National Institutes of Health, 2021. URL https://www.nih.gov/about-nih/nih-wide-strategic-plan.

[31] National Science Board, National Science Foundation. Science and engineering indicators 2024: The state of u.s. science and engineering. Technical Report NSB-2024-3, National Science Foundation, Alexandria, VA, 2024. URL https://ncses.nsf.gov/pubs/nsb20243.

[32] Nous Research. Hermes agent cli. https://hermes-agent.ai/tools/hermes-agent-cli, 2026. Accessed: 2026-05-05.

[33] OpenAI. Codex: A cloud-based software engineering agent. https://openai.com/index/introducing-codex/, 2025. Accessed: 2026-04-09.

[34] OpenAI. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/, 2026. Accessed: 2026-05-07.

[35] OpenAI. Introducing gpt-5.5. https://openai.com/index/introducing-gpt-5-5/, 2026. Accessed: 2026-05-07.

[36] OpenClaw. Agent runtime. https://docs.openclaw.ai/agent, 2026. Accessed: 2026-05-05.

[37] Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, Natalie S. Kim, Patrick Chao, Samuel Miserendino, Gildas Chabot, David Li, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. Gdpval: Evaluating ai model performance on real-world economically valuable tasks, 2025.

[38] N. G. Peterson, M. D. Mumford, W. C. Borman, P. R. Jeanneret, et al. Understanding work using the occupational information network (o*net): Implications for practice and research. *Personnel Psychology*, 54(2):451–492, 2001. doi: 10.1111/j.1744-6570.2001.tb00100.x.

[39] Long Phan, Alice Gatti, Nathaniel Li, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, Dan Hendrycks, Ziwen Han, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, et al. A benchmark of expert-level academic questions to assess ai capabilities. *Nature*, 649(8099):1139–1146, January 2026. doi: 10.1038/s41586-025-09962-4.

[40] Dilip K. Prasad, Deepu Rajan, Lily Rachmawati, Eshan Rajabally, and Chai Quek. Video processing from electro-optical sensors for object detection and tracking in a maritime environment: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):1993–2016, 2017. doi: 10.1109/TITS.2016.2634580.

[41] Qwen Team. Qwen3.7: The agent frontier. https://qwen.ai/blog?id=qwen3.7, 2026. Accessed: 2026-06-11.

[42] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023.

[43] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

[44] Elham Tabassi. Artificial intelligence risk management framework (ai rmf 1.0). Technical Report NIST AI 100-1, National Institute of Standards and Technology, Gaithersburg, MD, 2023.

[45] Tailcall. ForgeCode: Ai enabled pair programmer for claude, gpt, o series, grok, deepseek, gemini and 300+ models. https://github.com/tailcallhq/forgecode, 2024. Accessed: 2026-04-30.

[46] U.S. Bureau of Labor Statistics. 2018 standard occupational classification definitions, 2018.

[47] U.S. Congress. Chips and science act of 2022, public law 117-167, 2022. URL https://www.congress.gov/117/plaws/publ167/PLAW-117publ167.pdf.

[48] U.S. National Science Foundation. Chips and science, 2024. URL https://www.nsf.gov/chips.

[49] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

[50] xAI. Models and pricing. https://docs.x.ai/developers/models, 2026. Accessed: 2026-05-05.

[51] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[52] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[53] Qianyu Yang, Yang Liu, Jiaqi Li, Jun Bai, Hao Chen, Kaiyuan Chen, Tiliang Duan, Jiayun Dong, Xiaobo Hu, Zixia Jia, Yang Liu, Tao Peng, Yixin Ren, Ran Tian, Zaiyuan Wang, Yanglihong Xiao, Gang Yao, Lingyue Yin, Ge Zhang, Chun Zhang, Jianpeng Jiao, Zilong Zheng, and Yuan Gong. $onemillion-bench: How far are language agents from human experts?, 2026. URL `https://arxiv.org/abs/2603.07980`.

[54] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[55] Z.AI. Glm-5.1. `https://docs.z.ai/guides/llm/glm-5.1`, 2026. Accessed: 2026-05-05.

[56] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Appendix Table of Contents

A Authors **16**

A.1 Contributors & Affiliations 16

B Benchmark Construction Details **18**

B.1 Taxonomy Details 18

B.2 Task Construction Pipeline Details 20

B.3 Industry Task Cards and Metadata 21

C Evaluation Pipeline Details **27**

C.1 Pipeline Architecture Details 27

C.2 Task Specification Protocol 27

C.3 Evaluation Modes: Full Taxonomy and Worked Examples 28

C.4 Agent Harness Internals 32

D Extended Experiment Results and Analysis **34**

D.1 Public-Subset Representativeness 34

D.2 Timeout Analysis 34

D.3 Failure Taxonomy Classification 35

D.4 Model vs. Harness Effect 36

D.5 Cost, Time, and Token Efficiency 37

D.6 Per-Task Instance Score Heatmaps 37

A Authors

Each contributor below is listed with their full affiliation (superscript), keyed to the numbered institution list at the end of this section.

A.1 Contributors & Affiliations

Organization & Execution Team. Yiyou Sun[1], Xinyang Han[1], Weichen Zhang[1], Yuanbo Pang[1], Tianyu Wang[3], Yuhan Cao[1], Yixiao Huang[1], Xiang Liu[2], Zonglin Di[11,1], Chris Duroiu[1], Haoyun Zhang[1], Jeffrey Lin[1], Weishu Zhang[1], Tyler Zeng[1], Ying Yan[3], Bo Liu[5], Hanson Wen[1], Mingyang Xu[6], Xiaoyuan Liu[1], Zimeng Chen[1], Zhaowei Zhang[4], Weiyan Shi[7], Amanda Dsouza[8], Vincent Sunn Chen[8], Jingxu Xie[1], Dewen Lei[9], Hanxuan Zhang[10], Yinhe Wang[110], Xinran Zhang[62], Chao Zhou[3], Sheldon Huang[109], Dawn Song[1]

Advisory Committee. Patrick Bryant[12], Carl Boettiger[1], Yamini Rangan[13], Bradley Rothenberg[14], Kyle Steinfeld[1], Arvind Rao[6], Tapio Schneider[15], Georgios Yannakakis[16], Laure Zanna[17], Kaan Ozbay[17], Ida Sim[18], Tarek Zohdi[1], George Em Karniadakis[19], Jack Gallant[1], Teresa Head-Gordon[1]

Data Contributors. Chris Yuhao Liu[11], Rongwang Hu[3], Yafei Cheng[3], Aras Bacho[15], Zengyi Qin[20], Alan Yuille[21], Yixiong Chen[21], Hao Liu[5], Lin Zeng[3], Bo Chen[3], Zhendong Li[22], Litian Gong[23], Shashank Muralidhar Bharadwaj[24], Thorben Jansen[25], HAIXIANG TANG[26], Jianhong Tu[11], Jixian Liu[21], Maojia Song[27], Ruheng Wang[28], Shengcao Cao[26], Yanxin Zhang[24], Yingxuan Yang[1], Zongzheng Zhang[3], Honglin Bao[29], Hyunjun Kim[30], Peiran Xu[31], Shuo Lu[3], Zheng Zhang[32], Zhonghua Wang[33], Aditya Prakash[34], Bohan Lyu[1], Chang Ma[35], Haoxuan Chen[5], Hengduan Fan[3], Molei Tao[36], Roman Sitelew[37], Russell Alan Poldrack[5], Yang Liu[38], Yang Liu[11], Yichen Wu[39], Yuanze Lin[40], Zhenglu Li[41], Zibu Wei[3], Zijiao Chen[5], Abderrahim Bendahi[42], Alexander Schubert[18], Chase Broderick[35], Chengzhi Liu[43], Haoran Sun[44], Haoyang Li[45], Haoyue Hua[3], Jiayuan Ding[46], Jing Huang[47], Shangding Gu[1], Tuan Vinh[48], Xiatao Sun[44], Yanqiong Jiang[41], Yiran Hu[1,9], Zechen Xu[49], Ziao Yang[50], Chenguang Wang[11], Costas Spanos[1], Dongrong Yang[5], Haocheng Wang[1], Jiayi Cheng[17], JICHENG WANG[51], Junbo Peng[52], Kangqi Zhang[6], Keying Kuang[1], Lichao Sun[22], Ming Yin[53], Orson Xu[54], Vincent Siu[11], Wanzhe Liao[3], Wenqi Shi[28], Wenxi Deng[1], Xiaopan Zhang[36], Yangjunfeng Zhu[55], Yifan Shen[56], Yi Jiang[28], Yinsheng Li[57], Youbang Pan[3], Yuan Li[47], Yuchen Zhu[36], Yunshi Zhang[59], Zhenning Yang[6], Ziyan Rao[60], Alireza Rafiei[52], Andrew Glover[35], Bixuan Ren[49], Boqin Yuan[45], Chonghao Cai[61], Deren Lei[42], Hanwen Xing[62], Haoran Jin[6], Hector Gomez[63], Ishaan Singh Chandok[64], Jiaming Hu[65], Jianxin Sun[66], Justin Purnomo[1], Kunyang Sun[1], Lei Ding[11], Longtai Liao[67], Lukas Kim[1], Ming Liu[68], Qingpo Wuwu[3], Ran Jin[69], Ruiqi Zhang[1], Ting Sun[3], Wenbo Huang[26], Xuandong Zhao[1], Peter Chen[1], Chuanyang Jin[21], Qiming Bao[111], Xuhuan Shen[1], Yifan Zhang[26], Yi Li[1], Yingze Wang[1], Yixin Wang[5], Yushi Yang[40], Zhen Dong[43], Zhiling Yan[22], Zhongxing Xu[33], Zibo Zhao[70], Zixin Jiang[49], Ali Emami[52], Andy Zeyi Liu[44], Angela Wu[1], Aylin Caliskan[35], Baihe Huang[1], Baixiang Huang[52], Benjamin Yang[35], Benran Zhang[41], Brian Zhao[71], Carlo Bosio[1], Charlie Guo[72], Che Liu[73], Chenchen Du[74], Chen Dai[5], Chenlong Zhang[43], Chenyu Li[75], Chenyu Zhu[3], Chi Zhang[3], Core Francisco Park[64], Daofeng Li[76], Dehao Wu[26], Dowling Wong[77], Erin Kasson[76], Fangyu Wu[54], Fan Xu[1], Fiona Shen[44], Haimin Hu[21], Haitao Li[3], Hang Zhou[78], Hannah Szlyk[76], Hanning Zhang[43], Hanwen Zhang[64], Hao Fan[76], Haoran Ye[3], HaoYang Shang[79], Haoyue Bai[24], Hengyu Fu[1], Hongwei Li[43], Huiqi Wang[1], Jiachen Li[36], Jialu Wang[11], Jiaming Liu[56], Jiawei Shen[76], Jierui Zuo[35], Jingxuan Fan[64], Jingyan Li[3], Jirong Yang[6], Jonah So[43], Jorin Overwiening[64], Joseph Cavanagh[1], Joseph Lee[35], Junge Zhang[36], Kaiyuan Liu[35], Kaustubh Dhole[52], Kevin Qinghong Lin[40], Kunpeng Zhang[35], Kyle Montgomery[11], Layna Paraboschi[76], Liana[3], Lihui Liu[80], Liwei Zhou[21], Lu Lu[1], Mengyuan "Millie" Wu[54], Mikko Tolonen[81], Mingxuan Yan[36], Minqi Shi[3], Murphy Niu[43], Pan Lu[5], Patricia Cavazos-Rehg[76], Qijian Zhao[3], Qiuyang Mang[1], Rimika Jaiswal[43], Ruiling Xu[26], Run-Ze Fan[82], Ruoxi Ning[9], Santanu Bosu Antu[44], Scott Rankin[1], Sen Yang[44], Shi Gu[83], Shi Qiu[84], Sichen Zhu[36], Sihan Xu[6], Siqi Zhu[26], Song Zhang[3], Sophia Sun[62], Stefano De Castro[1], Swati Rajwal[52], Tianneng Shi[1], Tianxing Chen[3], Tianyue Zhou[1], Ting Wang[76], Ventus Shaw[3], Vladislav Susoy[64], Weihang Xiao[62], Wei Wang[47], Wenbo Lv[3], Wenjin Zhang[76], Wenshuo Zhang[3], Wenyu Liang[3], Xiaohu Yu[85], Xiao Li[76], Xiaoli Yang[5], Xin Lan[86], Xinyu Wang[24], Xuan Liu[45], Xuewen Chen[3], Xu Pan[64], Yan Zhu[1], Yicheng Lyu[3], Yifan Gu[3], Yimin Zhao[35], Ying Zhang[3], Yin Ren[64], Yipu Chen[36], Yiren Rong[1], Yi Shao[57], Yishu Zhu[1], Yizhi Zhang[87], Yongyi Xiong[88], Yuangang Li[89], Yuchen Wu[35], Yuhaohua Zheng[7], Yuhao Wu[76], Yunfu Deng[24], Yushan Li[1], Yu Wu[81], Yuzheng Xu[90,91], Yuzhou Nie[43], Zehao Wang[36], Zhehang Du[47], Zhengming Yu[83], Zhihan Jiang[54], Zhi Li[92], Zhixu

