LH-Bench: Skill-Grounded 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

Binary success metrics work when a task has a single correct answer. They fail for long-horizon enterprise work, where agents must coordinate tools over dozens of steps, produce multiple intermediate artifacts, and satisfy subjective process constraints such as design-system discipline, source grounding, and safe iterative editing. In this setting, evaluation requires procedural knowledge, not just an outcome checker.

We present **LH-Bench**, a benchmark and evaluation design in which expert-authored `SKILL.md` artifacts serve as the bridge between execution and evaluation. Skills encode workflow expectations as observable rubric boundaries, while curated artifact contracts and human preference judgments provide independent validation. We instantiate LH-Bench in two environments: **Figma-to-code** (33 real `.fig` tasks against the Figma API via MCP) and **Programmatic content** (183 chapters across 41 courses), and evaluate three autonomous agent harness families end-to-end.

Expert-authored skills make LLM judges materially more reliable than LLM-authored rubrics alone (κ=0.60 vs. 0.46 on the same runs), and independent human preferences recover the same primary ranking boundary (p<0.05). Skill-level decomposition exposes agent trade-offs that aggregate artifact scores hide, and structured verifier feedback enables recovery from 70.3% of observed execution errors. We release the benchmark artifacts, rubrics, and a source-grounded human reasoning dataset spanning subject matter expert annotations, chapter plans, and pairwise preferences.

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

Keywords: Agent skills, agent evaluation, rubric-based evaluation, long-horizon agents, procedural knowledge, enterprise benchmarks

*Equal contribution.

Agent Skills '26, San Jose, CA
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The strongest existing agent benchmarks are still dominated by tasks with crisp verifiers: unit tests, exact answers, or binary environment success. Enterprise work is different. Agents must maintain state over long horizons, reason over multiple artifacts, and satisfy workflow constraints that are real but not directly executable as tests. A front-end implementation can compile yet violate the design system; a generated tutorial can be fluent yet ungrounded in the provided sources. For these tasks, binary correctness is both too weak to rank systems reliably and too coarse to diagnose failure.

This is fundamentally a *skills* problem. What makes a long-horizon run good is not just the final artifact, but whether the agent executes the right procedure: inspect before editing, extract tokens before components, verify before shipping, cite before claiming. Those expectations are procedural knowledge. If they remain implicit, evaluation becomes ad hoc and hard to scale. If they are encoded explicitly as expert-authored skills, the same artifact can guide execution and define observable criteria for judging execution afterward.

We therefore frame **LH-Bench** as a skill-grounded evaluation design for subjective enterprise work. In LH-Bench, expert-authored `SKILL.md` artifacts specify workflow phases, failure modes, and completion criteria. We pair these skills with curated artifact contracts and human preference judgments, yielding a three-pillar design that can score both process quality and output quality. We instantiate LH-Bench in two environments:

- **Figma-to-code:** 33 real `.fig` tasks in which agents navigate the Figma API via MCP, extract structure and assets, and iteratively build front-end implementations.
- **Programmatic content:** 183 chapters across 41 courses in which agents work inside curated data rooms to create code-driven videos, animations, and presentations with source-grounded citations.

These environments sit squarely in the long-horizon regime identified by prior capability studies [10]: success depends on setup, decomposition, recovery, and context management over many interdependent steps. They also

stress capabilities outside the verifiable settings where recent reasoning post-training has been most successful, especially mathematics and coding [20].

Our contributions are:

1. **A skill-grounded evaluation design** for subjective long-horizon work, combining expert-authored skills, artifact contracts, and human preference validation.
2. **Evidence that expert-authored skills improve judge reliability:** on the same 92 Figma-to-code runs, expert-authored rubric skills raise mean pairwise agreement from κ=0.46 to 0.60.
3. **A two-environment benchmark** that evaluates three autonomous harness families end-to-end and exposes harness-level differences invisible to binary pass/fail scores.
4. **A recovery analysis** showing that agents recover from 70.3% of observed errors when verifier hooks return structured feedback, with recoverability driven more by feedback quality than by raw error frequency.
5. **An open release** of benchmark artifacts, rubrics, and a human reasoning dataset containing subject matter expert (SME) annotations, chapter plans, cited source spans, and pairwise preference data.

2 Related Work

Binary agent benchmarks are insufficient for subjective enterprise work. WebArena [29], OSWorld [26], SWE-bench [8], AgentBench [14], and GAIA [16] established interactive evaluation for LLM agents, but their dominant feedback signal is still binary success or unit-test completion. Even more demanding settings such as WorkArena [6] and SWE-Bench Pro [5] remain primarily completion- or unit-test-graded. LH-Bench differs in making subjective process quality first-class: it evaluates whether the agent executed the right workflow, not only whether it produced a superficially acceptable end state.

Our environments extend prior design-to-code and multi-tool settings. Design2Code [19], FronTalk [25], FullFront [21], and FrontendBench [30] study front-end generation, but largely from screenshots or text requirements. LH-Bench instead starts from real Figma artifacts, requiring API-based inspection, asset extraction, token discovery, and regression-safe iteration. Likewise, MINT [23] and ToolLLM [17] study multi-turn tool use, but not the long-horizon, multi-artifact workflows where verification and repair dominate.

