Solve Vision with Code: Coding Agents Are Stronger Visual Reasoners Than Video Models

Hokin Deng [1] **Zehong Zhao** [2] **Ran Ji** [2] **Maijunxian Wang** [3] **Qingying Gao** [4] **Fengyuan Yu** [1]
Zhengze Jiang [5] **Yilan Zhang** [6]

[1] Carnegie Mellon University [2] University of California, San Diego [3] University of California, Berkeley [4] Johns Hopkins University [5] Columbia University [6] University of California, Los Angeles

[†] Correspondence to: Hokin Deng <hokind@andrew.cmu.edu>.

[Code] [Website] [Results]

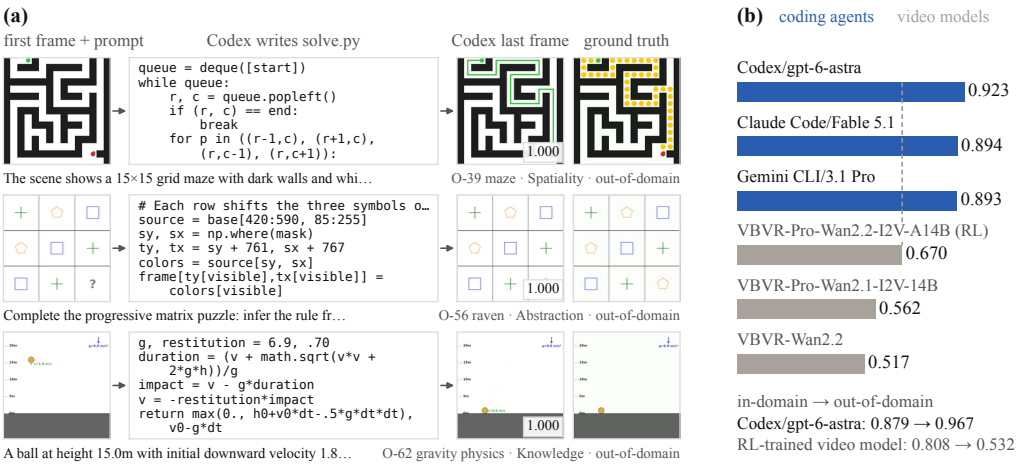


Figure 1 Solve vision with code. (a) Three VBVR-Pro-Bench tasks, one per row (O-39 maze, O-56 Raven matrix, O-62 gravity): the first frame and prompt the solver receives, lines from the `solve.py` Codex wrote in answer, and the last frame that program renders, with its evaluator score, beside the reference last frame. (b) Overall score of the three closed-model coding agents and the three best video models on the same 500 instances, scored by the same official evaluator; the best agent scores 0.923 against 0.670 for the best video model, which was RL-trained on the benchmark's own in-domain task families. From the in-domain to the out-of-domain half Codex rises (0.879 to 0.967) and that video model falls (0.808 to 0.532).

ABSTRACT

VBVR-Pro asks whether a generative model can reason natively in the visual medium. Given a first frame and a prompt, the model must produce the video in which the task is carried out, and a rule-based evaluator scores the result (Xu et al., 2026). Video generation models do poorly on it. The best of them, a Wan2.2 model RL-trained on the benchmark's own task families, scores 0.670 on VBVR-Pro-Bench. We keep the task, the inputs, the output container and the evaluator and change only the medium of the answer. Instead of denoising pixels, a coding agent writes a program that renders the video. The agent sees the first frame and the prompt inside a sandbox, with no ground truth, no metadata and no image or video generation model, and it gets one attempt. Three closed-model agents reach 0.923 (Codex, gpt-6-astra), 0.894 (Claude Code, Fable 5.1) and 0.893 (Gemini CLI, Gemini 3.1 Pro), each above 0.89 and 0.25 above the best video model. Their scores rise on the out-of-domain half of the benchmark, where every video model trained on it falls. Across 24 open-weight models the best reaches 0.572, above every video model except the RL-trained one; the weakest never call a tool and produce no video. Code and results are at `https://github.com/hokindeng/solve-vision-with-code`. Every video and program of the three closed-model agents, side by side with the ground truth, is at `https:`

//hokindeng.github.io/solve-vision-with-code/; open-weight lane videos and run logs are available on request. The project page is https://hokindeng.com.

1. Introduction

Video generation models now produce photorealistic, temporally coherent footage, and on that basis they are increasingly described as world models that can reason about the visual world (OpenAI, 2024; Google DeepMind, 2025; Wan Team, 2025; Kuaishou Technology, 2025; ByteDance Seed, 2025). VBVR-Pro puts that description to a test (Xu et al., 2026). It asks whether a generative model can reason *natively* in the visual medium. Given a first frame and a prompt, the model must produce the video in which the prompt is carried out, and a rule-based evaluator checks the content of the frames. The suite is built from 300 procedural generators in five cognitive categories (abstraction, perception, spatiality, transformation, knowledge). Its benchmark, VBVR-Pro-Bench, holds 100 tasks with five instances each, 50 in-domain families with training data and 50 out-of-domain families held out from all training. Video models score low on it. General-purpose systems land between 0.1 and 0.5, and the best, a Wan2.2 model RL-trained on the in-domain families with the evaluator as reward, reaches 0.670 overall and 0.532 out of domain.

We ask a different question about the same test. Every VBVR-Pro task is generated by a program, and its answer is a deterministic function of the first frame and the prompt. A model that denoises pixels must recover that function implicitly. A model that writes code can state it and let a renderer do the rest. So we swap the medium of the answer and nothing else. A coding agent is given the same first frame and the same prompt, together with the required container spec (frame size, frame rate, frame count, duration), and must write a program that renders the answer video. It sees no ground truth, metadata or generator, may not call an image or video generation model, runs in a CPU sandbox with common Python imaging libraries and a 30-minute limit, and gets one attempt per instance. The output goes to the unmodified official evaluator; an instance without a video scores zero, as it does for a video model that fails to generate (Fig. 1). This is the program-synthesis route to visual reasoning (Gao et al., 2023; Chen et al., 2023; Surís et al., 2023; Gupta & Kembhavi, 2023), applied to a benchmark that was built for video models and scored on their terms.

We ran this protocol on all 500 instances of VBVR-Pro-Bench with three closed-model agents (Codex with gpt-6-astra, Claude Code with Fable 5.1, Gemini CLI with Gemini 3.1 Pro) and, through the OpenCode harness, with 24 open-weight models. Video-model scores are the published VBVR-Pro leaderboard numbers and were not rerun.

The gap is large (Fig. 2, Tab. 2). The three closed-model agents score 0.923, 0.894 and 0.893, all above 0.89, against 0.670 for the best video model. The RL-trained video model earns its score in domain (0.808) and loses it out of domain (0.532), and every trained video model falls on the held-out families. Every coding agent above the noise floor, closed or open, rises on them; Codex goes from 0.879 to 0.967 (Fig. 3). Among open-weight models the best, GLM-5, scores 0.572, above every video model except the RL-trained one. The bottom of the ranking is a failure of tool use. Models that never call a tool produce no video in 500 attempts, and models that stop after one or two calls score near zero. The strongest agent is also cheap. Codex makes 3.2 tool calls per instance with a median wall-clock of 22 seconds (Tab. 4).

Contributions.

1. **A protocol.** A coding agent answers VBVR-Pro tasks by writing a program that renders the video, under the same inputs, output container and official evaluator as the video models. The benchmark itself is unchanged.
2. **A leaderboard.** Three closed-model coding agents and 24 open-weight models scored on all 500 instances of VBVR-Pro-Bench, next to the published video-model leaderboard: 0.923 against 0.670.
3. **The out-of-domain reversal.** Trained video models lose their advantage on held-out task families and coding agents gain on them, reported per split and per cognitive category.

4. **An anatomy of agent behaviour.** Tool calls, wall-clock, tokens and failure modes per agent from the agents' own event logs, and the tasks on which even the strongest agents fail.

Code and results, including the per-instance evaluation results and run statistics behind every table in this paper, are at `https://github.com/hokindeng/solve-vision-with-code`. Every video and program of the three closed-model agents, side by side with the ground truth, is at `https://hokindeng.github.io/solve-vision-with-code/`; open-weight lane videos and run logs are available on request. The project page is `https://hokindeng.com`.

2. Related Work

A line of recent work asks whether a video generation model can itself be the reasoner: given a first frame and a prompt, the model renders the solution as a video rather than describing it. Wiedemer et al. (2025) document such zero-shot "chain-of-frames" behaviour in Veo 3 across perception, manipulation and maze tasks, and Tong et al. (2025) frame video generation as a general multimodal reasoning paradigm. VBVR (Xu et al., 2026) and its successor VBVR-Pro (Xu et al., 2026) make the question measurable at scale. Hundreds of procedural generators produce first-frame, prompt and answer-video triples, a rule-based evaluator scores each family, and VBVR-Pro-Bench holds out 50 families from all training to separate memorisation from generalisation. VBVR-Pro also post-trains video models with the same evaluators as verifiable rewards, following GRPO (Shao et al., 2024; DeepSeek-AI, 2025) as adapted to flow and diffusion models by Flow-GRPO (Liu et al., 2025) and DanceGRPO (Xue et al., 2025). Its RL-trained checkpoints are the strongest video models on the benchmark and the natural ceiling for the comparison in this paper. We keep the benchmark, the evaluator and the published video-model numbers untouched and change only the family of system under test.

Writing a program is an older route to visual reasoning. Visual Programming (Gupta & Kembhavi, 2023) and ViperGPT (Surís et al., 2023) have a language model compose vision modules into an executable program, and Visual Sketchpad (Hu et al., 2024) lets a multimodal model draw intermediate results with code. These systems answer questions in text. Ours must render a full video, which turns the program from a scaffold into the deliverable. On the language side, program-aided reasoning (Gao et al., 2023; Chen et al., 2023) and Code as Policies (Liang et al., 2023) showed that offloading computation to an interpreter improves reliability. The agents we evaluate are the current form of that idea, Codex-style models (Chen et al., 2021) wrapped in a terminal loop that reads files, runs commands and iterates on errors. SWE-bench and SWE-agent (Jimenez et al., 2024; Yang et al., 2024) established the evaluation pattern we borrow (one sandboxed container per instance, one attempt, a verifier decides), and the terminal-agent benchmark tradition (Laude Institute, 2025) packages tasks the same way. We are not aware of prior work that runs off-the-shelf coding agents, unmodified, against a video-reasoning leaderboard.

Video benchmarks mostly score with learned components. VBench (Huang et al., 2024) mixes feature-based metrics, VideoScore (He et al., 2024) trains a video-language judge, and LLM-as-a-judge protocols inherit known position and verbosity biases (Zheng et al., 2023). Physics-IQ (Motamed et al., 2025) and VBVR-Pro instead compute scores from the pixels with classical computer vision against symbolic ground truth, so the same output always receives the same score and the score can serve as a reward. The same property makes the comparison in this paper exact, since a coding agent and a video model are graded by one deterministic function on the same 500 instances.

3. Method

The experiment is a substitution. VBVR-Pro-Bench gives a video model a first frame and a prompt and asks for the video in which the task gets solved. We hand the same two inputs to a coding agent in a sandbox, ask it to *write a program* that renders that video, and score the result with the benchmark's own evaluator, unchanged. Everything downstream of the video file is identical for both families of system.