Li[23], Zhuoran Yi[93], Tao Sun[1], Zhun Wang[1], Justin Xu[40], Shi Bo[65], Helena Casademunt[64], Wenjia Chen[94], Cheng Cheng[6], Nawen Deng[95], Rao Fu[19], Tianfu Fu[56], Yifan Han[3], Zhenyu He[1], Qiao Jin[48], Langlang Li[1], Yuetai Li[35], Sylvia Liu[3], Subhabrata Mukherjee[46], Yunqi Ouyang[3], He Ren[96], Dawei Shi[97], Haoran Wu[98], Zhiyue Wu[3], Hannah Yao[99], Jenny Yu[94], Rhea Zhan[1], Luqing Zhou[100], Blake Zhu[97], Junfan Zhu[3], Sanjiv Das[101], Yanjun Gao[102], Wenbo Guo[43], Erika J Schneider[49], Bing Liu[42], Zhiyong Lu[48], Yian Ma[45], Mark Mueller[1], Radha Poovendran[35], Somayeh Sojoudi[1], Huaxiu Yao[84], Yinglun Zhu[23], Károly Zsolnai-Fehér[112], Yegor Denisov-Blanch[5], Changran Hu[1], Vincent Koc[103], Hezheng Yin[104], Naomi Xia[104], Xinyi Bai[58], Arsh Shah Dilbagi[105], Jing Gu[106], Yutong Liu[42], Lin Shi[75], Qizheng Zhang[5], Yuexing Hao[107], Xiaoming Li[107], Xiangyi Li[108], Bingran You[108,1], Wenjun Ke[15], Daoan Zhang[113], Ari Nemet[114], Tianyi Zhang[41], Xingda Lyu[35], Pranit Chawla[37], Genta Indra Winata[115], Maosen Tang[75], Purva Chiniya[62], Qiuyi Chen[116], Steven Dillmann[5], Wanghan Xu[117], Wenqi Yang[37], Xiaojun Wu[37], Xin Luo[6], Yue Huang[118]

Affiliations

1. University of California, Berkeley
2. National University of Singapore
3. Independent Contributor
4. Individual Researcher
5. Stanford University
6. University of Michigan
7. Northeastern University
8. Snorkel AI
9. University of Waterloo
10. Databricks
11. University of California, Santa Cruz
12. SciLifeLab
13. HubSpot
14. nTop
15. California Institute of Technology
16. University of Malta
17. New York University
18. University of California, San Francisco
19. Brown University
20. OpenAGI Research Foundation
21. Johns Hopkins University
22. Lehigh University
23. University of California, Riverside
24. University of Wisconsin-Madison
25. Leibniz Institute for Science and Mathematics Education
26. University of Illinois Urbana-Champaign
27. Singapore University of Technology and Design
28. University of Texas Southwestern Medical Center
29. University of Chicago
30. KAIST
31. University of California, Los Angeles
32. Adobe
33. Monash University
34. IMPACT Lab at Children's National Hospital
35. University of Washington
36. Georgia Institute of Technology
37. Independent Researcher
38. North Carolina Central University
39. University of Nottingham
40. University of Oxford
41. University of Southern California
42. Meta
43. University of California, Santa Barbara
44. Yale University
45. University of California, San Diego
46. Hippocratic AI
47. University of Pennsylvania
48. U.S. National Institutes of Health
49. Syracuse University
50. Michtom School of Computer Science, Brandeis University
51. University of California, Davis
52. Emory University
53. Duke University
54. Columbia University
55. Tsinghua University
56. Massachusetts Institute of Technology
57. McGill University
58. Google
59. BUPT
60. UMass Chan Medical School
61. ETH Zurich
62. Amazon
63. Purdue University
64. Harvard University
65. Boston University
66. University of Nebraska-Lincoln
67. Oracle
68. Iowa State University
69. AstraZeneca
70. Arizona State University
71. Mission San Jose High School
72. X School
73. Imperial College London
74. Aalto University
75. Cornell University
76. Washington University in St. Louis
77. Karlsruhe Institute of Technology
78. LeverArch.ai
79. BreathingCORE Limited
80. Wayne State University
81. University of Helsinki
82. University of Massachusetts Amherst
83. Texas A&M University
84. University of North Carolina at Chapel Hill
85. University of Melbourne
86. Michigan State University
87. Jiji Information & Communication Technology Branch
88. Carnegie Mellon University
89. University of California, Irvine
90. University of Tokyo
91. NanoFrontier
92. University of Colorado Boulder
93. University of Utah
94. Morgan Stanley
95. Photon Fund
96. PIMCO
97. Brix Labs
98. Intellipro

99. JPMorgan Chase
100. Goldman Sachs
101. Santa Clara University
102. University of Colorado Anschutz Medical Campus
103. OpenClaw Foundation
104. MoE Fund
105. Adaline
106. Elorian AI
107. MatrAIx
108. BenchFlow
109. University of Toronto
110. Savannah College of Art and Design
111. University of Auckland
112. Two Minute Papers
113. University of Rochester
114. Mountain View High School
115. Capital One
116. Hithink RoyalFlush Information Network Co., Ltd.
117. Shanghai Jiao Tong University
118. University of Notre Dame

B Benchmark Construction Details

B.1 Taxonomy Details

B.1.1 Taxonomy Definition

Scope. ALE evaluates whether generalist computer-use agents, operating through digital interfaces, software tools, files, and APIs, can complete valuable professional work across fields. The taxonomy centers on workflows whose primary outputs can be produced at a computer, depend on domain expertise, and yield artifacts that are objectively evaluable.

Occupational backbone. We use SOC 2018 as the occupational backbone and O*NET to interpret occupational content through linked task, work-activity, and tools-and-technology records [46, 38, 29]. SOC 2018 provides a standard classification of U.S. occupations, while O*NET provides occupation-level records describing work content, activities, tasks, and technology use. SOC 2018 describes the structure of U.S. work across 23 major groups, 98 minor groups, 459 broad occupations, and 867 detailed occupations. To derive ALE's workflow-level taxonomy, we screen these occupation records for in-scope digital workflows, consolidate the retained occupational evidence into subdomains, and supplement SOC/O*NET where stable frontier workflows are absent or under-specified.

Screening procedure. The initial screen applies the scope definition to the 1,016 entries in O*NET 30.2 Occupation Data [29] using a fixed GPT-4o mini prompt at temperature 0. Each prompt instance includes the SOC code and title, O*NET description, task statements, work activities, and technology examples. After O*NET variants are consolidated under shared SOC base codes, 117 unique SOC base codes remain.

Subdomain construction. The screening stage identifies occupation-level evidence, which we then organize into workflow-level subdomains. We group SOC codes whose task statements, work activities, and tools-and-technology records describe a common class of artifact-producing professional work. Field, methodology, and work product are the primary dimensions used to set subdomain boundaries, with LLM-assisted research and domain-expert review used to check borderline assignments. A SOC code can be assigned to more than one subdomain when its work content contains separable workflows. This process yields 51 SOC-anchored subdomains.

Frontier extension. SOC/O*NET anchors the taxonomy in established occupations. ALE further adds a frontier supplement for emerging digital workflows not yet represented in SOC 2018 but already present in current research and professional practice, as reflected in recent NIH, NSF, and field-specific technical roadmaps [30, 31, 47, 48, 44]. The supplement adds four frontier subdomains and seven extensions to SOC-anchored subdomains, yielding 55 subdomains in total.

Result. The final taxonomy contains 55 workflow-level subdomains grouped into 13 domains.

B.1.2 Industry Landscape Review

Collection context. Subdomain workflow landscapes provide collection context for task workflows within each subdomain. Each landscape summarizes the work setting, practitioner roles, digital inputs, workflow dependencies, and output artifacts associated with a class of professional workflows. The records are based on field references, workflow documentation, LLM-assisted research, and expert review. They are used to organize candidate task families and to check that collected tasks specify realistic inputs, practice-appropriate deliverables, and verifiable success criteria. This section includes four illustrative subdomain workflow landscapes.

B.1.3 Manufacturing & Industrial Operations

Manufacturing & Industrial Operations covers workflows that turn CAD geometry, bills of materials, process specifications, and production targets into routings, toolpaths, line layouts, control logic, and inspection plans. A recurring handoff risk is consistency between the digital plan and the physical process: controller dialects, fixture geometry, alarm priorities, and tolerance stacks are tracked across CAM, controls, industrial engineering, and quality review.

Work starts from a part model, material requirements, and a production target. Process planners assign machine classes and routings; CAM programmers generate toolpaths, post G-code, and verify cutter motion in stock simulation; industrial engineers balance station times against takt; controls engineers write PLC logic, SCADA screens, alarm priorities, and safety interlocks; quality engineers author SPC plans, control limits, and first-article inspection procedures. The resulting artifacts include routing plans, posted code, simulation outputs, control files, SPC plans, and inspection packages. When a feature drifts out of specification, the inspection record can be compared with the process history and, when needed, route the workflow back to fixture, parameter, or toolpath revision.

B.1.4 Biomolecular Structure & Design

Biomolecular Structure & Design covers computational workflows that precede wet-lab execution. Typical inputs include a target hypothesis, candidate sequences or ligands, structural templates, assay constraints, and host-organism requirements. A recurring handoff risk is consistency between the model and the biological material: protonation states, retained waters, MSA depth, codon usage, and assembly-junction fidelity are tracked from computational design through experimental handoff.

Work starts from a protein to inhibit, a binder to design, or a pathway to express. Computational chemists dock candidate ligands and run molecular dynamics; structural biologists query structure predictors for relevant conformations; protein designers filter sequences through inverse-folding and stability models; bioinformaticians build MSAs and codon-optimize for the host; synthetic biologists draft plasmid maps and assembly junctions. The resulting artifacts include ranked ligand poses, structure models, sequence designs, codon-optimized constructs, plasmid maps, and design registry entries. If a binder fails to bind, a predicted pose is contradicted, or a pathway misses titer, the recorded assumptions may help identify the in-silico layer to revisit.

