PairCoder++: Pair Programming as a Universal Paradigm for Verified Code-Driven Multimodal and Structured-Artifact Generation

Junhao Chen[1] Xiang Li[2] Mingjin Chen[3] Boran Zhang[4] Henghaofan Zhang[5]
Yibin Xu[6] Yuehan Cui[7] Fangsheng Weng[8] Fei Ma[9]
Qi Tian[9] Ruqi Huang[1] Hao Zhao[1,10]

[1]THU [2]PKU [3]PolyU, Hong Kong [4]USTC [5]UESTC [6]Tongji University
[7]Tianjin University [8]Independent Researcher [9]Guangming Lab [10]BAAI

Project page: http://yisuanwang.github.io/PairCoder

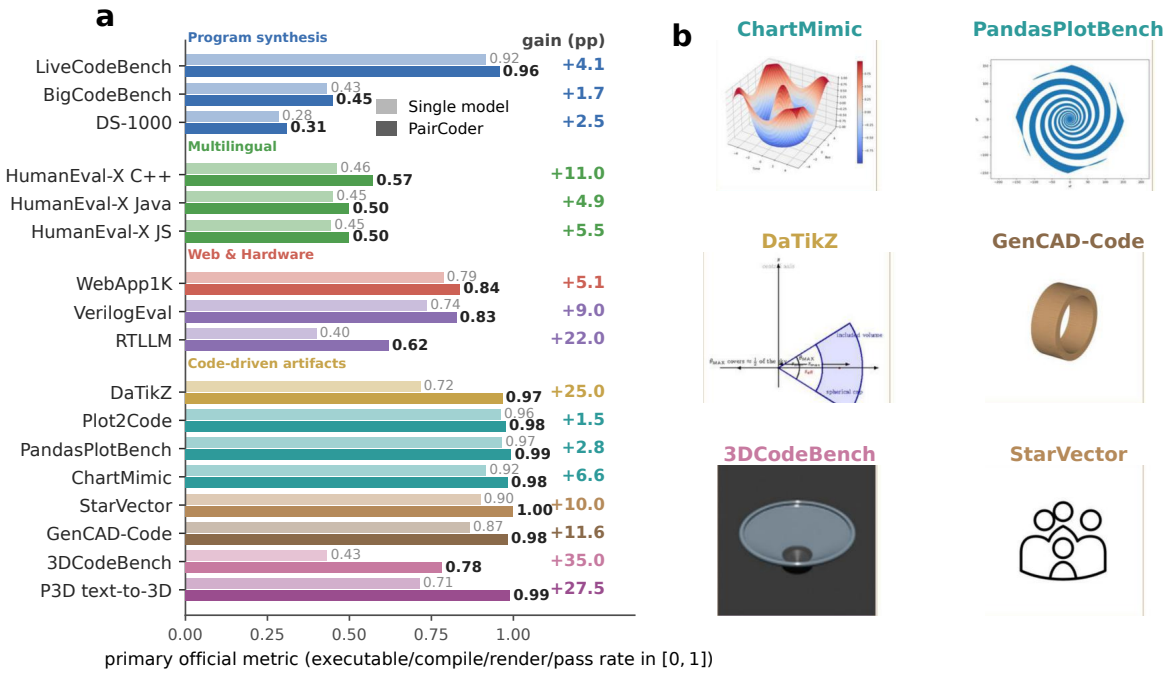


Figure 1: PairCoder improves code generation across every artifact domain, not only Python program synthesis. **(a)** All 17 public benchmarks at `gpt-5.4` (thinking off): the light bar is single-model, the dark bar (bold value) is PairCoder, coloured by benchmark family, with the absolute gain (pp) at right; each bar is the benchmark's primary official metric, an executable/compile/render/pass rate in $[0, 1]$. P3D-Bench is shown on its text-to-3D track (400 cases); its image-to-3D and assembly-3D tracks are reported as a 3-case demonstration in the appendix (Table 14). **(b)** Representative artifacts across six modalities: a chart (ChartMimic), a scientific figure (PandasPlotBench), a diagram (DaTikZ), a CAD model (GenCAD-Code), a 3D scene (3DCodeBench), and a vector graphic (StarVector).

Abstract

Code is the medium through which large language models generate structured artifacts: charts, scientific figures, vector graphics, CAD models, 3D scenes, and hardware designs are all produced by writing programs. In this regime single pass inference is brittle, because the compiler, renderer, or simulator that decides whether the artifact exists is invisible to the model. We present **PairCoder**, which grounds review in the toolchain and realizes it as two agent pair programming: a Driver agent writes the program, a Navigator agent reviews it against verification evidence (diagnostics, execution results, and renderings of the current artifact beside the target), and the two switch roles when errors persist. Across 17 public benchmarks and seven models from three vendors, PairCoder improves essentially every benchmark whose artifact is verifiable, on full official metric suites rather than execution alone (for example, Blender scene executability 0.20 to 0.78; Ti*k*Z compile rate up 10 to 30 points on every model), at 2.9 to 9.2 times single model cost (about 7 times overall). The improvements concentrate where the toolchain provides an informative oracle and the baseline leaves headroom, and the method ties or mildly regresses where the oracle is weak; we frame pair programming as a reliable recipe for verified code driven generation.

1 Introduction

Code generation for structured data and multimodal content asks a model to write a program whose value lies in the artifact it renders: scientific figures as Ti*k*Z, charts as matplotlib, icons as SVG, mechanical parts as CAD programs, 3D scenes as Blender scripts, web applications as React projects, and circuits as Verilog RTL (Belouadi et al., 2024a; Yang et al., 2024a; Wu et al., 2024; Galimzyanov et al., 2024; Rodriguez et al., 2025; Alam and Ahmed, 2024; Gao et al., 2026; Cui, 2024; Guo et al., 2025b; Liu et al., 2023b; Lu et al., 2024). The task is broadly useful and hard in a specific way: the program must survive an external toolchain (a compiler, renderer, or simulator) that the model cannot observe while writing, and the artifact must then match the target in structure, semantics, and geometry. Even strong LLMs (OpenAI, 2023; Bai et al., 2023; Liu et al., 2024; Li et al., 2023b) routinely emit programs that do not compile, do not run, or render something far from the request (Li et al., 2023c; Zhang et al., 2025; Tian et al., 2025).

Existing remedies fall into two camps. One scales up collaboration: team style multi agent frameworks (Hong et al., 2023; Qian et al., 2023; Li et al., 2023a; Wu et al., 2023) and role specialized pipelines (Huang et al., 2023; Dong et al., 2024; Islam et al., 2024; Chen et al., 2023c) improve correctness but pay up to an order of magnitude token overhead, and recent agentic creation systems show the same closed loop instinct in single domains (Pfaff et al., 2026; Chen et al., 2025a). The other camp keeps one model and adds self correction from runtime feedback (Shinn et al., 2024; Chen et al., 2024; Madaan et al., 2023; Gou et al., 2024), which is cheap but inherits the blind spots of a single perspective. Neither camp offers a domain general mechanism that anchors an independent reviewer to the toolchain, which is exactly where code driven creation fails.

We argue that this task needs the smallest collaborative structure in which an independent reviewer is grounded in verification, and human software engineering already has it: pair programming (Umapathy and Ritzhaupt, 2017; Lui and Chan, 2006; Hannay et al., 2009). We propose **PairCoder**, a two agent framework whose core idea is to turn code driven generation into a toolchain verified dialogue: one agent writes the program, the other reviews it against concrete verification evidence, and control of the keyboard follows the diagnosis.

PairCoder instantiates this idea with three components. (1) Driver and Navigator roles with Self-Mirror prompting: the Driver generates and revises while the Navigator reviews, each under a role specific system prompt over the shared task and memory, preserving two genuinely different perspectives on the same code. (2) Verification grounded review: each candidate is compiled, executed, or rendered by the benchmark's own toolchain, and the evidence, including diagnostics, test outcomes, and a rendering of the current artifact beside the target, is attached to the review; the Navigator must cite a concrete error or accept with `[NOERROR]`, which prevents churn on already correct programs. (3) Error triggered role switching with a quality fallback: persistent errors swap the roles so the agent that diagnosed the fault repairs it, and an exhausted budget returns the candidate with the best verified quality rather than the last one.

Across 17 public benchmarks spanning program synthesis, multilingual code, data science, web, hardware, and six artifact modalities, under seven models from three vendors, PairCoder improves essentially every benchmark whose artifact is verifiable (Figure 1), on full official metric suites rather than execution alone: Blender scene executability rises from 0.20 to 0.78, Ti*k*Z compile rate gains 10 to 30 points on every model, and chart, SVG, and CAD benchmarks improve in execution rate, SSIM, and CLIP, while on classical synthesis PairCoder stays competitive and often improves further (e.g. LiveCodeBench pass@1 from 0.94 to 0.99 at `gpt-5.4-mini`) at 2.9 to 9.2 times the single pass cost. A consistent pattern emerges: gains concentrate where the toolchain gives an informative oracle and the baseline leaves headroom, with ties at saturated oracles and mild regressions where the oracle is weak.