3.1. The Benchmark

VBVR-Pro-Bench. We use the video setting of VBVR-Pro-Bench (Xu et al., 2026). The benchmark has 100 task families, each backed by a procedural generator and a hand-written, task-specific evaluator; every family contributes 5 instances, for 500 in total. The 50 *In-Domain* families have training data in the VBVR-Pro SFT and RL corpora, so a video model trained on VBVR-Pro has seen the family, though not the instance. The 50 *Out-of-Domain* families are held out from every training split. The split measures generalisation for the video models it was designed around. For the coding agents, none of which were trained on VBVR-Pro, both halves are unseen.

Inputs and target. An instance consists of `first_frame.png`, a 1024×1024 image of the initial scene, and `prompt.txt`, a natural-language description of what the video should show. The reference video is 16 fps and runs from 10 to 282 frames (0.6 to 17.6 s) depending on the family. It ships with its final frame and a `metadata.json` holding the symbolic ground truth; the evaluator reads these, and no system under test sees them.

Categories. The kit assigns every task family to one of five faculties: Abstraction (25 tasks), Perception (28), Spatiality (16), Transformation (12) and Knowledge (19). The label is carried in the evaluator's per-instance output and is the one we use. Category means are plain means over the instances of the tasks carrying the label.

3.2. Coding Agents as Visual Reasoners

What the agent sees. Each instance runs in a fresh Docker container whose only mounted directory, `/app`, holds three files: `first_frame.png`, `prompt.txt` and an `instruction.md` generated by the harness. The instruction is the same for every agent and every model. In essence it says: *"`first_frame.png` is the first frame of a short video and `prompt.txt` describes what happens in it. Write code that produces the video. Deliver `output/video.mp4` (H.264,* 1024×1024*, 16 fps, about N frames) whose first frame is `first_frame.png` and whose last frame shows the completed result, and keep the program at `solve.py` so it regenerates the video. The video is drawn by your code; do not call an image or video generation model. Change only what the prompt asks for; every other pixel stays as in the first frame. Pace the action over the full duration."* The prompt is quoted verbatim inside the instruction. N and the duration are probed from the reference video, so the agent knows the container spec the evaluator expects, as the video models do. Nothing else is disclosed.

Sandbox. The container image is `python:3.11-slim` with `ffmpeg`, NumPy, Pillow, OpenCV, imageio, SciPy and Matplotlib preinstalled, plus DejaVu, Liberation and Noto CJK fonts. The agent runs as an unprivileged user with 2 CPUs, 3 GB of memory and no swap, a limit of 512 processes, and a wall-clock budget of 1,800 s per instance, after which the harness kills the container and records no video. Network access is left on so that the agent CLI can reach its model API and `pip` can fetch a package. The rules forbid calling any generative image or video model, and a search of the closed agents' 1,500 event logs and of the 7,230 `solve.py` programs retained from the open-weight lanes finds no generation endpoint and no deep-learning framework; the packages agents did install were OCR and scientific-computing libraries. Each instance gets exactly one attempt. A container that exits without `output/video.mp4` is a failed instance, with no retry, no best-of-n and no human intervention. All runs took place on one `c7i.4xlarge` CPU host between 12 and 16 September 2026.

Agents. An *agent* here is an off-the-shelf command-line coding tool driving a model. It reads files, runs commands, edits code and iterates until it decides it is done. We invoke each tool through its official non-interactive entry point with approvals bypassed, and add no system prompt, skill or modification. Table 1 lists the three closed-model agents. Codex CLI receives the first frame as an image attachment as well as a file; the other agents open it from disk with their own file tools or inspect it programmatically, as they see fit. Claude Code is routed through Amazon Bedrock with the model identifier passed straight through. Codex CLI runs with its default reasoning effort; no reasoning-effort override was set for any lane.

Table 1 The three closed-model coding agents. Each is the vendor's own command-line tool at the pinned version, run through its non-interactive mode with approvals bypassed; the 24 open-weight models are all driven by OpenCode 1.18.30 and are listed in Appendix B.

Agent (CLI)	Version	Model
Codex CLI (OpenAI, 2026)	0.154.0	gpt-6-astra
Claude Code (Anthropic, 2026)	2.1.268	Claude Fable 5.1 (Amazon Bedrock)
Gemini CLI (Google, 2026)	0.59.0	gemini-3.1-pro-preview

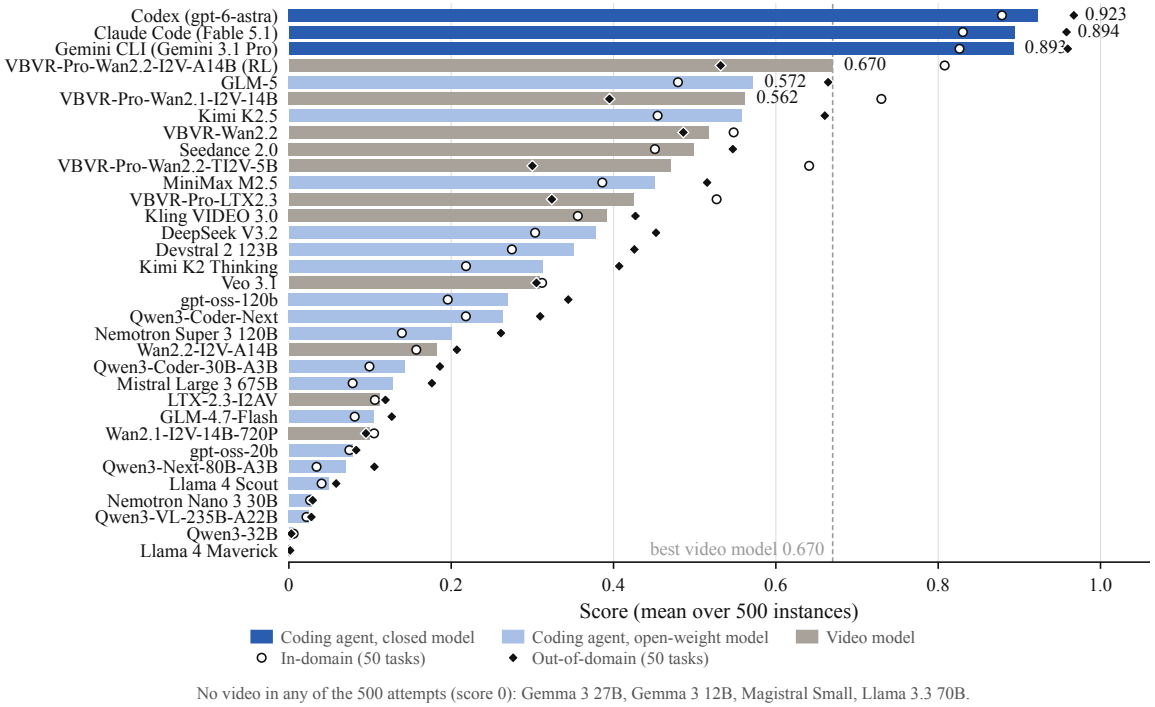
Open-weight roster. To ask whether the result depends on a frontier closed model, we run the same protocol with OpenCode (OpenCode contributors, 2026), an open-source agent that drives any model behind an OpenAI-compatible endpoint, over 24 open-weight models served by Amazon Bedrock from the GLM, Kimi, MiniMax, DeepSeek, Qwen, gpt-oss, Nemotron, Mistral, Llama and Gemma families (Appendix B). The three Llama models are not served by Bedrock's OpenAI-compatible endpoint, and its streaming Converse API rejects tool use for them, so a small shim on the host accepts OpenCode's streaming request, calls the non-streaming Converse API and replays the answer as a short event stream; tool calls map one-to-one. DeepSeek R1 has no tool use on Bedrock in either mode, cannot act as a coding agent, and is listed as unsupported rather than scored. Every open-weight model receives the identical instruction, sandbox and budget. Several are text-only. The protocol does not require vision, since a program can read the pixels.

3.3. Scoring

Every video is scored by the official VBVR-Pro-Bench kit, `run_evaluation_video.py`, without modification. The kit hands each instance to the family's evaluator, which decodes the video, compares it with the symbolic ground truth using classical computer vision (colour and shape primitives, Hungarian matching, multi-object tracking, OCR where the task shows text) and returns a score in $[0, 1]$. We run it in `task_specific_only` mode, the mode behind the published leaderboard, on CPU. The score of an instance for which the agent produced no video is 0, exactly as for a video model that fails to generate. A lane's score is the mean over all 500 instances; In-Domain and Out-of-Domain means are over their 250 instances; category means follow the tag weighting of Section 3.1. The video-model rows we compare against are the leaderboard numbers published with VBVR-Pro (Xu et al., 2026), computed by the same kit; we do not rerun those models. The kit must run under its pinned dependencies (NumPy < 2 plus its tracking and OCR packages), because under NumPy 2 four evaluators silently score every video, ground truth included, at 0. All numbers here come from a conforming environment (Appendix A).

3.4. Efficiency Accounting

Each agent CLI emits a machine-readable event stream, which the harness keeps per instance as `events.jsonl` next to the container's exit code and wall time. From it we compute three quantities. *Tool calls* count the actions the agent took: for Codex CLI, completed command-execution, file-change, MCP and web-search items; for Claude Code, `tool_use` blocks in assistant messages; for Gemini CLI and OpenCode, their `tool_use` events. *Tokens* are the agent's own usage report summed over the run: Codex's per-turn usage; Claude Code's final result usage, with cache-creation and cache-read input counted as input; Gemini CLI's result statistics; OpenCode's per-step token record, with cached input counted as input and reasoning tokens counted as output. *Wall time* is measured by the harness from container start to exit, so it includes model latency; we report the median over the 500 attempts. We also flag whether the agent ever called a tool, whether it hit the time limit, and whether a failed instance failed with or without tool use. These flags drive the failure anatomy in Section 5. Token counts are comparable within an agent and only indicative across agents, since tokenizers and caching differ.



No video in any of the 500 attempts (score 0): Gemma 3 27B, Gemma 3 12B, Magistral Small, Llama 3.3 70B.

Figure 2 The 33 systems that produced at least one video, sorted by overall score (bar: mean over the 500 instances, coloured by system class), with the in-domain score as a hollow circle and the out-of-domain score as a solid diamond on each bar and a dashed line at the best video model (0.670). For every coding agent above the noise floor the diamond lies to the right of the circle, and for every video model trained on the benchmark's task families it lies to the left. The four systems with no video in 500 attempts are named below the axis.

4. Results

Table 2 and Figure 2 report the score of every system on the 500 instances of VBVR-Pro-Bench (video setting): the three closed-model coding agents, the 23 open-weight models that can call a tool (24 in the roster; DeepSeek R1 cannot, Appendix B), and the eleven video models of the published leaderboard (Xu et al., 2026). All 37 systems are scored by the same unmodified official evaluator, and an instance with no video counts 0. Every number in this section is produced by `paper/figures/make_figures.py` from the released result files.