B.1.5 3D, Animation & Interactive Media

3D, Animation & Interactive Media covers workflows that convert scripts, storyboards, look briefs, and gameplay requirements into rendered frames or runtime states. Assets move through geometry, rigging, material, animation, lighting, compositing, and engine stages. A recurring handoff risk is that scale, coordinate frames, import settings, color spaces, and timing metadata are preserved as authored assets move across tools.

Work starts from a script, storyboard, and look brief. Concept artists establish the visual language; modelers build geometry within scale, topology, and polygon-budget constraints; look-development artists author materials and textures; riggers wire skeletons and deformers; animators key motion against the rig and camera; lighting technical directors stage scenes; FX artists simulate particles, fluids, and destruction; compositors layer rendered elements, while runtime engineers and technical artists integrate assets into gameplay logic, shaders, naming conventions, and pipeline layouts. The resulting artifacts include models, rigs, texture sets, material graphs, animation clips, lighting setups, compositing files, and engine scenes. When a shot or build returns with incorrect scale, color, or timing, the asset graph can help locate the layer that introduced the drift.

B.1.6 Robotics & Autonomous Systems

Robotics & Autonomous Systems covers workflows that translate task specifications and environment models into robot descriptions, perception calibration, controllers, planners, safety logic, and simulation assets. A recurring handoff risk is the sim-to-real gap: kinematic descriptions, sensor transforms, controller gains, planner parameters, and simulator assumptions are tracked from digital twin to hardware.

Work starts from a task specification, such as picking a part, reaching a target site, or navigating a constrained route. Mechanical engineers fix the URDF and joint limits; perception engineers

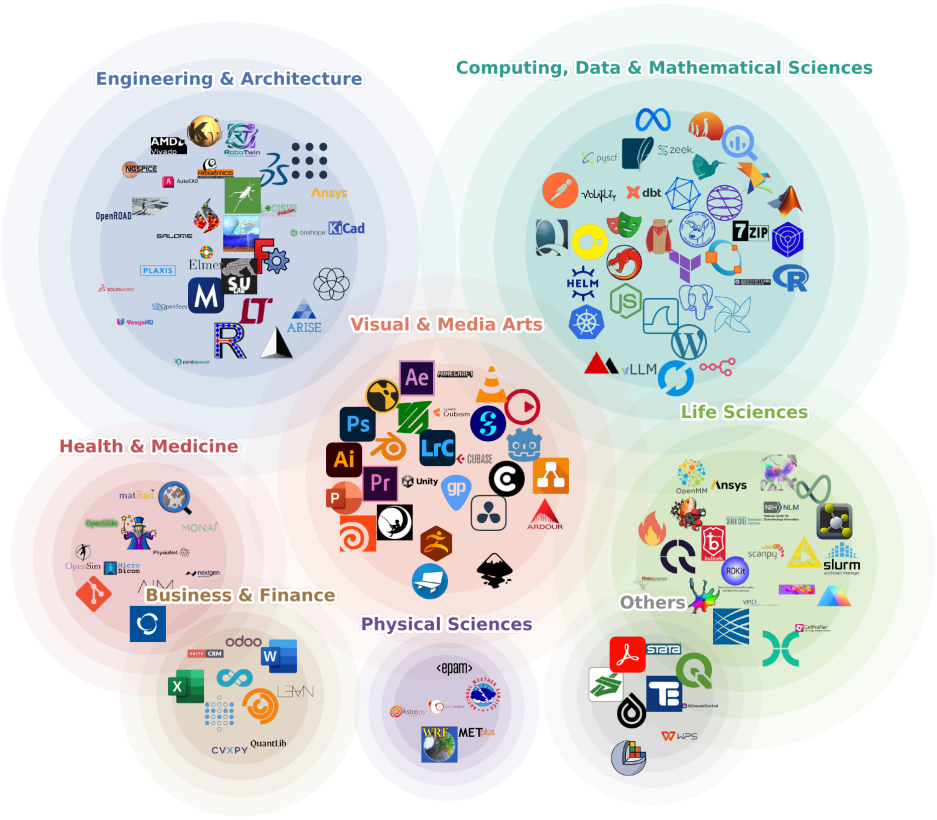


Figure 10: **Software ecosystem covered by ALE tasks.** Each icon is a distinct application or toolchain that appears in at least one task workflow, positioned within its primary ALE domain. Overlap regions hold tools that span multiple domains (e.g., creative-suite applications shared between Visual & Media Arts and Engineering). The figure is qualitative; quantitative per-subdomain instance counts appear in Figure 2.

calibrate sensors, validate transforms, and test perception against expected noise; controls engineers tune gains and design safety controllers; motion planners and trajectory optimizers compose plans under kinematic and dynamic constraints; behavior engineers connect state machines, behavior trees, or learned policies to failure modes; simulation engineers maintain the digital twin; safety engineers map the system to hazard analyses. The resulting artifacts include URDFs, calibration files, controller parameters, planner configurations, behavior specifications, simulation scenarios, and hardware-in-the-loop test records. When hardware behavior diverges from simulation, the recorded mismatch may inform updates to the simulator, controller, or policy before the system returns to the validation bench.

B.2 Task Construction Pipeline Details

This appendix expands on the staged construction protocol summarized in Section 2.3 and depicted in Figure 4. The protocol converts an expert's workflow submission into a benchmark instance through five gates: expert sourcing, task submission and editing, first-pass review, task implementation, and final QC and acceptance.

Expert sourcing. The pipeline begins with targeted expert outreach. To ensure coverage across the defined taxonomies, we established an advisory committee comprising leading industry professionals and expert practitioners. This committee anchors the recruitment of domain specialists who perform complex software workflows in their daily practice.

Task submission and editing. Tasks originate directly from these practitioners through a dedicated web submission portal (`https://agents-last-exam.org/submit/new/form`). The structural overhead for experts is kept deliberately low; they are asked to upload their own past projects that historically took them days or weeks to complete. AI-assisted tools on the portal help practitioners iteratively refine their proposals until the core components are fully specified: *a natural language description, the input files, the target software and tools, the expected output deliverable, and a clear evaluation specification*. For source-grounded tasks, public papers [40, 19], datasets, standards, and workflow documentation may also define input assets, label sets, or reference artifacts; the task bundle retains these source records as part of its material provenance.

First-pass review. Upon submission, tasks undergo a first review round that functions as a screening gate. Submissions receive feedback in the form of standard conference-style decisions: `major / minor revision`, `borderline accept`, `accept`, and `strong accept`. Any revision-based decision loops the proposal back to the expert for further editing.

Task implementation. Proposed tasks must then be translated from written specifications into executable benchmark environments. During implementation, an engineering team converts the expert's intent into runnable assets, provisions the necessary software containers, and codifies the evaluation logic. This process involves a rigorous engineer review and dry-run execution. If the engineer discovers gaps in the task logic or missing dependencies, the system triggers an automatic email notification, routing the task log back to the expert to unblock development.

Final QC and acceptance. Before a fully implemented task is admitted to the benchmark, it passes a final quality control gate overseen by the expert committee. This peer-review step evaluates both reproducibility and evaluation integrity. Specifically, reviewers check whether the expert's reference output is fundamentally correct, whether the evaluation bounds are properly calibrated (e.g., neither impossibly narrow nor spuriously permissive), and whether the problem context provides sufficient information to reach the final state. Issues found at this stage send the task back for final adjustments prior to acceptance.

B.3 Industry Task Cards and Metadata

B.3.1 Representative Task Cards

We present representative task cards to illustrate how ALE task instances are specified, executed, and scored. Each card summarizes the agent-facing request, input materials, expected deliverables, evaluation rubric, observed score, and observed outcome for one executed task instance. All trajectories shown here were produced by the Claude Code harness running Claude Opus 4.7. These cards are intended to make the benchmark construction and evaluation protocol concrete at the instance level, complementing the aggregate coverage and performance results reported in the main text.

Task Card 1: Injection Mold-Flow Analysis

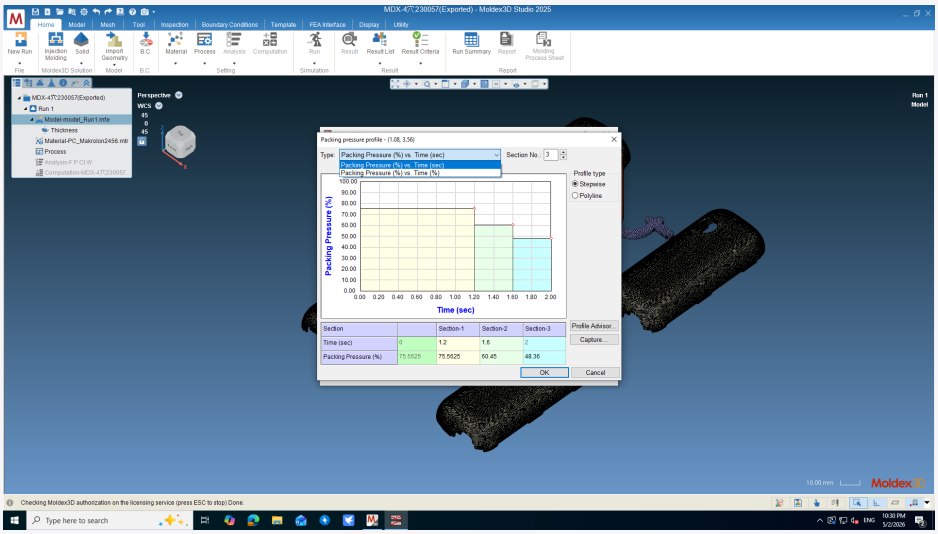
PARTIAL

Task ID: `manufacturing/mold-flow`
Observed score: 0.476 Manufacturing Moldex3D CAE

TASK DESCRIPTION

Injection mold-flow simulation supports tooling and process decisions. The task provides a meshed Moldex3D project, a process specification, and a results template. The agent must open project 230057, apply process_spec.json, run fill, pack, cooling, and warp analysis, and write output/results.json with solver-derived pressure, force, time, volume, and weight values.

SOFTWARE / EXECUTION VIEW



Moldex3D CAE workflow: the agent is editing the packing-pressure curve for a four-cavity injection-mold simulation. This is the process-setup stage; scoring depends on completing the solver run and extracting the pressure, force, cycle-time, volume, and weight metrics into results.json.

INPUT MATERIALS

- Staged Moldex3D project with mesh, material, and four-cavity geometry.
- process_spec.json specifying fill, pack, cool, and warp process settings.
- results_template.json defining every metric required in the final JSON.

REFERENCE OUTPUT

- Saved Moldex3D project after the configured simulation run.
- output/results.json populated with injection pressure, V/P switch pressure, clamp force, cooling/cycle times, volume, and weight fields.

EVALUATION RUBRIC

- Hard gates reject missing, malformed, or all-null results.json.
- The main score compares each numeric field with a hidden solver reference within 1 percent relative tolerance.
- Generated project artifacts can provide limited partial credit only when the main metrics are weak.

OBSERVED OUTCOME

- Score: 0.4762. The agent reached the right Moldex3D setup workflow and submitted results.json, but the score indicates only partial numeric/result credit.
- Failure mode: the submitted metrics were not reliably extracted from completed solver outputs; some values were estimated rather than measured CAE results.
- Interpretation: successful GUI setup was insufficient without closed-loop numerical verification: wait for result files, inspect or export solver data, then populate the JSON.