Our contributions are threefold: (1) PairCoder, a two agent framework that grounds pair programming in toolchain verification, unifying verification grounded review and error triggered role switching into one protocol across modalities; (2) the first systematic evidence that pair programming transfers from Python synthesis to structured data and multimodal artifact generation, improving executability, visual similarity (SSIM, CLIP, DINO, SigLIP-2), and geometry (Chamfer) across 17 benchmarks, seven models, and three vendors; and (3) a qualitative heuristic that gains scale with oracle informativeness and baseline headroom, characterizing when the collaboration helps, ties, or is wasted.

2 Related Work

2.1 Large Language Models

LLMs (Achiam et al., 2023; Touvron et al., 2023; Bai et al., 2023; Hui et al., 2024; Yang et al., 2025; Liu et al., 2024; Guo et al., 2025a) have made code one of their most impactful application domains (Chen et al., 2021; Liu et al., 2023a; Jiang et al., 2024), supported by code specialized model families (Nijkamp et al., 2023; Zheng et al., 2023; Li et al., 2023b; Lozhkov et al., 2024; Roziere et al., 2023) and increasingly demanding benchmarks (Jain et al., 2024; Zhuo et al., 2024; Lai et al., 2023; Jimenez et al., 2024; Zhang et al., 2023; Chen et al., 2025b; Guo et al., 2025b). Foundation models also reach into multimodal perception, multimodal generation, and robustness (Chen et al., 2023b; Miao et al., 2026; Ye et al., 2024; Chen et al., 2023a, 2026c), yet single pass inference remains prone to logical errors and hallucinations (Zhang et al., 2025; Tian et al., 2025; Ji et al., 2023). PairCoder is model agnostic: any of these models can play Driver or Navigator, and the collaboration adds reliability on top of whatever base capability the chosen model provides.

2.2 Structured Data Generation

A fast growing line of work, recently surveyed under the banner of multimodal code intelligence, treats programs as the medium for generating structured and multimodal content, where the value of the code lies in the artifact it renders: scientific figures (Belouadi et al., 2024a,b), charts and plots (Yang et al., 2024a; Wu et al., 2024; Galimzyanov et al., 2024; Yang et al., 2024c; Zhao et al., 2025), vector graphics and animation (Rodriguez et al., 2025; Wu et al., 2025; Chen et al., 2026b; Qiu et al., 2025), garment patterns (Weng et al., 2026a), parametric CAD (Alam and Ahmed, 2024; Alrashedy et al., 2025), procedural 3D scenes (Hu et al., 2024; Sun et al., 2023; Gao et al., 2026), front end code (Si et al., 2025; Cui, 2024), and hardware RTL (Liu et al., 2023b; Lu et al., 2024). These benchmarks score the rendered artifact with perceptual and geometric metrics (Wang et al., 2004; Radford et al., 2021; Oquab et al., 2024; Tschannen et al., 2025), yet the methods they evaluate are almost exclusively single pass generators. A parallel line of work synthesizes multimodal content directly with learned models rather than programs, spanning 3D editing, texture, rigging, skeleton, and human centric video generation (Weng et al., 2026b; Chen et al., 2026e; Sun et al., 2025, 2026; Chen et al., 2026d,a); PairCoder instead keeps the program as the medium, which makes the toolchain itself available as a verifier. PairCoder targets exactly this family of tasks and improves the full official metric suites rather than execution success alone; it is a concrete instance of the verification centered multimodal code intelligence that recent surveys call for.

2.3 Multi-Agent Collaboration

Two strands of work attack the brittleness of single pass generation. Multi agent systems divide labor across roles, from simulated teams and conversation scaffolds (Hong et al., 2023; Qian et al., 2023; Li et al., 2023a; Wu et al., 2023) to role specialized pipelines (Huang et al., 2023; Dong et al., 2024; Islam et al., 2024; Chen et al., 2023c) and domain specific agentic creation (Pfaff et al., 2026; Chen et al., 2025a); the other closes the loop with external signals, through self correction and tool grounded critique (Shinn et al., 2024; Chen et al., 2024; Madaan et al., 2023; Gou et al., 2024), learned verifiers and repair loops (Ni et al., 2023; Tsai et al., 2024), and agentic software systems (Yang et al., 2024b; Wang et al., 2025). Team scale systems pay an order of magnitude token overhead and the domain built ones are engineered per task, while single model self correction inherits its own blind spots. PairCoder occupies the intersection they leave open: the smallest collaborative unit, two agents in the Driver and Navigator pattern of human pair programming (Umapathy and Ritzhaupt, 2017; Lui and Chan, 2006; Hannay et al., 2009; Williams et al., 2000), with toolchain grounded review and error triggered role switching, applied as one uniform protocol from program synthesis to code driven multimodal generation.

3 Methods

3.1 Overview

The PairCoder framework, illustrated in Fig. 2, adapts pair programming to automated code generation with Large Language Models (LLMs). Two agents alternate between proposal and review. At each iteration, the Driver produces a candidate solution and the Navigator critiques it, after which the Driver revises the code. This interaction continues until the solution is accepted or the iteration budget is exhausted.

Formally, given a programming task $q \in \mathcal{Q}$,

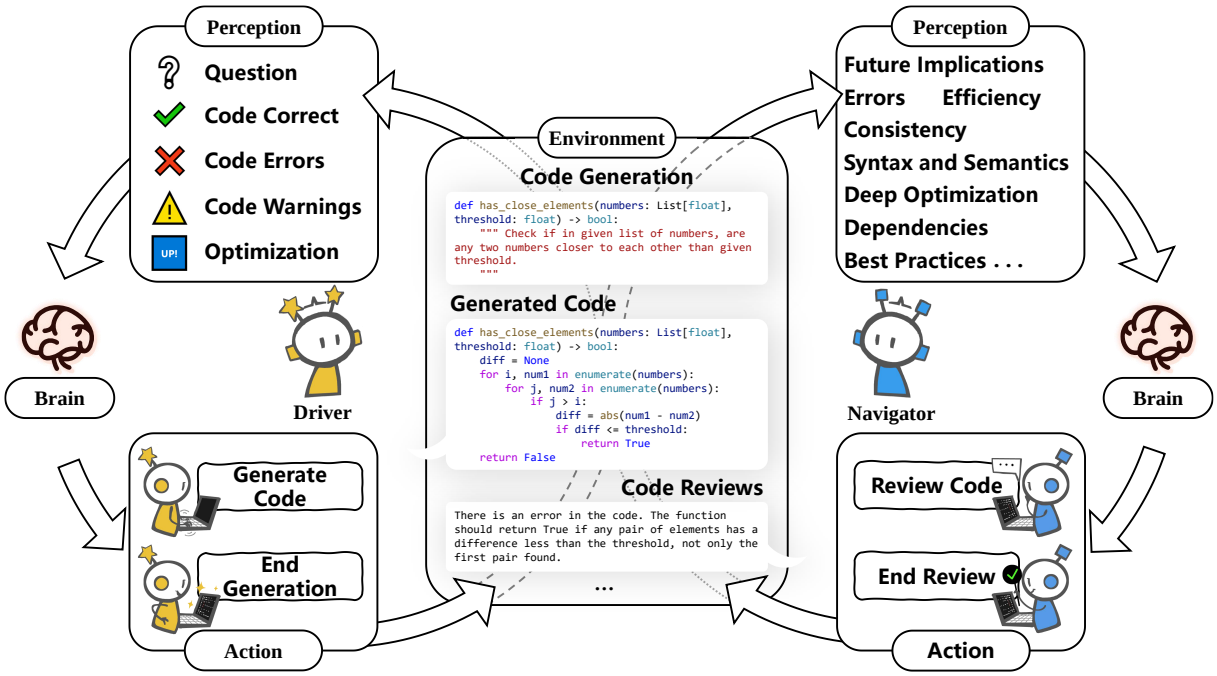


Figure 2: The PairCoder framework: the Driver generates and revises code, the Navigator reviews it against verification evidence, and persistent errors switch the roles.

PairCoder seeks to generate code $c^* \in \mathcal{C}$ that maximizes both correctness and quality:

$$c^* = \arg\max_{c \in \mathcal{C}} P(c \mid q) \cdot \text{Quality}(c, q) \quad (1)$$

3.2 Role Assignment

At each time step, a role function $\rho : \mathcal{A} \times T \to \mathcal{R}$ assigns Driver and Navigator to the two agents, where $\mathcal{A} = \{a_1, a_2\}$ denotes the agents, T the time steps, and $\mathcal{R} = \{Driver, Navigator\}$ the available roles.