We build on long-horizon evaluation and LLM judging, but contribute the missing procedural layer. Prior work shows that reliable task completion degrades sharply with horizon length [10, 15], that sparse final rewards are inadequate for long episodes [24], and that conversational long-horizon evaluation in API settings poses distinct control challenges [1]. In parallel, LLM-as-a-Judge work demonstrates strong alignment with human preferences but also exposes sensitivity to prompt framing and rubric quality; multi-agent judging improves robustness but still leaves the rubric-design problem open [3, 7, 22, 28]. Our contribution is not confidence calibration, judge ensembling, or judge selection. It is a skill-grounded rubric design for subjective enterprise trajectories, and an empirical demonstration that *expert-authored skills with observable boundaries* make judging long-horizon work materially more reliable.

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

Benchmark	Tasks	Multi-turn	Real env.	Expert rubrics	Artifact eval
WebArena [29]	812	×	Web	×	Binary
VisualWebArena [9]	910	×	Web	×	Binary
SWE-bench [8]	2294	×	Code	×	Unit tests
OSWorld [26]	369	×	OS	×	Binary
τ-bench [27]	200	✓	API	×	Binary
Design2Code [19]	484	×	×	×	VLM
MINT [23]	varies	✓	×	×	Binary
LH-Bench (ours)	**216**	✓	✓	✓	**Multi-tier**

LH-Bench is an Agent Skills benchmark. SkillsBench [13] shows that skill-level decomposition reveals failure modes hidden by aggregate scores, ResearchRubrics [18] shows that rubric design matters for ranking reliability, and DeepResearch Bench II [12] shows the value of expert-derived fine-grained rubrics for long-form report evaluation. LH-Bench contributes a different object: long-horizon agent *trajectories* in enterprise environments. Its novelty is not expert authorship alone, but treating `SKILL.md` as a dual-use artifact that both guides execution in-loop and defines observable rubric boundaries for post-hoc evaluation, exposing harness-level differences in orchestration, compaction, and recovery.

3 Skills as Execution and Evaluation Artifacts

3.1 Why skills are the core artifact

LH-Bench evaluates end-to-end *agent harnesses*, not base models in isolation. Across Claude Code, Codex CLI, and Gemini CLI, the shared substrate is now familiar: sandboxed shell execution, file-system tools, MCP-based extensibility, and long interactive trajectories. What differs in practice is not tool availability but orchestration:

when the agent inspects, when it commits to an implementation plan, when it verifies, and how it recovers. Those decisions are exactly what expert-authored skills encode.

In LH-Bench, each environment ships with a `SKILL.md` artifact that plays a dual role. *At execution time*, it specifies the recommended workflow, common failure modes, and non-negotiable constraints. *At evaluation time*, the same workflow phases define rubric boundaries that are observable from the transcript. For example, "token file created before components" is a verifiable event, not a vague notion of good planning. This dual use is the key design choice: skills are not auxiliary documentation, but the mechanism that makes subjective process quality inspectable at scale.

3.2 Autonomous execution and context management

All harnesses run autonomously inside sandboxes and interact with task environments through controlled APIs such as design extraction, browser automation, source extraction, rendering, and deployment. Tasks are stateful and routinely exceed context limits, so harnesses must compact prior interaction history while preserving commitments, open TODOs, and verified facts. We therefore evaluate the full harness stack, including compaction policy, retry behavior, and repair strategy, rather than a single model call.

3.3 Verifier hooks and recovery signals

Verification is a first-class interface rather than a final post-hoc check. Build failures, preview errors, rubric violations, and visual mismatches are surfaced as structured machine-readable feedback that the agent can act on immediately. In Figma-to-code, for example, a preview call triggers automated checks for runtime exceptions, blank renders, and broken routes; the returned payload includes both a diagnosis and the next required action. This turns verification into an in-loop repair signal instead of an offline score.

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 expose an equivalent native hook mechanism; it can invoke the `create_app_preview` MCP tool but receives raw tool output without automatic post-processing.

3.4 Extensible Tool Interface

All three CLIs support tool extensibility via MCP (Model Context Protocol) servers or equivalent custom tool definitions. LH-Bench exposes environment-specific capabilities—Figma structure inspection, asset export, source extraction, scene generation, previewing, and deployment—through standalone tools that every harness can call. This decouples benchmark logic from harness internals and keeps the comparison focused on reasoning and orchestration.

4 Benchmark Environments

LH-Bench currently contains two enterprise environments. Both are tool-rich, stateful, and evaluated with identical tool access plus environment-specific expert skills. The environments were selected because they force agents to coordinate planning, retrieval, editing, verification, and repair over long horizons, while still admitting concrete artifact contracts.

4.1 Figma-to-Code Environment

In Figma-to-code, agents take real Figma artifacts as input and iteratively produce front-end implementations across 33 `.fig` tasks. The key challenge is that the agent cannot treat the task as screenshot copying: it must inspect nested structure, discover routes, extract reusable tokens and assets, and preserve prior work across repeated edits.

Action space. Agents interact through (i) Figma MCP calls for design structure, styles, components, and asset export, (ii) file and shell tools for implementation, (iii) a preview tool that catches build and runtime failures, and (iv) a deployment tool that publishes static builds for evaluation and reproducible inspection.

Workflow constraints. The expert skill enforces workflow discipline: inspect before coding, implement all frames rather than cherry-picking, avoid interactive scaffolding, use deployment-safe asset paths, and checkpoint through preview before major changes. These constraints are intentionally transcript-visible so they can become rubric boundaries at evaluation time.