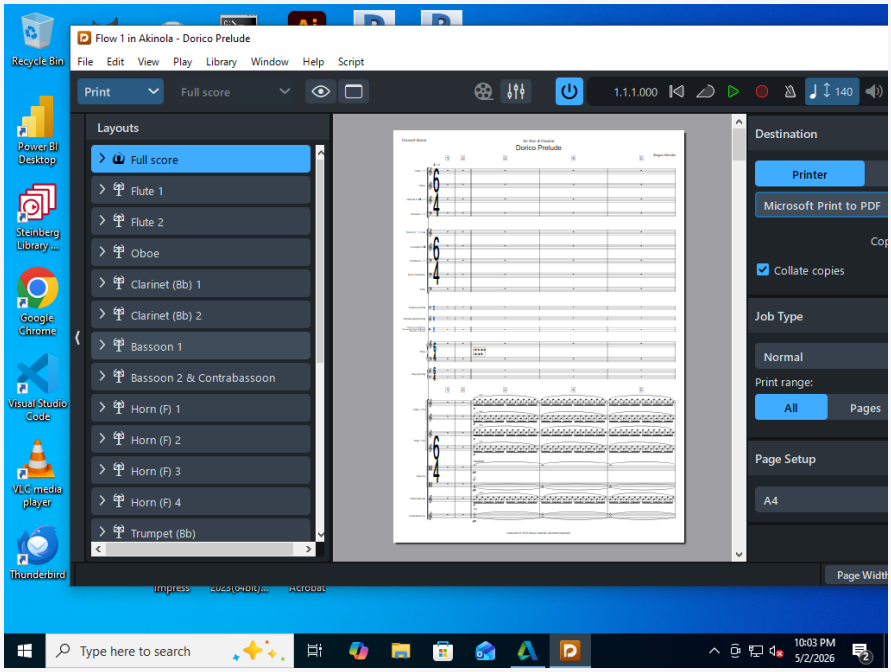
Task Card 2: Orchestral Music Transcription FAIL

Task ID: `audio/music_transcription`
Observed score: 0.000 Audio production Notation Dorico

TASK DESCRIPTION

Orchestral transcription converts an audio recording into a printable score and multitrack MIDI. The task provides an audio brief, a Dorico environment, a named piece, tempo, instrumentation contract, and exact output-file requirements. The agent must follow the Dorico Prelude / Akinola / tempo 140 / 27-instrument specification and deliver transcription.pdf, transcription.mid, and overview.png.

SOFTWARE / EXECUTION VIEW



Music-engraving workflow: Dorico is open on an orchestral score in print/export mode. The task required converting the audio brief into a readable full score and MIDI, then exporting the PDF, MIDI, and overview screenshot to the exact output paths.

INPUT MATERIALS

- Audio reference and task_brief.json with title, composer, tempo, instruments, and GM programs.
- Dorico environment for score engraving and export.
- Output-path requirements for PDF, MIDI, and overview screenshot.

REFERENCE OUTPUT

- transcription.pdf showing a complete, readable full score.
- transcription.mid with multitrack timing, pitches, dynamics, and GM Program Change assignments.
- overview.png showing a notation UI with the produced score.

EVALUATION RUBRIC

- Missing PDF, missing MIDI, or missing/invalid notation screenshot triggers a hard zero.
- After the gates pass, pitch and rhythm each contribute 30 percent, dynamics 20 percent, instrument assignment 10 percent, and score layout 10 percent.
- The MIDI is compared to a reference MIDI after track pairing and quantization.

OBSERVED OUTCOME

- Score: 0.0. The agent found a matching Dorico project and exported transcription.mid, but the required transcription.pdf and overview.png were not confirmed.
- Failure mode: the submission contained part of the required package, but the scoring hard gates require a complete score/MIDI/screenshot bundle before musical accuracy is evaluated.
- Interpretation: partial tool use did not translate into reliable multi-artifact delivery for a creative-production task.

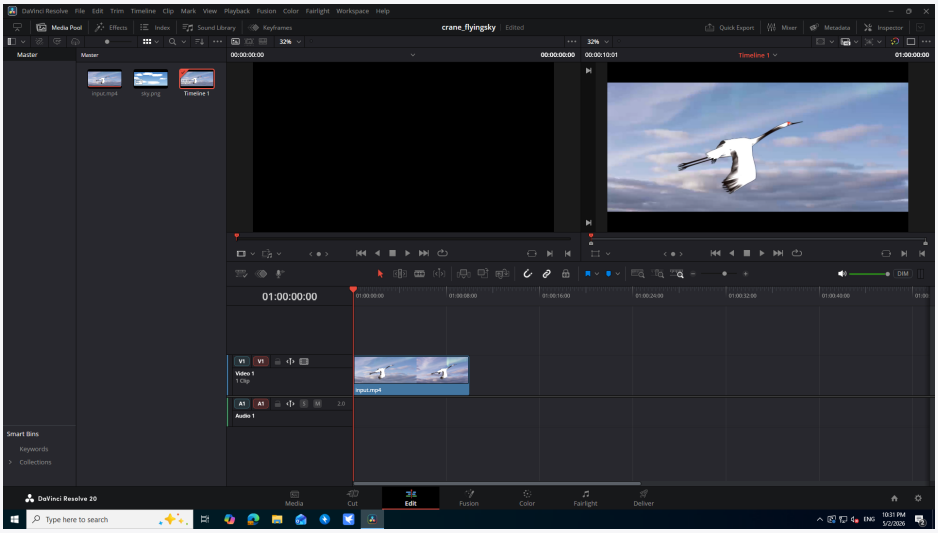
Task Card 3: Reference-Based Chroma Key Compositing FAIL

Task ID: `media/chroma_key_from_reference`
Observed score: 0.000 VFX DaVinci Resolve Compositing

TASK DESCRIPTION

Chroma keying is a VFX finishing task: remove a green-screen background, preserve foreground edges, place the subject over the intended plate, and match a reference frame. The task provides a bird foreground clip, a reference image, and a DaVinci Resolve environment. The agent must create a Resolve project, key the bird from input.mp4, match the composition implied by input.png, and export output/output.mp4.

SOFTWARE / EXECUTION VIEW



VFX compositing workflow: DaVinci Resolve shows a timeline and preview monitor with the bird footage. The task required identifying the green-screen foreground, keying it, compositing it over the intended sky plate, and matching the reference frame.

INPUT MATERIALS

- input.mp4 source clip for the keying workflow.
- input.png reference frame defining target composition and foreground placement.
- DaVinci Resolve desktop environment and required output path.

REFERENCE OUTPUT

- output/output.mp4 containing the composited bird shot.
- Frames with foreground preserved, green screen removed, and contour/placement close to the reference.

EVALUATION RUBRIC

- Remote scoring samples frames from reference/breakpoint.json.
- Foreground preservation is gated by roi_input_cv, and quality uses full-frame plus ROI edge IoU.
- A local vision judge must also confirm that a real keying-related edit was made rather than a raw-source copy.

OBSERVED OUTCOME

- Score: 0.0. The agent used Resolve and exported an MP4, but the output did not satisfy the reference-based chroma-key transformation.
- Failure mode: the submission treated a plausible bird-over-sky clip as the target instead of preserving the intended foreground/background relationship from the reference materials.
- Interpretation: the error was in visual task grounding rather than software operation alone.

Task Card 4: Skeletal Animation Reproduction PARTIAL

Task ID: `game/skeletal_animation_reproduction`
Observed score: 0.429 Game / 3D Blender Motion matching

TASK DESCRIPTION

Animation production turns a static character asset into a rigged, replayable performance. The task provides a Blender environment, Singing.obj, and a reference motion video. The agent must rig the character, reproduce the body motion from reference.mp4, include 19 exact bone names, and submit both an editable final.blend and a replayable preview.mp4.

SOFTWARE / EXECUTION VIEW



3D animation output view: this render frame shows the rigged singer character from the agent's Blender submission. The task was to reproduce the body motion from a reference video, so a single valid-looking frame is only partial evidence; timing, pose range, and replay consistency determine success.

INPUT MATERIALS

- Singing.obj static character mesh.
- reference.mp4 body-motion target and reference/package.json bone-name requirements.
- Blender runtime for rigging, keyframing, rendering, and replay validation.

REFERENCE OUTPUT

- final.blend containing a usable animated rig with the required bone names.
- preview.mp4 whose body motion, timing, and pose range match the hidden clean reference.

EVALUATION RUBRIC

- Validity gates require both artifacts and a non-trivial animated rig.
- Scoring combines video match, replay consistency from final.blend, minimal skeleton coverage, and binary vision checks.
- Short or damped motion can lose credit even when the required files are present.

OBSERVED OUTCOME

- Score: 0.4285. The required Blender and preview artifacts existed and contained animation, so the run earned partial credit.
- Failure mode: preview.mp4 was about 4 seconds while the reference was about 13.6 seconds, and the motion was damped to reduce mesh distortion.
- Interpretation: the submission satisfied file-level validity but not the full motion-fidelity requirement.

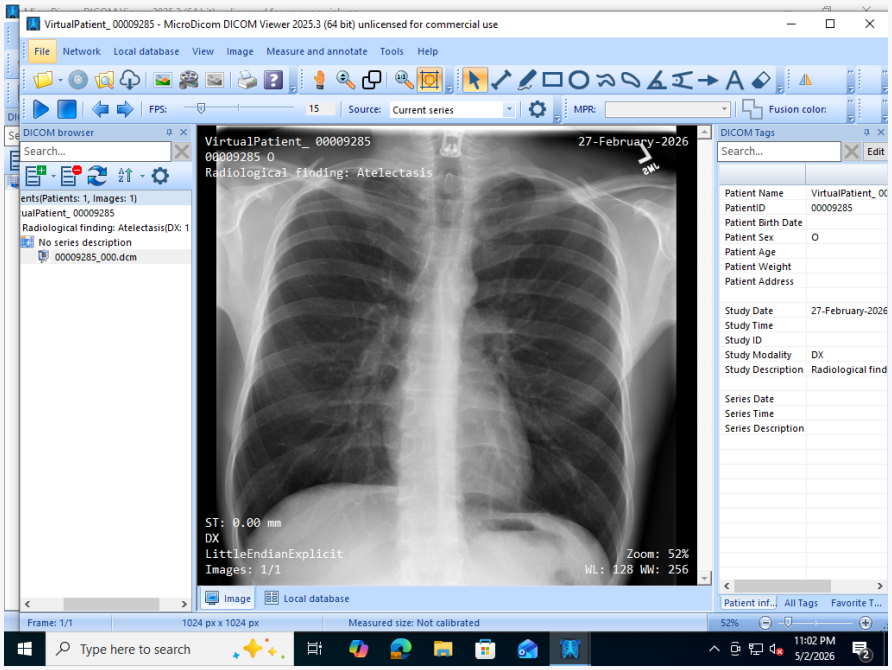
Task Card 5: MicroDicom Chest X-Ray Adjudication PARTIAL

Task ID: `radiology/microdicom_nih_cxr_reader_adjudication`
Observed score: 0.333 Radiology DICOM GUI Annotation QA

TASK DESCRIPTION

Radiology dataset curation often requires adjudication between competing reader annotations. The task provides nine NIH CXR DICOM cases, two proposed atelectasis boxes per case, reader notes, and fixed TSV schemas. The agent must inspect each case in MicroDicom, choose the better reader annotation, and submit adjudicated_boxes.tsv, adjudication_log.tsv, and final_impressions.tsv.

SOFTWARE / EXECUTION VIEW



Radiology adjudication workflow: MicroDicom displays a chest X-ray with DICOM metadata and annotation tools. The task required reviewing each case, comparing two reader boxes for atelectasis, and writing TSV decisions; the deliverable depends on visual adjudication for each case.