The Driver is responsible for code construction and revision, whereas the Navigator checks correctness, identifies deficiencies, and returns actionable feedback. A `REVISE` verdict is itself an error signal, and the switch is taken *before* the repair so that the agent that diagnosed the fault takes the keyboard and a different agent reviews the resulting fix. The default policy switches on each error ($\eta = 1$); we also study switching only after η consecutive errors, a fixed interval every k rounds, and never switching (Sec. 4.6).

Driver Agent. The Driver generates code conditioned on the task specification and interaction history. As illustrated in Fig. 3, its generation process follows:

$$C_t = F_{driver}(I_{dri}, Q, M_{t-1}) \quad (2)$$

where I_{dri} denotes the driver instructions, Q is the task description, C_t is the candidate code at iteration t, R_i is the review outcome at iteration i, and M_{t-1} is the accumulated interaction history:

$$M_t = \{(C_i, R_i) \mid 1 \le i \le t\} \quad (3)$$

Concretely, on the first round the Driver is prompted with the task Q alone and emits C_1; on every subsequent round it receives the previous candidate C_{t-1}, the Navigator's review R_{t-1}, and the same verification evidence ψ_{t-1} the Navigator saw, and is asked to return the full revised program. There is no auxiliary objective or fine tuning: the Driver is a frozen LLM steered entirely through its prompt and the role-specific system message.

Navigator Agent The Navigator evaluates the current candidate against concrete verification evidence and returns structured feedback:

$$R_t = F_{navigator}(I_{nav}, Q, C_t, M_{t-1}, \psi_t) \quad (4)$$

where I_{nav} specifies navigator-specific review guidelines and ψ_t is the verification evidence attached to C_t (defined below).

The Navigator re-reads the requirements, traces the candidate on concrete inputs, checks that names and interfaces match the specification, and weighs

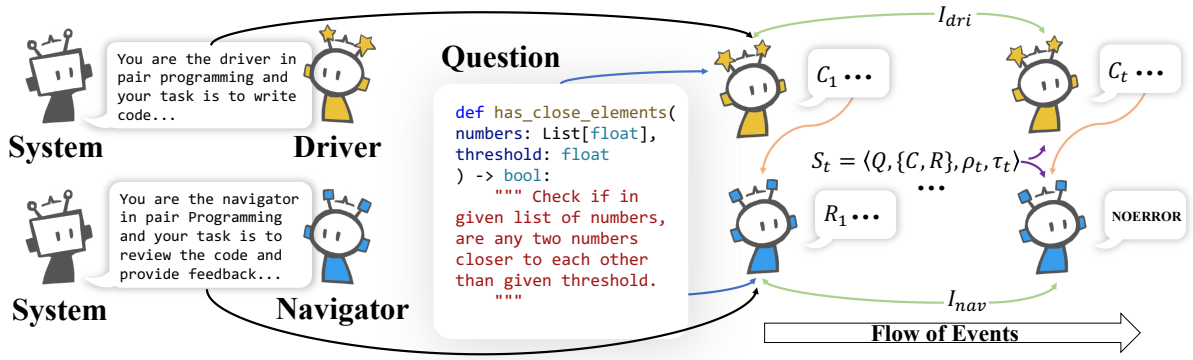


Figure 3: PairCoder workflow: the Driver proposes and revises, the Navigator reviews against toolchain evidence, and persistent errors trigger a role switch.

the evidence ψ_t, then issues one of two verdicts:

$$R_t = \begin{cases} \texttt{[NOERROR]} & \text{accept} \\ \text{REVISE}(\Delta_t) & \text{otherwise} \end{cases} \tag{5}$$

where Δ_t quotes the offending line, the requirement it violates, and the specific fix. The protocol is deliberately conservative: when ψ_t already passes and the Navigator cannot point to a definite fault it must accept with [NOERROR] rather than churn working code on style or vague suspicion. The interaction history is then updated as:

$$M_t = M_{t-1} \cup \{(C_t, R_t, \psi_t)\} \tag{6}$$

Verification Evidence ψ_t. Before each review the candidate C_t is run through the benchmark's own toolchain and the result is attached to the Navigator's prompt as evidence ψ_t. In our implementation ψ_t combines up to three signals. (1) A compile or execute predicate from the renderer, interpreter, or simulator (LaTeX, CadQuery, matplotlib, the SVG rasterizer, Blender, or an RTL simulator), together with the captured diagnostic on failure. (2) For tasks that ship a specification, the outcome of a test the Navigator authors *before* it reviews, a test-driven check of $\text{satisfy}(C_t, \text{spec}(Q))$. (3) For image conditioned tasks, a rendering of the current artifact placed beside the target, so the Navigator can cite concrete visual differences. The Driver receives the same ψ_t when it revises. This grounding is what makes the second perspective informative rather than a paraphrase of the first, and it is the only domain specific part of the loop; everything else is identical across the seventeen benchmarks.

3.3 System Components

Shared Environment. The framework maintains a shared state $S_t = \langle Q, M_t, \rho_t, \tau_t \rangle$ with four components. The query Q is the original task specification. The memory M_t is the running interaction history (candidates, reviews, and the attached evidence), kept per role and bounded in practice only by the model's context window. The role assignment ρ_t records the current mapping between agents and roles, and the iteration counter τ_t tracks collaboration progress.

Prompt construction. Each agent is a frozen chat LLM accessed through an OpenAI compatible API and carries its own message history; the two histories together realize the shared memory M_t. The role prompt is injected as a system message, a one line Driver or Navigator instruction plus an optional benchmark specific hint, and is re-asserted whenever the roles switch, which is the Self-Mirror mechanism in implementation terms. The Driver's user message is the task Q on the first round and the previous program, the Navigator's review, and the evidence ψ_{t-1} on every round after; the Navigator's user message is a review instruction, the candidate program, and ψ_t. For image conditioned benchmarks these messages are multimodal: the Driver is shown the target image, and the Navigator is additionally shown a rendering of the current candidate next to the target. The single model baseline is one direct generation with the same task prompt and no review.

Agent Brain. As depicted in Fig. 4, each agent uses an LLM conditioned on a role specific prompt and the current shared state:

$$\text{Brain}_{\text{role}} = \text{LLM}(P_{\text{role}}, S_t) \tag{7}$$

where P_{role} encodes the responsibilities of the current role and S_t provides the task context, dialogue history, and current role assignment.

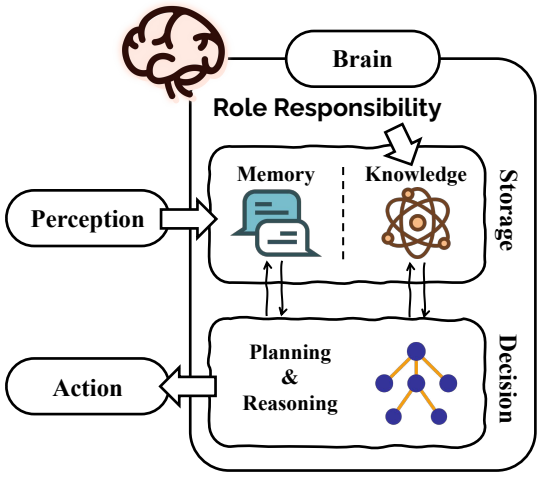


Figure 4: Agent brain architecture: LLM-based decision making with role-specific configuration.

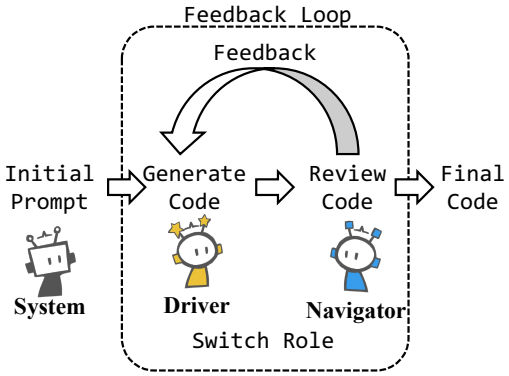


Figure 5: Collaboration mechanism with iterative feedback loop between Driver and Navigator agents.

Action Space. Each agent operates within a discrete action space $\mathcal{A}_{\text{role}}$:

$$\mathcal{A}_{driver} = \{\textsc{Generate}, \textsc{Refine}, \textsc{End}\} \quad (8)$$

$$\mathcal{A}_{navigator} = \{\textsc{Review}, \textsc{Accept}\} \quad (9)$$

3.4 Collaboration Protocol

The collaboration follows an iterative refinement process formalized in Algorithm 1. The feedback loop, illustrated in Fig. 5, alternates between Navigator review and Driver revision, yielding progressive improvement over successive iterations.