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 PNG exports per frame. The evaluated build is exercised through Playwright to capture screenshots, and a VLM judge compares those screenshots against ground truth frame by frame. We also run programmatic checks for successful builds, deployment accessibility, and component coverage. Pure infrastructure failures 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.

4.2 Programmatic Content Environment

Programmatic content captures a different but equally important enterprise workflow: given a curated data room, the agent must produce code-driven media that explains complex concepts faithfully for a particular audience. The outputs are not raw video generations but *programs* that render to media: Remotion videos with narration, Manim animations, or React/Framer Motion presentations. We evaluate 183 chapters across 41 courses.

Action space. Agents use three tool layers: (1) *source tools* that normalize heterogeneous documents into line-numbered markdown with a structured index, enabling line-level citation; (2) *generation tools* that create scenes, animations, or slides with synchronized audio; and (3) *verification tools* that check rendering, synchronization, and viewport constraints.

Multi-turn conversation simulation. Tasks simulate iterative expert interaction: the agent produces a sequence of chapters over a persistent workspace, then receives revision requests such as "compare GRPO with PPO" or "add a gradient-flow visual." Strong performance therefore requires *scene coherence* across turns and *fast editing*: targeted changes without wholesale regeneration.

Grounding and verification. Source extraction produces a persistent structured index with per-source line counts and section headings, enabling judges to verify source grounding and penalize unsupported claims. We detail the verifier hooks here because they are central to recovery analysis: before compilation, generated React or animation code is parsed for structural errors; during rendering, Remotion compilation failures are surfaced directly to the agent; after TTS generation, audio–video duration synchronization is checked at render time; and presentation outputs are validated with screenshot and viewport checks. These verifier hooks make programmatic content amenable to the same recovery-style analysis as Figma-to-code, even though the artifacts are multimedia rather than webpages.

Table 2. Representative LH-Bench tasks, included in the main text to illustrate why binary pass/fail misses the procedural burden of these environments.

Environment	Representative task	Why it is hard
Figma-to-code	Multi-route product or SaaS interface from a real `.fig` file	The agent must inspect nested frames, extract assets and tokens, implement all routes, and preserve previously-correct pages during iterative edits.
Programmatic content	Chapter revision such as "compare GRPO with PPO" or "add a gradient-flow visual"	The agent must retrieve supporting evidence from multiple sources, maintain prior visual style, edit only the affected scenes, and keep narration synchronized with updated visuals.

5 Benchmark Construction

We construct LH-Bench from expert workflows and enterprise-representative artifacts. Every task includes (i) environment state, such as a `.fig` file or a data room of sources; (ii) expert-authored skills and rubrics; and (iii) verifiers over intermediate and final artifacts.

5.1 Subject matter expert artifacts

For each environment, subject matter experts (SMEs) author the workflow artifacts that drive both execution and evaluation. In Figma-to-code, four experienced front-end engineers author `SKILL.md` guidance plus the process rubrics used to score trajectories. In programmatic content, instructional-design SMEs author chapter plans, design notes, cited source spans, and the rubric criteria used for evaluation. The crucial design principle is that rubric boundaries are *observable*: they refer to events that can be verified from the transcript or generated artifacts, rather than to vague notions of quality.

5.2 SME annotation workflows

Figma-to-code dataset curation. Ground truth is built from an automated pipeline with expert curation. We source candidate designs from Figma Community and enterprise design partners, filter for enterprise complexity rather than template simplicity, duplicate them into evaluation accounts, and extract structure through the production Figma API. Expert selection balances the four complexity axes from Section 4.1.

Programmatic content source-grounded annotation. To build faithful ground truth for programmatic content, we provide SMEs with an annotation interface that supports chapter-level task specification and fine-grained source grounding. The interface segments sources into line-addressable spans and uses a highlight-to-cite interaction to attach exact evidence to each chapter, producing high-quality citations with explicit metadata.

5.3 Artifact contracts and structured traces

LH-Bench makes subjective tasks scorable by requiring agents to emit *interim artifacts*—for example per-frame ground-truth images, manifests, chapter citation targets, and renderable scene programs—that serve as deterministic hooks for downstream verification (schemas in Appendix C). We also preserve structured traces aligned to workflow phases, so judges consume a decision-relevant action sequence rather than an unbounded raw transcript.

5.4 Versioned infrastructure and parallel judging

Task definitions, rubrics, and skill versions are tracked independently. The pipeline (Appendix A) exposes execution, evaluation, and leaderboard endpoints; agents build artifacts in ephemeral sandboxes and upload final outputs to object storage. Evaluation runs three judges in parallel (Table 3): a trajectory judge for planning and recovery, a process judge for skill execution, and an output judge for artifact fidelity. Judges emit structured JSON grades per tier, enabling both leaderboard ranking and fine-grained diagnosis.

5.5 Open release

We release more than prompts and final scores. The LH-Bench release includes task artifacts, rubric versions, environment-specific skills, verifier configurations, and a human reasoning dataset. For programmatic content this dataset includes SME chapter decompositions, global design notes, line-level citation spans, and pairwise preference outcomes; for Figma-to-code it includes expert rubric definitions and preference judgments over deployed artifacts. The intended use is twofold: reproducible evaluation of long-horizon agents and supervision data for future work on skill induction, rubric learning, and post-training for enterprise tasks.

