Beyond Binary Correctness: Scaling Evaluation of Long-Horizon Agents on Subjective Enterprise Tasks

Ishan Gupta
Independent
USA
345ishaan@gmail.com

Abhishek Chandwani
Metaphi Inc
USA
abhishek@metaphi.ai

Abstract

Large language models excel on objectively verifiable tasks such as math and programming, where evaluation reduces to unit tests or a single correct answer. In contrast, real-world enterprise work is often subjective and context-dependent: success hinges on organizational goals, user intent, and the quality of intermediate artifacts produced across long, multi-tool workflows.

We introduce **LH-Bench**, a three-pillar evaluation design that moves beyond binary correctness to score autonomous, long-horizon execution on subjective enterprise tasks. The pillars are: (i) expert-grounded rubrics that give LLM judges the domain context needed to score subjective work, (ii) curated ground-truth artifacts that enable stepwise reward signals (e.g., chapter-level annotation for content tasks), and (iii) pairwise human preference evaluation for convergent validation. We show that domain-authored rubrics provide substantially more reliable evaluation signals than LLM-authored rubrics (κ=0.60 vs. 0.46), and that human preference judgments confirm the same top-tier separation (p<0.05), evidence that expert-grounded evaluation can scale without sacrificing reliability.

We additionally find that test-time verification—runtime hooks that surface structured feedback during execution—enables agents to recover from 70% of errors, with recovery strongly dependent on error-message quality. We release public datasets and report results on two environments: **Figma-to-code** (33 real `.fig` tasks against the Figma API via MCP) and **Programmatic content** (41 courses comprising 183 individually-evaluated chapters on a course platform serving 30+ daily users).

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; • **Human-centered computing** → *Human computer interaction (HCI).*

Keywords

Agent evaluation, rubric-based evaluation, expert-grounded rubrics, long-horizon agents, enterprise benchmarks

1 Introduction

LLM-based agents are increasingly deployed on complex enterprise workflows, yet the benchmarks used to evaluate them remain anchored to binary correctness: a unit test passes or fails, a math proof checks out or does not. Real enterprise work resists this framing because it is inherently procedural and context-dependent. What constitutes a correct application development lifecycle varies by enterprise—organizational style guides, design systems, and deployment conventions differ across companies. What qualifies as a good presentation or learning video varies by user persona, audience expertise, and pedagogical context. These outcomes cannot be reduced to deterministic pass/fail. Evaluating agents on such tasks requires expert-curated stepwise rewards and task-specific skill-based verifiers that grade process quality, not just final outputs.

This setting is hard for three compounding reasons: (i) *long-horizon* execution requires state tracking across dozens of interdependent steps, (ii) *subjective quality* cannot be reduced to a single correct answer, and (iii) *multi-artifact workflows* produce intermediate outputs whose quality determines downstream success. Binary correctness collapses these dimensions into a single outcome and cannot diagnose where or why an agent fails.

Recent measurement of autonomous agent capability [13] reveals that the frontier of reliable task completion has been doubling roughly every seven months, yet current models still collapse on tasks exceeding a few hours of equivalent human effort. The highest-value enterprise work—design-system implementation, source-grounded content production—sits squarely beyond this frontier, and exposes a *train–test gap*: post-training datasets overwhelmingly target code and math, so reasoning patterns may not transfer to the messy, multi-artifact workflows that dominate enterprise environments.

We introduce **LH-Bench**, a three-pillar evaluation design, and operationalize it in two enterprise environments:

- **Figma-to-code:** agents take *actual Figma artifacts* as inputs (curated as an evaluation dataset) and iteratively produce and revise front-end implementations.
- **Programmatic content:** agents operate in a structured "data room" containing expert-curated sources and must create *programmatic content*—code-generated videos (Remotion + TTS) or web-native presentations (React/Framer Motion)—for tutoring modules and product demos, following chapter-level ground truth with citations and iteratively editing based on feedback.

Across both environments, we compare agent trajectories and output artifacts against domain-expert ground truth, and grade performance using skill-specific rubrics plus hybrid evaluation (LLMs and expert humans).

Audience and Use. LH-Bench targets researchers who need realistic, environment-grounded evaluations and post-training datasets for improving long-horizon agent reliability. LH-Bench is designed as a reusable evaluation environment. New Figma design files or data-room configurations can be dropped into the pipeline, and three-pillar evaluation scores agents without per-task human annotation. This is possible because expert-authored rubrics, curated ground-truth artifacts, and workflow-specific `SKILL.md` references

encode the domain knowledge that LLM judges need to simulate expert evaluation—addressing the critical bottleneck of scaling agent evaluation on subjective enterprise tasks.

Our contributions are:

(1) **Evaluation design:** LH-Bench moves beyond binary correctness by combining expert-grounded rubrics for LLM judges, curated ground-truth artifacts for stepwise rewards, and pairwise human preference evaluation.
(2) **Validation evidence:** all three tiers converge on the same primary separation (Table 9); expert-authored rubrics yield substantially higher inter-judge agreement than LLM-authored rubrics (κ=0.60 vs. 0.46).
(3) **Benchmark instantiation:** two enterprise environments (Figma-to-code and programmatic content) with end-to-end evaluation of three autonomous agent harnesses.
(4) **Recovery analysis:** runtime verification hooks enable self-correction; across 96 runs, agents recover from 70% of errors, with recovery dependent on error-message quality.

2 Research Questions

We organize the paper around the following research questions (RQs), motivated by the need for evaluation designs that can score subjective, context-dependent enterprise work over long horizons:

- **RQ1 (Evaluation): How can subjective, context-dependent enterprise work be evaluated reliably beyond binary correctness?** We study rubric-based evaluation and artifact verification, compare expert-authored and LLM-authored rubrics for inter-judge reliability, and show how different tiers (artifact, skill, behavior) capture complementary aspects of performance.
- **RQ2 (Benchmarking): What failure modes emerge when agents autonomously execute long-horizon enterprise tasks in tool-rich environments?** We characterize errors in navigation, state tracking, multi-tool coordination, and artifact regression across iterative edits.
- **RQ3 (Test-time recovery): How effectively can agents self-correct from structured verification feedback?** We evaluate recovery behavior when agents receive build/deploy failures and visual/rubric violations as actionable feedback during execution.

3 Related Work

Agent Benchmarks and Environments. Existing benchmarks evaluate LLM agents in interactive environments—WebArena [34], OSWorld [29], SWE-bench [10], AgentBench [17], GAIA [19]—but predominantly use binary success metrics on single-turn or short-horizon tasks. WorkArena [7] evaluates 682 tasks in ServiceNow, the closest existing enterprise benchmark, but uses binary completion metrics. SWE-Bench Pro [6] extends to 1,865 long-horizon software engineering tasks but remains unit-test-graded. LH-Bench differs by targeting subjective professional knowledge work requiring iterative artifact editing, using multi-tier evaluation rather than binary pass/fail, and grounding tasks in real enterprise artifacts (Table 1).

UI Generation and Design-to-Code. Design2Code [23] benchmarks screenshot-to-code generation on 484 real webpages, and FronTalk [28] extends to multi-turn front-end generation with visual feedback, highlighting a "forgetting issue" where agents overwrite prior features. FullFront [24] benchmarks the full front-end engineering workflow and FrontendBench [35] evaluates 148 automated front-end tasks, but both use screenshots as inputs. LH-Bench extends this line of work by using *actual Figma design files* as inputs (not screenshots), requiring agents to navigate design-tool APIs for structure extraction, asset export, and token discovery before coding—and to iterate across multiple verification cycles without regressing earlier work.

Tool Use and Multi-Turn Interaction. MINT [26] evaluates multi-turn tool use with language feedback, and ToolLLM [21] scales to 16,000+ real-world APIs. LH-Bench focuses on *multi-tool orchestration over long sessions*: agents coordinate design-extraction, code-generation, build, preview, and deployment tools across dozens of turns, with structured verification feedback enabling test-time recovery.

Long-Horizon Agent Evaluation. Kwa et al. [13] show that reliable task completion has been doubling every ∼7 months, but collapses beyond a few hours of equivalent human effort—precisely the regime enterprise workflows occupy. UltraHorizon [18] benchmarks trajectories exceeding 200k tokens, identifying error types including in-context locking. τ^2-bench [1] extends dual-control evaluation to conversational agents but focuses on short-horizon API interactions. Turn-level reward design [27] demonstrates that sparse final rewards are insufficient for credit assignment in long episodes. LH-Bench contributes enterprise-specific long-horizon evaluation with skill-level scoring that provides dense, diagnostic signals aligned to expert-defined workflow phases, building on the ReAct paradigm [31].

LLM-Based Evaluation and Judging. Zheng et al. [33] established the LLM-as-a-Judge paradigm, achieving over 80% agreement with human preferences. Subsequent work has catalogued judge biases [3, 8, 25] and proposed multi-agent judging [4]. LH-Bench uses *three LLM judges* from different model families with expert-authored rubrics and cross-judge variance tracking, applied to multi-dimensional enterprise artifacts.

Rubric-Based and Skill-Based Evaluation. ResearchRubrics [22] demonstrates that rubric granularity significantly affects ranking reliability, DeepResearch Bench II [15] uses expert-authored rubrics to diagnose research agents, and SkillsBench [16] finds that skill-level decomposition reveals failure modes invisible to aggregate scores. LH-Bench builds on both lines: domain-expert rubrics with skill-level decomposition produce dense, diagnostic signals for long-horizon enterprise tasks.

Enterprise AI and Compound Systems. The shift to compound AI systems [32] motivates system-level evaluation including orchestration and feedback loops. Recent surveys [9, 20] note the scarcity of benchmarks targeting real enterprise workflows. LH-Bench addresses this gap by benchmarking full agent harnesses on enterprise tasks, with signals designed to diagnose harness-level differences in context management, tool orchestration, and recovery.

Table 1: Positioning of LH-Bench relative to existing agent benchmarks. Existing benchmarks rely on binary or unit-test evaluation; LH-Bench introduces multi-tier, expert-grounded scoring.

Benchmark	Tasks	Multi-turn	Real env.	Expert rubrics	Artifact eval
WebArena [34]	812	×	Web	×	Binary
VisualWebArena [11]	910	×	Web	×	Binary
SWE-bench [10]	2294	×	Code	×	Unit tests
OSWorld [29]	369	×	OS	×	Binary
τ-bench [30]	200	✓	API	×	Binary
Design2Code [23]	484	×	×	×	VLM
MINT [26]	varies	✓	×	×	Binary
LH-Bench (ours)	**216**	✓	✓	✓	**Multi-tier**

4 Agent Harness Design

LH-Bench evaluates end-to-end *agent harnesses* rather than base models in isolation. A recent convergence in agent-CLI design—driven by rapid improvements in model reasoning for code, tool use, and file manipulation—has produced a shared architectural pattern across Claude Code (Anthropic), Codex CLI (OpenAI), and Gemini CLI (Google): all three provide a sandboxed shell environment, file-system tools, and extensible tool interfaces (e.g., MCP servers). This generality means that environment-specific capabilities such as Figma extraction, preview verification, or source-grounded generation can be defined once and run identically across harness families, enabling controlled comparison of the *orchestration and reasoning* differences that actually differentiate harnesses on long-horizon tasks.

We design LH-Bench harnesses to be *autonomous by default*: agents execute in sandboxes, invoke tools programmatically, and recover from failures using runtime feedback rather than relying on manual intervention. Expert-authored `SKILL.md` workflow references are loaded identically across all three CLIs, isolating the effect of harness-level differences (context management, retry policies, compaction strategies) from environment knowledge. Below we describe the key features of our harness design.