Termination Protocol. The system terminates when condition $\Omega(S_t)$ is satisfied:

$$\Omega(S_t) = \begin{cases} \text{true} & \text{if } R_t = \texttt{[NOERROR]} \\ \text{true} & \text{if } \tau \geq T \\ \text{false} & \text{otherwise} \end{cases} \quad (10)$$

If the loop stops on `[NOERROR]` the accepted candidate is returned. If instead the budget T is exhausted, PairCoder does not return the last attempt but the best one seen, $\arg\max_{C_i \in M_\tau} \text{Quality}(C_i)$, where Quality orders candidates first by whether the verification evidence ψ passes, then by the benchmark's continuous score when one is available (for example structural similarity to the target), and finally by recency. This makes an unresolved review debate degrade to a safe fallback rather than a failure.

Algorithm 1 PairCoder Collaboration Protocol

Input: Task Q, parameters $\Theta = \{T, k, \eta\}$ (budget, fixed interval, error threshold)
Output: Optimized code C^*
Initialize $S_0 \leftarrow \langle Q, \emptyset, \rho_0, 0 \rangle$
while $\tau < T$ and not terminated **do**
  Driver phase with self mirror
  $P_{dri} \leftarrow \text{SelfMirror}(\text{Driver}, I_{dri})$
  $C_\tau \leftarrow F_{driver}(P_{dri}, Q, M_{\tau-1})$
  Navigator phase with self mirror
  $\psi_\tau \leftarrow \text{Verify}(C_\tau)$ ▷ compile/execute, opt. test, opt. render
  $P_{nav} \leftarrow \text{SelfMirror}(\text{Navigator}, I_{nav})$
  $R_\tau \leftarrow F_{navigator}(P_{nav}, Q, C_\tau, M_{\tau-1}, \psi_\tau)$
  if $R_\tau = \texttt{[NOERROR]}$ **then**
    return C_τ
  end if
  Update shared state
  $M_\tau \leftarrow \text{Update}(M_{\tau-1}, C_\tau, R_\tau)$
  $\rho_{\tau+1} \leftarrow \text{CheckSwitch}(\rho_\tau, R_\tau, \tau)$
  $\tau \leftarrow \tau + 1$
end while
return $\arg\max_{C_i \in M_\tau} \text{Quality}(C_i)$

Self-Mirror Mechanism. To preserve role consistency, each agent receives an identity aware prompt before generation or review:

$$\text{SelfMirror}(\rho_t, Q, M_t) = \text{concat}(\text{RolePrompt}(\rho_t), Q, M_t) \quad (11)$$

where $\text{RolePrompt}(\rho_t)$ explicitly states whether the agent should act as Driver or Navigator. This mechanism keeps the Driver focused on code production and revision, and keeps the Navigator focused on diagnosis, verification, and targeted feedback throughout the interaction.

Implementation. Both arms use identical prompts and the same per benchmark extractor, which recovers the program from the raw reply only to run the verifier ψ; the dialogue itself keeps

the raw text. Empty or failed API responses are retried with exponential backoff and do not pollute the history. Reasoning is disabled for the main grid (a thinking-enabled study is reported separately), and the loop uses at most three to four review rounds in the main experiments (up to eight in the role-switching study of Sec. 4.6). The verifier ψ is the only per benchmark component; it wraps the benchmark's official toolchain (compiler, interpreter, renderer, simulator, or unit tests). We log prompt and completion tokens separately for each arm, which is the basis for the token accounting in Sec. 4.5.

Role Switching Strategies. Our framework supports heterogeneous agent configurations in which different LLMs can serve as Driver and Navigator, allowing us to leverage complementary model strengths. To optimize collaboration dynamics in this setting, we implement two switching policies $\pi : S_t \to \{0, 1\}$:

Fixed interval switching.

$$\pi_{\text{fixed}}(S_t) = (\tau \bmod k = 0) \tag{12}$$

ensuring balanced participation every k iterations.

Error triggered switching.

$$\pi_{\text{error}}(S_t) = \left(\sum_{i=\tau-w}^{\tau} \mathbb{I}[R_i = \text{REVISE}] \geq \eta \right) \tag{13}$$

activating when error count exceeds threshold η within window w.

4 Experiments

4.1 Experimental Setup

Benchmarks. Systematic LLM evaluation has been studied across many domains, from Chinese language understanding (Zhang et al., 2023) and strategic reasoning (Chen et al., 2025b) to web interface comprehension (Guo et al., 2025b). Following this tradition, we evaluate PairCoder on 17 public benchmarks organized by the artifact a program must produce. *Program synthesis*: LiveCodeBench (Jain et al., 2024) (recent contest problems), BigCodeBench (Zhuo et al., 2024) (library intensive tasks), and DS-1000 (Lai et al., 2023) (data science). *Multilingual code*: the C++, Java, and JavaScript splits of HumanEval-X (Zheng et al., 2023). *Web applications*: WebApp1K (Cui, 2024) (React user journeys scored by hidden Jest tests). *Hardware*: VerilogEval (Liu et al., 2023b) and RTLLM (Lu et al., 2024) (simulation against reference testbenches). *Code driven artifacts*: DaTikZ (Belouadi et al., 2024a) (caption to Ti*k*Z), Plot2Code (Wu et al., 2024), Pandas-PlotBench (Galimzyanov et al., 2024), and ChartMimic (Yang et al., 2024a) (chart or data to matplotlib), StarVector (Rodriguez et al., 2025) (image to SVG), GenCAD-Code (Alam and Ahmed, 2024) (image to CadQuery), 3DCodeBench (Gao et al., 2026) (text to Blender script), and P3D-Bench (Yang et al., 2026) (text to parametric 3D CAD, detailed in Sec. 4.4). Each benchmark is scored with its official metric suite: pass@1 or execution rate where unit tests or toolchains define correctness, plus SSIM (Wang et al., 2004), CLIP (Radford et al., 2021), DINO (Oquab et al., 2024), SigLIP-2 (Tschannen et al., 2025), and Chamfer distance for rendered artifacts; full per benchmark scoring and reproducibility details are in Appendix E.

Models and protocol. We test seven models from three vendors: `gpt-5.4-mini`, `gpt-5.4`, and `gpt-5.5` (OpenAI); `doubao-1.5-lite` and `doubao-seed-2.0-mini` (Doubao); `deepseek-v3.2` and `deepseek-v4-flash` (DeepSeek), all with thinking disabled. The baseline is a single direct generation (one call per problem). PairCoder runs the loop of Sec. 3.2 with at most three to four review rounds: the Driver generates, the verification predicate ψ compiles, executes, or renders the candidate, the Navigator reviews the code together with this evidence (and, for visually grounded tasks, a rendering of the current artifact next to the target) and either accepts with `[NOERROR]` or requests a concrete revision, and roles switch when errors persist. Both arms use identical prompts and extraction; image input benchmarks are restricted to the vision capable models. We log every token in both arms.

4.2 Main Results

Table 1 reports the full grid: 17 benchmarks under all seven models. On the three OpenAI models PairCoder improves 44 of the 48 cells; the four exceptions are three exact ties at saturated oracles (ChartMimic execution at `gpt-5.4-mini`, BigCodeBench at `gpt-5.5`, and StarVector rendering at `gpt-5.5`) and one small regression (Plot2Code execution at `gpt-5.5`, 0.985 to 0.977, where the baseline already runs near the ceiling and SSIM still improves, see Sec. 5). Three pat-

Table 1: Main results: 17 benchmarks × seven models from three vendors (thinking off), each scored with its official metric suite. Each cell is single model → **PairCoder**; bold marks the better arm (↓ metrics are better when lower; visual metrics are aggregates that score non-rendering generations as 0), green marks improvements. † marks image-input benchmarks under vision-less models (`doubao-1.5-lite`, `deepseek-v3.2`, `deepseek-v4-flash`). **P3D-Bench** denotes its *text-to-3D / minimal-JSON* track (Sec. 4.4); image-to-3D and assembly-3D are reported as a demonstration in Table 14.