Public artifacts. All datasets are released on HuggingFace:

- Figma-to-code tasks: https://huggingface.co/datasets/metaphilabs/frontend-figma-to-code
- Figma-to-code expert preferences: https://huggingface.co/datasets/metaphilabs/figma2code-expert-preferences
- Programmatic content tasks: https://huggingface.co/datasets/metaphilabs/remotion-video-gen
- Programmatic content expert preferences: https://huggingface.co/datasets/metaphilabs/remotion-video-preferences

Table 3. 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

6 Evaluation Framework

Evaluation uses two primary tiers, each scored on anchored 1–5 scales and reported separately to preserve diagnostic value.

Process tier (skill score). Three LLM judges (Gemini 3.1 Pro, Claude Sonnet 4.6, GPT-5.2) score structured traces and transcripts on four expert-authored process rubrics with observable boundaries aligned to workflow phases: design inspection, token extraction, component architecture, and build verification. We report means and cross-judge variance (Section 7.4).

Output tier (artifact score). A VLM judge (Gemini 3) compares Playwright-captured screenshots against expert ground-truth images frame by frame on eight rubrics covering visual fidelity, layout accuracy, and interaction correctness (Appendix G).

Scoring. Each tier is a weighted average ($S_{\text{tier}} = \sum_i w_i s_i, \sum w_i = 1$). Output scores provide an aggregate quality signal; process scores preserve the skill-level detail needed for diagnosis, ablation, and future post-training.

7 Experiments

7.1 Experimental Setup

We evaluate three agent harness families—Claude Code, Codex CLI, and Gemini CLI—across seven configurations (harness × model; Appendix D) on two environments: Figma-to-code (33 tasks) and programmatic content (183 chapters across 41 courses). Each configuration executes autonomously with identical tool access and the same environment-specific expert skill. Skill-tier evaluation uses one flagship judge model from each family. A controlled ablation compares matched Figma-to-code runs with and without `SKILL.md` (Section 7.5).

For programmatic content, SMEs provide both reasoning artifacts and scoring criteria: chapter decompositions, global design notes, source-span annotations, and a five-skill rubric covering content selection, narrative structure, visual hierarchy, information density, and source grounding. These artifacts serve as ground truth for rubric synthesis and VLM-based evaluation.

Table 4. 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

Table 5. 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 6. 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

7.2 Figma-to-Code Results

Table 4 reports output scores, Table 5 reports skill scores, and Table 6 decomposes the skill score by rubric. All three primary candidates include 95% bootstrap confidence intervals (1,000 resamples over tasks).

7.3 Programmatic Content Results

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

Table 7. 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

Rubric granularity analysis. Both 3-point and 5-point scales preserve the same harness ranking (Claude Code > Gemini CLI > Codex), confirming that the result is not an artifact of rubric granularity. The 5-point scale yields tighter confidence intervals (38% reduction) and reduces VLM ties by 49%, bringing VLM tie behavior closer to human annotators (Appendix J).

7.4 Rubric-Level Analysis

Table 6 reveals why the skill tier is useful: it separates bottlenecks in workflow execution from bottlenecks in output quality.

Design token and style extraction is a universal bottleneck. All three agents score lowest on *token & style extraction* (Claude Code 2.88, Codex 2.64, Gemini CLI 1.66). This is the clearest evidence that long-horizon front-end work is not merely code generation. Even strong models can implement components competently once coding begins, but they often fail to externalize the design system first. That gap then propagates into downstream fidelity errors.

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 models are comparatively good at structural implementation once the workflow has been set up correctly.

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 strongest build-verification behavior, suggesting a harness that is relatively persistent at repair even when upstream extraction is weak. These compensatory profiles are precisely what aggregate leaderboard scores hide.

Task-level difficulty gradient. The deliberate complexity stratification (Section 4.1) produces a meaningful difficulty spread rather than a benchmark full of trivial wins. Applying the expert pass/fail threshold ($\leq$3 = fail, $\geq$4 = pass) across all rated tasks (n=31), 71% of tasks

are discriminating: at least one harness passes and at least one fails. This is where skill-level differences matter most.

7.5 Ablation: Removing Expert Skills

We ablate the effect of expert-authored `SKILL.md` in Figma-to-code by rerunning all three harness families *without* the expert workflow (single-line task description, no procedural guidance) on three tasks spanning the difficulty gradient, then comparing against matched runs *with* `SKILL.md`. Tool access is held constant. Table 8 reports per-harness skill scores (n=7 paired runs).

Preliminary evidence from this small-scale ablation (n=7 paired runs) suggests that expert skills matter, but not uniformly across harnesses. Codex gains the most (+0.87 overall), with build verification improving from 1.00 to 2.67; without `SKILL.md`, it effectively fails to preview and iterate. Claude Code gains modestly (+0.15), with improvements concentrated in early-stage inspection and token extraction. Gemini CLI changes little (+0.05), suggesting that its built-in iteration pattern partly compensates for missing workflow guidance. Across harnesses, the strongest operational signal is deployment success: only 2 of 7 runs without `SKILL.md` produce a deployable artifact. We also observe a 42% execution-cost reduction for Claude Code (mean \$6.45→\$3.75) when expert skills reduce exploratory backtracking. Because the study is underpowered (2–3 runs per harness), we treat it as directional rather than definitive; Appendix M describes the infrastructure for scaling this analysis.