4.1 Sandboxed Execution and Reproducibility

Agents interact with environments through controlled APIs (e.g., design extraction, browser automation, build/deploy commands) and operate against task-specific state snapshots. This reduces hidden state and improves reproducibility of trajectories and artifacts.

4.2 Context Compaction and Pre-Compaction

Long-horizon tasks routinely exceed context limits. Our harnesses implement automatic compaction policies that summarize prior turns, tool outputs, and intermediate artifacts while preserving commitments, open TODOs, and verified facts. We additionally use proactive pre-compaction: before expensive tool calls or major edits, the harness produces a compact "working set" view of the current state to reduce regressions and support edit locality.

4.3 Runtime Verification and Recovery

Harnesses integrate verification as a first-class interface. Build/deploy failures, rubric violations, and visual mismatches are surfaced as structured, machine-readable feedback that the agent can consume to plan repairs. For example, in Figma-to-code the harness exposes a preview-verification hook: after the agent calls a preview tool (e.g., `create_app_preview`), automated checks run and return a structured error if issues are detected—runtime exceptions (e.g., React router nesting errors), blank-page renders, or other failures. The error payload includes a diagnosis and an explicit next action (e.g., "fix these errors, then call `create_app_preview` again"), enabling iterative self-correction within the same episode.

Verification hook availability. Preview verification hooks are implemented as post-tool-use hooks for Claude Code (via the Agent SDK hook interface) and as after-tool subprocess hooks for Gemini CLI (via `.gemini/hooks/`). Codex CLI does not support native hook mechanisms; it can invoke the `create_app_preview` MCP tool but receives raw tool output without automatic verification.

4.4 Extensible Tool Interface

All three CLIs support tool extensibility via MCP (Model Context Protocol) servers and custom tool definitions. LH-Bench defines environment-specific tools—Figma structure inspection, asset export, scene generation, source extraction—as standalone servers that any harness can invoke. This decouples environment logic from harness internals and ensures that performance differences reflect orchestration quality rather than tool availability.

5 Benchmark Environments

LH-Bench currently includes two enterprise environments. Each environment is a tool-rich, interactive setting in which an agent must complete long-horizon tasks by producing and iteratively editing artifacts. All environments are evaluated across three agent harness families—Claude Code (Anthropic), Codex CLI (OpenAI), and Gemini CLI (Google)—using identical tool access and expert-authored `SKILL.md` workflow references (Section 6). Below we describe the task design, action space, and verification constraints for each environment.

5.1 Figma-to-Code Environment

In the Figma-to-code environment, agents take real Figma design artifacts as inputs and iteratively produce front-end implementations across 33 `.fig` tasks. Tasks require navigation of design structure, faithful implementation, and safe localized edits across multiple iterations.

Action space. Agents interact through a constrained tool set: (a) Figma MCP calls for design structure, styles, components, and asset exports; (b) file and shell tools for writing code and running builds; (c) a sandboxed preview tool that catches build/runtime failures; and (d) a deployment tool that publishes static builds for artifact evaluation and reproducible inspection.

Workflow constraints. We enforce conventions encoded in expert-authored `SKILL.md`: agents must extract from Figma before coding (no guessing of colors/fonts), implement all frames (no cherry-picking), use non-interactive scaffolding (no `npm create`/`npx create-*`), use relative asset paths suitable for subdirectory deployments, and checkpoint via preview at multiple milestones to enable recovery.

Ground truth and verification. Each task includes a `ground_truth/` directory with a `manifest.json` (frame metadata: name, node id, viewport, target route) and 2x-scale PNG exports per frame. The evaluated agent builds the UI, which is exercised via Playwright (MCP) to capture screenshots; a VLM judge (Gemini 3) compares the built UI against ground truth frame-by-frame. We also run programmatic checks: build verification (`npm build`), deployment accessibility, and component-coverage (matched/total). Pure infrastructure failures (score=1) are excluded from model comparisons.

Task complexity axes. We curate tasks along four complexity axes to ensure the benchmark spans a wide capability range: (i) *application category*—e-commerce, SaaS dashboards, portfolio sites, landing pages—each imposing different design-system conventions; (ii) *frame count* (ranging from single-frame layouts to designs with 70+ frames), which determines the scope of navigation and cross-page consistency the agent must maintain; (iii) *image and asset density*, from icon-light text layouts to media-heavy product grids requiring bulk asset export and format selection; and (iv) *route navigation complexity*, from single-page designs to multi-route applications with nested navigation hierarchies and responsive breakpoints. This deliberate stratification ensures the benchmark tests agents across the full difficulty spectrum rather than clustering at a single complexity level.

5.2 Programmatic Content Environment

Explaining complex concepts visually—for onboarding, training, or learning content—is among the highest-value knowledge work in enterprises. The same source material (e.g., a research paper, a product specification) may need to be rendered as a narrated video for executives, an interactive slide deck for engineers, or a mathematical animation for researchers. This diversity of audience and format makes the task genuinely long-horizon: agents must retrieve relevant evidence from large document collections, choose an appropriate visual style, and compose code-driven media artifacts that are faithful to the source material.

In the programmatic content environment, agents operate in a structured "data room" of expert-curated sources and must generate code-driven content—not pixel-level video from diffusion models, but *composable programs* that render to media. Agents produce either (i) a programmatic video (Remotion + TTS, exported to MP4), (ii) a mathematical animation (Manim, rendered to MP4), or (iii) a web-native presentation (React/Framer Motion slide deck). All three output modes share the same harness and tool interface, enabling controlled comparison across content types. We evaluate on 183 chapters across 41 full-fledged tutor-directed courses.

Action space. Agents interact through a layered tool set organized by output mode. *Common tools* available in all modes include: a *source extractor* that normalizes PDFs, DOCX, PPTX, and web pages into line-numbered markdown with a structured index (`sources.json`); a *context researcher* sub-agent that searches extracted documents and returns findings with line-level citations (`source_id:line_number`); and an *image generator* for concept art and diagrams. *Video-specific tools* include a parallel scene generator (`generate_video_scenes`) that produces React components and TTS audio simultaneously, a Manim generator (`generate_manim_scene`) for mathematical animations, and a root-composition generator that assembles scenes with precise audio–video timing. *Presentation-specific tools* include a storyboard generator (narrative arc planning), a React slide generator with Framer Motion animations, a chart generator (CSV to PNG), and preview/screenshot tools for iterative viewport validation.

Multi-turn conversation simulation. Tasks in this environment simulate how domain experts interact with learning content: a user provides a document collection (e.g., three arXiv papers on GRPO), and the agent must produce a sequence of chapters—each building on prior context. Subsequent turns introduce the kind of requests a real learner would make: "now compare GRPO with PPO," "add a visual showing the gradient flow," or "make the timeline start from 2012." The agent must maintain *scene coherence* across turns (e.g., a neural-network diagram introduced in Turn 1 must retain its visual style in Turn 3 unless explicitly changed) and perform *fast editing*: isolated changes must produce targeted CSS or Manim edits without regenerating entire scenes. Session IDs chain Claude sessions across turns, and the workspace (including extracted context and generated artifacts) persists, so the agent accumulates state over the full episode.

Grounding and verification. Source extraction produces a persistent context directory (`context/`) and a structured index (`context/sources.json`) with per-source line counts and section headings, enabling judges to verify that generated scripts are supported by provided sources and to penalize hallucinated or uncited material. For video mode, agents must plan 4–8 scenes (10–25 seconds each, max 2 minutes total) and render a final MP4 via Remotion at 15 fps, with audio–video duration matching enforced at the frame level. For Manim mode, audio is generated first and the animation is constrained to match the audio duration exactly. For presentation mode, every slide must fit the viewport (100vh, no scrolling) and is validated via automated preview and screenshot checkpoints before deployment.

6 Benchmark Construction

We construct LH-Bench from expert-designed workflows and enterprise-representative artifacts. Each task includes (i) environment state (e.g., a `.fig` file or a data room of sources), (ii) an expert-authored rubric mapping to skills, and (iii) verifiers over intermediate and final artifacts.

6.1 Expert Rubrics and `SKILL.md`

For each environment, domain experts author `SKILL.md` guidance capturing environment-specific conventions, common failure modes, and recommended tool usage. Separately, experts define skill rubrics used to score agent trajectories (e.g., navigation, error recovery, edit locality, and spec adherence). Rubrics are written by four experienced front-end engineers (6+ years in B2C internet companies) for Figma-to-code, and by domain experts in instructional design for the programmatic content environment.

6.2 SME Annotation Workflows

Figma-to-code dataset curation. Ground truth is constructed from an automated pipeline with expert curation. We source candidate designs from Figma Community, pre-filtering by engagement and selecting specifically for enterprise complexity: multi-frame layouts with real typography systems, brand color palettes, component hierarchies, and responsive breakpoints—not single-page templates or simple wireframes. Selected designs are sourced from Figma Community and enterprise design partners across healthcare, e-commerce, and professional-services verticals, then duplicated into evaluation accounts via Playwright. Design structure is extracted via the production Figma API, and LLM-based quality scoring is applied before final expert selection. Expert selection balances the four complexity axes (Section 5.1) to ensure coverage across application categories, frame counts, asset densities, and navigation topologies.

Programmatic content source-grounded annotation. To build faithful ground truth for programmatic content, we provide SMEs with a dedicated annotation interface that supports chapter-level task specification and fine-grained source grounding. Unlike screenshot-only workflows, which are difficult to scale and provide limited traceability, our interface segments sources into line-addressable spans and uses a "highlight-to-cite" interaction to attach exact evidence to each chapter. This produces high-quality, granular citations with explicit metadata (source, span, chapter), reducing noise and improving annotation speed.

6.3 Artifact Contracts

LH-Bench makes subjective tasks scorable by requiring agents to emit *interim artifacts*—per-frame ground truth images, structured manifests, citation targets—that serve as deterministic hooks for downstream verification (schemas in Appendix C).

6.4 Environment Infrastructure

Task definitions and rubrics are versioned independently. The pipeline (Appendix A) exposes execution, evaluation, and leaderboard endpoints; agents build artifacts in ephemeral sandboxes and upload finalized builds to object storage.

6.5 Parallel Judging

Evaluation runs three judges in parallel (Table 2): a task-agnostic trajectory judge (planning, constraint-following, recovery), a task-specific process judge (workflow compliance and skill execution), and an output judge (artifact fidelity from rendered artifacts and runtime checks). Judges emit structured JSON grades per tier, enabling both leaderboard ranking and fine-grained diagnostics. We will release public versions of the task datasets for research use upon publication.

Table 2: LH-Bench judges and the artifacts they consume.

Judge	Inputs	Scores
Trajectory	Transcript + tool traces	Planning, recovery
Process	Transcript + skill rubric	Workflow compliance
Output	Ground truth + screenshots	Visual fidelity

7 Evaluation Framework

We propose a hybrid evaluation approach combining expert rubrics, trajectory grading, and programmatic verification of artifacts (including visual outputs).

7.1 Rubrics, Tiers, and Aggregation

Each rubric dimension is scored on a 1–5 scale with anchors: 1=Inadequate/Poor, 2=Developing/Below Average, 3=Proficient/Adequate, 4=Advanced/Good, 5=Expert/Excellent. Rubrics are organized into tiers and aggregated as weighted averages within tier and (optionally) across tiers.