Benchmark	Metric	`gpt-5.4-mini`	`gpt-5.4`	`gpt-5.5`	`doubao-1.5-lite`	`doubao-seed-2.0-mini`	`deepseek-v3.2`	`deepseek-v4-flash`
Program synthesis								
LiveCodeBench	pass@1 ↑	0.942→**0.992**	0.917→**0.958**	0.933→**0.975**	0.500→**0.533**	0.825→**0.867**	0.950→0.908	0.933→**0.975**
BigCodeBench	pass@1 ↑	0.450→**0.458**	0.433→**0.450**	0.458→0.458	0.350→0.317	0.408→0.392	0.467→0.450	0.417→**0.425**
DS-1000	pass@1 ↑	0.250→**0.255**	0.285→**0.310**	0.385→**0.425**	0.075→0.075	0.100→0.085	0.155→0.080	0.170→0.110
Multilingual								
HumanEval-X C++	pass@1 ↑	0.335→**0.415**	0.463→**0.573**	0.524→**0.835**	0.183→**0.220**	0.427→0.421	0.463→0.451	0.561→**0.598**
HumanEval-X Java	pass@1 ↑	0.341→**0.384**	0.451→**0.500**	0.396→**0.561**	0.250→**0.256**	0.335→0.335	0.500→0.463	0.494→0.463
HumanEval-X JS	pass@1 ↑	0.323→**0.341**	0.445→**0.500**	0.384→**0.543**	0.256→0.250	0.293→**0.305**	0.463→**0.482**	0.317→**0.366**
Web & Hardware								
WebApp1K	pass@1 ↑	0.700→**0.838**	0.787→**0.838**	0.787→**0.887**	0.013→0.013	0.525→**0.750**	0.575→**0.750**	0.650→**0.675**
VerilogEval	pass@1 ↑	0.763→**0.808**	0.737→**0.827**	0.821→**0.897**	0.429→**0.436**	0.647→**0.699**	0.596→**0.724**	0.692→**0.705**
RTLLM	pass@1 ↑	0.656→**0.719**	0.400→**0.620**	0.620→**0.660**	0.360→0.300	0.500→0.480	0.560→**0.640**	0.560→0.520
Code-driven artifacts								
DaTikZ	compile ↑	0.500→**0.633**	0.717→**0.967**	0.783→**1.000**	0.550→**0.850**	0.567→**0.800**	0.683→**0.917**	0.633→**0.733**
	SSIM ↑	0.215→**0.256**	0.363→**0.512**	0.394→**0.485**	0.406→**0.601**	0.325→**0.449**	0.365→**0.544**	0.348→**0.406**
	CLIP ↑	0.327→**0.384**	0.566→**0.770**	0.602→**0.767**	0.391→**0.621**	0.441→**0.606**	0.508→**0.702**	0.505→**0.578**
	DINO ↑	0.303→**0.338**	0.530→**0.698**	0.554→**0.730**	0.259→**0.391**	0.383→**0.545**	0.465→**0.596**	0.445→**0.519**
Plot2Code	exec ↑	0.841→**0.962**	0.962→**0.977**	0.985→0.977	†	0.909→**0.917**	†	†
	SSIM ↑	0.412→**0.474**	0.476→**0.495**	0.487→**0.496**	†	0.410→**0.443**	†	†
	CLIP ↑	0.802→**0.912**	0.916→**0.920**	0.943→0.931	†	0.835→0.823	†	†
PandasPlotBench	exec ↑	0.789→**0.851**	0.966→**0.994**	0.960→**0.977**	†	0.909→**0.989**	†	†
	SSIM ↑	0.423→**0.461**	0.567→**0.610**	0.597→**0.644**	†	0.491→**0.534**	†	†
	CLIP ↑	0.728→**0.794**	0.909→**0.944**	0.907→**0.942**	†	0.846→**0.922**	†	†
ChartMimic	exec ↑	0.967→0.967	0.917→**0.983**	0.983→**1.000**	†	0.900→**0.950**	†	†
	SSIM ↑	0.578→**0.578**	0.533→**0.582**	0.593→**0.613**	†	0.443→**0.476**	†	†
	CLIP ↑	0.871→0.861	0.820→**0.848**	0.874→**0.887**	†	0.770→**0.808**	†	†
StarVector	render ↑	0.983→**1.000**	0.900→**1.000**	1.000→1.000	†	0.983→**1.000**	†	†
	SSIM ↑	0.784→**0.801**	0.776→**0.873**	0.777→**0.787**	†	0.758→**0.770**	†	†
	CLIP ↑	0.929→**0.944**	0.859→**0.954**	0.961→0.960	†	0.883→**0.915**	†	†
	DINO ↑	0.874→**0.885**	0.820→**0.906**	0.938→0.934	†	0.793→**0.819**	†	†
GenCAD-Code	exec ↑	0.900→**0.983**	0.867→**0.983**	0.917→**1.000**	†	0.833→**0.967**	†	†
	Chamfer ↓	0.233→**0.166**	0.259→**0.155**	0.215→**0.126**	†	0.308→**0.212**	†	†
3DCodeBench	exec ↑	0.200→**0.783**	0.433→**0.783**	0.383→**0.417**	0.167→**0.383**	0.200→**0.533**	0.333→**0.633**	0.600→0.533
	SigLIP-2 ↑	0.181→**0.702**	0.263→**0.842**	0.741→**0.790**	0.225→**0.302**	0.180→**0.521**	0.271→**0.446**	0.658→0.487
	DINO ↑	0.101→**0.376**	0.129→**0.512**	0.362→**0.420**	0.138→0.132	0.097→**0.300**	0.175→**0.261**	0.372→0.290
	Chamfer ↓	4.850→**2.618**	4.658→**1.658**	2.368→2.482	4.762→**3.955**	4.965→**3.400**	5.279→**3.407**	2.694→3.258
P3D-Bench text-to-3D (minimal-JSON)	valid ↑	0.973→**1.000**	0.715→**0.990**	0.672→**0.943**	0.140→**0.427**	0.812→**0.990**	0.950→**0.993**	0.945→**1.000**
	geometry ↑	0.600→**0.618**	0.347→**0.483**	0.321→**0.438**	0.073→**0.218**	0.419→**0.497**	0.549→**0.576**	0.581→**0.611**
	F@0.05 ↑	0.746→**0.767**	0.443→**0.619**	0.419→**0.570**	0.094→**0.275**	0.537→**0.633**	0.682→**0.714**	0.727→**0.764**
	F@0.01 ↑	0.316→**0.326**	0.153→**0.215**	0.143→**0.188**	0.037→**0.104**	0.196→**0.230**	0.282→**0.298**	0.298→**0.311**
	NC ↑	0.678→**0.698**	0.435→**0.605**	0.402→**0.556**	0.087→**0.262**	0.504→**0.609**	0.640→**0.671**	0.650→**0.685**
	IoU ↑	0.486→**0.501**	0.226→**0.334**	0.207→**0.288**	0.054→**0.157**	0.302→**0.360**	0.434→**0.460**	0.474→**0.498**
	topology ↑	0.968→**0.995**	0.708→**0.983**	0.669→**0.938**	0.136→**0.416**	0.808→**0.985**	0.946→**0.989**	0.943→**0.998**

terns stand out. First, the largest gains concentrate where the toolchain provides the richest signal: 3DCodeBench executability rises by up to 58 points, DaTikZ compile rate by 10 to 30 points, and WebApp1K and RTLLM each by up to 22 points. Second, gains often grow with model capability rather than shrink: HumanEval-X C++ improves by 8.0, 11.0, and 31.1 points on `gpt-5.4-mini`, `gpt-5.4`, and `gpt-5.5` respectively, because a stronger Navigator reviews more accurately and a stronger Driver exploits the feedback better. Third, on benchmarks whose baseline is already near the ceiling the loop terminates immediately via `[NOERROR]` and safely ties instead of regressing, which is the designed behavior of the conservative review protocol.

4.3 Cross-Vendor Generality

The four right columns of Table 1 repeat the study on models from two further vendors. The headline largely transfers: artifact domains with a strong toolchain signal improve on nearly every applicable model. DaTikZ gains 10 to 30 compile points on all four models, 3DCodeBench gains 22 to 33 executability points on three of four, WebApp1K gains 2.5 to 22.5 points wherever the model can write React at all, and the vision capable `doubao-seed-2.0-mini` improves every chart, SVG, and CAD metric. The same columns also delimit the method honestly: on benchmarks where the verification signal is weak (DS-1000 signature execution without functional tests, BigCodeBench library calls, RTLLM compile only evidence), the weaker or non GPT navigators occasionally churn working code and small regressions appear (for example DS-1000 on the DeepSeek models). This is exactly the boundary in the qualitative pattern above: where the toolchain cannot adjudicate, review quality becomes model bound; where it can, gains appear across all vendors; Appendix C (Figures 19–23) shows the same tasks rendered across these models side by side. As a negative control we also ran GeoCodeBench (Li et al., 2026), a PhD level 3D vision benchmark on which both arms score at the floor (about 0.02), and PairCoder ties rather than regresses.