7.6 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 [4] as our primary agreement measures.

Figma-to-code. Mean pairwise Cohen's κ (quadratic weights) is 0.60 (n=92 runs, 360 ratings), indicating moderate-to-substantial agreement [11]; 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 [7].

Expert-authored vs. LLM-authored rubrics. We compare agreement under two rubric versions on the same 92 runs (Table 9). 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 result is the paper's central empirical point: better skills produce better judges, even when the judge models and evaluated runs are fixed.

Table 8. `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**

Table 9. 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**

Convergent validity across evaluation tiers. Three independent signals—VLM artifact fidelity, three-judge skill evaluation, and pairwise human preference (Table 10)—all recover the same primary ranking boundary. Because the tiers use different inputs, different judges, and different criteria, this convergence is strong evidence that expert-authored skill verifiers are useful ranking signals rather than artifacts of a single judging setup.

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 10). 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 nearly transitive (4.2% cycle rate). Winner quality ratings also differ across agents (Kruskal–Wallis p=0.048): Codex wins are rated higher on average (3.88/5) than Claude Code (3.40/5) or Gemini CLI (3.35/5), suggesting higher output peaks despite similar top-tier preference rates.

Expert pass/fail classification. The domain expert also assigns absolute quality ratings on a 5-point

Table 10. 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 11. 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%

scale: ≤3 = fail, 4 = design system well-defined but assets and fine-grained spacing still require one engineering sprint, 5 = production-ready. Overall, 60% of winning outputs pass (≥4), with substantial variance across models: Codex GPT-5.2 Pro reaches 77.8% (21/27), Codex GPT-5.2 reaches 57.9% (33/57), Claude Code reaches 54.5% (24/44), and Gemini CLI reaches 52.4% (11/21). This reinforces that pairwise preference and absolute artifact quality are related but not interchangeable.

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 the LLM judges nominally separate the top two where human preferences do not, cautioning against interpreting fine-grained LLM score gaps as perceptible quality differences.

Human–VLM alignment (programmatic content). We validate VLM-as-judge scoring against n=275 matched chapter-level human SME pairwise preferences (Table 11).

Both human and VLM judges agree that Claude Code is the top-ranked harness (human WR 81.5%, VLM WR 69.0%), and Codex win rates align closely (34.2% vs. 33.1%). The main divergence is at position 2: humans rate Codex and Gemini equally, while the VLM favors Gemini, likely because production polish is overweighted relative to content accuracy. Pairwise agreement improves from 46.5% to 52.0% under 5-point rubrics as finer granularity reduces tie asymmetry (Appendix J).

Table 12. 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 13. 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

7.7 Failure Taxonomy and Test-Time Recovery

We extract structured error→recovery events from all 96 Figma-to-code agent transcripts using an LLM-based analysis pipeline (Tables 12, 13).

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). The clearest pattern is not which errors occur, but which errors are actionable: structured compiler-style feedback (syntax, type, build) yields >85% recovery, whereas ambiguous configuration signals yield only 17%.

Agents recover from 70% of errors, with distinct correction strategies. Across 590 errors, agents self-correct 70.3% of the time (Table 13), showing that verifier hooks are not only evaluators but useful runtime skills in their own right. The harnesses adopt distinct repair styles: Claude Code is *proactive*, previewing early and often; Codex is *reactive*, encountering more errors but recovering from many of them; Gemini CLI is *persistent*, driving consistently to deployment.

8 Conclusion

The core claim of this paper is simple: evaluating long-horizon enterprise agents requires explicit procedural knowledge, and expert-authored skills are a practical way to provide it. In LH-Bench, skills do double duty. They

guide the agent at execution time, and they define observable boundaries for judging the resulting trajectory. That design makes subjective enterprise work scorable without collapsing everything into a brittle binary metric.

Across Figma-to-code and programmatic content, expert-authored skills improve judge reliability, human preferences confirm the main ranking boundary, and structured verifier feedback supports real recovery during execution. Just as importantly, skill-level decomposition reveals which parts of the workflow are failing. That is the signal needed for benchmark design, harness engineering, and post-training alike.

9 Limitations and Future Work

(1) Our findings are limited to two environments, albeit two that stress very different skills; extending the framework to additional enterprise domains is necessary before making claims of broad external validity. (2) We evaluate concrete commercial harnesses rather than a model-agnostic open implementation, so model capability and harness design remain partially entangled. (3) The `SKILL.md` ablation is underpowered and should be read directionally. A natural next step is to scale this study and investigate automated skill induction from successful and failed trajectories. (4) Human preference validation remains expensive; improving the faithfulness of multimodal judges, especially on content accuracy versus surface polish, is still an open problem.

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] 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*.

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

[5] 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).

[6] 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)*.

[7] 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).

[8] 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)*.

[9] 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)*.

[10] 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/.

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

[12] 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).

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

[14] 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)*.

[15] 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).

[16] 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)*.

[17] 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)*.

[18] 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).

[19] 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)*.

[20] Yi Su, Dian Yu, Linfeng Song, Juntao Li, Haitao Mi, Zhaopeng Tu, Min Zhang, and Dong Yu. 2025. Crossing the Reward Bridge: Expanding RL with Verifiable Rewards Across Diverse Domains. *arXiv preprint arXiv:2503.23829* (2025).

[21] 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).

[22] 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).