7.1.1 Process tier rubrics (Figma-to-code). Our process tier uses expert-authored, transcript-evaluable rubrics with observable boundaries aligned to sequential workflow phases. In our current Figma-to-code release, we use four core process rubrics: (i) design inspection and asset extraction, (ii) design token and style extraction, (iii) component and layout architecture, and (iv) build verification and iteration. These criteria explicitly reward early design inspection, semantic asset organization, token centralization prior to component implementation, and preview-driven iteration—all of which reduce regression and improve output fidelity over long horizons.

7.1.2 Output tier rubrics (Figma-to-code). Our output tier grades rendered artifacts against exported ground truth frames and runtime checks. We score component coverage (with absolute matched/total fields), layout accuracy, color accuracy, typography accuracy, asset display, overall visual fidelity, responsive behavior, and interaction fidelity. These rubrics are designed to be decomposable (for diagnosis) while still supporting a single leaderboard score via weighted aggregation.

Tier 1: Artifact Score (Figma-to-code). We compute a weighted average over eight artifact rubrics—component coverage, layout accuracy, colors, typography, asset display, visual fidelity, responsive behavior, and interaction fidelity—comparing the deployed UI against design ground truth via a VLM judge (Gemini 3) that performs frame-by-frame comparison of Playwright-captured screenshots against expert ground-truth images. Weights and definitions are in Appendix G.

Tier 2: Skill Score (Figma-to-code). We compute a weighted average over four core process rubrics: design inspection and asset extraction, design token and style extraction, component and layout architecture, and build verification and iteration (see Section 8.4 for per-rubric analysis). Scores are produced by three LLM judges (Gemini 3.1 Pro, Claude Sonnet 4.6, GPT-5.2) and averaged across judges; variance is tracked.

Table 3: Figma-to-code output scores (VLM judge, 1–5 scale). Seven model configurations across three harness families. 95% bootstrap CI reported for primary candidates.

Rank	Agent harness / model	Score	n
1	Codex (GPT-5.2 Pro)	4.27	18
2	Claude Code (Opus 4.6)	4.19 ± 0.28	32
3	Codex (GPT-5.2)	3.94 ± 0.36	31
4	Claude Code (Opus 4.5)	3.88	29
5	Gemini CLI (Gemini 3.1 Pro)	3.73 ± 0.44	29
6	Claude Code (Sonnet 4.5)	3.66	22
7	Gemini CLI (Gemini 3 Pro)	3.59	22

Tier 3: Behavior Score (Optional). We optionally score task-agnostic trajectory rubrics (equal-weighted) covering tool usage, error recovery, instruction following, planning, information gathering, efficiency, task completion, and safety/constraints. This tier is disabled by default.

7.2 Scoring and Aggregation

Tier scores are weighted averages of rubric dimensions ($S_{\text{tier}} = \sum_i w_i s_i$, $\sum w_i = 1$). When multiple judges score the same tier, we report the mean and track cross-judge variance. We report output and process tiers separately; process scores serve as dense diagnostic signals for long-horizon learning.

8 Experiments

8.1 Experimental Setup

We evaluate three agent harness families—Claude Code, Codex CLI, and Gemini CLI—across seven configurations (harness × model; see Appendix D) on two environments: Figma-to-code (33 tasks) and Programmatic content (183 chapters across 41 courses). Each configuration executes autonomously in a sandboxed environment with identical tool access and workflow constraints defined by an expert-authored `SKILL.md`. Skill-tier evaluation uses the flagship model from each family. A controlled ablation compares execution with and without `SKILL.md` on Figma-to-code (Section 8.5).

For the programmatic content environment, human SMEs provide (i) reasoning on how they would design content given each user instruction, structured as a 5-skill rubric covering content selection, narrative structure, visual hierarchy, information density, and source grounding; and (ii) per-instruction annotations indicating which documents and assets from the data room they would focus on. These annotations serve as ground truth for rubric synthesis and VLM-based artifact scoring.

8.2 Figma-to-Code Results

Table 3 reports **output scores** (visual artifact fidelity evaluated by a VLM judge), Table 4 reports **skill scores** (process quality evaluated by three LLM judges from different model families), and Table 5 decomposes skill scores by rubric to expose per-skill strengths and bottlenecks. All tables include 95% bootstrap confidence intervals (1,000 resamples over tasks) for the three primary candidates.

Table 4: Figma-to-code skill scores (3 LLM judges, 1–5 scale). 95% bootstrap CI over tasks. Per-judge columns: Gemini 3.1 Pro, Claude Sonnet 4.6, GPT-5.2. Var. is population variance across judges.

Rank	Agent / model	Score (n)	Gemini	Claude	GPT	Var.
1	Claude Code (Opus 4.6)	3.27 ± 0.14 (31)	3.50	3.37	2.92	0.14
2	Codex (GPT-5.2)	3.16 ± 0.15 (31)	3.51	3.11	2.82	0.15
3	Gemini CLI (3.1 Pro)	2.80 ± 0.11 (29)	3.06	2.70	2.67	0.06

Table 5: Figma-to-code skill scores decomposed by rubric (average across 3 LLM judges, 1–5 scale). 95% bootstrap CI on overall score. Rubric columns: Inspect.=Design Inspection, Token=Token & Style Extraction, Comp.=Component Architecture, Build=Build Verification. Bold indicates highest per-rubric score. κ row shows mean pairwise Cohen's kappa.

Rank	Agent / model	Score (n)	Inspect.	Token	Comp.	Build
1	Claude Code (Opus 4.6)	3.27 ± 0.14 (31)	3.40	**2.88**	**3.58**	3.21
2	Codex (GPT-5.2)	3.16 ± 0.15 (31)	**3.52**	2.64	3.34	2.97
3	Gemini CLI (3.1 Pro)	2.80 ± 0.11 (29)	2.95	1.66	3.28	**3.45**
	κ (agreement)	0.60	0.50	0.58	0.34	0.67

8.3 Programmatic Content Results

We evaluate programmatic content generation using a VLM-as-judge approach: a Gemini 3.1 Pro judge scores each rendered chapter against SME-authored rubrics encompassing content relevance, visual design, pedagogical effectiveness, audio-visual synchronization, and technical accuracy (see Appendix I for full rubric definitions). Scores are normalized to $[0, 1]$ and reported as course-level means with 95% bootstrap confidence intervals (n=41 courses, 183 chapters). We further validate VLM rankings against human expert pairwise preferences (n=275 matched comparisons).

Rubric granularity analysis. Table 6 reports VLM artifact scores under two rubric granularities: a coarser 3-point scale (0/1/2 per criterion) and a finer 5-point scale (1–5 per criterion). Both scales preserve the same harness ranking (Claude Code > Gemini CLI > Codex), confirming that system-level conclusions are robust to rubric granularity. The 5-point scores are uniformly higher (+0.03 on average) because the finer scale captures partial credit—a "partially correct" element scores 3/5 (0.60) rather than 1/2 (0.50) under the 3-point scale. The 5-point scale also yields substantially tighter confidence intervals (e.g., Codex ±0.044 vs. ±0.071, a 38% reduction), indicating that finer granularity reduces measurement noise. Critically, the 5-point scale reduces VLM ties in pairwise comparisons by 49% (38 ties vs. 74 with the 3-point scale), bringing VLM tie behavior in line with human annotator tie rates (36 ties) and eliminating the structural mismatch that deflates inter-rater κ (see Appendix J). This validates the rubric design principle discussed in Section 8.6: finer-grained scales improve evaluation precision and discriminative power without altering system-level conclusions (see Appendix I for full rubric definitions).

Table 6: Programmatic content: VLM artifact scores (Gemini 3.1 Pro judge, normalized 0–1). Scores are course-level means of per-chapter normalized scores with 95% bootstrap CIs (n=41 courses, 183 chapters).

Agent / model	3-pt score	5-pt score
Claude Code (Opus 4.6)	0.588 ± 0.093	0.612 ± 0.069
Gemini CLI (3.1 Pro)	0.507 ± 0.087	0.526 ± 0.063
Codex (GPT-5.2)	0.441 ± 0.071	0.478 ± 0.044

8.4 Rubric-Level Analysis

Table 5 decomposes skill scores into the four expert-authored process rubrics, revealing consistent patterns across agents.

Design token and style extraction is a universal bottleneck. All three agents score lowest on the *token & style extraction* rubric (Claude Code 2.88, Codex 2.64, Gemini CLI 1.66). Gemini CLI's score of 1.66 is particularly notable—it is the only sub-2.0 rubric average in the table—suggesting that its harness rarely performs systematic token discovery before coding. This is consistent with the "prompt-to-pixels" failure pattern where agents skip design inspection and guess visual properties, resulting in cascading fidelity errors. That even the top-ranked agent achieves only 2.88 on this rubric suggests token extraction remains a broadly unsolved challenge for current harnesses.

Component and layout architecture is consistently strong. All agents score highest or near-highest on *component & layout architecture* (Claude Code 3.58, Codex 3.34, Gemini CLI 3.28), indicating that current code-generation capabilities translate well to structural implementation once the agent begins coding.

Compensatory skill profiles emerge across agents. Codex leads on *design inspection* (3.52 vs. Claude Code's 3.40) but trails on *build verification* (2.97 vs. Gemini CLI's 3.45). Gemini CLI, despite the lowest overall score, achieves the *highest* build verification score (3.45), suggesting its harness excels at iterative preview-driven correction even when upstream design extraction is weak. These compensatory profiles indicate that overall score alone obscures important harness-level trade-offs, and that rubric-level decomposition provides actionable diagnostic signal for improving agent workflows.

Task-level difficulty gradient. The deliberate complexity stratification (Section 5.1) produces a measured difficulty gradient in the results. Applying the expert pass/fail threshold (≤3 = fail, ≥4 = pass) across all rated tasks (n=31): 16% are universally easy (100% pass rate across all agents), 13% are universally hard (0% pass—no agent produces shipping-quality output), and 71% are discriminating (at least one agent passes, at least one fails). The discriminating majority is where harness-level differences emerge: Claude Code achieves a 54.5% pass rate, Codex 57.9%, and Gemini CLI 52.4%, with the 4 universally hard tasks identifying current capability ceilings at the intersection of high frame count, dense asset hierarchies, and multi-route navigation (per-task breakdown in Appendix L).

Table 7: `SKILL.md` ablation: mean skill scores (3-judge avg, 1–5) with and without expert workflow guidance across 3 tasks. n = number of paired task runs per harness.

Agent	Condition	Insp.	Token	Arch.	Build	Overall
Claude Code (n=3)	Without	3.44	2.78	3.44	3.22	3.23
	With	3.78	3.11	3.56	2.89	3.38
	Δ	+0.34	+0.33	+0.12	−0.33	+0.15
Codex (n=2)	Without	2.17	2.50	2.34	1.00	2.06
	With	3.00	2.84	3.17	2.67	2.93
	Δ	+0.83	+0.34	+0.83	+1.67	+0.87
Gemini CLI (n=2)	Without	3.50	1.50	3.00	3.00	2.78
	With	3.17	1.67	3.17	3.34	2.83
	Δ	−0.33	+0.17	+0.17	+0.34	+0.05
Pooled (n=7)	Δ	+0.28	+0.28	+0.37	+0.56	**+0.33**

8.5 Ablation: Expert Workflow Guidance

We ablate the effect of expert-authored `SKILL.md` on the Figma-to-code environment by running all three harness families *without* the expert workflow (a single-line task description, no procedural guidance) on 3 tasks spanning the difficulty gradient, and comparing against matched *with*-`SKILL.md` runs on the same tasks. Tool access is held constant within each harness. Table 7 reports per-harness skill scores (n=7 paired runs).