4.4 Generalization to Parametric 3D Code (P3D-Bench)

To test whether the verification-grounded loop transfers to a target where the artifact is a *3D solid* rather than a program output, we evaluate on P3D-Bench (Yang et al., 2026), a benchmark in which the model writes a parametric construction program that is compiled to a mesh and scored against ground-truth geometry. P3D-Bench spans three tracks (text-to-3D, image-to-3D, and assembly-3D) across several program formats (minimal-JSON, OpenSCAD, CadQuery, Three.js). We evaluate the *text-to-3D* track primarily in the minimal-JSON format, the one configuration whose ground truth is materializable from the openly licensed Text2CAD v1.1 corpus, and additionally report executable validity for the OpenSCAD format. For the image-to-3D and assembly-3D tracks, whose full evaluation sets build on the licensed Fusion 360 Gallery, we report a three-case in-repo demonstration (Appendix D, Table 14), on which PairCoder improves validity, geometry, and topology in the CadQuery format of both tracks and ties on Three.js, where the baseline already compiles all three cases.[1] The Navigator's verification predicate ψ is simply the benchmark's own compiler ("does the program build a valid solid?"), and we report the official metrics: executable validity, a geometry score (Chamfer distance, F-score, normal consistency and volumetric IoU, combined as the worst-filled composite that zeroes invalid programs), and a topology score. We run the full 400-case text-to-3D split with thinking disabled.

The P3D-Bench rows of Table 1 show the same headroom-tracking pattern as the rest of the grid: PairCoder improves *every* metric on *every* model with no regressions. The weakest baseline (`doubao-1.5-lite`, 14.0% valid) gains $+28.7$ points of validity, $+0.145$ geometry and $+0.280$ topology, and `gpt-5.4` (71.5%) gains $+27.5/+0.136/+0.275$, while near-saturated models such as `gpt-5.4-mini` (97.3%) and `deepseek-v4-flash` (94.5%) still improve rather than regress. Crucially, on programs the baseline already compiles the geometry is unchanged: PairCoder intervenes only when the artifact fails to build, so coverage rises without disturbing correct solids, the conservative-review behavior of Sec. 4.2, now measured on 3D geometry and topology. Appendix D breaks the geometry and topology columns into their official sub-metrics (capped Chamfer, F-score@0.05/0.01, normal consistency, and volumetric IoU for geometry, Table 11; no-open-edge, inverted-normal, and

[1] We report P3D-Bench's official metrics: executable validity, geometry (Chamfer, F-score, normal consistency, IoU), and topology.

non-manifold ratios for topology, Table 12): PairCoder improves every sub-metric on every model, while on the programs the single model already compiles PairCoder keeps that program unchanged, so its geometry is *identical* there and the Chamfer distance on that common set matches the baseline exactly. This makes the source of the gain unambiguous: it is coverage of non-executable programs, not re-optimization of already-valid geometry: PairCoder repairs programs the single model fails to build at all into valid solids. Appendix D additionally illustrates a vision-grounded geometry-refinement variant that improves already-valid but inaccurate solids (Fig. 24), for example raising F-score@0.05 from 0.23 to 0.95. A second program format tells the same story: with OpenSCAD in place of minimal-JSON, PairCoder drives validity to a perfect 1.000 on all three models tested, though the gain is smaller because the baseline is already more fluent in OpenSCAD (Appendix D, Table 13).

4.5 Token Economics

We log every token in both arms. Across the full grid PairCoder consumes 2.9× to 9.2× the tokens of the single model baseline depending on the benchmark (for example 2.9× on GenCAD-Code, 3.4× on StarVector, 4.0× on WebApp1K, 5.9× on ChartMimic, 6.9× on DaTikZ, 7.9× on RTLLM where simulation failures trigger the most revision rounds, and 9.2× on Plot2Code), with an overall ratio of 7.4× over 138M logged tokens (the per benchmark breakdown is in Table 10). This is the cost of the review rounds themselves: tasks accepted in the first round cost roughly 2× a single pass (one generation plus one review), and the multiplier grows only when the Navigator actually finds errors, that is, exactly when the additional spend converts into accuracy (Fig. 6). Team style multi agent frameworks reported in prior work consume an order of magnitude more than a single call; while we do not re-run those systems on our benchmarks, PairCoder's measured 3 to 9 times envelope is well below that order of magnitude overhead.

4.6 Role Switching Analysis

The error triggered role switch is a design choice; here we test it directly at `gpt-5.4-mini` on four benchmarks, against three alternatives at matched budget: *no switch* (the Driver keeps the keyboard, the Navigator always reviews), *switch on each error* (the Navigator that just flagged an error takes

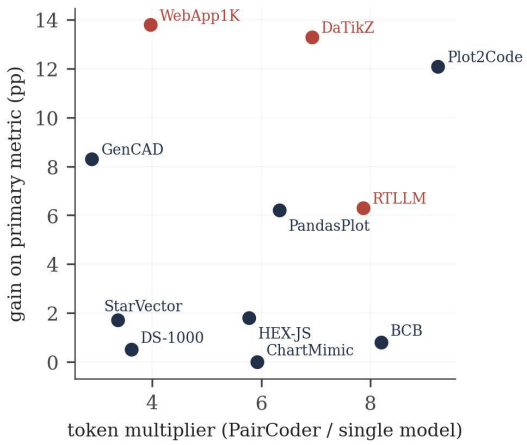


Figure 6: Cost versus benefit at `gpt-5.4-mini`: per-benchmark token multiplier against primary-metric gain; colours follow the families of Fig. 1.

over immediately), *switch after two consecutive errors*, and a *fixed interval* switch every two rounds regardless of the signal. Figure 7 reports the official metric and the token cost for each policy.

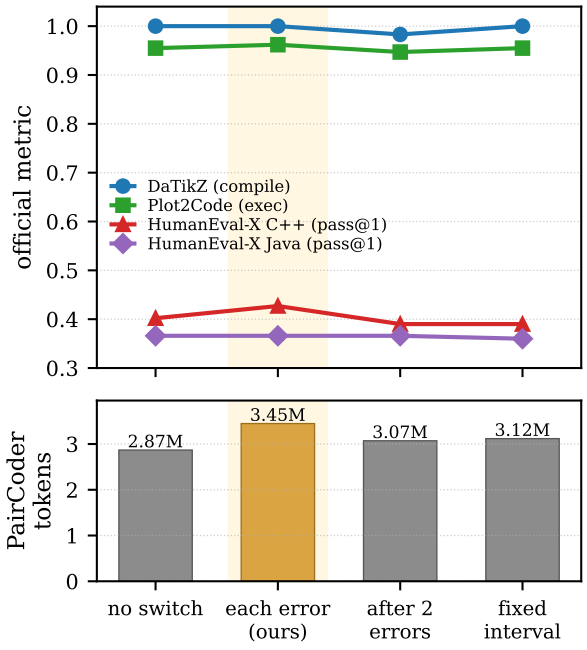


Figure 7: Role-switching policy at `gpt-5.4-mini`: official metric (top) and PairCoder token cost (bottom) for four policies; see Sec. 4.6.

Switching on each error is the best or tied-best policy on all four benchmarks, and never the worst. The effect is small where the loop is short (DaTikZ saturates near a perfect compile rate under every policy) but clear where iteration matters: on HumanEval-X C++, switching immediately reaches 0.427 while delaying the switch to

two consecutive errors drops to 0.390, the same as never switching, because the agent that diagnosed the fault is the one best positioned to fix it, and keeping it in the reviewer seat wastes that diagnosis. No switch and the fixed interval are competitive but consistently a step behind. We therefore adopt switch on each error as the default in all other experiments. This updates the earlier preference for a two error threshold, which we had observed on Python only; on the broader benchmark set immediate switching is the more reliable choice.

4.7 Effect of Reasoning Effort

The main results disable model thinking. We additionally test whether enabling reasoning changes the picture, on the three Python benchmarks where the no-thinking gains are smallest, at `gpt-5.4-mini`. Enabling high reasoning effort raises both arms' baselines, as expected, and PairCoder still helps where the verification signal is informative: on BigCodeBench the gain grows from $+0.8$ points (no thinking, 0.450 to 0.458) to $+5.0$ points (high, 0.425 to 0.475), because a stronger Driver produces candidates whose remaining errors are more often caught by the Navigator's doctest evidence. On DS-1000, whose only signal is that a function executes on its signature (no functional tests), thinking lifts the baseline from 0.250 to 0.310 but PairCoder neither helps nor hurts beyond noise (0.310 to 0.300), consistent with the qualitative pattern above: where the toolchain cannot adjudicate correctness, extra reasoning improves the Driver but gives the Navigator nothing new to verify. Reasoning roughly triples per-call token cost, so we use it only for this analysis and report the main grid with thinking disabled. (We omit hardware RTL from this comparison: at high effort the small model spends most of its budget on reasoning and frequently truncates the actual Verilog module, an artifact of the model rather than of the collaboration.)