[23] 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)*.

[24] 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*.

[25] 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).

[26] 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)*.

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

[28] 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)*.

[29] 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)*.

[30] 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

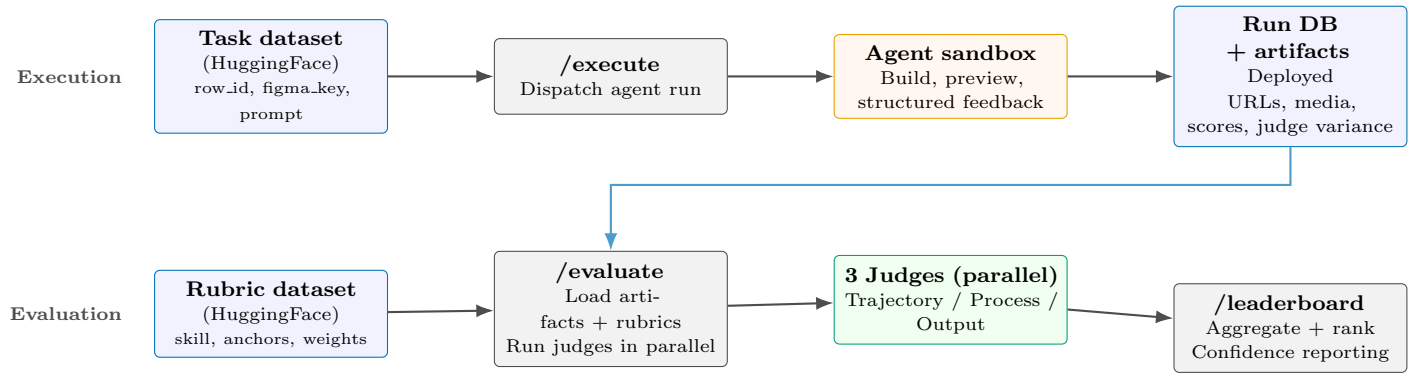

Figure 1. LH-Bench execution and evaluation pipeline. Tasks and rubrics are versioned independently in HuggingFace. Agent runs produce persistent artifacts, which are graded by three judges in parallel; results flow into leaderboards.

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 7.3) and encode the 5-skill design rubric (content selection, narrative structure, visual hierarchy, information density, source grounding).

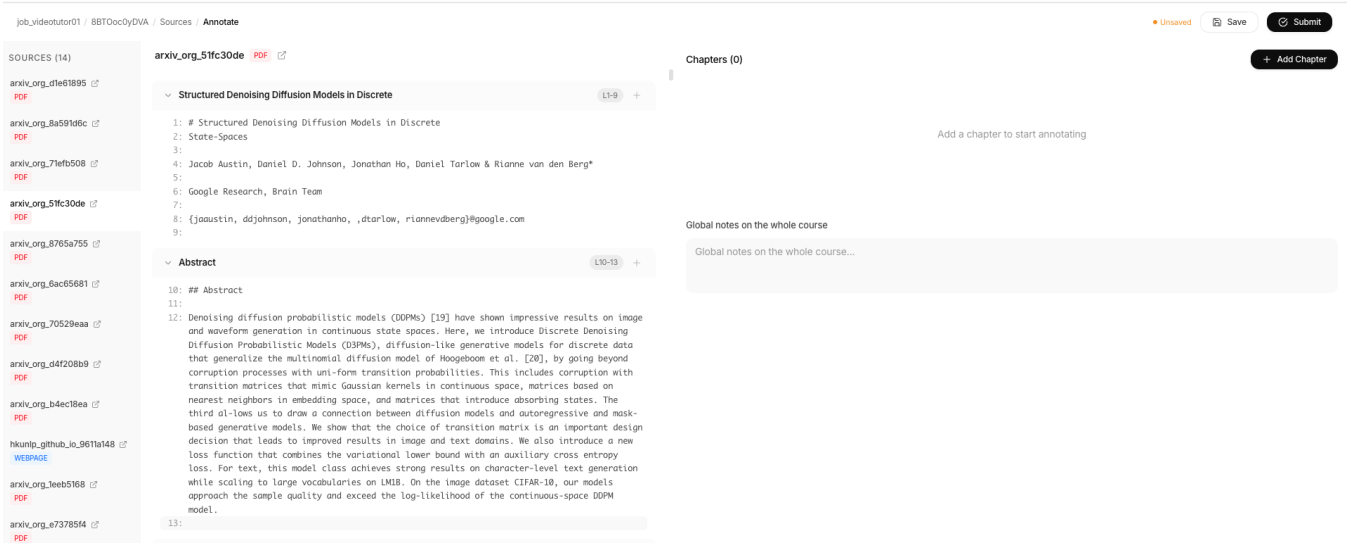

Figure 2. SME annotation interface for programmatic content. Left: source documents in the data room. Center: line-numbered document viewer with collapsible sections and highlight-to-cite interaction. Right: chapter definitions with attached source spans and global design notes.

C Ground Truth Schema Examples

Figma-to-code manifest. (see Section 5.3):

```
{
  "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 5.3):

```
{
  "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 14 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.

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 15 compares the two rubric versions used in the inter-judge agreement analysis (Section 7.6).

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.

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.

Table 14. 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 hardcoded 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

Table 15. 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

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 9).