Preliminary evidence from this small-scale ablation (n=7 paired runs) suggests the benefit of expert workflow guidance varies by harness. Codex gains the most (+0.87 overall), with build verification improving from 1.00 to 2.67—without `SKILL.md`, Codex never successfully previews or iterates, producing a floor score on that rubric. Claude Code gains modestly (+0.15), consistent with its native familiarity with the Skills specification, though design inspection and token extraction each improve by ∼0.3. Gemini CLI gains minimally (+0.05), suggesting its harness-level iteration patterns (100% deploy rate, strong build verification) may compensate for the absence of procedural guidance. Across harnesses, execution cost drops 42% for Claude Code (mean \$6.45→\$3.75) as expert workflows reduce exploratory backtracking. The sharpest operational signal is deployment success: only 2 of 7 runs without `SKILL.md` produce a deployable artifact, compared to near-universal deployment with it. We note that this ablation is underpowered (2–3 runs per harness) and treat these results as directional; scaling to more tasks per condition is needed for definitive conclusions. Versioning infrastructure for scaling this ablation is described in Appendix M.

8.6 Inter-Judge Agreement

We use three judges from different model families (Gemini 3.1 Pro, Claude Sonnet 4.6, GPT-5.2) and report cross-judge variance and Cohen's κ [5] as measures of agreement.

Figma-to-code. Mean pairwise Cohen's κ (quadratic weights) is 0.60 (n=92 runs, 360 ratings), indicating moderate-to-substantial agreement [14]; per-rubric κ ranges from 0.34 (component architecture) to 0.67 (build verification). All three judges preserve the same rank ordering across harnesses despite absolute-score offsets consistent with known calibration differences across LLM families [8].

Table 8: Inter-judge agreement by rubric version (same judge families, same 92 runs). κ is quadratic-weighted Cohen's kappa.

Metric	v1.1 (LLM, 8 rubrics)	v1.2 (Expert, 4 rubrics)
Mean pairwise κ	0.46	**0.60**
Gemini–Claude	0.44	**0.65**
Gemini–GPT	0.31	**0.52**
Claude–GPT	0.63	**0.64**
Mean variance	0.25	**0.10**

Expert-authored vs. LLM-authored rubrics. We compare agreement under two rubric versions on the same 92 runs (Table 8). Under v1.1 (8 LLM-authored rubrics), $\kappa = 0.46$; under v1.2 (4 expert-authored rubrics), $\kappa = 0.60$—a +0.15 improvement. The largest gains occur in Gemini-involved judge pairs (+0.21), suggesting that fewer, well-anchored expert dimensions reduce scoring ambiguity for judges most sensitive to rubric phrasing.

Convergent validity across evaluation tiers. Three independent evaluation signals—VLM artifact fidelity (output tier), 3-judge transcript evaluation (skill tier), and pairwise human preference (Table 9)—all agree on the same primary ranking boundary. The output and skill tiers use entirely different inputs, judges, and rubrics; the human evaluator uses direct side-by-side comparison with no access to LLM scores. This three-way convergence provides strong evidence that expert-authored skill verifiers are reliable, scalable indicators for ranking autonomous agents on subjective enterprise tasks.

Human preference evaluation. To validate LLM-based rankings with direct human judgment, a domain expert performed 135 pairwise preference evaluations on Figma-to-code outputs across 31 tasks, comparing agent-built UIs side-by-side without access to LLM scores or agent identity (Table 9). Bradley-Terry Elo rankings [2] place Codex and Claude Code in a statistically indistinguishable top tier (p=0.67, Cohen's h=0.08), with both significantly preferred over Gemini CLI (p=0.036 and p=0.047 respectively). Position bias is absent (52% A-rate, binomial p=0.72), and preferences are near-perfectly transitive (4.2% cycle rate). Winner quality ratings differ significantly across agents (Kruskal-Wallis p=0.048): Codex victories are rated higher (3.88/5) than Claude Code (3.40/5) or Gemini CLI (3.35/5), suggesting Codex produces more variable but higher-peak outputs.

Expert pass/fail classification. The domain expert assigns quality ratings on a 5-point scale: ≤3 = fail, 4 = design system well-defined but assets and fine-grained spacing require one engineering sprint, 5 = production-ready. Applying this threshold to winner ratings, overall 60% of winning outputs pass (≥4). Pass rates vary substantially by model: Codex GPT-5.2 Pro leads at 77.8% (21/27), followed by Codex GPT-5.2 at 57.9% (33/57), Claude Code at 54.5% (24/44), and Gemini CLI at 52.4% (11/21). The separation between GPT-5.2 Pro and the rest suggests that when GPT-5.2 Pro wins a matchup, it produces meaningfully higher-quality output—consistent with the higher winner ratings reported above.

Table 9: Human pairwise preference (Figma-to-code): Bradley-Terry Elo from 135 votes by 1 domain expert. The top two harnesses are statistically indistinguishable (p=0.67, h=0.08); both significantly outperform Gemini CLI (p<0.05).

Agent harness	Elo	95% CI	Win %	n_{wins}
Codex (GPT-5.2)	1054	[1005, 1114]	56.8	72
Claude Code (Opus 4.6)	1039	[987, 1093]	55.1	43
Gemini CLI (3.1 Pro)	907	[842, 965]	31.1	21

Table 10: Programmatic content: pairwise win rates from human SME preferences vs. VLM-derived outcomes (n=275 matched chapter-level comparisons).

Agent	Human WR	VLM WR (3-pt)	VLM WR (5-pt)
Claude Code (Opus 4.6)	81.5%	66.8%	69.0%
Codex (GPT-5.2)	34.2%	33.9%	33.1%
Gemini CLI (3.1 Pro)	34.2%	49.2%	47.8%

At the *individual run* level, human and LLM judges show weak concordance (κ=0.08 output, 0.06 skill). At the *aggregate* level, both agree on the primary ranking boundary, but LLM judges nominally separate the top two where human preferences do not (Table 9), cautioning against interpreting fine-grained LLM score gaps as perceptible quality differences.

Programmatic content. For programmatic content, we evaluate VLM judge reliability through human–VLM alignment rather than multi-judge agreement, since a single VLM judge (Gemini 3.1 Pro) scores artifacts against chapter-specific rubrics. Human–VLM pairwise agreement is 52.0% with Cohen's $\kappa = 0.082$ under 5-point rubrics (n=275), with system-level concordance on the top-ranked harness.

Human–VLM alignment (programmatic content). To validate VLM-as-judge scoring against human expert preferences, we compare pairwise outcomes from VLM artifact scores against human SME pairwise preferences on n=275 matched chapter-level comparisons. The human preference interface allows both strict preference and explicit ties. Table 10 reports win rates.

Both human and VLM judges agree that Claude Code is the top-ranked harness (human WR 81.5%, VLM WR 69.0%). Codex win rates are closely aligned between human and VLM (34.2% vs. 33.1%). The primary divergence is at position 2: humans rate Codex and Gemini equally (both 34.2%), while the VLM favors Gemini (47.8% vs. 33.1%), likely reflecting VLM sensitivity to production-quality dimensions where Gemini performs competitively, whereas human experts weight content accuracy and pedagogical effectiveness more heavily.

Raw pairwise agreement is 46.5% (3-point) and 52.0% (5-point); Cohen's κ is 0.073 and 0.082 respectively. Notably, κ *improves* with the 5-point scale, driven by a reduction in tie asymmetry: the VLM produces 74 ties under 3-point rubrics but only 38 under 5-point, closely matching the 36 human ties. The 5-point scale also yields the largest agreement gains on the hardest comparisons (Codex vs. Gemini: 41.8% → 49.5%; Claude vs. Gemini: 40.2% → 48.9%), while

Table 11: Failure taxonomy across all Figma-to-code runs (n=96). Recovery rate is the fraction of errors from which the agent successfully self-corrected.

Error type	Count	Recovered	Recovery %
Tool call failure	419	278	66.3
Git error	64	48	75.0
Syntax error	33	30	90.9
Dependency error	28	22	78.6
Preview deny	18	16	88.9
Build error	11	11	100.0
Type error	7	6	85.7
Config error	6	1	16.7
Runtime error	4	3	75.0
Total	**590**	**415**	**70.3**

Table 12: Per-agent recovery summary (Figma-to-code). Failure rate = failed tool calls / total tool calls. Recovery rate = errors recovered / total errors.

Agent	Runs	Errors	Failure %	Recovery %	Preview	Deploy	Tool calls
Claude Code (Opus 4.6)	32	152	7.3	71.1	30/32	27/32	2692
Codex (GPT-5.2)	34	273	9.4	74.0	5/34	21/34	3339
Gemini CLI (3.1 Pro)	30	165	8.4	63.6	30/30	30/30	2092

the easy comparison (Claude vs. Codex) remains stable at 57.6%. System-level concordance on the top-ranked harness, combined with directional alignment on win rates and improved κ under finer granularity, validates the VLM-as-judge approach for subjective multimedia evaluation. Full per-pair breakdowns are in Appendix J.

8.7 Failure Taxonomy and Test-Time Recovery

We extract structured error→recovery events from all 96 Figma-to-code agent transcripts using an LLM-based pipeline (Tables 11, 12).

Tool call failures dominate; error message quality determines recoverability. Tool call failures account for 71% of all errors (419/590), including MCP timeouts, file-not-found errors, and permission denials (Figure 3). Harness-specific patterns emerge: Codex concentrates 78% of git errors and 79% of dependency errors, driven by a recurring cascade where `.git/index.lock` files and unresolved version conflicts compound across tool invocations. The sharpest diagnostic signal is recoverability by feedback type: errors with structured compiler output (syntax, type, build) yield >85% recovery, while ambiguous signals (configuration errors) yield only 17% (Table 11, Figure 4).

Agents recover from 70% of errors, with distinct correction strategies. Across 590 errors, agents self-correct 70.3% of the time (Table 12), demonstrating that test-time verification feedback enables meaningful self-correction [12]. The three harnesses exhibit distinct profiles: Claude Code is *proactive*—94% preview usage, iterating on structured feedback before proceeding; Codex is *reactive*—encountering the most errors per run yet achieving the highest recovery rate (74%); Gemini CLI is *persistent*—100% deploy rate across all runs, pushing through to deployment even with unresolved errors.

9 Conclusion

Binary benchmarks cannot capture the idiosyncratic quality criteria of enterprise work. LH-Bench demonstrates that stepwise verifiers and expert-grounded evaluation solve this problem.

Across two environments—application development lifecycle (Figma-to-code) and programmatic content generation from enterprise knowledge—we show that LH-Bench, our three-pillar evaluation design, produces reliable, diagnostic signals for grading autonomous agents on subjective tasks. Three properties make this design viable at scale:

Scalable: expert-authored rubrics, curated ground-truth artifacts, and workflow-specific `SKILL.md` references encode the domain knowledge that LLM judges need to simulate expert evaluation, eliminating per-task dependence on human judgment. Expert-authored rubrics yield substantially higher inter-judge agreement than LLM-authored rubrics (κ=0.60 vs. 0.46), validating this encoding.

Reusable: the environment infrastructure accepts new task configurations—Figma design files, data-room document collections—and evaluates new agents through the same multi-harness pipeline without re-engineering the evaluation.