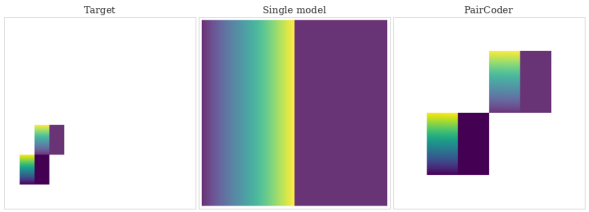
5 Code-Driven Multimodal Generation

The central claim of this work is that pair programming turns code driven creation, not just program synthesis, into a verified process. We test it on seven public benchmarks whose outputs are executable artifacts in other modalities: scientific figures (DaTikZ (Belouadi et al., 2024a): caption to Ti*k*Z/LaTeX), charts (Plot2Code (Wu et al., 2024), PandasPlotBench (Galimzyanov et al., 2024), ChartMimic (Yang et al., 2024a): image or data to matplotlib), vector graphics (StarVector (Rodriguez et al., 2025): image to SVG), parametric CAD (GenCAD-Code (Alam and Ahmed, 2024): image to CadQuery), and 3D scenes (3DCodeBench (Gao et al., 2026): text to Blender script). In each domain the Navigator's review is grounded in the same verification predicates ψ_i as in Sec. 3.2: the candidate program is compiled, executed, or rendered, and the result (including the rendered image for multimodal tasks) is supplied to the Navigator as review evidence before it issues `[NOERROR]` or a concrete revision request. We report each benchmark's official metric suite, namely execution or compile rate plus visual similarity (SSIM (Wang et al., 2004), CLIP (Radford et al., 2021), DINO (Oquab et al., 2024), SigLIP-2 (Tschannen et al., 2025)) and geometric fidelity (Chamfer distance), rather than execution success alone.

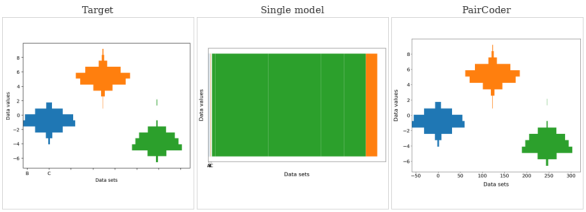
Table 2: Multimodal code generation with `gpt-5.4-mini` (thinking off); each cell is single model → **PairCoder** (↓ better when lower).

Benchmark	Metric	single → PairCoder
DaTikZ (60)	compile rate ↑	0.500 → **0.633**
	SSIM / CLIP / DINO ↑	.215/.327/.303 → **.256/.384/.338**
Plot2Code (132)	execution rate ↑	0.841 → **0.962**
	SSIM / CLIP ↑	.412/.802 → **.474/.912**
PandasPlotBench (175)	execution rate ↑	0.789 → **0.851**
	SSIM / CLIP ↑	.423/.728 → **.461/.794**
ChartMimic (60)	execution rate ↑	0.967 → 0.967
	SSIM / CLIP ↑	.578/.871 → .578/.861
StarVector (60)	render rate ↑	0.983 → **1.000**
	SSIM / CLIP / DINO ↑	.784/.929/.874 → **.801/.944/.885**
GenCAD-Code (60)	execution rate ↑	0.900 → **0.983**
	Chamfer (aggregate) ↓	0.233 → **0.166**
3DCodeBench (60)	executability ↑	0.200 → **0.783**
	SigLIP-2 / DINO (agg.) ↑	.181/.101 → **.702/.376**
	Chamfer (aggregate) ↓	4.85 → **2.62**

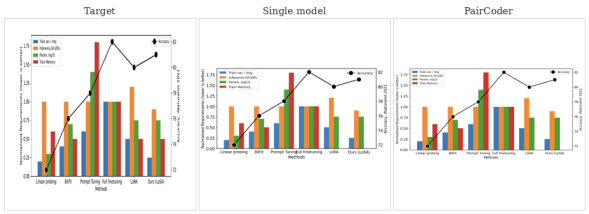
Quantitative results. Table 2 summarizes the `gpt-5.4-mini` results. PairCoder improves every official metric on six of the seven benchmarks: not only are far more generated programs executable, the artifacts they produce are also closer to the reference in pixel structure (SSIM), semantics (CLIP, SigLIP-2, DINO), and geometry (Chamfer). The only exception is ChartMimic, where the baseline already executes near the ceiling (about 0.97 to 1.0) and the loop ties; notably, at `gpt-5.4` the same benchmark turns positive (execution 0.92 to 0.98, SSIM 0.53 to 0.58), confirming that the tie reflects a saturated oracle rather than a limit of the collaboration. These gains transfer across vendors: with the volcano engine models, DaTikZ improves on



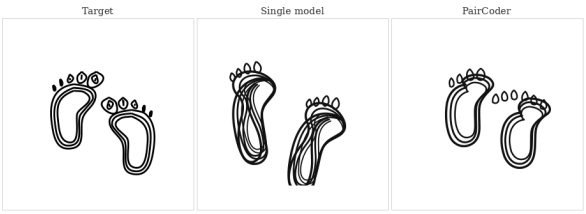
(a) Plot2Code: image to matplotlib



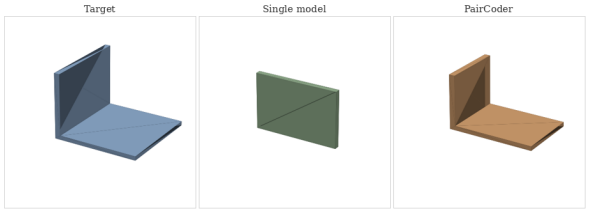
(b) PandasPlotBench: data to matplotlib



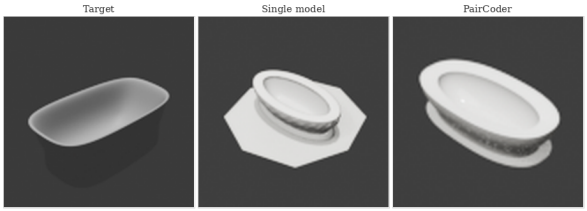
(c) ChartMimic: chart to matplotlib



(d) StarVector: image to SVG



(e) GenCAD-Code: image to CadQuery



(f) 3DCodeBench: text to Blender

Figure 8: Qualitative comparison across six artifact domains: target/reference (left), single model (middle), PairCoder (right).

all four (doubao-1.5-lite +30.0, doubao-seed-2.0-mini +23.3, deepseek-v3.2 +23.4, deepseek-v4-flash +10.0 compile points), 3DCodeBench gains +21.6/+33.3/+30.0 executability points, and the multimodal doubao-seed-2.0-mini improves nearly every chart, SVG, and CAD metric (for example, PandasPlotBench execution 0.91 to 0.99).

Qualitative comparison. Figure 8 illustrates the dominant failure repair pattern behind these numbers. Single model generations in these domains frequently die at the toolchain boundary: LaTeX that does not compile, matplotlib scripts that raise exceptions, Blender scripts that produce no geometry. PairCoder converts a large fraction of these hard failures because the Navigator receives the concrete compiler or runtime error (and, for image conditioned tasks, a rendering of the current candidate next to the target) and returns an actionable critique; after the error triggered role switch, the agent that diagnosed the fault repairs it directly. When the baseline already renders, the same visual review instead nudges layout, proportions, and styling toward the target, which is reflected in the consistent SSIM, CLIP, and DINO gains. Appendix B expands this into one page per benchmark, with galleries (Figures 12–18) and the full measured metric suites (Tables 3–9); Appendix A walks through complete multi-round traces (Cases A.1–A.4).

Cost. The collaboration costs about 3 to 9 times the tokens of single model inference in these domains (for example, 2.9× on GenCAD-Code, 3.4× on StarVector, 5.9× on ChartMimic, 6.9× on DaTikZ, and 9.2× on Plot2Code, measured per arm), with an overall ratio of about 7× across the full grid; review rounds terminate early via `[NOERROR]` on the majority of tasks that succeed immediately.

6 Conclusion

Code is increasingly how language models make things, from figures, charts, and vector graphics to CAD parts, 3D scenes, and circuits, and in this regime the decisive failures occur at the boundary between the program and the toolchain that renders it. We presented PairCoder, a minimal two agent framework that closes this boundary: a Driver writes the program, a Navigator reviews it with verdicts grounded in compiler, execution, and rendering evidence, and error triggered role switching keeps the collaboration moving. Across 17 public benchmarks, seven models, and three vendors, PairCoder improves essentially every ver-

ifiable benchmark on its entire official metric suite, covering executability, visual similarity, and geometric fidelity alike, while remaining within 2.9 to 9.2 times a single pass in cost (about 7 times overall), far below team simulation systems; on classical synthesis it remains competitive with and often improves strong single model inference (for example LiveCodeBench pass@1 from 0.94 to 0.99). The broader lesson is a qualitative pattern for collaborative generation: gains concentrate where the artifact is verifiable and the baseline leaves headroom, and fade where the oracle is weak. Wherever a trustworthy toolchain can be made to speak, and in code driven creation it almost always can, pair programming converts that signal into reliably better artifacts, making it a practical recipe for trustworthy code mediated generation of structured data and multimodal content wherever such an oracle exists.