INPUT MATERIALS

- Nine DICOM chest X-ray cases with a case manifest.
- reader_a and reader_b annotation TSVs plus clinical notes.
- Rules fixing labels, allowed selected_reader values, and output schemas.

REFERENCE OUTPUT

- adjudicated_boxes.tsv with the final reader choice and box per case.
- adjudication_log.tsv and final_impressions.tsv with the required rows and fixed labels.

EVALUATION RUBRIC

- Each TSV has hard gates for missing files, malformed headers, invalid reader values, or wrong fixed labels.
- Log and impression files must match hidden references exactly by case_id.
- Box output must select the hidden-supported reader and achieve IoU at least 0.50 against the gold box.

OBSERVED OUTCOME

- Score: 0.3333. The agent wrote the three required TSV files covering all nine cases, but the score indicates only partial agreement with the hidden references.
- Failure mode: detailed image review appears only for a few cases; later decisions relied on coordinates and heuristics rather than systematic visual adjudication.
- Interpretation: file-format compliance was stronger than case-by-case visual-domain judgment.

B.3.2 Executed Task Inventory

The full inventory of the 150 selected public tasks, with task name, task identifier, domain, and a brief description for each, is browsable online under the *Benchmark Splits (ALE-V1, 2026/06)* tab at `https://agenthle.org/demo`.

C Evaluation Pipeline Details

C.1 Pipeline Architecture Details

This appendix expands on the three decoupled components summarized in Section 3.1 and depicted in Figure 6: the task specification, the agent, and the environment.

Task Specification. The task specification is the executable form of an expert submission. It encapsulates five elements provided during the construction pipeline (Section 2.3): a natural-language *description*, *input assets*, the required *software*, *reference assets* (ground-truth outputs), and *evaluation criteria*. These are encoded in a single `main.py` file that exposes three lifecycle functions: `load()` declares the task description, metadata, and compute requirements; `start()` provisions the virtual machine into a deterministic starting state by copying input assets and launching the required software; and `evaluate()` scores the agent's output artifacts against references or rubrics, returning a normalized score in $[0, 1]$. The full protocol is detailed in Appendix C.2.

Agent. The agent is the system under evaluation, composed of a *harness* (orchestration middleware) and a *model* (the foundation model). Upon receiving the task configuration, which consists of a description and associated metadata, the agent enters an action loop: it observes the environment (via screenshots, shell output, or file contents), selects an action (mouse clicks, keystrokes, shell commands, file edits, or API calls), executes it, and repeats until it decides to terminate.

Environment. Each task executes inside a remote virtual machine that hosts the required industrial software and exposes a standardized filesystem layout. Four directories partition the workspace: `input/` contains assets the agent reads (e.g., design files, game binaries, raw data); `software/` holds pre-installed applications and their dependencies; `output/` is the sole writable target where the agent deposits its deliverables; and `reference/` stores ground-truth artifacts used exclusively by the scoring function (the agent has no access to this directory during execution). This layout enforces a clean contract: the agent reads from `input/`, writes to `output/`, and is scored by comparison against `reference/`.

Compute environment. All task instances are executed on Google Cloud Platform (GCP) virtual machines. The default configuration is `c4-standard-4` (4 vCPUs, 16 GB RAM). Tasks that require GPU acceleration (e.g., 3D rendering, simulation) use `g2-standard-8` instances equipped with an NVIDIA L4 GPU. A small number of tasks involving heavy numerical simulation are provisioned with higher-memory or multi-core configurations as dictated by the task's compute requirements declared in `load()`. These resource assignments are determined per-task based on the software and workload involved.

Decoupled design guarantees. The decoupled design yields two practical guarantees. First, any agent, regardless of its internal architecture, model backbone, or tool configuration, can be evaluated on any task provided it conforms to the action interface (shell commands, GUI interactions, and file I/O). Second, the same task specification can be deployed across different environment backends (cloud VMs or local containers) without modification.

C.2 Task Specification Protocol

Every task specification implements three lifecycle phases that together ensure deterministic, reproducible evaluation.

Phase 1: `load()` (Initialization). The `load()` function is purely declarative: it returns a structured task object containing the natural-language description visible to the agent, metadata (filesystem paths, configuration parameters, task-specific constants), and compute requirements (operating system type, hardware specifications). No remote connection is established and no environment state is modified. This phase defines *what* the task is.

Phase 2: `start()` **(Environment Preparation).** The `start()` function transforms the virtual machine into the task's deterministic starting state. Operating through a *session API* that provides programmatic access to the remote desktop, including file-system operations (create, copy, delete), application management (launch, install), keyboard and mouse control, and screen capture, etc.

Phase 3: `evaluate()` **(Scoring).** After the agent terminates, the `evaluate()` function retrieves the agent's output artifacts from the remote environment and scores them against the reference assets or rubric criteria defined in the specification. The function returns a normalized score in $[0, 1]$. The concrete evaluation methodology (deliverable-based extraction versus milestone-based reporting, rubric comparison versus reference matching, and the anti-gaming provenance gates that guard against shortcut solutions) is summarized in Section 3.3 and detailed in Appendix C.3.

C.3 Evaluation Modes: Full Taxonomy and Worked Examples

This appendix expands Section 3.3. We document (i) where the scoring code runs, (ii) the artifact modes that a task author can choose for the comparison, (iii) the score-composition patterns observed across the benchmark, and (iv) the helper layer that backs LLM-as-judge evaluations. Concrete task implementations are cited throughout so that readers can inspect the full source.

C.3.1 Execution Locale

Host-side scoring (default). The harness pulls the agent's artifact off the VM with `session.read_file` or `session.read_bytes`, then runs the scoring code in the host Python process. This is the default whenever (a) the artifact is small enough to transfer and (b) the scoring tooling is available off-VM. Examples:

- `finance/equity_research_summary` reads the produced LibreOffice workbook bytes and calls `score_workbook_bytes` against a manifest.
- `cybersecurity/snake_crackme` reads a flag file, normalizes the text, and compares the SHA-256 of every candidate against the expected digest.
- `photography/raw_photo_processing` compares exported file pairs against a reference manifest.

Host-side scoring is preferred because the scoring code is more easily reviewed, version-controlled, and re-run offline against a saved trajectory.

VM-side verifier. When the artifact requires software that cannot be reasonably moved off the VM (CAD/CAM kernels, headless 3D renderers, vendor-licensed engines, very large geometry), `evaluate()` uploads a per-task script from `tasks/<task>/scripts/` into a temporary directory on the VM and invokes it with `session.run_command`; the script prints a JSON result on stdout, which the host parses into a score. Examples:

- `manufacturing/gcode` uploads `check_collision.py`, `simulate_agent.py`, and `verify_stl.py` to drive PowerMill's COM API for collision detection and stock simulation, then to score the resulting STL surface against a hidden reference STL.
- `finance/sec_10k_financial_parsing` uploads `score_outputs.py` which uses the VM's Python environment to score the parsed filings against a multi-file reference manifest.

The contract is uniform: VM-side verifiers communicate with the host strictly through their stdout JSON, never by writing into `output/`.

C.3.2 Artifact Modes

Each task workflow author selects one or more of the following modes for the comparison step. Table 3 summarizes the modes and representative task workflows.

Exact / hashed values. The deliverable is a short string (a flag, a parsed answer, an identifier). Scoring is byte-equal or hash-equal after normalization. Because the answer is small but the search space is large, this mode is dominant in cybersecurity and a subset of mathematics task workflows.

Structured tabular / numeric. The deliverable is a table or a multi-field record (cells of a workbook, line items of a financial filing, parameters of a calibration). The reference is a manifest of (field,

Mode	Reference form	Locale	Judge	Example task workflow
Exact / hashed value	secret / digest	host	code	`cybersecurity/snake_crackme`
Structured tabular	manifest of (field, value, tolerance)	host or VM	code	`finance/sec_10k_financial_parsing`
Geometric / spatial	STL / mesh / point cloud	VM	code	`manufacturing/gcode`
Visual appearance	reference screenshots	host	vision LLM	`game/mota_reproduction`
Behavioral / world state	deterministic state dump	VM	code	`architecture/parametric_energy_simulation`
Free-text / semantic	rubric	host	LLM	`finance/equity_research_summary` (rubric component)
Executable artifact	test set / oracle program	host or VM	code	`data_computer_science/data_pipeline_etl_instance_1`

Table 3: Evaluation modes available to task workflow authors. Most ALE task workflows combine two or more modes (e.g., a behavioral gate with a geometric score).

value, tolerance) tuples; per-field credit is granted within tolerance and aggregated. This mode is dominant in finance, accounting, and clinical-data-standards task workflows.

Geometric / spatial. The deliverable is a 3D mesh, a point cloud, or any spatially-embedded artifact. Scoring uses surface-distance functions (e.g., 10,000 surface samples scored against a reference STL in `gcode`, with credit for each fraction-within-threshold band).

Visual appearance. The deliverable's correctness is most naturally judged by human eye (a rendered scene, a recolored photo, a UI screenshot). The host calls a vision LLM judge with the agent and reference images side by side and a rubric question.

Behavioral / world state. The deliverable is the state of an interactive system after the agent's edits. Scoring replays the system under a fixed trajectory and dumps a comparable state (game map, simulation log, NPC events).

Free-text / semantic. The deliverable is a written report. Scoring is a rubric of yes/no or graded sub-criteria, evaluated by an LLM judge.

Executable artifact. The deliverable is a program, a model, or a pipeline. Scoring runs the artifact against a held-out test set or oracle and aggregates per-instance correctness.

C.3.3 Score-Composition Patterns

Per-mode scores are combined into the final $[0, 1]$ value by one of four patterns:

Gate-and-score. A hard precondition that forces 0 on failure, followed by a continuous score. Used to prevent reward hacking by superficially close artifacts. Canonical example: in `manufacturing/gcode`, a PowerMill collision/gouge gate must pass before any geometric similarity is awarded; otherwise the task workflow scores 0 regardless of how close the simulated stock model is to the reference.

Weighted rubric. An expert-defined weighted sum over multiple sub-metrics, e.g. `gcode`'s

$$\text{score} = 0.70 \cdot \text{frac_within}(0.3\,\text{mm}) + 0.30 \cdot \text{frac_within}(2.0\,\text{mm})$$

on the surface-distance distribution between agent and reference STLs. Weights are part of the task spec and are reviewed during the QC stage of Section 2.3.

Binary checklist averaging. The deliverable is judged by N independent yes/no questions, and the score is the mean. The `game/mota_reproduction` task workflow uses this pattern over a set of vision-LLM probes (engine identification, character sprite presence, map layout match).

Pairwise file aggregation. When the deliverable is a directory of files matched against a reference directory, the helper `utils.evaluation.collect_matching_files` pairs each agent file with its reference, runs the same scoring function on every pair, and returns the mean.

C.3.4 Judge Types and the LLM-Judge Helper Layer

Avoid LLM judges by default. ALE deliberately suppresses the use of LLM-as-judge wherever a deterministic alternative exists. The standing rule for every accepted task workflow is: if the deliverable can be reduced to bytes, fields, geometry, world state, or executable behavior, the scoring code must operate on those signals, not on a model's holistic opinion of the result. A task that proposes "ask GPT-4 whether the result looks correct" is rejected at the QC stage of Section 2.3 and either re-engineered to expose a checkable artifact or dropped. This is enforced for three reasons: (i) judge models drift across releases, which would silently re-rank agents; (ii) general-purpose "does this look right?" prompts are too soft to discriminate near-correct from correct, which is exactly the regime where the benchmark needs to resolve agents; and (iii) deterministic judges can be re-run offline against a saved trajectory by anyone who has the artifact, with no API cost.