4.1. Main Comparison

The three closed-model coding agents occupy the top three positions. Codex with gpt-6-astra scores 0.923, Claude Code with Fable 5.1 scores 0.894, and Gemini CLI with Gemini 3.1 Pro scores 0.893, all with a video on 500 of 500 instances. The best video model, VBVR-Pro-Wan2.2-I2V-A14B, which was RL-trained on the in-domain task families with the same evaluator as reward, scores 0.670. The gap between the best agent and the best video model is 0.253; the second-best video model, VBVR-Pro-Wan2.1-I2V-14B, is a further 0.108 behind at 0.562. The three agents are within 0.030 of each other, so the result does not hinge on one model.

The gap is not an evaluator artefact. The checker is the benchmark's own, run unchanged, and the video-model rows are the authors' numbers with the same checker. The agents never see the ground truth, the generator or the task metadata, and have no image or video model. Each writes a program that reads the first frame, computes the answer and renders it.

Table 2 VBVR-Pro-Bench (video setting, 100 tasks × 5 instances) leaderboard. Every system is scored by the unmodified official rule-based evaluator of the benchmark kit; the score is the mean over all 500 instances, and an instance for which the system produced no video counts 0. Coding-agent rows are our runs (one attempt per instance, sandboxed, no image or video model available); video-model rows are the published VBVR-Pro leaderboard numbers (Xu et al., 2026), not rerun by us. *Rank* is the position among all 37 systems. *Videos* is the number of instances for which the agent wrote a video file (— for video models, which we did not run). In-domain (ID) tasks are the 50 families with training data in VBVR-Pro, out-of-domain (OOD) the 50 families held out from all training. Best per column in bold, second underlined; the best coding agent and the best video model are shaded. †No video in any of the 500 attempts (the model answers in prose and never calls a tool).

Rank	System	Overall	ID	OOD	Videos / 500
Coding agents, closed models					
1	Codex (gpt-6-astra)	**0.923**	**0.879**	**0.967**	500
2	Claude Code (Fable 5.1)	<u>0.894</u>	<u>0.830</u>	0.958	500
3	Gemini CLI (Gemini 3.1 Pro)	0.893	0.826	<u>0.960</u>	500
Coding agents, open-weight models (OpenCode)					
5	GLM-5	0.572	0.479	0.664	495
7	Kimi K2.5	0.557	0.454	0.660	498
11	MiniMax M2.5	0.451	0.386	0.515	488
14	DeepSeek V3.2	0.378	0.303	0.452	475
15	Devstral 2 123B	0.350	0.275	0.426	494
16	Kimi K2 Thinking	0.313	0.218	0.407	406
18	gpt-oss-120b	0.270	0.196	0.344	380
19	Qwen3-Coder-Next	0.264	0.218	0.310	498
20	Nemotron Super 3 120B	0.200	0.139	0.261	436
22	Qwen3-Coder-30B-A3B	0.143	0.099	0.186	480
23	Mistral Large 3 675B	0.127	0.079	0.176	254
25	GLM-4.7-Flash	0.104	0.081	0.127	470
27	gpt-oss-20b	0.079	0.075	0.083	148
28	Qwen3-Next-80B-A3B	0.070	0.034	0.105	167
29	Llama 4 Scout	0.049	0.040	0.058	269
30	Nemotron Nano 3 30B	0.028	0.026	0.029	91
31	Qwen3-VL-235B-A22B	0.025	0.022	0.028	70
32	Qwen3-32B	0.004	0.006	0.003	25
33	Llama 4 Maverick	0.001	0.001	0.002	7
34	Gemma 3 27B	0.000	0.000	0.000	0†
35	Gemma 3 12B	0.000	0.000	0.000	0†
36	Magistral Small	0.000	0.000	0.000	0†
37	Llama 3.3 70B	0.000	0.000	0.000	0†
Video models (VBVR-Pro leaderboard)					
4	VBVR-Pro-Wan2.2-I2V-A14B (RL)	0.670	0.808	0.532	—
6	VBVR-Pro-Wan2.1-I2V-14B	0.562	0.730	0.395	—
8	VBVR-Wan2.2	0.517	0.548	0.486	—
9	Seedance 2.0	0.499	0.451	0.547	—
10	VBVR-Pro-Wan2.2-TI2V-5B	0.470	0.641	0.300	—
12	VBVR-Pro-LTX2.3	0.425	0.527	0.324	—
13	Kling VIDEO 3.0	0.392	0.356	0.427	—
17	Veo 3.1	0.309	0.312	0.305	—
21	Wan2.2-I2V-A14B	0.182	0.157	0.207	—
24	LTX-2.3-I2AV	0.112	0.106	0.119	—
26	Wan2.1-I2V-14B-720P	0.100	0.105	0.095	—

4.2. In-Domain versus Out-of-Domain

The benchmark splits its 100 tasks into 50 in-domain families, for which VBVR-Pro provides training data, and 50 out-of-domain families held out from all training. Figure 3 plots OOD minus ID for every system that produced at least one video. The two classes move in opposite directions.

Coding agents score higher out of domain. Codex goes from 0.879 in domain to 0.967 out of domain (+0.089), Claude Code from 0.830 to 0.958 (+0.128), Gemini CLI from 0.826 to 0.960 (+0.133). The open-weight models gain more: GLM-5 +0.185, Kimi K2.5 +0.206. Of the 22 coding agents with a video, 21 have a positive difference; the exception, Qwen3-32B, produced 25 videos and scores 0.006 versus 0.003 (−0.003, noise). No agent was trained on any family, so the split has no meaning for them. The held-out families are simply easier for a program; 11 of the 12 hardest tasks for the closed agents are in-domain (Table 5).

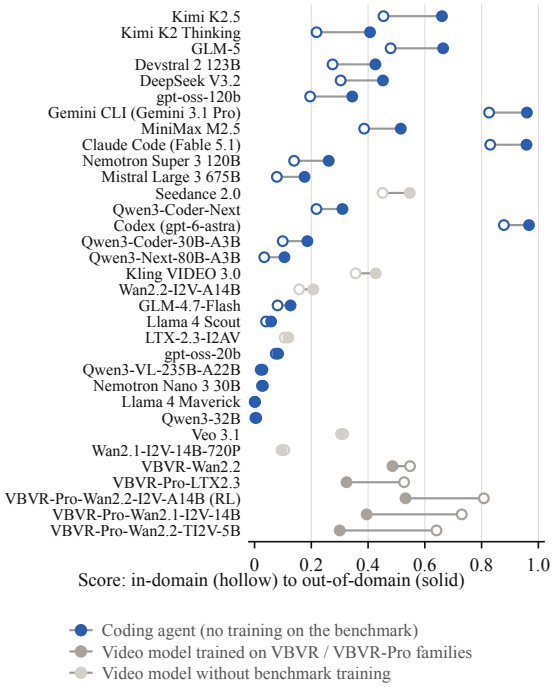


Figure 3 In-domain score (hollow) and out-of-domain score (solid) of every system with at least one video, sorted by the difference between the two. Coding agents (blue), none trained on the benchmark, gain on the held-out families; the five video models trained on VBVR or VBVR-Pro task families (grey) lose between 0.062 and 0.341, and video models without benchmark training (light grey) sit near zero. Values are the ID and OOD columns of Table 2.

Video models trained on the benchmark score lower out of domain. All five VBVR- and VBVR-Pro-trained models drop: VBVR-Pro-Wan2.2-I2V-A14B from 0.808 to 0.532 (−0.276), VBVR-Pro-Wan2.1-I2V-14B from 0.730 to 0.395 (−0.335), VBVR-Pro-Wan2.2-TI2V-5B from 0.641 to 0.300 (−0.341), VBVR-Pro-LTX2.3 from 0.527 to 0.324 (−0.203), VBVR-Wan2.2 from 0.548 to 0.486 (−0.062). The video models without benchmark training are flat or slightly up: Seedance 2.0 +0.096, Kling 3.0 +0.071, Wan2.2-I2V-A14B +0.050, LTX-2.3-I2AV +0.013, Veo 3.1 −0.007, Wan2.1-I2V-14B-720P −0.010. Training a video model on the task families buys in-domain score at the cost of out-of-domain score. The RL-trained A14B is the only video model above 0.8 in domain, and out of domain it falls to 0.532, below Seedance 2.0 (0.547), which never saw the benchmark. Out of domain every closed agent is above 0.95 and the best video model is at 0.547. In domain the agents still lead, with Codex at 0.879, Claude Code at 0.830 and Gemini CLI at 0.826 against 0.808 for the A14B, on the split it was trained on with the evaluator as its reward.

4.3. Results by Category

Table 3 breaks the score down by the benchmark's five task categories: Abstraction (25 tasks), Perception (28), Spatiality (16), Transformation (12) and Knowledge (19). The closed agents are above 0.8 in every category. Codex is best on Abstraction (0.957) and Knowledge (0.863), Gemini CLI on Perception (0.972), and Claude Code on Spatiality (0.908) and Transformation (0.943). The lowest closed-agent cell is Gemini CLI on Spatiality (0.816), the highest Gemini CLI on Perception (0.972).

Table 3 Score by task category for the three closed-model agents and the eight best open-weight models (mean over the 5 instances of every task in the category, both splits; task counts per category in the second header row; best per column in bold, second underlined). The last row is the in-domain category profile of the released VBVR-Pro baseline video model as reported in the VBVR-Pro paper (Xu et al., 2026, Table 8), and because it covers the 50 in-domain tasks only, the block above it restricts the three closed-model agents to the same 50 tasks, computed from the per-instance results. Rows in the in-domain block are not ranked.

System *Tasks per category*	Abstraction (25)	Perception (28)	Spatiality (16)	Transformation (12)	Knowledge (19)	Overall (100)
Coding agents, closed models						
Codex (gpt-6-astra)	**0.957**	0.953	<u>0.903</u>	<u>0.902</u>	**0.863**	**0.923**
Claude Code (Fable 5.1)	0.838	<u>0.957</u>	**0.908**	**0.943**	0.834	<u>0.894</u>
Gemini CLI (Gemini 3.1 Pro)	<u>0.878</u>	**0.972**	0.816	0.894	<u>0.861</u>	0.893
Coding agents, open-weight models (OpenCode)						
GLM-5	0.478	0.748	0.491	0.569	0.506	0.572
Kimi K2.5	0.427	0.727	0.527	0.582	0.490	0.557
MiniMax M2.5	0.347	0.633	0.412	0.439	0.358	0.451
DeepSeek V3.2	0.308	0.538	0.358	0.288	0.307	0.378
Devstral 2 123B	0.290	0.478	0.382	0.296	0.248	0.350
Kimi K2 Thinking	0.231	0.476	0.359	0.195	0.216	0.313
gpt-oss-120b	0.224	0.348	0.300	0.167	0.255	0.270
Qwen3-Coder-Next	0.260	0.364	0.195	0.180	0.232	0.264
In-domain tasks only						
Tasks per category	(13)	(7)	(10)	(6)	(14)	(50)
Codex (gpt-6-astra)	0.928	0.948	0.872	0.823	0.826	0.879
Claude Code (Fable 5.1)	0.763	0.931	0.874	0.905	0.780	0.830
Gemini CLI (Gemini 3.1 Pro)	0.817	0.910	0.767	0.848	0.826	0.826
VBVR-Pro-Wan2.2-TI2V-5B *(video model)*	0.578	0.476	0.480	0.724	0.511	0.641

The open-weight models have a different profile. Perception is their strongest category by a wide margin (GLM-5 0.748, Kimi K2.5 0.727, MiniMax M2.5 0.633), Abstraction and Knowledge the weakest (GLM-5 0.478 and 0.506). Reading the first frame correctly is within reach of the mid-tier models; the categories that need a multi-step plan separate them from the closed models.