Generalizable: the three pillars converge across both environments and all three harness families: independent evaluation tiers agree on the same ranking boundaries, and rubric-level decomposition reveals compensatory skill profiles invisible to aggregate scores. Our recovery analysis further shows that error message quality, not error frequency, determines self-correction—underscoring that evaluation must look beyond final outputs to the execution process itself.

LH-Bench is a reusable environment that enables enterprises to implement multi-harness agent runners and a proven three-pillar evaluation design to reliably grade new agents on new task configurations.

10 Limitations and Future Work

(1) Findings are limited to two environments. We aim to secure additional compute and domain-expert budget to scale LH-Bench in environment diversity and task complexity in future work. (2) We evaluate Codex CLI rather than the OpenAI Agent SDK, which may support different orchestration patterns. (3) All three harnesses are tightly coupled to their model families. An open-source, model-agnostic harness would enable controlled experiments decoupling model capability from harness orchestration. We release public datasets for both environments.

References

[1] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment. *arXiv preprint arXiv:2506.07982* (2025).

[2] Ralph Allan Bradley and Milton E. Terry. 1952. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika* 39, 3/4 (1952), 324–345.

[3] Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. 2024. Humans or LLMs as the Judge? A Study on Judgement Biases. *arXiv preprint arXiv:2402.10669* (2024).

[4] Jiaju Chen, Yuxuan Lu, Xiaojie Wang, Huimin Zeng, Jing Huang, Jiri Gesi, Ying Xu, Bingsheng Yao, and Dakuo Wang. 2025. Multi-Agent-as-Judge: Aligning LLM-Agent-Based Automated Evaluation with Multi-Dimensional Human Evaluation. In *NeurIPS Workshop on Multi-Turn Interactions.*

[5] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46.

[6] Xiang Deng, Jeff Da, Edwin Pan, et al. 2025. SWE-Bench Pro: Can AI Agents Solve Long-Horizon Software Engineering Tasks? *arXiv preprint arXiv:2509.16941* (2025).

[7] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. WorkArena: How Capable Are Web Agents at Solving Common Knowledge Work Tasks?. In *International Conference on Machine Learning (ICML)*.

[8] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2024. A Survey on LLM-as-a-Judge. *arXiv preprint arXiv:2411.15594* (2024).

[9] Shengyue Guan, Jindong Wang, Jiang Bian, Bin Zhu, Jian-Guang Lou, and Haoyi Xiong. 2025. Evaluating LLM-based Agents for Multi-Turn Conversations: A Survey. *arXiv preprint arXiv:2503.22458* (2025).

[10] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *International Conference on Learning Representations (ICLR)*.

[11] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

[12] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, et al. 2025. Training Language Models to Self-Correct via Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*.

[13] Thomas Kwa, Ben West, Joel Becker, et al. 2025. Measuring AI Ability to Complete Long Tasks. https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/.

[14] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174.

[15] Ruizhe Li, Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. 2026. DeepResearch Bench II: Diagnosing Deep Research Agents via Rubrics from Expert Report. *arXiv preprint arXiv:2601.08536* (2026).

[16] Xiangyi Li et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670* (2026).

[17] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2024. AgentBench: Evaluating LLMs as Agents. In *International Conference on Learning Representations (ICLR)*.

[18] Haotian Luo, Huaisong Zhang, Xuelin Zhang, Haoyu Wang, Zeyu Qin, Wenjie Lu, Guozheng Ma, Haiying He, Yingsha Xie, Qiyang Zhou, Zixuan Hu, Hongze Mi, Yibo Wang, Naiqiang Tan, Hong Chen, Yi R. Fung, Chun Yuan, and Li Shen. 2025. UltraHorizon: Benchmarking Agent Capabilities in Ultra Long-Horizon Scenarios. *arXiv preprint arXiv:2509.21766* (2025).

[19] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: A Benchmark for General AI Assistants. In *International Conference on Learning Representations (ICLR)*.

[20] Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. 2025. Evaluation and Benchmarking of LLM Agents: A Survey. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[21] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations (ICLR)*.

[22] Manasi Sharma, Chen Bo Calvin Zhang, Chaithanya Bandi, Clinton Wang, Ankit Aich, Huy Nghiem, Tahseen Rabbani, Ye Htet, Brian Jang, Sumana Basu, Aishwarya Balwani, Denis Peskoff, Marcos Ayestaran, Sean M. Hendryx, Brad Kenstler, and Bing Liu. 2025. ResearchRubrics: A Benchmark of Prompts and Rubrics For Evaluating Deep Research Agents. *arXiv preprint arXiv:2511.07685* (2025).

[23] Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2025. Design2Code: Benchmarking Multimodal Code Generation for Automated Front-End Engineering. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

[24] Haoyu Sun, Huichen Will Wang, Jiawei Gu, Linjie Li, and Yu Cheng. 2025. FullFront: Benchmarking MLLMs Across the Full Front-End Engineering Workflow. *arXiv preprint arXiv:2505.17399* (2025).

[25] Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. 2024. Judging the Judges: Evaluating Alignment and Vulnerabilities in LLMs-as-Judges. *arXiv preprint arXiv:2406.12624* (2024).

[26] Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024. MINT: Evaluating LLMs in Multi-turn Interaction with Tools and Language Feedback. In *International Conference on Learning Representations (ICLR)*.

[27] Quan Wei, Siliang Zeng, Chenliang Li, William Brown, Oana Frunza, Wei Deng, Anderson Schneider, Yuriy Nevmyvaka, Yang Katie Zhao, Alfredo Garcia, and Mingyi Hong. 2025. Reinforcing Multi-Turn Reasoning in LLM Agents via Turn-Level Reward Design and Credit Assignment. In *NeurIPS Workshop on Multi-Turn Interactions*.

[28] Xueqing Wu, Zihan Xue, Da Yin, Shuyan Zhou, Kai-Wei Chang, Nanyun Peng, and Yeming Wen. 2026. FronTalk: Benchmarking Front-End Development as Conversational Code Generation with Multi-Modal Feedback. *arXiv preprint arXiv:2601.04203* (2026).

[29] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[30] Shunyu Yao, Noah Shinn, Karthik Narasimhan, et al. 2024. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint arXiv:2406.12045* (2024).

[31] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*.

[32] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. 2024. The Shift from Models to Compound AI Systems. Berkeley Artificial Intelligence Research Blog. Accessed: 2026.

[33] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[34] 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. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *International Conference on Learning Representations (ICLR)*.

[35] Hongda Zhu, Yiwen Zhang, Bing Zhao, Jingzhe Ding, Siyao Liu, Tong Liu, Dandan Wang, Yanan Liu, and Zhaojian Li. 2025. FrontendBench: A Benchmark for Evaluating LLMs on Front-End Development via Automatic Evaluation. *arXiv preprint arXiv:2506.13832* (2025).

Appendix

A Pipeline Diagram

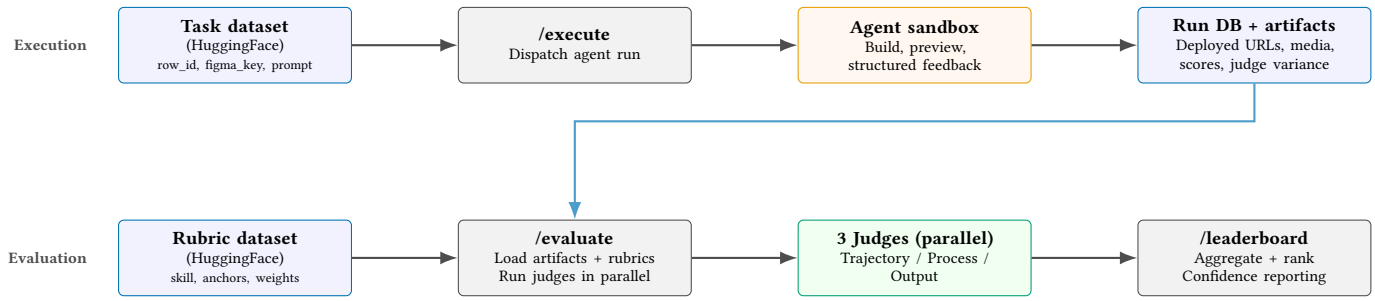

Figure 1: LH-Bench execution and evaluation pipeline. Tasks and rubrics are versioned independently in HuggingFace. Agent runs produce persistent artifacts, which are graded by three judges in parallel; results flow into leaderboards.

B SME Annotation Tool

Figure 2 shows the SME annotation interface used for programmatic content ground-truth construction. The tool presents three coordinated panels:

Source panel (left). Lists all documents in the task's data room (e.g., arXiv PDFs, web pages) with type badges. SMEs click a source to load it in the center panel.

Document viewer (center). Renders the selected source as line-numbered markdown. Each section is collapsible and displays its line range (e.g., `L1–9`, `L10–13`). SMEs can select text spans inline using a "highlight-to-cite" interaction: selecting lines attaches the span (with source ID and line numbers) to the active chapter, producing structured citations of the form `source_id:start_line–end_line`.

Chapter panel (right). SMEs define chapters—the units of content the agent must produce—and attach source spans to each chapter. Each chapter includes a title, ordering, and the set of cited spans from the source panel. A "Global notes" field captures high-level design reasoning (e.g., narrative arc, emphasis priorities) that applies across all chapters.

This three-panel design enables SMEs to build granular, source-grounded annotations efficiently: the annotator reads a source, highlights relevant passages, and assigns them to chapters in a single workflow, rather than context-switching between separate tools. The resulting annotations power rubric synthesis for VLM-based artifact scoring (Section 8.3) and encode the 5-skill design rubric (content selection, narrative structure, visual hierarchy, information density, source grounding).

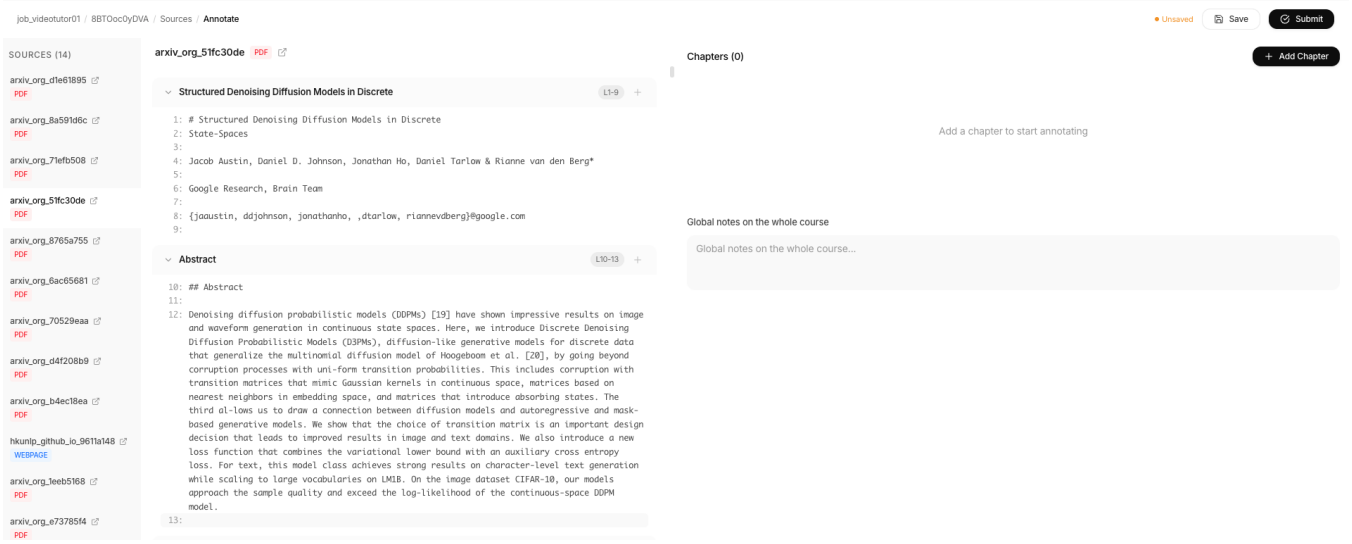

Figure 2: SME annotation interface for programmatic content. Left: source documents in the data room. Center: line-numbered document viewer with collapsible sections and highlight-to-cite interaction. Right: chapter definitions with attached source spans and global design notes.