When LLM judging is unavoidable. A small set of task workflows have no objective code-based reference, primarily creative or perceptual deliverables such as rendered scenes, music-production sessions, UV-mapped textures, and animation previews. For these, ALE uses the helper layer in `utils/evaluation.py`:

- `llm_vision_yes_no_judge`: a single targeted yes/no visual question over an (agent image, reference image) pair.
- `llm_vision_binary_questions_sync` / `llm_vision_binary_checklist_judge`: a list of independent yes/no probes whose final score is a fraction.
- `llm_vision_judge` / `llm_vision_json_judge`: a graded or JSON-structured rubric over a small fixed set of fields.

Targeted probes, not general judging. The single most important property of ALE's LLM-judged workflows is that the prompts *never* ask the model to score the artifact in the abstract. Every prompt is a narrow, evidence-anchored yes/no probe, written by the task author with reference to (a) the specific software the workflow uses and (b) failure modes the author has observed in pilot agent runs. The model is asked one structurally tight question at a time; the score is composed by code from those answers. The following are verbatim probes from current task workflows:

- `game/mota_reproduction` (replay-time engine and state checks):
  - "Does the first image show that the game is developed using RPGMakerXP? One can identify whether there is an 'orange sun-like circle' in the top-left corner of the game window." (gate)
  - "Does the first image show with the same map layout as in the original game?"
  - "Does the first image show with the same player status as in the original game?"
- `audio/timbre_synthesis` (proves the agent actually used the DAW):
  - "Does this image show either (a) a software synthesizer / VST plugin interface with visible parameters such as oscillators, filters, envelopes, LFOs, or effects, OR (b) a DAW (e.g., Cubase, Ableton, FL Studio, Logic, Reaper) arrangement / piano-roll / mixer view that contains one or more tracks with audio or MIDI clips?" (gate)
  - "Does this screenshot provide evidence that the user has actually worked on the project, for example a DAW arrangement containing multiple tracks with audio or MIDI clips, a piano roll with notes entered, or a synthesizer plugin whose knobs/sliders are clearly not all in their default/init positions?"
- `game/uv_reproduction` (UV-mapping artifact checks against a fixed reference):
  - "Is the texture placement and orientation correct enough to pass?"
  - "Does the candidate preserve the reference material appearance and color palette well enough to pass?"
  - "Are obvious UV seams, stretching, or texture artifacts absent enough for the result to pass?"
- `game/high_to_low_modeling` (decimation correctness):
  - "Does the candidate preserve the overall highpoly shape well enough to pass?"
  - "Does the candidate preserve the main silhouettes across the sampled views well enough to pass?"
  - "Does the candidate achieve a meaningful lowpoly reduction rather than effectively submitting the highpoly again?"
  - "Are obvious visual artifacts or shape collapses absent enough for the result to pass?"
- `game/object_generation` (missing-geometry restoration):
  - "Is the missing geometry restored well enough to pass?"
  - "Is the part placement and alignment correct enough to pass?"

  - "Is the whole object coherent and complete enough to pass?"
  - "Is the final material appearance acceptable enough to pass?"
- `game/skeletal_animation_reproduction` (replay self-consistency):
  - "Does the submitted preview match the reference body motion well enough to pass?"
  - "Does the replay rendered from `final.blend` agree with the submitted preview well enough to pass?"
  - "Do the visible skeleton states and poses look natural and non-broken enough to pass?"

Three patterns recur across these probes. (1) Each question targets one identifiable artifact (a circle in a corner, a track in a DAW, a UV seam, a silhouette, a pose), not the gestalt of "is the deliverable good." (2) Many probes are written as gates, i.e., a binary precondition that must hold before the rest of the rubric is even checked, so that an unrelated screenshot or a placeholder file scores 0 before any quality judgment is asked. (3) The remaining probes are phrased as "...enough to pass?" rather than as graded preferences, which converts the model into a same-vs.-different comparator against a fixed reference rather than a free-form quality oracle. Together these conventions limit the burden on the LLM to a series of decisions that a domain expert could replicate from the same image, and keep the LLM out of the role of integrator: the integration (weighting, gating, and aggregation across instances) always happens in code.

C.3.5 Empirical Distribution of Judge Types

To make the design choice "code-based by default, LLM only when unavoidable" concrete, we report the actual distribution of judge types and execution locales at the task-workflow level. The proportions in Table 4 are obtained by static analysis of every `main.py` in the open-sourced reference task tree together with its accompanying `scripts/` directory, scanning for direct invocations of the LLM-judge helpers in `utils/evaluation.py` and for the use of `session.run_command` on uploaded Python verifiers.

Judge type	Share
Code-based (deterministic)	93.2%
LLM-as-judge	6.8%

(a) Judge type per task workflow.

Execution locale	Share
Host-side	88.5%
VM-side verifier	11.5%

(b) Execution locale of the scoring code.

Table 4: Distribution of judge type and execution locale across the open-sourced task workflows in the ALE reference task tree.

Within the LLM-judged subset. Vision-grounded primitives dominate: the most-used helpers are `llm_vision_judge`, `llm_vision_binary_checklist_judge`, `llm_vision_binary_questions_sync`, and `llm_vision_yes_no_judge`. The remaining helpers (`llm_multimodal_binary_questions_sync`, `llm_multimodal_text`, `llm_multimodal_json`, `llm_vision_json_judge`, and the video-rubric `gemini_video_json_judge`) appear in only a handful of task workflows each. Helpers may co-occur within a single task workflow. The concentration on vision helpers reflects the fact that LLM judging is reserved for cases where the deliverable is a rendered scene, photo, screenshot, or short video that has no objective code-based reference.

Within the VM-side subset. VM-side task workflows are dominated by industries whose deliverable cannot be scored without the on-VM software stack: CAD/CAM (PowerMill, SolidWorks), licensed financial workbooks, and headless 3D rendering. The contract is uniform: a per-task-workflow verifier under `tasks/<task-workflow>/scripts/` is uploaded by `evaluate()` into a temporary directory on the VM, executed via `session.run_command`, and its JSON stdout is parsed back into a score on the host.

Composition patterns. A static-analysis estimate finds that the great majority of `evaluate()` bodies contain at least two early `return [0.0]` sites preceding a continuous-score path, consistent with the gate-and-score pattern of Section C.3.3. Explicit weighted-rubric expressions of the form $a \cdot x + b \cdot y$ appear in only a small fraction of `main.py` files; in most task workflows the weighting is encoded inside per-task-workflow scoring scripts under `scripts/` (which the static count above does not pick up), so this is a lower bound on the true prevalence of weighted aggregation.

C.3.6 Reference Isolation and Robustness

Reference isolation. The `reference/` directory lives outside the agent's workspace and is not exposed through any session API a non-evaluator caller would invoke. Most `evaluate()` implementations begin with a sanity check that lists every required reference path and returns 0.0 early if any path is missing, which both prevents misconfigured runs from silently producing inflated scores and documents the reference contract in the same file as the scoring logic.

Output presence and shape checks. Workflows consistently treat "no output produced" or "output in the wrong shape" as score 0 rather than as a crash, so that an agent that times out or refuses still emits a well-defined number.

Determinism. Code-based judges are deterministic by construction. For LLM-judged task workflows, we record the judge model and prompt with the result and make sure that any score can be re-derived from the agent's saved artifacts.

C.3.7 Workflows and Task Instances

A single task workflow (one `main.py`) exposes a list of *task instances*, encoded in the codebase as the `VARIANTS` tuple list: each instance carries instance-specific configuration but shares the same `evaluate()`. For example, the `manufacturing/gcode` task workflow declares 18 workpiece instances, each pointing at a different blank PowerMill project but scored by the same collision-gate-then-STL pipeline. Per-instance scores are averaged into a task workflow score, task workflow scores are averaged into industry scores, and industry scores aggregate into the cluster-level results reported in Section 4. The current ALE release contains 960 task workflows and 1,490 task instances in total.

C.4 Agent Harness Internals

This appendix elaborates on the internal structure of the agent harness introduced in Section 3.2. The architecture described here is shared, at a macro level, across mainstream harness implementations such as Claude Code [4], Codex [33], and OpenClaw, and is faithfully reproduced in our own native implementation.

Main agent loop. The harness operates a six-phase control loop: (0) *Initialization* configures the system prompt and tool bindings; (1) *Context Building* assembles the current conversation state; (2) *LLM Call* queries the foundation model; (3) *Decide* routes the model's output to either a final delivery or a tool invocation; (4) *Collect Tool Result* gathers the execution outcome; (5) *Overflow Check* evaluates whether the accumulated context exceeds a compaction threshold. If not, the loop returns to phase 1; otherwise, context compaction is triggered before the next iteration. The loop terminates when the model elects to deliver rather than act.

System prompt builder. At initialization, the harness constructs a system prompt from modular components: *Identity* (agent persona), *Memory* (persistent cross-session state), *Tool Guidance* (usage conventions for each tool), *Runtime* (environment metadata), *Behavioral Rules* (safety and policy constraints), and *Skills* (domain-specific capabilities). These components are typically authored through configuration files such as `CLAUDE.md` or `AGENTS.md`.

Tool system. The harness exposes a unified tool interface that the model invokes by name: file operations (read, write, glob, grep), shell execution, web search and fetch, and sub-agent management (spawn, list, wait, terminate). Each tool returns structured results that are appended to the conversation context.

GUI-as-Tool: CUA MCP bridge. The GUI-as-Tool mode extends the tool system with 14 desktop-action tools exposed through an MCP server that wraps a CUA (Computer-Use Agent) HTTP API running on the VM. Table 5 lists the full tool surface.

C.4.1 Tool Surface and Terminology

Tool taxonomy and per-agent availability. Tool names differ across harnesses, so the analysis in Figure 9 maps raw tool calls into a common taxonomy before aggregation. *Bash* denotes direct shell or terminal execution, including tools named `Bash`, `bash`, `shell`, `exec`, `run_shell_command`, `terminal`, `Execute`, `bash_command`, and `execute_code`. *File* denotes direct file-system tools

Table 5: **GUI-as-Tool: 14 desktop-action tools** exposed via the CUA MCP bridge.

Group	Tool	Description
Keyboard	`key`	Press and release one or more keys (supports hotkeys, e.g. `["ctrl","c"]`)
	`key_down`	Press keys down without releasing (for modifier holds)
	`key_up`	Release previously held keys
	`type`	Type text into the currently focused input field
	`hold_key`	Hold keys for a specified duration, then release
Mouse	`mouse_move`	Move the cursor to a coordinate
	`click`	Click at a coordinate (left/right/middle; single/double/triple)
	`drag`	Drag from a start coordinate to an end coordinate
	`mouse_down`	Press a mouse button without releasing
	`mouse_up`	Release a mouse button
	`scroll`	Scroll in a direction (up/down/left/right) by a specified amount
Utility	`screenshot`	Capture the current screen; optionally save to a VM path
	`cursor_position`	Return the current cursor coordinates
	`wait`	Pause execution for a specified duration

such as `Read`, `Write`, `Edit`, `Glob`, `Grep`, `read_file`, `write_file`, `edit_file`, `patch`, and `list_directory`. *GUI* denotes the CUA desktop-action surface in Table 5, including wrapper-specific names such as `mcp__cua__click`, `cua___screenshot`, and `mcp_cua_key`. *Web* denotes browser or retrieval tools such as `WebSearch`, `WebFetch`, `web_search`, `web_fetch`, `webSearch`, `webFetch`, and `browser_navigate`. *Planning/delegation* denotes explicit planning, task-tracking, memory, or sub-agent tools such as `TodoWrite`, `task`, `think`, `delegate`, `subagents`, and `memory_get`. *Other* captures process, session, finish, or harness-internal utilities that do not fit the preceding groups. Finally, *Azure desktop* refers to the hosted Windows remote-desktop backend used for Windows GUI task execution; it is an execution substrate, not a model provider and not a separate tool class.