G Output Tier Rubric Weights

Table 16 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 16. 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

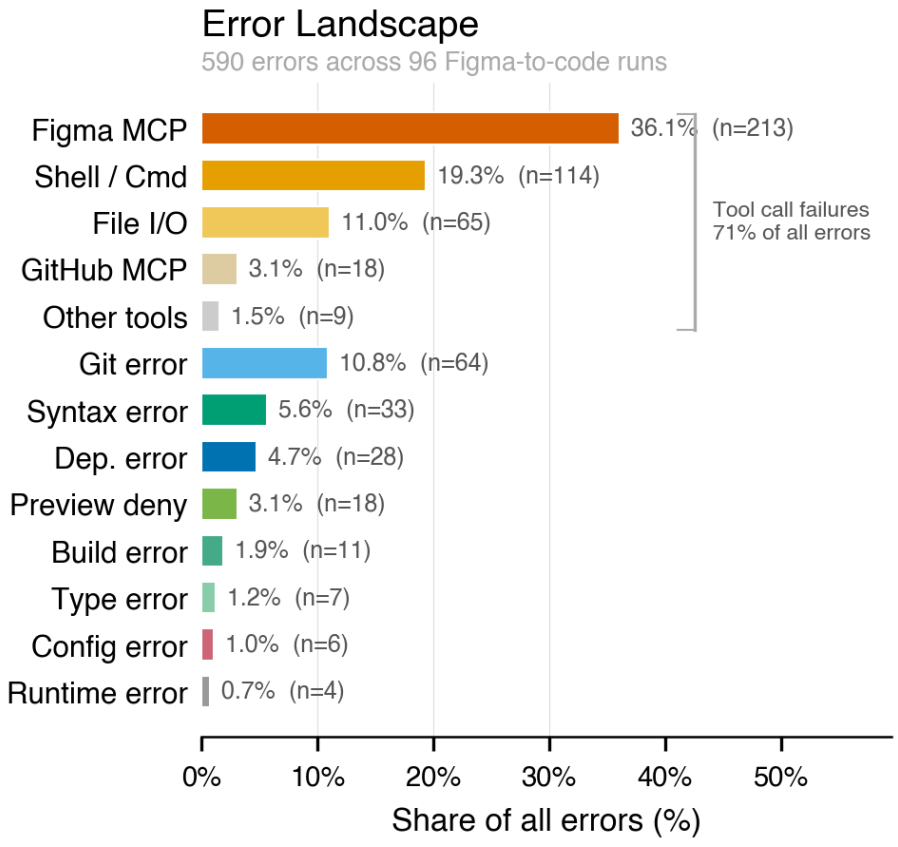

Figure 3. Error landscape across 96 Figma-to-code runs (590 total errors). Tool call failures account for 71% of all errors; within these, Figma MCP operations are the dominant source (51%), reflecting the difficulty of reliably invoking design-extraction APIs at scale.

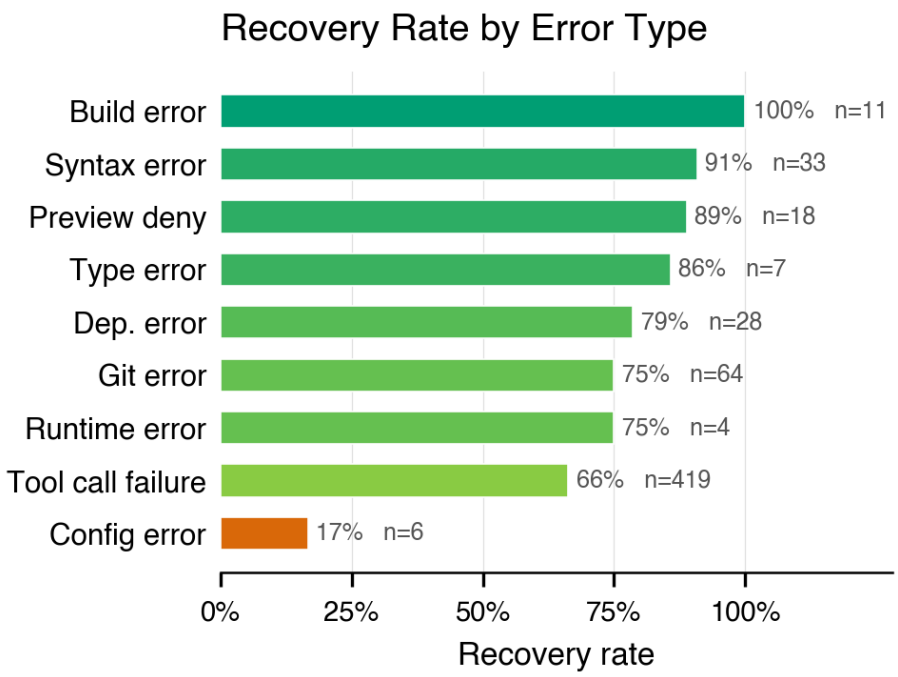

Figure 4. Recovery rates by error type. Structured compiler feedback (syntax, type, build errors) yields >85% recovery; ambiguous signals (configuration errors) yield only 17%.