The only published per-category numbers for a video model are those of the VBVR-Pro-Wan2.2-TI2V-5B baseline (Xu et al., 2026, Table 8). We compare against its in-domain profile, and the last block of Table 3 restricts the closed agents to the same 50 tasks. All three agents lead the baseline in every category. Codex leads it by 0.099 on Transformation (0.823 versus 0.724), the video model's best category, and by 0.472 on Perception (0.948 versus 0.476), its worst, which is the strongest category of all three agents (0.910 to 0.948).

4.4. Open-Weight Models

The best open-weight coding agent beats every video model except the RL-trained one. GLM-5 through OpenCode scores 0.572, fifth among all 37 systems, above VBVR-Pro-Wan2.1-I2V-14B (0.562), VBVR-Wan2.2 (0.517), Seedance 2.0 (0.499) and the seven video models below them; only the RL-trained A14B (0.670) is ahead. Kimi K2.5 (0.557) is seventh. Out of domain, GLM-5 (0.664) and Kimi K2.5 (0.660) are above every video model, whose best OOD score is 0.547.

The open-weight ranking is a ranking of agentic coding ability. Scores fall from GLM-5 (0.572) through MiniMax M2.5 (0.451) and DeepSeek V3.2 (0.378) to a long tail below 0.3, and the *Videos* column of Table 2 tracks the drop. The top six wrote a video on 406 to 498 of 500 instances, Mistral Large 3 on 254, gpt-oss-20b on 148, Qwen3-32B on 25, Llama 4 Maverick on 7. Gemma 3 27B, Gemma 3 12B, Magistral Small and Llama 3.3 70B produced no video in 500 attempts and score 0, by the same rule that scores a video model

that fails to generate. Table 4 traces these failures to tool use; the four zero lanes never call a tool and answer in prose. Whether a lane beats the trained video models or scores 0 turns first on whether the model runs a program at all.

5. Analysis

5.1. Efficiency

The three closed agents land within 0.030 of each other, but they do not pay the same price (Tab. 4). Codex solves an instance in 3.2 tool calls, a median of 22 s and 56.8k input tokens. Claude Code uses 2.9× the calls, 4.8× the time and 4.1× the input tokens for a score 0.029 lower; Gemini CLI uses 8.0× the calls, 9.9× the time and 11.6× the input tokens for a score 0.030 lower. The difference lies in how each agent looks at the frame. Codex receives the frame as an image attached to its prompt, and a typical trace is a directory listing, one probe of a few pixels and the program, three commands in all. Claude Code opens the image with its file reader on 465 of the 500 instances, then measures coordinates with short scripts. Gemini CLI opens the image on only 10 of 500 instances. It reconstructs the scene numerically, script by script, and the 25.3 calls and 659k input tokens per instance are the cost of that reconstruction.

Figure 4 puts every lane on one axis. Calls do not buy score. The best open-weight lanes, GLM-5 and Kimi K2.5, make 48.0 and 41.9 calls per instance and read 1.57M and 1.21M input tokens, 15× and 13× Codex's calls and 28× and 21× its tokens, for 0.572 and 0.557. The lanes that call tools the least also score the least, so tool use is necessary, and yet it fails to separate the top three from the rest.

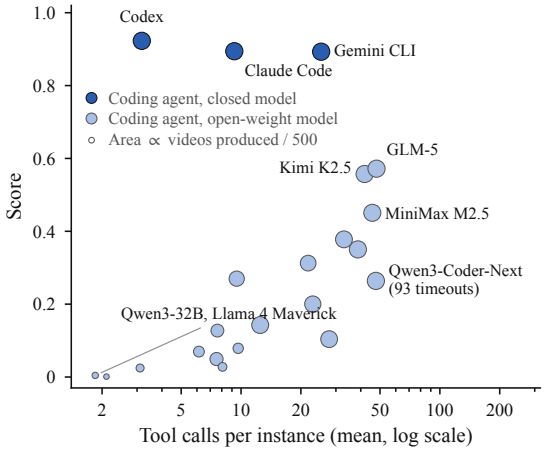


Figure 4 Score against mean tool calls per instance (log scale) for every lane that produced at least one video, the disc area proportional to the number of videos produced out of 500; the four lanes with no video are omitted. Dark blue discs are the closed agents, light blue OpenCode with an open-weight model; the three closed agents sit at the top left, and no lane to their right reaches them. Values from Tab. 4 and its full-length source, `bench/paper/table3_efficiency.csv`.

Table 4 Cost of a solution for the three closed agents and the six best open-weight lanes: per-instance means over the 500 instances (median for wall-clock seconds) from each agent's own event log. Tool use is the fraction of instances with at least one tool call, tokens are input and output per instance in thousands, and timeouts are instances killed at the 1800 s wall clock. Every lane that produced a video appears in Fig. 4.

Agent	Score	Videos	Tool use	Calls	Median s	In (k)	Out (k)	Timeouts
Codex (gpt-6-astra)	**0.923**	**500**	**1.00**	**3.2**	**22**	**57**	**1.2**	**0**
Claude Code (Fable 5.1)	0.894	500	1.00	9.2	105	233	7.3	0
Gemini CLI (Gemini 3.1 Pro)	0.893	500	1.00	25.3	218	659	7.5	0
GLM-5	0.572	495	1.00	48.0	378	1569	16.3	12
Kimi K2.5	0.557	498	0.99	41.9	336	1213	19.2	0
MiniMax M2.5	0.451	488	0.99	45.7	420	1295	19.8	0
DeepSeek V3.2	0.378	475	1.00	32.9	277	636	13.6	0
Devstral 2 123B	0.350	494	0.99	38.7	344	839	13.1	0
Kimi K2 Thinking	0.313	406	0.96	21.8	125	509	15.5	1

5.2. Failure Anatomy of the Open-Weight Lanes

An instance fails in one of two ways: a *delivery failure*, in which no `video.mp4` exists when the container exits, or a *content failure*, in which a video exists and the evaluator rejects it. Table 4 separates the two through the *Videos* column, and the source CSV further splits delivery failures by whether the agent called a tool at all.

No tool call. Gemma 3 27B, Gemma 3 12B, Llama 3.3 70B and Magistral Small never call a tool on any of their 500 instances. They answer the instruction in prose. Gemma 3 27B on G-13 writes "Okay, I will write a Python script to generate the video as described in the prompt" and then prints a script whose coordinates are `green_start = (100, 100)` with the comment "This is a placeholder, and you'll need to implement logic to detect these colors" (lane `bedrock-gemma-3-27b`, G-13, instance 00000). Llama 3.3 70B emits the tool call itself as text, `{"type": "function", "name": "write", "parameters": ...}`, with `waypoints = [(100, 100), (200, 200), (300, 300)] # Replace with actual waypoints` inside it (lane `bedrock-llama3.3-70b`, G-13, instance 00000); the call is never executed. All four score 0 for the same reason a video model that fails to generate scores 0.

One or two calls, then stop. Qwen3-32B averages 1.85 calls per instance and delivers 25 videos; Llama 4 Maverick averages 2.10 calls and delivers 7. Qwen3-32B reads a file or writes a skeleton and hands the work back: "The skeleton code for the video generator has been created at `/app/solve.py`. Next, you need to implement the logic for finding the green and red positions" (lane `bedrock-qwen3-32b`, G-13, instance 00000). Llama 4 Maverick delegates to a sub-agent type that does not exist, fetches a web page, and stops. 475 and 482 of their instances are delivery failures with a tool call on record.

Timeouts. Qwen3-Coder-Next is the opposite failure. It calls 47.7 tools per instance, reads 1.82M input tokens, delivers a video on 498 instances, and hits the 1800 s wall clock on 93 of them, more than every other lane combined; its score is 0.264.

Diligent but wrong. GLM-5, Kimi K2.5 and MiniMax M2.5 make 42–48 calls per instance, read 1.2–1.6M input tokens, and deliver a video on 495, 498 and 488 instances. Their failures are almost all content failures: their scores over delivered videos, 0.578, 0.560 and 0.462, are within 0.011 of their scores over all 500. These models run the loop the closed agents run, at a higher price, and produce a wrong video.

5.3. Where the Closed Agents Fail

On 53 of the 100 tasks all three agents score at least 0.95, and on 13 the three-agent mean is below 0.7 (Fig. 5). Table 5 lists the twelve hardest; eleven are in-domain. We read the programs and the evaluators behind these tasks and find four causes; Fig. 6 shows the agents' last frames beside the ground truth on ten tasks, three of them from Tab. 5.

The answer belongs where the diagram puts it. O-13 and O-14 show an analogy $A \to B \to C :: D \to ? \to ?$ and ask to "apply the same two-step transformation". The two `?` cells are where the answer belongs, and the top row's "filled" and "outline" styles differ only in stroke width. Claude Code animates D in place on all 10 instances of the two tasks, thinning its stroke and moving it, and scores below 0.1 on every one; Gemini CLI does so on 4 of 10. Codex writes into the `?` cells on all 10 and scores 0.9 or above on 9; on the tenth it redraws the shape instead of copying its pixels and scores 0.

Unstated conventions decide the score. On G-218 every program we read detects the three vertices and computes the largest interior angle in the same way; the scores follow the radius of the drawn circle, not the vertex. The evaluator credits a circle only if it covers 70% of an 80×80 pixel box around the vertex, and the ground-truth ring has radius 40. Programs that drew a radius of 34 or less score 0 (four of Codex's five instances, two of Claude's); those that drew 37 or more score 1.0 (all five of Gemini's). On G-43 the three agents box the same room on four of the five instances, with boxes that differ by at most 20 pixels, and score 0.00, 0.00 and 0.78 on instance 00000. The evaluator's green range, $R \leq 100$, $G \geq 100$, $B \leq 100$, contains the floorplan's grey $(100, 100, 100)$ outlines exactly at its corner. How many of them survive video encoding sets the count of "green" contours, and the count penalty removes the whole score.

A correct answer that is not the reference. G-15 asks for a shortest path around obstacles. Any shortest path

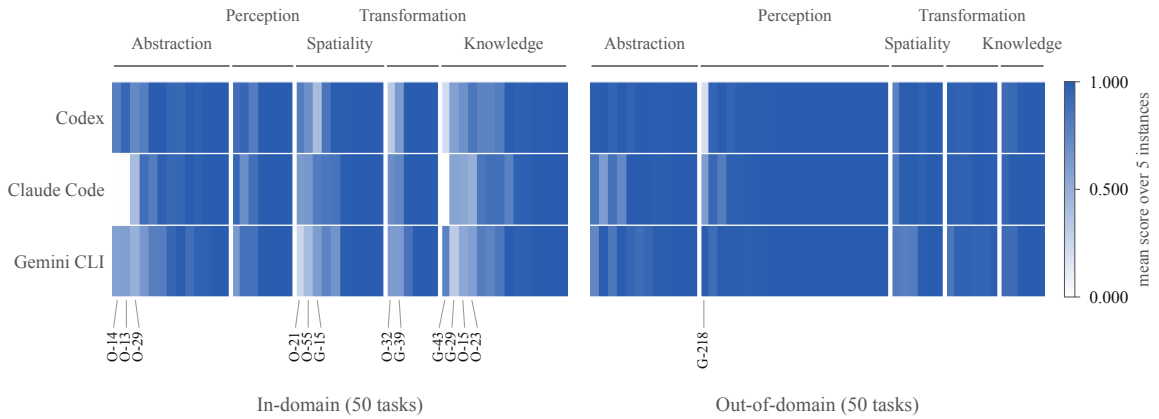


Figure 5 Per-task mean score of the three closed agents on all 100 tasks, the 50 in-domain tasks in the left block and the 50 out-of-domain tasks in the right, grouped by category (labels above). Colour is one ramp from 0 (white) to 1 (dark blue); the 13 tasks with a three-agent mean below 0.7 are named under the axis, and the twelve of Tab. 5 are among them. Values from `bench/paper/table4_per_task_closed.csv`; categories from the evaluator output.