C Ground Truth Schema Examples

Figma-to-code manifest. (see Section 6.3):

```
{
  "figma_file_key": "oSNDllo...8YMSlD",
  "total_frames": 5,
  "frames": [
    { "name": "PDP",
      "node_id": "2176:167104",
      "gt_image": "2176-167104.png",
      "target_route": "/pdp" },
    { "name": "PLP_Category",
      "node_id": "2176:167875",
      "gt_image": "2176-167875.png",
      "target_route": "/" }
  ]
}
```

Programmatic content annotation. (see Section 6.3):

```
{
  "task_id": "video-tutor-042",
  "context_gt": [
    { "source_id": "arxiv-2301.00234",
      "start_line": 42, "end_line": 58,
      "quote": "Attention is computed as..." }
  ],
  "rubrics": [
    { "criterion": "source_grounding",
      "scale": "0/1/2",
      "anchor_2": "All claims cited" }
  ]
}
```

D Harness Specifications

This appendix provides formal descriptions of the agent harnesses, tool interface, skill injection mechanism, and session recovery protocol used in LH-Bench.

D.1 Harness Descriptions

We evaluate three commercial agent harnesses. Each tightly couples specific models with proprietary agent logic, representing real-world deployment conditions:

- **Claude Code** (Anthropic): Anthropic's coding agent with native Skill integration. Uses the Claude Agent SDK with subprocess CLI execution, MCP server support via `McpStdioServerConfig`, automatic context compaction, and session resumption. Operates in permission-bypass mode (`acceptEdits`) for autonomous execution.
- **Codex CLI** (OpenAI): OpenAI's lightweight coding agent. MCP servers are registered via a global registry (`codex mcp add`). Runs in `dangerously-bypass-approvals-and-sandbox` mode for full autonomy. Outputs structured JSONL event streams with transcript archival.
- **Gemini CLI** (Google): Google's open-source terminal agent. MCP servers are auto-discovered from a per-project `.gemini/settings.json` configuration. Uses Vertex AI authentication with wrapper scripts for environment variable isolation. Runs in trust mode (`-yolo`) for auto-approval.

Model family consideration. Claude models have been trained with awareness of the Agent Skills specification, which may confer advantages when processing Skill-formatted instructions. Codex and Gemini harnesses receive equivalent skill content but through their respective native mechanisms.

D.2 Tool Interface

All agents interact with environments through the Model Context Protocol (MCP). Each MCP server exposes domain-specific tools following a standardized request/response interface:

Figma-to-code tool categories.

- **Design extraction** (Figma MCP): `get_figma_file`, `get_node_tree`, `export_node_images`—retrieve design structure, component hierarchy, and rasterized assets.
- **File and shell** (built-in): Read, Write, Edit, Glob, Grep, Bash—standard file manipulation and command execution within the sandbox.
- **Preview verification** (App Preview MCP): `create_app_preview`, `get_preview_status`—build the agent's code in an ephemeral container and return structured error diagnostics (runtime exceptions, blank-page detection) or a live URL.
- **Deployment** (GCS MCP): `upload_dist_to_gcs`—publish finalized static builds for artifact evaluation and human inspection.
- **Browser automation** (Playwright MCP): `browser_navigate`, `browser_take_screenshot`—exercise deployed UIs and capture frame-level screenshots for visual verification.

D.3 Skill Injection Mechanism

`SKILL.md` files encode expert-authored procedural knowledge as structured Markdown with YAML frontmatter:

```
---
name: figma-to-code
description: Convert Figma designs to
  production-ready frontend code.
---
# Step 0: Check manifest for prior progress
# Step 1: Extract design structure via Figma MCP
# Step 2: Export assets and ground truth frames
...
```

Per-harness loading. Each harness discovers and loads Skills using its native mechanism:

- **Claude Code**: `setting_sources=["project"]` triggers scanning of `.claude/skills/*/SKILL.md` in the project root.
- **Gemini CLI**: Skills are placed in `.gemini/skills/*/SKILL.md` and loaded via file-based configuration in `settings.json`.
- **Codex CLI**: Skill content is inlined into the system prompt, as Codex lacks a native skill-discovery mechanism.

D.4 Manifest-Based Session Recovery

Agents maintain a flat `manifest.json` at the project root to track execution progress:

```
{
  "preview_url": "https://...",
  "deployed_url": "https://...",
```

```
  "completed_steps": [
    "Step 1: Extract from Figma",
    "Step 2: Export assets"
  ],
  "updated_at": "2026-01-31T..."
}
```

Step 0 of every `SKILL.md` requires reading the manifest to check prior progress, enabling session recovery across agent restarts without re-executing completed work. This is critical for long-horizon tasks where context limits or transient failures may interrupt a session.

D.5 Containerization and Deployment

Agent runs execute in sandboxed containers deployed via Modal (serverless). Each model configuration receives a dedicated persistent volume (e.g., `/figma-claude-opus`, `/figma-codex-52`, `/figma-gemini-31-pro`) for project state isolation. The container image includes Python 3.11, Node.js 20, and all agent CLI binaries. MCP tool servers run as co-processes within the same container, with credentials injected via `modal.Secret` at runtime.

E Skill Rubric Definitions (v2.1)

Table 13 presents the four expert-authored process rubrics used for Figma-to-code skill evaluation (v2.1). Each rubric uses a 1–5 anchored scale with observable transcript evidence. The rubrics are designed around sequential workflow phases with binary-observable boundaries between score levels.

Table 13: Figma-to-code process rubrics (v2.1, expert-authored). Weight indicates contribution to the aggregate skill score. Anchors summarize the 1–5 scale boundaries.

Rubric	Wt.	What it measures	Key boundary (3→4)
Design Inspection & Asset Extraction	0.30	Extent of Figma file inspection, asset export completeness, format correctness, multi-page navigation discovery	3 = all assets exported in correct formats; 4 = also organized (semantic filenames, directory structure, hierarchy inspection before coding)
Design Token & Style Extraction	0.25	Extraction and centralization of colors, typography, spacing, shadows, borders into a token/theme file	3 = token file covers 4+ of 6 categories, tokens referenced in code; 4 = also created before components, semantic naming, zero hard-coded leaks
Component & Layout Architecture	0.25	Component decomposition, pattern reuse, Auto Layout→CSS mapping, variant/state handling	3 = repeated patterns identified, layout correct, hover states; 4 = also planned upfront (visible in transcript), full variant coverage, props interface
Build Verification & Iteration	0.20	Build/preview execution, error diagnosis and fix iteration, visual verification against Figma design	3 = build compiles successfully; 4 = also opened preview AND made at least one fix based on visual inspection

Each scale point includes concrete transcript indicators. For example, at score 5 ("Production-grade") on Design Inspection, the agent uses WebP for photos, applies @2x scale factors, deduplicates repeated assets, converts SVGs to components, and maps the full navigation topology before coding. The full rubric specification with all 5 anchor levels per rubric is available in the repository at `verifiers/figma-to-code/process_rubrics.json`.

F Rubric Version Comparison (v1.1 vs v1.2)

Table 14 compares the two rubric versions used in the inter-judge agreement analysis (Section 8.6).

v1.1 (LLM-authored, 8 rubrics). Generated by prompting an LLM to produce evaluation criteria for Figma-to-code agent trajectories. The rubrics cover fine-grained workflow steps with unequal weights (0.07–0.20) and use generic proficiency anchors ("Inadequate" through "Expert") without binary-observable boundaries.

Table 14: Rubric version comparison. v1.1 uses 8 generic LLM-authored rubrics; v1.2 uses 4 domain-specific expert-authored rubrics with anchored scales.

	v1.1 (LLM-authored)	**v1.2 (Expert-authored)**
Rubric count	8	4
Scope	Figma-to-code (fine-grained)	Figma-to-code (workflow phases)
Anchor style	Generic ("Inadequate"–"Expert")	Observable (transcript evidence)
Boundary type	Subjective	Binary-observable
v1.1 rubrics (non-equal weight):	Design file inspection (0.07), Image asset extraction (0.20), Icon/vector extraction (0.15), Design token discovery (0.15), Component pattern recognition (0.10), Variant/state analysis (0.12), Layout structure analysis (0.13), Build verification (0.08)	
v1.2 rubrics (weighted):	Design inspection (0.30), Token extraction (0.25), Component architecture (0.25), Build verification (0.20)	
Mean pairwise κ	0.46	0.60
Mean variance	0.25	0.10

v1.2 (Expert-authored, 4 rubrics). Designed by domain experts with Figma-to-code workflow knowledge. Each rubric maps to a sequential workflow phase, uses binary-observable boundaries between score levels (e.g., "token file created before components" is verifiable in the transcript), and includes specific transcript evidence patterns.

The key design insight is that **domain-specific rubrics with binary-observable boundaries reduce scoring ambiguity**. For example, "Did the agent create a token file before writing components?" is unambiguously verifiable from the transcript, whereas "Did the agent demonstrate good planning?" requires subjective interpretation that varies across judges. This is reflected in the +0.15 kappa improvement (Table 8).

G Output Tier Rubric Weights

Table 15 lists the eight artifact rubrics used for the Figma-to-code output tier (Tier 1). Each rubric is scored on a 1–5 scale by a VLM judge (Gemini 3) comparing Playwright-captured screenshots against expert ground-truth frame images.

Table 15: Figma-to-code output tier rubric weights.