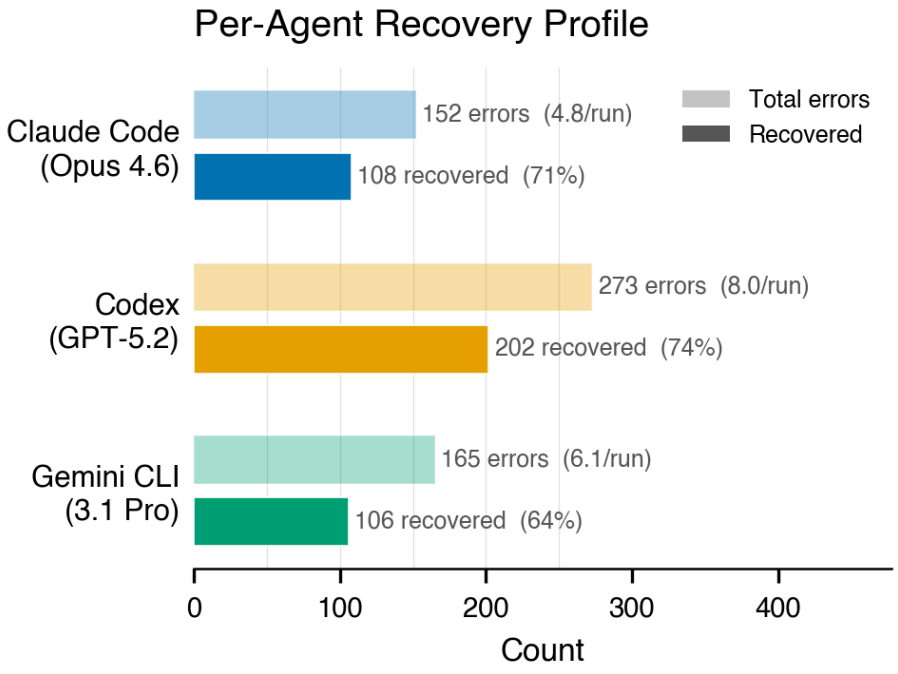

Figure 5. Per-agent recovery profiles. Codex encounters the most errors (8.0/run) yet achieves the highest recovery rate (74%); Claude Code encounters the fewest (4.8/run) with 71% recovery; Gemini CLI recovers 64% with 100% deploy completion.

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 7.3 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 7.6.

J.1 Per-Pair Agreement and Cohen's κ

Table 17 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).

Table 17. 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**

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.
- **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 11) 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 [28]. 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.

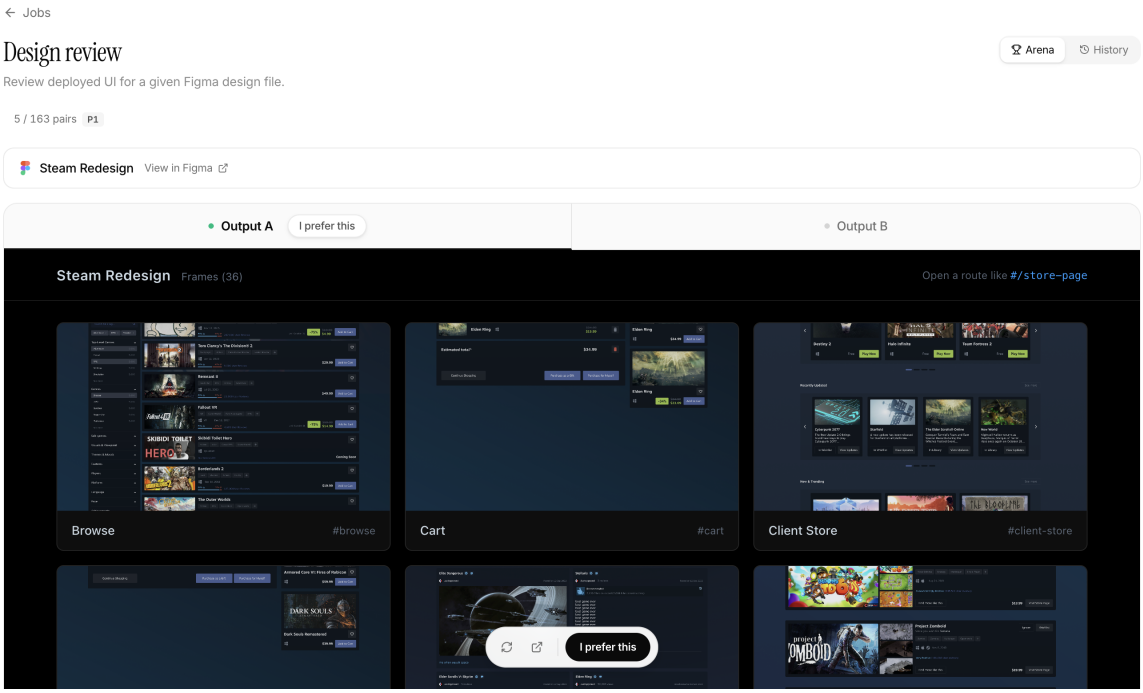

Figure 6. Preference arena interface for pairwise human evaluation. Two agent-built UIs are shown side-by-side for the same Figma design task, with frame-level thumbnails and deployed URLs. Position is randomized; agent identity is hidden.

L Task-Level Human Baseline

Table 18 reports per-task pass/fail classification from human expert evaluation, applying the quality threshold defined in Section 7.2 (≤ 3 = fail, ≥ 4 = pass). Tasks are sorted by overall pass rate to illustrate the difficulty gradient produced by the complexity stratification described in Section 4.1.

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.

Table 18. 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%