Table 5 The twelve hardest tasks for the closed agents, ranked by the mean of the three per-task scores (each a mean over 5 instances). Split is in-domain (ID) or out-of-domain (OOD), the category is the benchmark's, and eleven of the twelve are in-domain; Sec. 5.3 groups them by cause.

Task	Name	Split	Category	Codex	Claude	Gemini	Mean
G-43	understand scene structure	ID	Knowledge	0.21	0.00	**0.77**	0.33
O-14	shape scale then outline	ID	Abstraction	**0.78**	0.01	0.58	0.45
G-29	chart extreme with data	ID	Knowledge	**0.59**	0.57	0.32	0.49
O-13	shape outline then move	ID	Abstraction	**0.94**	0.01	0.59	0.51
O-29	ballcolor	ID	Abstraction	**0.73**	0.41	0.49	0.54
O-32	rolling ball	ID	Transformation	0.32	**0.70**	0.61	0.54
O-21	construction blueprint	ID	Spatiality	**0.80**	0.62	0.24	0.55
O-55	rotation	ID	Spatiality	**0.68**	0.64	0.41	0.58
O-15	ball bounces given time	ID	Knowledge	**0.68**	0.55	0.54	0.59
G-218	identify largest angle in triangle	OOD	Perception	0.20	0.60	**1.00**	0.60
G-15	grid avoid obstacles	ID	Spatiality	0.41	**0.81**	0.61	0.61
O-23	domino chain branch path prediction	ID	Knowledge	**0.83**	0.48	0.59	0.63

is correct, but the evaluator scores proximity to the ground truth's own path. The three programs for instance 00001 each run a breadth-first search; Claude Code's tie-break reproduces the reference path and scores 1.0, Codex's and Gemini's return other paths of the same length and score 0.03 each. On instance 00002, where the shortest path is a straight line, all three score at least 0.999.

The frame under-determines the video. O-55 rotates the camera 180° around a six-block sculpture, which reveals faces that are not visible in the frame. O-32 and O-15 are scored against a simulated trajectory whose speed and timing the prompt does not give. O-29, O-21 and O-23 grade a multi-step process on the order and state of its intermediate steps. G-29 requires reading numeric labels off a pie chart, and scores swing from 0.0 to 0.96 between instances. On these tasks a program alone is not enough, and a physical or perceptual prior would help. Eleven of the twelve hardest tasks are in-domain, and this is the source of the agents' higher score on the held-out half.

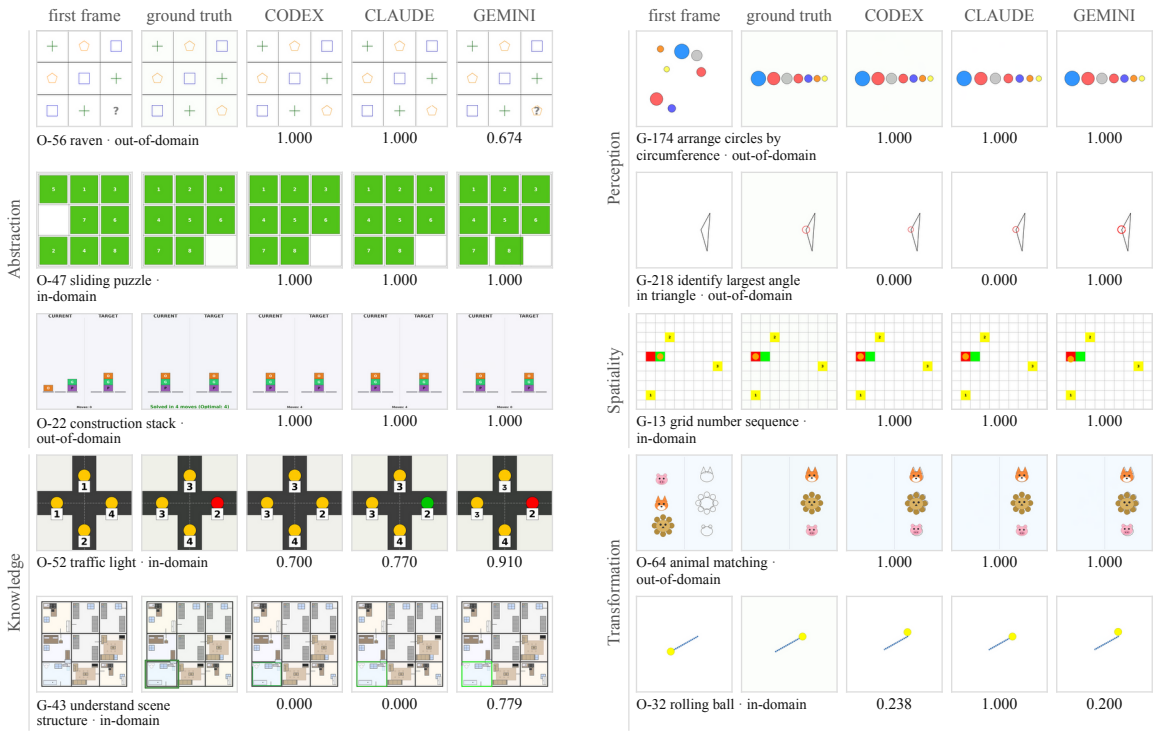


Figure 6 Last frames of the three closed agents beside the ground truth on ten tasks (instance 00000), grouped by category. Each row shows the first frame the agent was given, the ground-truth last frame and the last frame rendered by the program Codex, Claude Code and Gemini CLI wrote, with the official evaluator's score under each. Most rows are solved by all three; the failures are the unstated conventions of G-43 and G-218 (Sec. 5.3), where the drawn box or circle sits in the right place and scores 0, and the under-determined trajectory of O-32, with partial credit on O-52 and O-56.

5.4. What a Winning Program Does

Figure 7 shows the O-56 program, a 3×3 progressive matrix with an empty bottom-right cell. Codex never inspected a pixel. It read the frame attached to its prompt, stated the rule ("each row shifts the three symbols one position left"), and wrote the program in its second command. The program copies the pentagon's own pixels from the cell where it appears into the empty cell, so stroke and colour match the source exactly; it fades the question mark to white over eight frames and reveals the copied pixels in angular order over the next 25; and it asserts, on every frame, that the other eight cells are byte-identical to the input. Its 835 output tokens cover the whole run.

Figure 8 shows the G-13 program, which must guide the agent through numbered cells to the goal along shortest grid paths. Codex probed the frame once to find the sprite's bounding box, hard-coded the nine corner cells of the route from its reading of the image, with a comment giving the Manhattan length of each leg, and expanded the corners into a cell-by-cell route. Every frame is the input with the sprite erased from the start cell and re-stamped, pixel for pixel, at an interpolated position, so the board never changes.

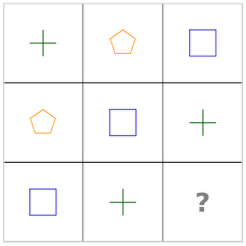
5.5. Scope of the Result

The comparison is exact because both families are graded by the same deterministic evaluator on the same 500 instances, and it is narrow for the same reason. The agents were told the container geometry, had one attempt, and were judged by rules with the sensitivities of Sec. 5.3; no human looked at a video. The result concerns the interface, a program against a denoiser, on a benchmark built to be verifiable. It does not show that the agents perceive the frame better than a video model in general.

```python
img = Image.open('first_frame.png').convert('RGB')
base = np.array(img)
# Each row shifts the three symbols one position left:
# square, plus, pentagon. Copy the pentagon's pixels.
source = base[420:590, 85:255]
mask = ((source[:,:,0] > 200) & (source[:,:,1] > 60)
        & (source[:,:,1] < 200) & (source[:,:,2] < 60))
sy, sx = np.where(mask); colors = source[sy, sx]
ty, tx = sy + 761, sx + 767
angles = np.arctan2(tx - 852, -(ty - 852)) % (2*np.pi)
question = np.zeros(base.shape[:2], dtype=bool)  # "?"
question[790:915, 810:895] = np.any(
    base[790:915, 810:895] != 255, axis=2)
for i in range(35):
    frame = base.copy()
    fade = min(i / 8, 1)
    q = base[question].astype(float)*(1-fade) + 255*fade
    frame[question] = np.rint(q).astype(np.uint8)
    progress = np.clip((i-8)/25, 0, 1)
    if progress > 0:
        visible = angles <= progress*2*np.pi
        frame[ty[visible],tx[visible]] = colors[visible]
    assert (frame[:683] == base[:683]).all()
    assert (frame[683:,:683] == base[683:,:683]).all()
    p.stdin.write(frame.tobytes())  # -> video.mp4
```

solve.py written by Codex for O-56 raven, instance 00000

first frame



Codex last frame · score 1.000

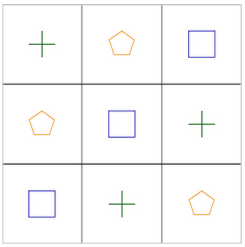
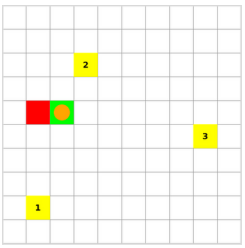


Figure 7 The Codex program for O-56 (raven, out-of-domain, instance 00000; 2 tool calls, 17 s, score 1.0) beside the task's first frame and the last frame it renders. The program is the agent's own `solve.py`, trimmed to the lines discussed in the text; it copies the pentagon's pixels from the cell where they appear into the empty cell and asserts on every frame that the other eight cells are unchanged.

```python
img = Image.open('first_frame.png').convert('RGB')
original = np.array(img)
orange = np.all(original == (255,165,0), axis=2) # agent
sy, sx = np.where(orange); dx, dy = sx - 255, sy - 459
board = original.copy(); board[orange] = (0, 255, 0)
# Each leg has Manhattan-minimal length: 5, 8, 8, 8.
corners = [(2,4), (2,8), (1,8), (3,8), (3,2),
           (8,2), (8,5), (8,4), (1,4)]
route = [corners[0]]
for target in corners[1:]:
    x, y = route[-1]
    while (x, y) != target:
        x += int(np.sign(target[0]-x))
        y += int(np.sign(target[1]-y))
        route.append((x, y))
centers = np.array([(51+102*x, 51+102*y)
                    for x, y in route])
for frame_idx in range(107):   # hold, travel, hold
    t = np.clip((frame_idx-3)/99, 0, 1) * (len(route)-1)
    i = min(int(t), len(route)-2)
    p = np.rint(centers[i]*(1-(t-i))
                + centers[i+1]*(t-i)).astype(int)
    frame = board.copy()
    frame[p[1]+dy, p[0]+dx] = (255, 165, 0)
    proc.stdin.write(frame.tobytes())  # -> video.mp4
```

solve.py written by Codex for G-13 grid number sequence, instance 00000

first frame



Codex last frame · score 1.000

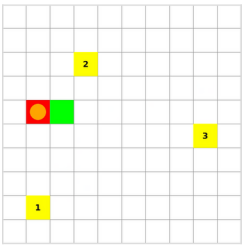


Figure 8 The Codex program for G-13 (grid number sequence, in-domain, instance 00000; 3 tool calls, 20 s, score 1.0) beside the task's first frame and the last frame it renders. The nine corner cells of the route are written down from the image, each leg annotated with its Manhattan length, and the sprite is erased and re-stamped pixel for pixel on every frame.