Sub-agents. Complex tasks benefit from delegation. The harness can spawn specialized sub-agents (a *General* sub-agent with access to all tools, an *Explore* sub-agent restricted to read-only operations, among others) that operate in isolated context windows and return summarized results to the parent loop. This mechanism enables parallel exploration and limits context consumption.

Context manager. Long-horizon professional tasks routinely generate context that exceeds model limits. The context manager implements a three-tier compaction strategy: (1) *Microcompaction* clears stale tool results in place; (2) *LLM-based summarization* compresses older conversation segments into structured checkpoints; (3) *Truncation* enforces hard context-window limits (e.g., 400K or 1M tokens). This graduated approach preserves recent detail while retaining long-range planning state.

ALE-Claw vs. OpenClaw. OpenClaw is a personal AI assistant with two main components: a user-interaction runtime and an agent loop. For ALE-Claw, we removed the components that keep a long-lived, multi-user AI assistant alive in production: the scheduled-prompt subsystem, including cron and heartbeats; multi-channel gateways; the skills system; and the plugin framework with lifecycle hooks. These components are not needed to solve a single benchmark task. The simplification reduces the system prompt by $\sim 65\%$.

The agent loop is similar in principle to the design in Figure 8: it takes a task instruction and turns it into a sequence of tool calls and observations until the task is complete. OpenClaw was originally developed in TypeScript [36], which makes direct adaptation to the CUA framework difficult. To resolve this, we rewrote the agent loop in Python and added a small set of CUA-specific adaptations: a composite computer-use tool that matches CUA's native GUI surface, and a vision-driven GUI sub-agent, `delegate_gui`, with no OpenClaw analogue. We retained OpenClaw's load-bearing context-management primitives near-verbatim.

ALE-Claw is also of independent interest. By isolating OpenClaw's agent loop and porting it to Python, we make the design accessible to the broader Python research ecosystem and to the CUA framework specifically, where the original TypeScript implementation cannot be plugged in directly.

The Python port also enables dynamic ablation of the agent harness. Components can be swapped or removed in place, and the resulting performance shift can be measured directly; this workflow is impractical against the original TypeScript runtime. As one concrete future direction, ALE-Claw can serve as a fixed scaffold for benchmarking different GUI models against the same task suite.

D Extended Experiment Results and Analysis

D.1 Public-Subset Representativeness

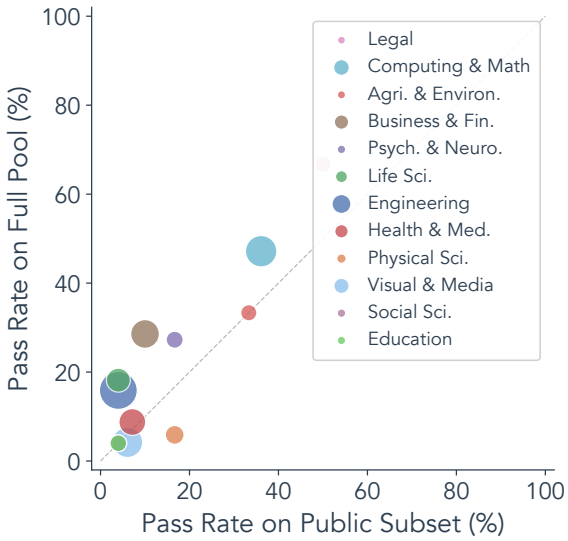


Figure 11: **Public-subset representativeness.** Pass rate per taxonomy cluster on the public subset (x) vs. the full task pool (y) for Claude Code + Opus 4.7. Point size $\propto$ total task instances per cluster. The strong correlation (r=0.89) confirms the public subset is representative.

To verify that the public subset is representative of the broader benchmark despite its limited size (Section 2.3), we ran Claude Code + Opus 4.7 on the full task pool. The full task pool yield a higher pass rate. The gap arises because the public set includes the full Last-Exam tier, while the private pool contains proportionally more Near-Term-level tasks. Figure 11 compares pass rates per taxonomy cluster on the public subset versus the full task pool. The two axes are strongly correlated (Pearson r=0.89, p<0.001), indicating that the public subset faithfully reflects full-pool difficulty across domains. Most clusters lie above the diagonal because the private pool's higher share of easier tasks raises each cluster's full-pool pass rate.

D.2 Timeout Analysis

ALE evaluations use a five hour wall clock cap per run. When a run reaches the cap, the harness stops the agent and the normal evaluator scores whatever artifacts are present in the output directory. In the runs used for Table 1, 3.8% of evaluated runs reached the cap. Runs that reached the cap have a mean score of 20.7, compared with 33.2 for runs that ended earlier.

Table 6: Timeout frequency by difficulty tier. Scores are mean normalized scores on the same 0 to 100 scale used in Table 1.

Tier	Timeout rate	Timeout score	Other score
Near-Term	3.1%	39.2	48.6
Full-Spectrum	3.2%	17.7	25.9
Last-Exam	6.4%	1.3	7.0

Table 7: Timeout frequency by harness for harnesses with at least one run that reached the cap.

Harness	Timeout rate	Timeout score	Other score
OpenClaw	5.7%	21.3	29.3
Cursor	4.8%	34.6	44.2
Claude Code	4.5%	22.0	40.1
Terminus	4.4%	0.0	35.8
Gemini CLI	4.2%	3.3	35.4
Codex	2.0%	35.1	36.8
ForgeCode	2.0%	0.0	33.3
ALE-Claw	1.2%	6.7	40.5
Grok CLI	1.0%	0.0	17.4
Droid	0.6%	3.0	45.9
Hermes	0.4%	0.0	34.8
OpenHands	0.3%	0.0	21.5

D.3 Failure Taxonomy Classification

This appendix documents the two-stage pipeline behind the failure root-cause taxonomy presented in Figure 9.

D.3.1 Stage 1: Trajectory analysis

For each failed task, an LLM (OpenAI Codex) was given access to the full run artifact directory, including the agent interaction log (`interaction_log.json`), run metadata (`run_result.json`, `agent_result.json`), evaluation output (`debug/eval/result.json`), and event traces (`events.jsonl`). The LLM was prompted to produce a structured *analysis card* in Markdown with five mandatory sections:

1. **Conclusion**: a one-sentence verdict, an overall judgment (success / partial success / failure), and the single most important problem.
2. **Task description**: a plain-language explanation of what the task asks, including key constraints and runtime metadata.
3. **What the agent did right**: correct behaviors, with evidence pointers to specific log entries.
4. **What the agent did wrong**: observed errors, with evidence pointers. The prompt requires separating confirmed errors from uncertain or inferred causes.
5. **Scoring**: the final score, raw score breakdown (if available), inferred evaluation criteria, and a confidence rating.

Each claim in the analysis card must cite a specific artifact file and field as evidence. The prompt prohibits reading the full transcript (`transcript.jsonl`) to keep generation cost bounded; the interaction log provides a sufficient behavioral summary.

D.3.2 Stage 2: Taxonomy classification

Each analysis card was then classified into a two-level failure taxonomy using GPT-4o (temperature 0). The classification prompt defines the following hierarchy:

- **Understanding**: the agent lacked knowledge or fabricated information.
  - *Domain Knowledge Gap*: the agent's errors trace back to missing specialized expertise. The prompt instructs: "Would a domain expert have avoided this mistake? If yes, classify here."
  - *Hallucination/Fabrication*: the agent invented data or results instead of computing them.
- **Approach**: the agent understood the domain but chose the wrong plan.
  - *Wrong Strategy*: the agent violated explicit task constraints or chose a fundamentally wrong method not attributable to domain ignorance.
  - *Incomplete/Abandoned*: the agent stopped early or failed to produce required deliverables.

- **Execution**: the approach was sound but the implementation was flawed.
  - *Implementation Bug*: logic errors, calculation mistakes, or data processing bugs.
  - *Output Format Error*: output in wrong format, location, or structure.
- **Infrastructure**: external constraints unrelated to agent capability.
  - *GUI/Browser Failure*: GUI or browser interaction failed due to tool issues.
  - *Timeout/Resources*: the agent ran out of time or computational resources.

D.3.3 Distribution

Nearly half (47%) of classifiable failures stem from Approach errors: wrong strategy (30%) or premature abandonment (17%). Understanding failures account for 31%, dominated by domain knowledge gaps (25%) with a smaller fraction (6%) involving hallucination or data fabrication. The remaining 22% are Execution errors: output format mismatches (10%), implementation bugs (8%), and GUI interaction failures (4%). Timeout and resource-exhaustion cases are excluded from this breakdown because they reflect environment constraints rather than agent reasoning failures; their prevalence is analyzed separately in Appendix D.2.

D.4 Model vs. Harness Effect

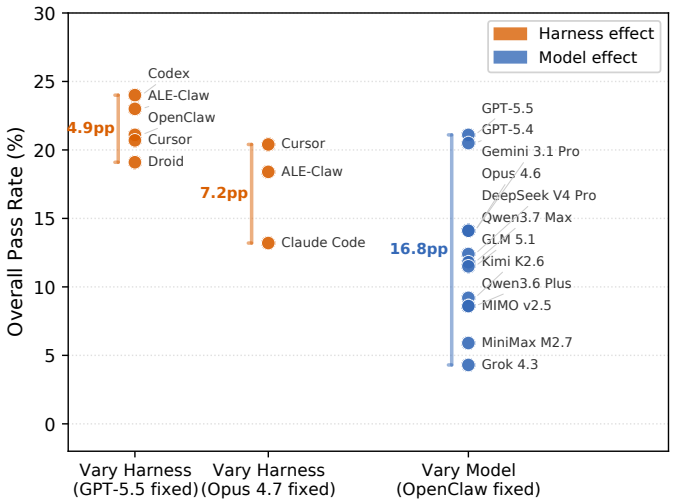


Figure 12: **Model choice vs. harness choice.** Each dot is one configuration; the vertical bracket shows the full range of overall pass rates. Varying the backbone model under a fixed harness (OpenClaw, 12 models) produces a 16.8 pp spread, roughly 3× the spread observed when varying the harness under a fixed backbone (4.9–7.2 pp).

A natural question raised by Table 1 is whether performance differences are driven primarily by the choice of foundation model or the choice of agent harness. Figure 12 isolates the two factors. Under a fixed OpenClaw harness, swapping the backbone model produces an overall pass-rate spread of 16.8 percentage points (from 4.3% for Grok 4.3 to 21.1% for GPT-5.5). Under a fixed backbone, swapping the harness yields a much narrower spread: 4.9 pp when the backbone is GPT-5.5 (five harnesses, 19.1–24.0%) and 7.2 pp when the backbone is Claude Opus 4.7 (three harnesses, 13.2–20.4%).

The pattern is consistent across both backbone choices: among the competitive harnesses evaluated here, engineering differences in prompting strategy, tool routing, and context management account for only a modest share of overall performance variation. The dominant factor is the foundation model's reasoning and domain knowledge, which aligns with the failure analysis in Section 4.2, where Understanding and Approach errors (both rooted in model capability) constitute the majority of failures.