Rubric	**Weight**	**What it measures**
Component coverage	0.20	Percentage of design components rendered
Layout accuracy	0.18	Spatial positioning/sizing and flex/grid correctness
Colors accuracy	0.14	Palette fidelity (fills, gradients, borders)
Typography accuracy	0.12	Font family/size/weight/line-height match
Asset display	0.10	Images/icons/vector assets render correctly
Visual fidelity	0.10	Overall visual similarity
Responsive behavior	0.08	Adapts to multiple viewports
Interaction fidelity	0.08	Hover/active/disabled states

H Recovery Analysis Figures

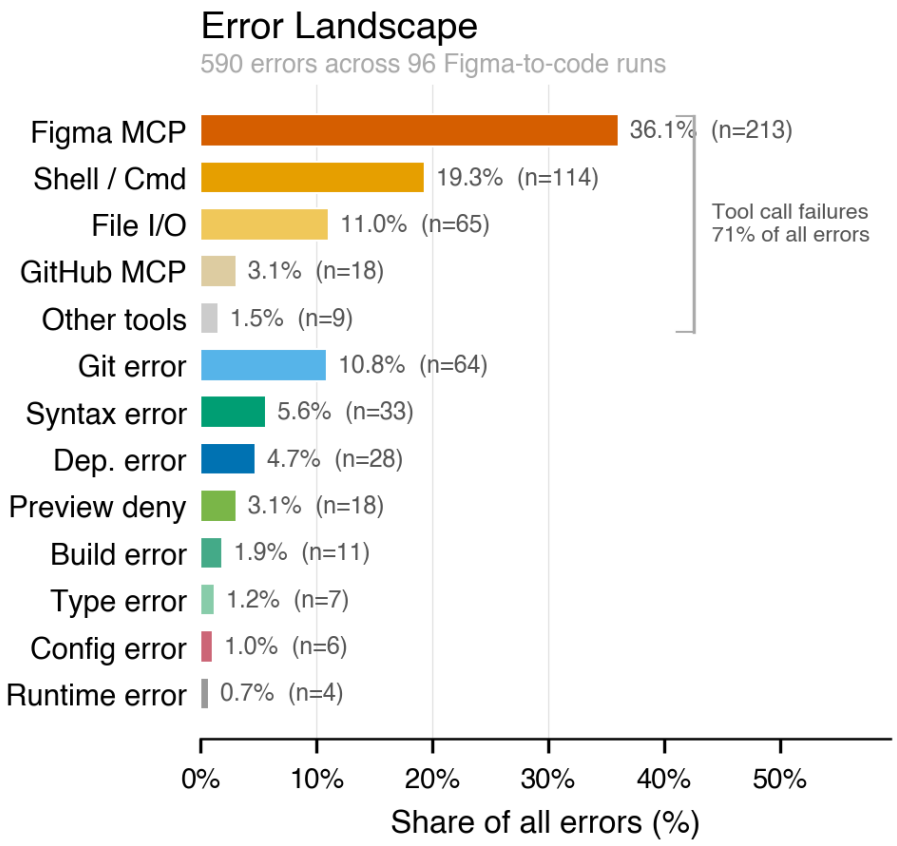

Figure 3: Error landscape across 96 Figma-to-code runs (590 total errors). Tool call failures account for 71% of all errors; within these, Figma MCP operations are the dominant source (51%), reflecting the difficulty of reliably invoking design-extraction APIs at scale.

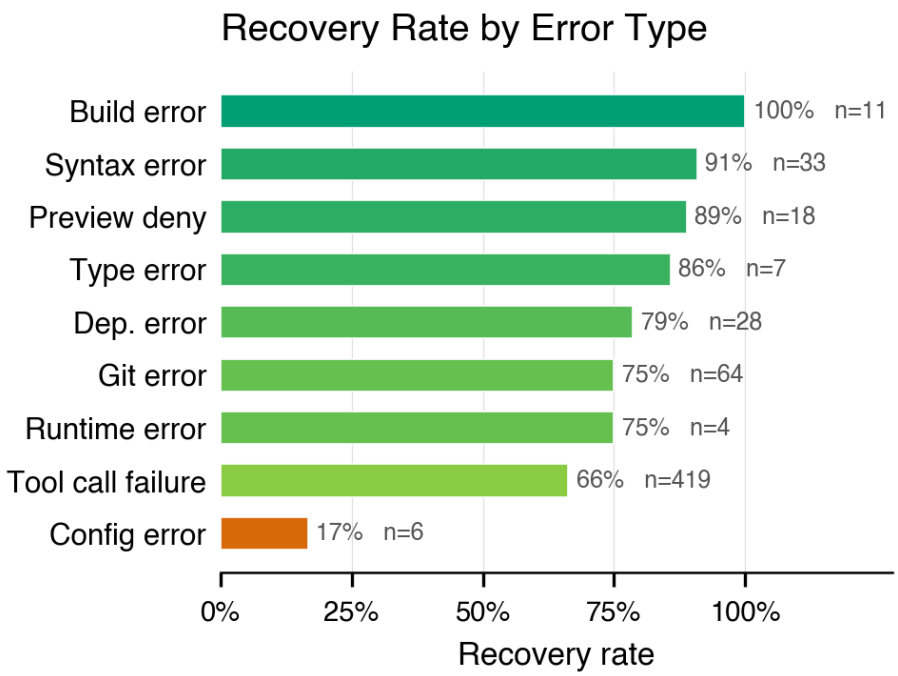

Figure 4: Recovery rates by error type. Structured compiler feedback (syntax, type, build errors) yields >85% recovery; ambiguous signals (configuration errors) yield only 17%.

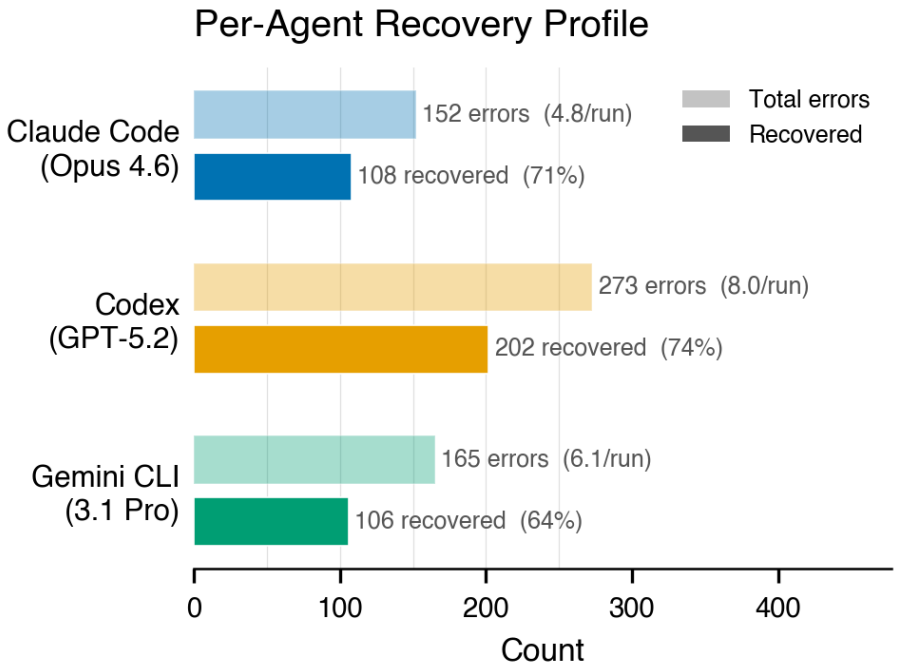

Figure 5: Per-agent recovery profiles. Codex encounters the most errors (8.0/run) yet achieves the highest recovery rate (74%); Claude Code encounters the fewest (4.8/run) with 71% recovery; Gemini CLI recovers 64% with 100% deploy completion.

I Programmatic Content Evaluation Rubrics

Each chapter in the programmatic content environment is graded against five generic SME-authored rubrics plus chapter-specific criteria synthesized from annotator evaluation notes. The generic rubrics are applied uniformly across all chapters; chapter-specific rubrics are LLM-synthesized from per-chapter annotator notes to capture task-specific quality dimensions (e.g., "correctly illustrates the attention mechanism" for a chapter on transformers). Below we present the five generic rubrics with their full 5-point scale definitions.

I.1 Generic Rubrics (5-point scale)

1. Content Relevance and Clarity. Evaluates whether the video content accurately addresses the chapter instruction and presents information in a clear, logically structured manner.

Score	Description
1	Content is largely irrelevant or incoherent; fails to address the chapter instruction.
2	Addresses the topic but with major gaps, inaccuracies, or disorganized presentation.
3	Covers core points adequately; minor gaps or unclear transitions but generally on-topic.
4	Clear, well-organized content that addresses all key aspects of the instruction with minor omissions.
5	Comprehensive, precisely targeted content with logical flow; every segment directly serves the chapter objective.

2. Visual Design and Production Quality. Evaluates the aesthetic quality, consistency, and professionalism of visual elements including typography, color, layout, and animations.

Score	Description
1	Visuals are broken, missing, or unreadable; severe rendering artifacts.
2	Functional but amateurish; inconsistent styling, poor contrast, or cluttered layouts.
3	Acceptable visual quality; consistent styling with minor polish issues.
4	Professional appearance; cohesive color palette, clean typography, smooth animations.
5	Exceptional production quality; polished transitions, purposeful motion design, broadcast-quality aesthetics.

3. Pedagogical Effectiveness. Evaluates how well the video teaches the intended concept, including pacing, scaffolding, and use of examples.

Score	Description
1	No discernible teaching structure; concepts presented without context or progression.
2	Attempts to explain but lacks scaffolding; jumps between concepts without bridging.
3	Reasonable pedagogical flow; builds on prior context with adequate pacing.
4	Effective teaching with clear scaffolding, well-timed examples, and concept reinforcement.
5	Exemplary pedagogy; progressive disclosure, concrete-to-abstract scaffolding, retrieval cues, and anticipation of learner misconceptions.

4. Audio–Visual Synchronization. Evaluates alignment between narration and on-screen visuals, including timing of transitions, text highlights, and animation triggers.

Score	Description
1	Severe desynchronization; narration and visuals are unrelated or offset by multiple seconds.
2	Noticeable timing mismatches; visuals often appear before or after the relevant narration.
3	Generally synchronized; occasional minor offsets that do not impede comprehension.
4	Well-synchronized; visuals reinforce narration with consistent timing.
5	Precise synchronization; animations trigger at exactly the right narration cue, enhancing comprehension through temporal alignment.

5. Technical Accuracy of Visualizations. Evaluates the correctness of diagrams, equations, code snippets, and data representations shown in the video.

Score	Description
1	Visualizations contain fundamental errors (wrong equations, incorrect diagrams, fabricated data).
2	Partially correct but with significant errors that could mislead learners.
3	Mostly correct; minor inaccuracies that do not alter the core message.
4	Accurate visualizations with proper notation, correct relationships, and faithful data representation.
5	Technically impeccable; visualizations are precise, properly labeled, and include appropriate caveats or simplification notes where relevant.

I.2 Chapter-Specific Rubrics

In addition to the five generic rubrics, each chapter receives 1–3 chapter-specific rubrics synthesized by an LLM from the human annotator's evaluation criteria for that chapter. For example, a chapter on "GRPO vs. PPO gradient flow" might receive a rubric for "Gradient diagram correctness: does the visualization accurately show the policy gradient computation path for both algorithms?" These chapter-specific rubrics use the same scale structure (either 3-point or 5-point) as the generic rubrics and are scored by the same VLM judge.

I.3 3-Point vs. 5-Point Scale Comparison

We evaluate all rubrics under both granularities. The 3-point scale uses levels 0 (absent/incorrect), 1 (partially correct), and 2 (fully correct). The 5-point scale provides finer discrimination as shown above. Section 8.3 reports aggregate results under both scales.