6. Conclusion

VBVR-Pro was built to ask whether generative models can reason in the visual medium. We kept its tasks, its inputs, its output container and its evaluator, and changed only how the answer is produced: a coding agent writes a program that renders the video instead of denoising it. On VBVR-Pro-Bench that one change moves the best score from 0.670, for a video model RL-trained on the benchmark's own in-domain families, to 0.923,

and all three closed-model agents exceed 0.89. The direction of generalisation flips as well. Trained video models fall on the held-out families and coding agents rise on them, so the agents' advantage is largest where the video models were not trained. Among open-weight models the best reaches 0.572, and the weakest fail before reasoning begins, by never using a tool or by giving up after one or two calls. On tasks whose answers are programs, a solver that writes the program is scored on the reasoning alone, while a solver that paints the pixels is scored on the reasoning and the painting at once. The gap measures that difference; it does not show that video models cannot reason.

Limitations. The official evaluator is used unmodified and inherits its properties. On a fresh seed a frozen first frame scores above 0.5 on four of the 100 task families and the generator's own ground truth scores below 0.95 on six, so a small part of every score is evaluator noise. All results are on one benchmark whose tasks are generated and rendered by code, the setting in which writing code is most natural. The comparison is between an agent with tools, a sandbox and up to 30 minutes of iteration and a video model that emits one sample in one shot; it measures what each route delivers under its own constraints, and says nothing about what each model could do under the other's. Agent cost is reported in tokens and wall-clock rather than dollars, and varies by two orders of magnitude across models. The natural next step is to train the code-writing policy with GRPO (Shao et al., 2024) using the official evaluator as the reward, mirroring the protocol VBVR-Pro applied to video models (Xu et al., 2026). The reward pipeline and a training step on AWS Trainium are in place, and no result from it is reported here.

References

Anthropic. Claude code. Software, version 2.1.268, 2026. URL https://github.com/anthropics/claude-code.

ByteDance Seed. Seedance 1.0: Exploring the boundaries of video generation models, 2025.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research (TMLR)*, 2023.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799, 2023.

Google. Gemini CLI. Software, version 0.59.0, 2026. URL https://github.com/google-gemini/gemini-cli.

Google DeepMind. Veo 3. Model card and technical documentation, 2025. URL https://deepmind.google/models/veo/.

Gupta, T. and Kembhavi, A. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14953–14962, 2023.

He, X., Jiang, D., Zhang, G., Ku, M., Soni, A., Siu, S., Chen, H., Chandra, A., Jiang, Z., Arulraj, A., Wang, K., Do, Q. D., Ni, Y., Lyu, B., Narsupalli, Y., Fan, R., Lyu, Z., Lin, Y., and Chen, W. VideoScore: Building automatic metrics to simulate fine-grained human feedback for video generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

Hu, Y., Shi, W., Fu, X., Roth, D., Ostendorf, M., Zettlemoyer, L., Smith, N. A., and Krishna, R. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Huang, Z., He, Y., Yu, J., Zhang, F., Si, C., Jiang, Y., Zhang, Y., Wu, T., Jin, Q., Chanpaisit, N., Wang, Y., Chen, X., Wang, L., Lin, D., Qiao, Y., and Liu, Z. VBench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Kuaishou Technology. Kling. Video generation model, 2025. URL https://klingai.com.

Laude Institute. Harbor: A framework for packaging and running agent tasks. Software, 2025. URL https://github.com/laude-institute/harbor.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Liu, J., Liu, G., Liang, J., Li, Y., Liu, J., Wang, X., Wan, P., Zhang, D., and Ouyang, W. Flow-GRPO: Training flow matching models via online RL, 2025.

Motamed, S., Culp, L., Swersky, K., Jaini, P., and Geirhos, R. Do generative video models understand physical principles?, 2025.

OpenAI. Video generation models as world simulators. Technical report, 2024. URL https://openai.com/research/video-generation-models-as-world-simulators.

OpenAI. Codex CLI. Software, version 0.154.0, 2026. URL https://github.com/openai/codex.

OpenCode contributors. OpenCode. Software, version 1.18.30, 2026. URL https://github.com/sst/opencode.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024.

Surís, D., Menon, S., and Vondrick, C. ViperGPT: Visual inference via Python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11888–11898, 2023.

Tong, J., Mou, Y., Li, H., Li, M., Yang, Y., Zhang, M., Chen, Q., Liang, T., Hu, X., Zheng, Y., Chen, X., Zhao, J., Huang, X., and Qiu, X. Thinking with video: Video generation as a promising multimodal reasoning paradigm, 2025.

von Werra, L., Belkada, Y., Tunstall, L., Beeching, E., Thrush, T., Lambert, N., Huang, S., Rasul, K., and Gallouédec, Q. TRL: Transformer reinforcement learning. Software, 2020. URL https://github.com/huggingface/trl.

Wan Team. Wan: Open and advanced large-scale video generative models, 2025.

Wiedemer, T., Li, Y., Vicol, P., Gu, S. S., Matarese, N., Swersky, K., Kim, B., Jaini, P., and Geirhos, R. Video models are zero-shot learners and reasoners, 2025.

Xu, J., Wang, R., Pu, F., Wang, M., Ji, R., Zhou, T., Gu, C., Zuo, J., Xiao, H., Geng, Y., Yin, W., Chen, W., Qian, O., Yan, Z., Huang, Z., Diao, H., Pan, L., Li, B., Fan, X., Luo, D., Yu, F., Zhao, Z., Gao, Q., Zhu, T., Zhang, Y., Tong, J., Feng, P., Jiang, Z., Wang, L., Guo, Z., Zhang, R., Chen, J., Joseph, S., Venhoff, C., Motamed, S., Yang, M., Sripada, C., Yuille, A., Torr, P., Zhang, L., Kumar, V., Khashabi, D., Kriegeskorte, N., Millière, R., Müller, V. C., Rao, A., Wang, Q., Liu, Z., Lin, D., Yang, L., Deng, H., and Cai, Z. VBVR-Pro: A scalable and verifiable suite for native visual reasoning, 2026. URL https://arxiv.org/abs/2608.26105.

Xue, Z., Wu, J., Gao, Y., Kong, F., Zhu, L., Chen, M., Liu, Z., Liu, W., Guo, Q., Huang, W., and Luo, P. DanceGRPO: Unleashing GRPO on visual generation, 2025.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

A. Evaluator Environment

Every score in this paper comes from the unmodified official VBVR-Pro-Bench evaluator (`run_evaluation_video.py`, 100 task-specific rule-based evaluators, `task_specific_only`, CPU) (Xu et al., 2026). The evaluator is run in an environment built from the kit's own `requirements.txt`: `numpy<2` (norfair requires it), OpenCV 4.x, norfair and easyocr. That environment is kept separate from the one the task generators use (numpy 2, OpenCV 5).

The separation matters because the evaluator fails silently. In an environment with numpy 2 and no norfair, four evaluators return 0 for every video they are given, including the benchmark's own reference videos. These are G-5 and G-8, where the norfair import fails, and O-18 and O-19, where numpy 2 rejects the kit's scalar indexing with `invalid index to scalar variable`. The kit runs its evaluators with `fail_on_error` off and maps an unsupported or failed evaluator to 0, so nothing appears in the log and the four tasks simply score 0 for every model. Our first scoring of the three closed-model lanes was done in such an environment, and the in-domain scores of these four tasks were 0 for every agent. All lanes were re-scored in the conforming environment, and every number in this paper is from that re-scoring.

The check that catches this is cheap, and we recommend it before trusting any number from the kit. Score the benchmark's own reference videos and require every one of them to score 1.0. In the conforming environment they do; in the non-conforming one, the four tasks above score 0. Sec. D applies the same self-check to freshly generated instances of all 100 families, where the generator's own ground truth is the reference.

B. Open-Weight Roster

Tab. 6 lists the 24 open-weight models evaluated in Tab. 2. All 24 run through the same agent, OpenCode 1.18.30 (OpenCode contributors, 2026), inside the same container and under the same limits as the three closed-model agents (Sec. F), and are served by Amazon Bedrock in `us-east-1`. Model ids are given as `aws bedrock list-foundation-models` prints them. Twenty models are reached through Bedrock's OpenAI-compatible endpoint. The three Llama models are not served by that endpoint, and their streaming Converse endpoint rejects tool use, so they run through a small shim on the host (`bench/bedrock_proxy.py`) that exposes the non-streaming Converse API behind an OpenAI-compatible interface. DeepSeek R1 has no tool use on Bedrock in either mode (checked 2026-09-14); a model that cannot call a tool cannot write, run or save a program, so the lane is marked unsupported after 401 attempts produced no video.

Five lanes produced no video at all: DeepSeek R1 (unsupported), and Gemma 3 27B, Gemma 3 12B, Magistral Small and Llama 3.3 70B, which never call a tool and answer the task in prose. Timeouts at the 1800 s wall clock are rare except for Qwen3-Coder-Next (93 of 500 instances) and GLM-5 (12); every other lane has at most one.

Table 6 The 24 open-weight models, all run through OpenCode 1.18.30 against Amazon Bedrock. *Route* is the Bedrock OpenAI-compatible endpoint, a Converse-API shim (`bench/bedrock_proxy.py`) for models that endpoint does not serve and whose streaming Converse endpoint rejects tool use, or unsupported (no tool use on Bedrock at all, so the model cannot act as a coding agent); *Produced* is the number of instances, out of 500, for which the agent wrote a video, missing videos score 0, and *Score* is the mean official-evaluator score over 500 instances. DeepSeek R1 was attempted on 401 instances, produced no video, and is marked unsupported.