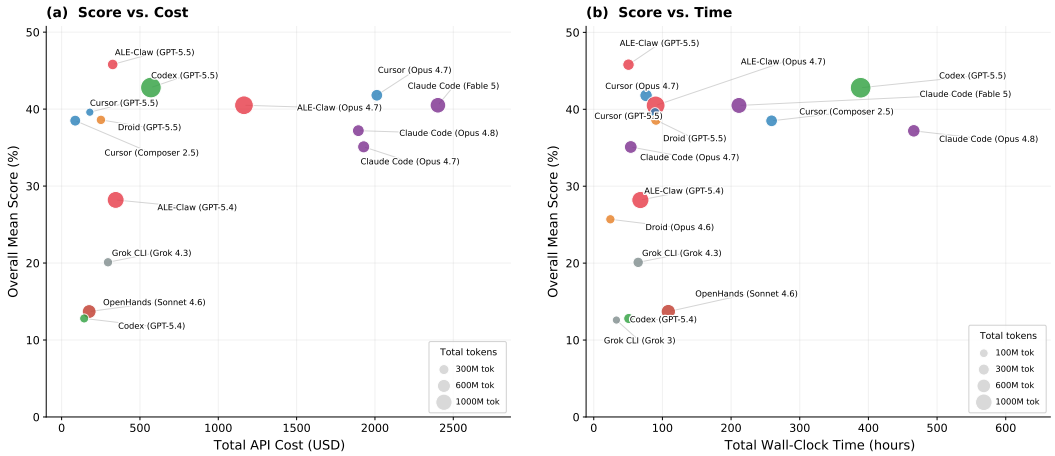


Figure 13: **Performance vs. resource consumption for mainstream agent harnesses.** Each bubble represents one harness–backbone configuration from Table 1; bubble area is proportional to total token consumption. **(a)** Overall mean score vs. total API cost (configurations with available cost data). **(b)** Overall mean score vs. total wall-clock time (all 16 configurations). The ideal operating point is the upper-left corner of each panel (high score, low resource use).

D.5 Cost, Time, and Token Efficiency

Figure 13 visualizes the relationship between performance and resource consumption across the 16 mainstream harness–backbone configurations in Table 1. Three observations emerge.

Cost and performance are only loosely correlated. In panel (a), ALE-Claw with GPT-5.5 achieves the highest overall mean score (45.8%) at $326 total API cost, while the same harness paired with Opus 4.7 spends 3.6× more ($1 164) yet scores 5.3 percentage points lower (40.5%). Cursor with Composer 2.5 is the most frugal configuration, reaching 38.5% for $87, whereas Claude Code with Fable 5 spends $2 402, the highest of any configuration, for a comparable 40.5%. The spread indicates that higher spending does not reliably translate to better results; the backbone model's fit to the task distribution and the harness's token efficiency jointly determine the cost–performance tradeoff.

Time efficiency varies widely. Panel (b) reveals that wall-clock time is largely decoupled from score. ALE-Claw (GPT-5.5) achieves the top score in approximately 51 hours of total wall-clock time, whereas Claude Code (Opus 4.8) requires 466 hours for a lower score (37.2%). Droid (Opus 4.6) is the fastest configuration (24 hours) but scores only 25.7%. The variation reflects differences in per-task timeout behavior, retry strategies, and the degree of parallelism in each harness's action loop.

Token consumption does not predict performance. Bubble sizes in both panels show that token-heavy configurations are not necessarily higher-scoring. ALE-Claw (Opus 4.7) consumes 1 373M tokens yet scores slightly below Cursor (Opus 4.7) at 460M tokens (40.5% vs. 41.8%). Conversely, Cursor (GPT-5.5) uses only 160M tokens while reaching 39.6%, suggesting that concise tool use and efficient context management can compensate for raw token volume.

D.6 Per-Task Instance Score Heatmaps

Figures 14–16 show the mean score of every task instance under each evaluated agent system. Rows are sorted by descending average score across all systems; columns are grouped by harness and sorted by descending average score within each group. Task instance labels are colored by taxonomy domain (legend inset). Gray cells indicate missing runs.



Figure 14: **Per-task instance scores: Near-Term tier** (67 task instances).

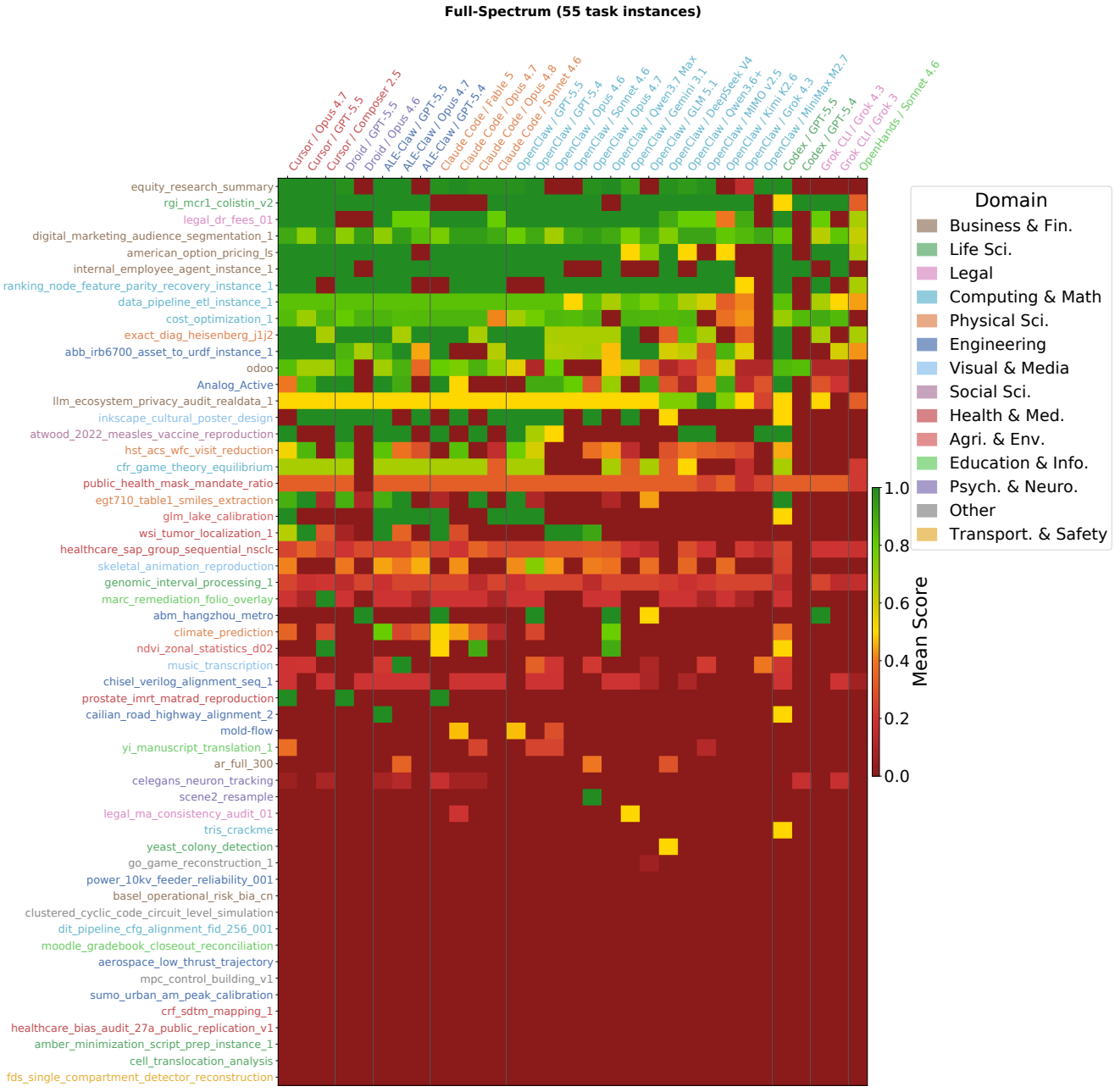


Figure 15: **Per-task instance scores: Full-Spectrum tier** (55 task instances).

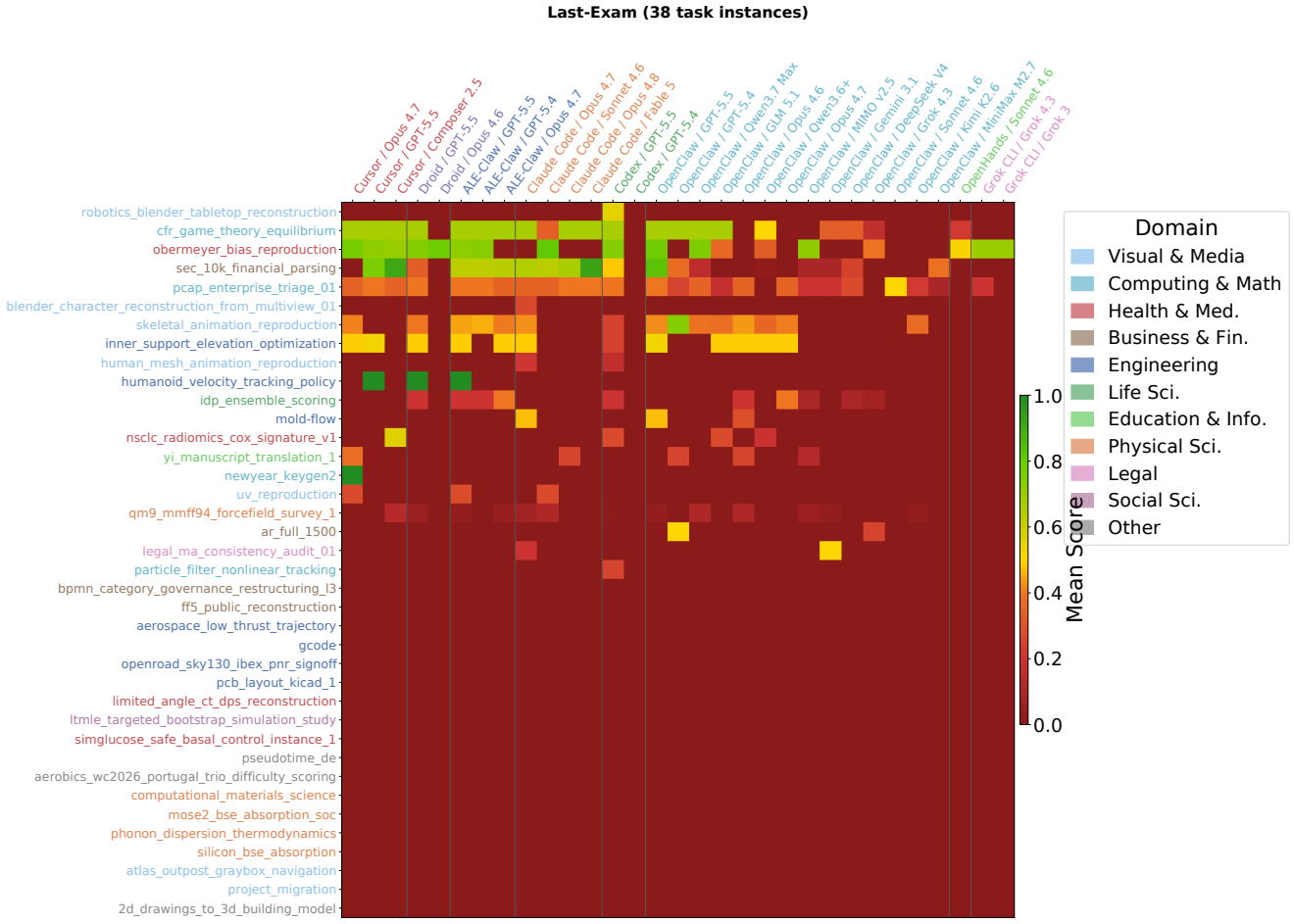


Figure 16: **Per-task instance scores: Last-Exam tier** (38 task instances).