I.4 Example VLM Judge Prompt (5-Point Scale)

Below is an abbreviated example of the prompt sent to the VLM judge (Gemini 3.1 Pro) for a single chapter evaluation. The prompt includes: (1) a system preamble enforcing strict, evidence-based scoring; (2) course context and outline; (3) the chapter instruction; (4) the full set of rubrics (5 generic + 2 chapter-specific in this example); and (5) structured output instructions. We show the system preamble, two representative rubrics (one generic, one chapter-specific), and the output format. The full prompt for all rubrics follows the same pattern.

```
You are a strict, impartial evaluator of
AI-generated educational videos. Score only what
you observe in the video, not what it could have
been. Be critical. Reserve top scores for
genuinely exceptional work.

## Course Context
[Course description and outline omitted for space]

## Chapter Instruction
> How do Denoising Score Matching with Langevin
> Dynamics (SMLD) and DDPM learn and sample from
> complex data distributions, and why are they
> fundamentally equivalent under a score-based
> perspective?

## Rubrics

### 1. Content Relevance and Clarity
Evaluation: Does the video stay focused on
explaining how SMLD and DDPM learn/sample, and
why they are equivalent under score-based
perspective?

Scale:
  1: Mostly off-topic or incoherent; fails to
     address how SMLD/DDPM learn and sample;
     major factual errors.
  2: Touches on diffusion but with major gaps;
     equivalence missing or asserted without
     explanation.
  3: Adequate overview of SMLD and DDPM with
     minor gaps; equivalence mentioned but not
     strongly supported.
  4: Clear and well-structured; explicitly
     explains equivalence under unified
     score-based view; negligible omissions.
  5: Exceptionally focused; rigorously and
     intuitively explains SMLD-DDPM equivalence;
     consistent notation; resolves common
     confusions.

[... 4 more generic rubrics: Visual Design,
 Pedagogical Effectiveness, Audio-Visual Sync,
 Technical Accuracy ...]
```

```
### 6. Chapter Mastery: Understanding SMLD
Evaluation: How well does the video explain SMLD
as learning scores via denoising score matching
across noise levels and sampling via Langevin
dynamics?

Scale:
  1: Mentions SMLD without meaningful
     explanation; sampling missing or incorrect.
  2: High-level description but vague about noise
     levels or sampling mechanism.
  3: Explains denoising score matching and noise
     levels; basic Langevin sampling idea but
     omits key details.
  4: Clearly explains score matching across noise
     levels and annealed Langevin dynamics with
     correct update structure.
  5: Crisp end-to-end account: forward
     perturbation, score learning, principled
     sampling via annealed Langevin dynamics;
     addresses typical pitfalls.

### 7. Chapter Mastery: DDPM and Score-Based
###    Equivalence to SMLD
[Similar 5-point scale structure]

## Instructions
For each rubric:
1. Note specific evidence (timestamps, visual
   elements, narration) relevant to that rubric.
2. Match observations against each scale level.
3. Assign the integer score. Do NOT interpolate.
4. Cite specific evidence in thinking_process.

Return JSON only:
{"rubric_scores": [
  {"rubric_name": "...",
   "score": "<integer>",
   "matched_level": "<scale description>",
   "thinking_process": "Specific evidence: ..."}
]}
```

The chapter-specific rubrics (rubrics 6–7 in this example) are synthesized from human annotator evaluation criteria for each chapter, ensuring that the VLM judge evaluates both generic production quality and chapter-specific conceptual mastery. The `design_context` field (omitted above) additionally instructs the judge on expected visual structure—e.g., "use parallel, side-by-side layout to emphasize SMLD–DDPM equivalence."

J Human–VLM Alignment Details

This appendix provides per-pair breakdowns of human–VLM agreement for the programmatic content environment, complementing the aggregate results in Section 8.6.

J.1 Per-Pair Agreement and Cohen's κ

Table 16 reports agreement and κ for each agent pair under both rubric granularities. Agreement is the fraction of comparisons where the human and VLM select the same winner (or both tie); κ is Cohen's kappa computed over the 3-class outcome (A wins, B wins, tie).

J.2 Tie Asymmetry and κ Interpretation

Under the 3-point rubric, a substantial tie asymmetry exists between human and VLM raters:

- **Humans tie infrequently.** Across 275 comparisons, human SMEs produce 36 ties (13.1%). The annotation interface supports both strict preference and explicit ties, but annotators overwhelmingly express directional preferences.

Table 16: Human–VLM agreement by agent pair (n = matched chapter-level pairwise comparisons).

Pair	n	Agree (3-pt)	κ (3-pt)	Agree (5-pt)	κ (5-pt)
Claude vs. Codex	92	57.6%	0.121	57.6%	0.044
Claude vs. Gemini	92	40.2%	−0.004	48.9%	0.024
Codex vs. Gemini	91	41.8%	0.102	49.5%	0.180
Overall	**275**	**46.5%**	**0.073**	**52.0%**	**0.082**

- **The VLM ties frequently under coarse scales.** Under the 3-point rubric, the VLM produces 74 ties (26.9% of comparisons). Under the 5-point rubric, this drops to 38 ties (13.8%)—a 49% reduction that brings VLM tie behavior in line with the human tie rate (13.1%).

Cohen's κ is computed over a 3-class outcome space (A wins, B wins, tie). When the two raters have markedly different tie rates, κ is deflated regardless of whether they agree on the *direction* of non-tie outcomes. This explains why κ improves from 0.073 (3-point) to 0.082 (5-point): the 5-point scale eliminates the structural tie mismatch, allowing κ to better reflect genuine agreement on directional preferences.

Why win rates complement κ here. The aggregate win rates (Table 10) show strong directional alignment: both human and VLM judges agree that Claude Code is the top-ranked harness (human WR 81.5%, VLM WR 69.0%). Codex win rates are closely aligned between human and VLM (34.2% vs. 33.1%). The primary divergence—Gemini's VLM win rate (47.8%) exceeding its human win rate (34.2%)—likely reflects the VLM's sensitivity to production quality dimensions where Gemini performs competitively, while human experts weight content accuracy and pedagogical effectiveness more heavily. This pattern is consistent with known biases in VLM evaluation of multimedia artifacts, where surface-level polish can inflate scores relative to content depth [33]. The 5-point scale yields the largest agreement gains on the hardest discriminations (Codex vs. Gemini: +7.7%, Claude vs. Gemini: +8.7%), confirming that finer rubric granularity is most valuable precisely where evaluation is most challenging.

K Preference Arena

Figure 6 shows the pairwise preference evaluation interface used for human baselining. Annotators see two agent-built outputs side-by-side for the same Figma task, with deployed UI frames rendered at full fidelity. The original Figma design is linked for reference. Position (left/right) is randomized per vote to mitigate ordering effects. Annotators select "I prefer this" for the better output, with no access to agent identity or LLM scores.

L Task-Level Human Baseline

Table 17 reports per-task pass/fail classification from human expert evaluation, applying the quality threshold defined in Section 8.2 ($\leq$3 = fail, $\geq$4 = pass). Tasks are sorted by overall pass rate to illustrate the difficulty gradient produced by the complexity stratification described in Section 5.1.

M Experiment Versioning Infrastructure

To support controlled ablations, each execution records a `versions` dictionary (e.g., `{"skill": "v0"}`) stored as a JSONB column alongside run metadata. `SKILL.md` files are versioned in a `versions/` subdirectory alongside the base skill file; at execution time, the agent runner overwrites the base `SKILL.md` with the specified version variant before launching the agent session. Analytics endpoints group results by `(harness, model, skill_version)`, enabling per-condition comparison at both the task and aggregate level. The infrastructure supports additional ablation dimensions (e.g., prompt version, tool access) via the same `versions` dictionary without schema changes.

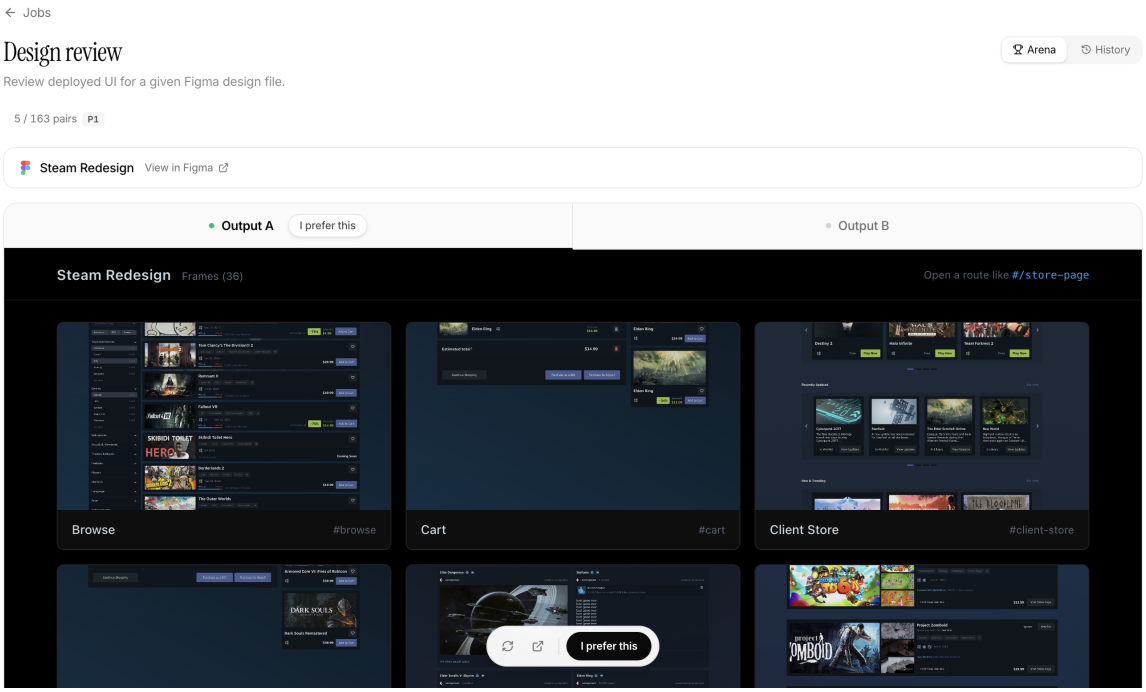

Figure 6: Preference arena interface for pairwise human evaluation. Two agent-built UIs are shown side-by-side for the same Figma design task, with frame-level thumbnails and deployed URLs. Position is randomized; agent identity is hidden.

Table 17: Per-task pass/fail from human expert evaluation (Figma-to-code, n=31 rated tasks). P/T = pass count / total rated runs for that harness on that task. Tasks sorted by overall pass rate.

Task ID	Claude	Codex	Gemini	Overall	Pass %
010ac575	0/3	–	0/1	0/4	0%
49bdd49a	0/2	0/1	–	0/3	0%
aafbd754	0/1	0/2	–	0/3	0%
d8611910	0/1	–	–	0/1	0%
6d91b0b6	0/1	0/2	1/2	1/5	20%
91ce9b15	0/2	0/2	1/1	1/5	20%
20ef0a04	0/2	2/4	–	2/6	33%
7cd22b0d	0/1	0/1	1/1	1/3	33%
85611489	1/1	0/2	–	1/3	33%
a93ffbb6	1/2	–	0/1	1/3	33%
792105af	2/2	0/3	–	2/5	40%
8b3bf60b	2/3	0/2	–	2/5	40%
63521424	–	3/5	0/1	3/6	50%
8efa99ee	1/1	–	0/1	1/2	50%
a713118e	–	1/6	4/4	5/10	50%
cead04c7	4/4	2/6	–	6/10	60%
3a8534f5	1/2	2/2	1/2	4/6	67%
706c87ae	1/1	1/2	–	2/3	67%
7c46867b	0/1	2/2	–	2/3	67%
d0b0a0e3	2/2	0/1	–	2/3	67%
e03bd339	–	1/1	1/2	2/3	67%
f35388e3	0/1	2/2	–	2/3	67%
6af7bf85	2/2	2/2	1/3	5/7	71%
43dd3b2d	2/2	3/3	0/1	5/6	83%
c413f4df	0/1	5/5	–	5/6	83%
65829f23	0/1	6/6	–	6/7	86%
518a5ddc	1/1	2/2	–	3/3	100%
70278991	1/1	5/5	–	6/6	100%
9674e37a	1/1	6/6	–	7/7	100%
d00f7a9b	–	9/9	1/1	10/10	100%
d399b2f6	2/2	–	–	2/2	100%