Model	Bedrock model id	Route	Produced / 500	Score
GLM-5	`zai.glm-5`	OpenAI-compatible	495	**0.572**
Kimi K2.5	`moonshotai.kimi-k2.5`	OpenAI-compatible	498	0.557
MiniMax M2.5	`minimax.minimax-m2.5`	OpenAI-compatible	488	0.451
DeepSeek V3.2	`deepseek.v3.2`	OpenAI-compatible	475	0.378
Devstral 2 123B	`mistral.devstral-2-123b`	OpenAI-compatible	494	0.350
Kimi K2 Thinking	`moonshot.kimi-k2-thinking`	OpenAI-compatible	406	0.313
gpt-oss-120b	`openai.gpt-oss-120b-1:0`	OpenAI-compatible	380	0.270
Qwen3-Coder-Next	`qwen.qwen3-coder-next`	OpenAI-compatible	498	0.264
Nemotron Super 3 120B	`nvidia.nemotron-super-3-120b`	OpenAI-compatible	436	0.200
Qwen3-Coder-30B-A3B	`qwen.qwen3-coder-30b-a3b-v1:0`	OpenAI-compatible	480	0.143
Mistral Large 3 675B	`mistral.mistral-large-3-675b-instruct`	OpenAI-compatible	254	0.127
GLM-4.7-Flash	`zai.glm-4.7-flash`	OpenAI-compatible	470	0.104
gpt-oss-20b	`openai.gpt-oss-20b-1:0`	OpenAI-compatible	148	0.079
Qwen3-Next-80B-A3B	`qwen.qwen3-next-80b-a3b`	OpenAI-compatible	167	0.070
Llama 4 Scout	`us.meta.llama4-scout-17b-instruct-v1:0`	Converse shim	269	0.049
Nemotron Nano 3 30B	`nvidia.nemotron-nano-3-30b`	OpenAI-compatible	91	0.028
Qwen3-VL-235B-A22B	`qwen.qwen3-vl-235b-a22b`	OpenAI-compatible	70	0.025
Qwen3-32B	`qwen.qwen3-32b-v1:0`	OpenAI-compatible	25	0.004
Llama 4 Maverick	`us.meta.llama4-maverick-17b-instruct-v1:0`	Converse shim	7	0.001
DeepSeek R1	`us.deepseek.r1-v1:0`	unsupported	0 / 401	0.000
Gemma 3 12B	`google.gemma-3-12b-it`	OpenAI-compatible	0	0.000
Gemma 3 27B	`google.gemma-3-27b-it`	OpenAI-compatible	0	0.000
Llama 3.3 70B	`us.meta.llama3-3-70b-instruct-v1:0`	Converse shim	0	0.000
Magistral Small	`mistral.magistral-small-2509`	OpenAI-compatible	0	0.000

C. Per-Task Results of the Closed-Model Agents

Tab. 7 gives the score of each of the three closed-model agents on every one of the 100 tasks, with the task's split and cognitive category as the benchmark assigns them. The split halves are not balanced by category: the 50 in-domain tasks are 14 Knowledge, 13 Abstraction, 10 Spatiality, 7 Perception and 6 Transformation; the 50 out-of-domain tasks are 21 Perception, 12 Abstraction, 6 Spatiality, 6 Transformation and 5 Knowledge. On 33 of the 100 tasks all three agents score 1.000 on all five instances. The lowest means are on G-43 (`understand_scene_structure`: 0.215, 0.003 and 0.770 for Codex, Claude Code and Gemini CLI), O-14 (`shape_scale_then_outline`: 0.779, 0.008, 0.576) and G-29 (`chart_extreme_with_data`: 0.586, 0.566, 0.320); Tab. 5 and the analysis section discuss them.

Table 7 Per-task scores of the three closed-model coding agents on VBVR-Pro-Bench. Each entry is the mean official-evaluator score over the five instances of the task, an instance without a video counts 0, and the best of the three is in bold; split means and the overall means match Table 2. In-Domain tasks (50) have training data in VBVR-Pro; Out-of-Domain tasks (50) are held out from all training.

Task	Generator	Split	Category	Codex	Claude Code	Gemini CLI
G-3	`stable_sort`	ID	Perception	**1.000**	**1.000**	**1.000**
G-5	`multi_object_placement`	ID	Transformation	**1.000**	**1.000**	**1.000**
G-8	`track_object_movement`	ID	Spatiality	**1.000**	**1.000**	**1.000**
G-9	`identify_objects_in_region`	ID	Perception	0.898	**0.938**	0.627
G-13	`grid_number_sequence`	ID	Spatiality	**1.000**	**1.000**	**1.000**
G-15	`grid_avoid_obstacles`	ID	Spatiality	0.414	**0.805**	0.609
G-16	`grid_go_through_block`	ID	Spatiality	**0.845**	0.843	0.758
G-18	`grid_shortest_path`	ID	Spatiality	**1.000**	**1.000**	**1.000**
G-21	`multiple_occlusions_vertical`	ID	Knowledge	**0.996**	**0.996**	**0.996**

continued on next page

Table 7 continued

Task	Generator	Split	Category	Codex	Claude Code	Gemini CLI
G-25	seperate_object_spinning	ID	Transformation	**0.999**	**0.999**	0.857
G-29	chart_extreme_with_data	ID	Knowledge	**0.586**	0.566	0.320
G-31	directed_graph_navigation	ID	Spatiality	0.987	**0.989**	**0.989**
G-39	attention_shift_different	ID	Transformation	0.620	**0.740**	0.618
G-41	grid_highest_cost	ID	Knowledge	**1.000**	0.770	**1.000**
G-43	understand_scene_structure	ID	Knowledge	0.215	0.003	**0.770**
G-45	key_door_matching	ID	Spatiality	**1.000**	**1.000**	**1.000**
G-51	predict_next_color	ID	Abstraction	0.926	0.934	**0.995**
G-131	select_next_figure_increasing_size_sequence	ID	Abstraction	**1.000**	0.799	0.786
G-134	select_next_figure_large_small_alternating_sequence	ID	Abstraction	**1.000**	**1.000**	0.971
G-138	spot_unique_non_repeated_color	ID	Perception	0.804	**0.883**	0.874
G-158	identify_all_hollow_points	ID	Perception	**1.000**	**1.000**	**1.000**
G-194	construct_concentric_ring	ID	Transformation	**1.000**	**1.000**	**1.000**
O-10	shape_outline_fill	ID	Abstraction	**0.998**	0.977	0.794
O-12	shape_color_then_scale	ID	Abstraction	**0.970**	0.969	0.964
O-13	shape_outline_then_move	ID	Abstraction	**0.941**	0.011	0.591
O-14	shape_scale_then_outline	ID	Abstraction	**0.779**	0.008	0.576
O-15	ball_bounces_given_time	ID	Knowledge	**0.675**	0.547	0.542
O-16	color_addition	ID	Knowledge	**1.000**	**1.000**	**1.000**
O-18	glass_refraction	ID	Knowledge	**1.000**	0.999	0.995
O-19	mirror_reflection	ID	Knowledge	0.765	**0.840**	0.797
O-21	construction_blueprint	ID	Spatiality	**0.798**	0.619	0.243
O-23	domino_chain_branch_path_prediction	ID	Knowledge	**0.833**	0.482	0.588
O-24	domino_chain_gap_analysis	ID	Knowledge	0.758	**0.886**	0.758
O-25	LEGO_construction_assembly	ID	Spatiality	**1.000**	0.852	0.662
O-29	ballcolor	ID	Abstraction	**0.728**	0.411	0.487
O-30	bookshelf	ID	Abstraction	**0.938**	0.923	**0.938**
O-31	ball_eating	ID	Abstraction	0.793	**0.890**	0.629
O-32	rolling_ball	ID	Transformation	0.324	**0.697**	0.612
O-33	counting_object	ID	Perception	**0.935**	0.693	0.872
O-34	dot_to_dot_task	ID	Knowledge	0.970	0.970	**0.980**
O-36	grid_shift	ID	Transformation	0.993	0.996	**0.999**
O-37	light_sequence	ID	Perception	**1.000**	**1.000**	**1.000**
O-38	majority_color	ID	Perception	**1.000**	**1.000**	**1.000**
O-44	rotation_puzzle	ID	Abstraction	0.990	**0.993**	0.893
O-45	sequence_completion	ID	Abstraction	**1.000**	**1.000**	**1.000**
O-47	sliding_puzzle	ID	Abstraction	**1.000**	**1.000**	**1.000**
O-52	traffic_light	ID	Knowledge	0.799	**0.876**	0.835
O-53	clock	ID	Knowledge	0.977	**0.983**	0.980
O-55	rotation	ID	Spatiality	**0.681**	0.635	0.412
O-75	communicating_vessels	ID	Knowledge	0.992	**1.000**	0.999
ID mean (50 tasks)				**0.879**	0.830	0.826
G-24	separate_objects_no_spin	OOD	Transformation	**0.964**	**0.964**	**0.964**
G-47	multiple_keys_for_one_door	OOD	Spatiality	**0.999**	**0.999**	**0.999**
G-54	connecting_color	OOD	Perception	**1.000**	**1.000**	**1.000**
G-135	select_next_figure_small_large_alternating_sequence	OOD	Abstraction	**1.000**	**1.000**	**1.000**
G-136	locate_point_in_overlapping_area	OOD	Perception	**1.000**	**1.000**	**1.000**
G-140	locate_topmost_unobscured_figure	OOD	Spatiality	0.966	0.978	**0.988**
G-147	identify_unique_figure_in_uniform_set	OOD	Perception	**1.000**	**1.000**	**1.000**
G-160	circle_largest_numerical_value	OOD	Knowledge	**1.000**	**1.000**	**1.000**
G-161	mark_second_largest_shape	OOD	Perception	**1.000**	**1.000**	**1.000**
G-167	select_longest_polygon_side	OOD	Perception	**1.000**	0.800	**1.000**
G-168	identify_nearest_to_square_rectangle	OOD	Perception	**1.000**	**1.000**	0.999
G-169	locate_intersection_of_segments	OOD	Perception	0.993	**1.000**	0.986
G-174	arrange_circles_by_circumference	OOD	Perception	**1.000**	**1.000**	**1.000**
G-189	draw_midpoint_perpendicular_line	OOD	Perception	0.986	**0.992**	0.978
G-193	draw_next_sized_shape	OOD	Abstraction	**0.989**	**0.989**	0.987
G-202	mark_wave_peaks	OOD	Perception	**1.000**	**1.000**	**1.000**
G-206	identify_pentagons	OOD	Perception	**1.000**	**1.000**	**1.000**
G-212	find_incorrect_arrow_direction	OOD	Perception	**1.000**	**1.000**	**1.000**
G-217	circle_central_dot	OOD	Perception	**1.000**	**1.000**	**1.000**
G-218	identify_largest_angle_in_triangle	OOD	Perception	0.200	0.600	**1.000**
G-219	select_leftmost_shape	OOD	Spatiality	**0.988**	**0.988**	0.789
G-221	outline_innermost_square	OOD	Spatiality	0.777	**0.818**	0.808
G-222	mark_tangent_point_of_circles	OOD	Perception	**1.000**	**1.000**	**1.000**
G-223	highlight_horizontal_lines	OOD	Perception	0.988	**1.000**	**1.000**
G-240	add_borders_to_unbordered_shapes	OOD	Perception	**0.946**	0.930	0.905
G-247	identify_chinese_character	OOD	Knowledge	**1.000**	**1.000**	**1.000**
G-248	mark_asymmetrical_shape	OOD	Perception	**1.000**	**1.000**	**1.000**
G-250	color_triple_intersection_red	OOD	Perception	0.982	0.972	**0.983**
G-273	high_density_liquid	OOD	Knowledge	0.905	**0.935**	0.900
O-2	pigment_color_mixing_subtractive	OOD	Knowledge	**1.000**	**1.000**	**1.000**
O-5	symbol_deletion	OOD	Abstraction	**1.000**	0.606	**1.000**

continued on next page

Table 7 continued

Task	Generator	Split	Category	Codex	Claude Code	Gemini CLI
O-6	2d_geometric_transformation	OOD	Transformation	**0.970**	**0.970**	0.842
O-9	shape_scaling	OOD	Abstraction	**0.990**	0.982	0.988
O-11	shape_color_then_move	OOD	Abstraction	**0.998**	0.732	0.991
O-22	construction_stack	OOD	Abstraction	**0.983**	0.850	0.745
O-27	move_2_object_to_2_target	OOD	Transformation	**1.000**	**1.000**	0.948
O-39	maze	OOD	Spatiality	**1.000**	**1.000**	**1.000**
O-43	object_subtraction	OOD	Perception	0.949	**0.981**	**0.981**
O-46	shape_sorter	OOD	Transformation	0.998	**0.999**	0.949
O-49	symmetry_completion	OOD	Spatiality	**1.000**	**1.000**	0.800
O-54	control_panel	OOD	Abstraction	**0.949**	0.870	0.837
O-56	raven	OOD	Abstraction	**1.000**	**1.000**	0.935
O-58	symbol_delete	OOD	Abstraction	**1.000**	**1.000**	**1.000**
O-59	symbol_insert	OOD	Abstraction	0.960	**1.000**	0.936
O-60	symbol_substitute	OOD	Abstraction	**1.000**	**1.000**	**1.000**
O-61	symbol_edit	OOD	Abstraction	**1.000**	**1.000**	0.896
O-62	gravity_physics	OOD	Knowledge	0.924	**1.000**	0.905
O-64	animal_matching	OOD	Transformation	**0.991**	**0.991**	0.982
O-65	animal_size_sorting	OOD	Perception	**1.000**	**1.000**	**1.000**
O-85	2d_object_rotation	OOD	Transformation	0.960	**0.962**	0.961
OOD mean (50 tasks)				**0.967**	0.958	0.960
Overall mean (100 tasks)				**0.923**	0.894	0.893

D. Reward Validation and Training Setup (Future Work)

No RL results are reported in this paper. This section documents the reward and the training setup we built and the checks they passed. It reports no training curve and no trained checkpoint.

Reward. The reward is the official evaluator, applied to freshly generated instances rather than to the benchmark's fixed ones. VBVR-Pro's evaluators are keyed by generator, so any instance of one of the 100 evaluator-covered generators can be scored, at any seed. The environment runs the generator at a fresh seed and writes the instance (first frame, prompt, ground-truth video, final frame, metadata); the policy's program is executed in a sandbox and its `output/video.mp4` is scored by `get_evaluator(task).evaluate(..., task_specific_only=True)`, the same call the benchmark uses. No hand-written verifier is used anywhere.

Validation. Before training on it, the reward was checked for discrimination on all 100 evaluator families (Tab. 8). For each family, one instance is built at a fresh seed; the generator's own ground-truth video is scored, and so is a video that freezes the first frame for the full duration, which is the answer of a policy that does nothing. A family is discriminating when the ground truth scores at least 0.95 and the frozen frame at most 0.5. Ninety of the 100 families pass. Six score their own ground truth below 0.95 on a fresh seed: G-47 (0.00, the generator's `optimal_path_infeasible` case), O-27 (0.45), O-24 (0.72), O-53 (0.75), G-167 (0.87) and O-46 (0.90). Four score a frozen first frame above 0.5: O-58 (1.00, so doing nothing scores full), O-39 (0.64), O-54 (0.59) and O-15 (0.58). The median frozen-frame score is 0.00 and the mean 0.11; a truth-plus-frozen pair takes 5.3 s to score. Of the 50 in-domain families that training would use, only O-24 and O-53 (ground truth below 0.95) and O-15 (frozen frame 0.58) are affected. These are properties of the official evaluator and are kept as they are; the benchmark's own RL training used the same scorer.

Table 8 Reward validation on all 100 evaluator families (fresh seed 4242, one instance per family, 2026-09-13): for each family the generator's own ground-truth video and a frozen copy of the first frame are scored by the official evaluator in the conforming environment. A family is *discriminating* when ground truth scores ≥ 0.95 and the frozen frame ≤ 0.5; the two lists below are disjoint. Families marked † are among the 50 in-domain families used for training.

Check	Families
Ground truth ≥ 0.95 and frozen frame ≤ 0.5	90 / 100
Ground truth < 0.95 on a fresh seed	6
Frozen frame > 0.5	4
Frozen-frame score, median / mean	0.00 / 0.11
Evaluator time per (truth, frozen) pair	5.3 s
Ground truth < 0.95 (family: score of its own ground truth)	
G-47	0.00
O-27	0.45
O-24†	0.72
O-53†	0.75
G-167	0.87
O-46	0.90
Frozen frame > 0.5 (family: score of the frozen first frame)	
O-58	1.00
O-39	0.64
O-54	0.59
O-15†	0.58

Training setup. The setup mirrors the RLVR protocol that VBVR-Pro applied to a video diffusion model (Xu et al., 2026; Xue et al., 2025), with the policy swapped for a code LLM and the action for a program. The policy is a dense open-weight code LLM trained with GRPO (Shao et al., 2024) through the TRL `GRPOTrainer` (von Werra et al., 2020), with $G = 16$ programs per prompt, 16 prompts per step and no KL term. The interaction is single-turn: the prompt is the task prompt, an answer-free textual perception dump of the first frame (colour buckets, connected components, their centres and boxes; 73–402 tokens, replacing the "open the image with OpenCV" step every coding agent performed first on the benchmark) and the output contract, and the completion is one Python program. The program runs in a sandbox (temporary directory, 180 s wall clock, 4 GB address space, 256 processes, no network where user namespaces allow it) and must write `output/video.mp4`. The reward is graded so that early groups have variance: 0 for a program that does not parse, 0.05 for one that crashes, 0.10 for one that runs without writing a video, and $0.15 + 0.85\times$ the evaluator score otherwise, so a perfect video still scores 1.0. Training instances come from the 50 in-domain families at fresh seeds (seed base 100,000, so the benchmark's and the RL split's own seeds are avoided by construction); the benchmark instances are never trained on and the 50 out-of-domain families are never seen, as in the benchmark's protocol. Evaluation is the benchmark itself, run with `bench/run_bench.py` against the current checkpoint. A GRPO training step of this pipeline was run end to end on AWS Trainium (Qwen3-0.6B policy, 4 ranks, native PyTorch on Neuron with FSDP through Accelerate, reward sandboxes on the host CPUs). The training run itself is future work.

E. Known Limitations of the Source Tasks

For the reward check we packaged all 300 VBVR-Pro generators as tasks mechanically: the prompt is the generator's prompt and the truth is the generator's video. The 100 benchmark families are among them (G and O ids; T ids denote the final `generator-fixes` set of the remaining generators). Build and audit passes read every prompt against its first frame. The recurring gaps, by task id, are listed below; they are properties of the source tasks and are left as they are.

- **Truth or prompt is wrong.** T094 (the queried cell is provably safe but labelled N), T097 (the played move loses to a mate next turn), T118 (prompt says the cubes stay blue; the truth ends grey), O-9 (a "200 %" enlargement renders as a 0.86 shrink), O-44 (rotations of 92° and 105° under a "90°" rule), O-52 (a yellow light's next colour is not derivable), O-62 (prompt elasticity 0.80, simulation 0.7), O-7 (colour names print as "unknown"), O-11 and O-12 (colours named `color_NNN`). O-13's prompt contradicted its render and is patched to match. O-37's default seed produced a no-op sample (the initial state already satisfied the target); the registry overrides its seed.
- **Answer not unique.** G-4/T004, G-13, G-16, G-41, T012 (tied shortest routes); T080, T083 (several legal moves, the truth picks one at random); T113 (several paths); T135 (42 valid nonograms); T141 (an 18-move solution among many); T142 (several valid colourings).
- **Prompt written for a text or multiple-choice answer, video conventions undescribed.** T054, T061, T084, T086, T090, T096, T100, T104, T108, T114, T115, T120, T121, O-55. (T125–T150 are all written as video prompts.)
- **Essential state only in metadata.** T031 (sprite art and motion profile), T033 (snake route), T007 (command list), T028 (cube placement), T049/T050 (the missing shape's geometry), T077 (target layout), T102 (generator scoring rules), T116 (random curve), T119 (camera), G-3 (row layout), G-54 (free curve), O-16/O-17 (mixing formula), O-66 (slot rule), O-87 (stochastic fluid).
- **Truth adds elements the prompt never mentions** (captions, header text, highlight boxes, construction lines, pre-animations): T032, T056, T058, T059, T088, T089, T099, T101, T105, T117, T120, T122, T123, T124, T144, and unstated fill or loop colours on several T134–T150 puzzles.
- **Styling is unstated** (ring radius, stroke width, exact shade, pacing) on most G-series marking tasks; the evaluator's 2 px boundary tolerance and 15 % time tolerance absorb some of it, not all. The current prompts add "Only modify what the task requires; keep the background and any other element unchanged" everywhere and state a few more conventions (trail colour on G-48, red circles on G-131 and G-133, simultaneous motion on G-5, G-24, G-25 and G-40).
- **Small or late changes.** On about a quarter of the tasks a frozen first frame already clears the video gate and only the final-frame gate rejects it; G-11 is the reverse (the object returns to its start, so only the video gate discriminates). Tab. 8 lists the families where the official evaluator itself does not separate a frozen frame from the truth.
- **Rendering environment.** Truth videos were rendered on macOS; the generators ship their own fonts, but those that look up system fonts by name can render differently inside the container. T036 and T037 render with Blender (10–60 minutes on CPU), which the task image does not install. Some generators need packages beyond the agent image (matplotlib, networkx, scipy, shapely, python-chess, pymunk); the oracle installs them, the agent image does not carry them.

F. Compute and Cost

All agent runs were made between 2026-09-12 and 2026-09-16 on a single Amazon EC2 `c7i.4xlarge` instance (16 vCPUs, no GPU). Each instance of each lane runs in its own Docker container (`harbor/agent/`: python 3.11, ffmpeg, numpy, Pillow, OpenCV, imageio, scipy, matplotlib, and the four agent CLIs) with 2 CPUs, 3 GB of memory and no swap, a 512-process limit and a wall clock of 1800 s, one attempt per instance; several containers run in parallel on the host. Model inference is remote for every lane (the closed models through their vendors' APIs or Amazon Bedrock, the open-weight models through Amazon Bedrock), so the host does no model computation. The official evaluator also runs on the host CPU. Token usage is taken from the agents' own event logs. Over the 500 instances, Codex with gpt-6-astra consumed 28.4 M input and 0.58 M output tokens (56.8 k and 1.2 k per instance); Claude Code with Fable 5.1, 116.7 M input and 3.66 M output tokens (233 k and 7.3 k); Gemini CLI with Gemini 3.1 Pro, 329 M input and 3.77 M output tokens (659 k and 7.5 k). Input counts include cached prompt tokens as each CLI reports them. Dollar cost was not measured.