Limitations

While PairCoder improves code generation accuracy, it still requires substantially more tokens and wall clock time than single model inference, with about seven times the cost per task on average in our experiments (2.9 to 9.2 times depending on the benchmark). Although one PairCoder run can reach accuracy levels that otherwise require several retries, the added latency and token budget still constrain practical deployment. Reducing this cost while preserving accuracy remains an important direction for future work.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Md Ferdous Alam and Faez Ahmed. 2024. Gencad: Image-conditioned computer-aided design generation with transformer-based contrastive representation and diffusion priors. *arXiv preprint arXiv:2409.16294*.

Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. 2025. Generating cad code with vision-language models for 3d designs. In *The Thirteenth International Conference on Learning Representations*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Jonas Belouadi, Anne Lauscher, and Steffen Eger. 2024a. Automatikz: Text-guided synthesis of scientific vector graphics with tikz. In *The Twelfth International Conference on Learning Representations*.

Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. 2024b. Detikzify: Synthesizing graphics programs for scientific figures and sketches with tikz. In *Advances in Neural Information Processing Systems*, volume 37, pages 85074–85108. Curran Associates, Inc.

Junhao Chen, Mingjin Chen, Jianjin Xu, Xiang Li, Junting Dong, Mingze Sun, Puhua Jiang, Hongxiang Li, Yuhang Yang, Hao Zhao, Xiao-Xiao Long, and Ruqi Huang. 2026a. Dancetogether: Generating interactive multi-person video without identity drifting. In *The Fourteenth International Conference on Learning Representations*.

Junhao Chen, Kejun Gao, Yuehan Cui, Mingze Sun, Mingjin Chen, Shaohui Wang, Xiaoxiao Long, Fei Ma, Qi Tian, Hao Zhao, and Ruqi Huang. 2026b. Lottiegpt: Tokenizing vector animation for autoregressive generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 31639–31651.

Junhao Chen, Xiang Li, Xiaojun Ye, Chao Li, Zhaoxin Fan, and Hao Zhao. 2025a. Idea23d: Collaborative lmm agents enable 3d model generation from interleaved multimodal inputs. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 4149–4166.

Junhao Chen, Peng Rong, Jingbo Sun, Chao Li, Xiang Li, and Hongwu Lv. 2023a. Soulstyler: Using large language model to guide image style transfer for target object. *arXiv preprint arXiv:2311.13562*.

Junhao Chen, Jingbo Sun, Xiang Li, Haidong Xin, Yuhao Xue, Yibin Xu, and Hao Zhao. 2025b. LLMsPark: A benchmark for evaluating large language models in strategic gaming contexts. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 182–194, Suzhou, China. Association for Computational Linguistics.

Junhao Chen, Xiaojun Ye, Jingbo Sun, and Chao Li. 2023b. Towards energy-efficient sentiment classification with spiking neural networks. In *International Conference on Artificial Neural Networks*, pages 518–529. Springer.

Junhao Chen, Boran Zhang, Mingjin Chen, Henghaofan Zhang, Saining Zhang, Congcong Zhu, Hao Zhao, Ruqi Huang, Zhihao Li, and Yufei Wang. 2026c. One video, one world: Turning monocular video into physical 4d scenes. *Preprint*, arXiv:2606.31388.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph,

Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Mingjin Chen, Junhao Chen, Zhaoxin Fan, Yujian Lee, Zichen Dang, Lili Wang, Yawen Cui, Lap-Pui Chau, and Yi Wang. 2026d. Hvg-3d: Bridging real and simulation domains for 3d-conditional hand-object interaction video synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15986–15997.

Mingjin Chen, Junhao Chen, Huan-ang Gao, Xiaoxue Chen, Zhaoxin Fan, and Hao Zhao. 2026e. Ultraman: ultra-fast and high-resolution texture generation for 3d human reconstruction from a single image. *Machine Vision and Applications*, 37(2):24.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. 2023c. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

Yi Cui. 2024. Webapp1k: A practical code-generation benchmark for web app development. *arXiv preprint arXiv:2408.00019*.

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–38.

Timur Galimzyanov, Sergey Titov, Yaroslav Golubev, and Egor Bogomolov. 2024. Drawing pandas: A benchmark for llms in generating plotting code. *arXiv preprint arXiv:2412.02764*.

Yipeng Gao, Lei Shu, Genzhi Ye, Xi Xiong, Ameesh Makadia, Meiqi Guo, Laurent Itti, and Jindong Chen. 2026. 3dcodebench: Benchmarking agentic procedural 3d modeling via code. *arXiv preprint arXiv:2606.01057*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. Critic: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Hongcheng Guo, Wei Zhang, Junhao Chen, Yaonan Gu, Jian Yang, Junjia Du, Shaosheng Cao, Binyuan Hui, Tianyu Liu, Jianxin Ma, Chang Zhou, and Zhoujun Li. 2025b. IW-bench: Evaluating large multimodal models for converting image-to-web. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6449–6466, Vienna, Austria. Association for Computational Linguistics.

Jo E Hannay, Tore Dybå, Erik Arisholm, and Dag IK Sjøberg. 2009. The effectiveness of pair programming: A meta-analysis. *Information and software technology*, 51(7):1110–1122.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. 2024. Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In *International Conference on Machine Learning*.

Dong Huang, Qingwen Bu, Jie M Zhang, Michael Luck, and Heming Cui. 2023. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Md Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. 2024. Mapcoder: Multi-agent code generation for competitive problem solving. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4912–4944.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

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*, volume 2024, pages 54107–54157.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. Camel: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023b. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Wenyi Li, Renkai Luo, Yue Yu, Huan-ang Gao, Mingju Gao, Li Yuan, Chaoyou Fu, and Hao Zhao. 2026. Benchmarking phd-level coding in 3d geometric computer vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 30974–30985.

Zongjie Li, Chaozheng Wang, Zhibo Liu, Haoxuan Wang, Dong Chen, Shuai Wang, and Cuiyun Gao. 2023c. Cctest: Testing and repairing code completion systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1238–1250. IEEE.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023a. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572.

Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. 2023b. Verilogeval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–8.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. 2024. Rtllm: An open-source benchmark for design rtl generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 722–727.

Kim Man Lui and Keith CC Chan. 2006. Pair programming productivity: Novice–novice vs. expert–expert. *International Journal of Human-computer studies*, 64(9):915–925.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36.

Xingyu Miao, Junting Dong, Qin Zhao, Yuhang Yang, Junhao Chen, and Yang Long. 2026. From frames to sequences: Temporally consistent human-centric dense prediction. *Preprint*, arXiv:2602.01661.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning*, pages 26106–26128.

Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. 2023. Codegen2: Lessons for training llms on programming and natural languages. *arXiv preprint arXiv:2305.02309*.

OpenAI. 2023. GPT-3.5 documentation. `https://platform.openai.com/docs/models/gpt-3-5`. Accessed: October 21, 2023.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. 2024. Dinov2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. 2026. Scenesmith: Agentic generation of simulation-ready indoor scenes. *arXiv preprint arXiv:2602.09153*.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 6.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Zhen Liu, Tim Z. Xiao, Katherine M. Collins, Joshua B. Tenenbaum, Adrian Weller, Michael J. Black, and Bernhard Schölkopf. 2025. Can large language models understand symbolic graphics programs? In *The Thirteenth International Conference on Learning Representations*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763.

Juan A Rodriguez, Abhay Puri, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. 2025. Starvector: Generating scalable vector graphics code

from images and text. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 16175–16186.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

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 *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3956–3974.

Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 2023. 3d-gpt: Procedural 3d modeling with large language models. *arXiv preprint arXiv:2310.12945*.

Mingze Sun, Junhao Chen, Junting Dong, Yurun Chen, Xinyu Jiang, Shiwei Mao, Puhua Jiang, Jingbo Wang, Bo Dai, and Ruqi Huang. 2025. Drive: Diffusion-based rigging empowers generation of versatile and expressive characters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21170–21180.

Mingze Sun, Cheng Zeng, Jiansong Pei, Junhao Chen, Chaoyue Song, Shaohui Wang, Tianyuan Chang, Bin Huang, Zijiao Zeng, and Ruqi Huang. 2026. Animator-centric skeleton generation on objects with fine-grained details. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17336–17345.

Yuchen Tian, Weixiang Yan, Qian Yang, Xuandong Zhao, Qian Chen, Wen Wang, Ziyang Luo, Lei Ma, and Dawn Song. 2025. Codehalu: Investigating code hallucinations in llms via execution-based verification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25300–25308.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

YunDa Tsai, Mingjie Liu, and Haoxing Ren. 2024. Rtlfixer: Automatically fixing rtl syntax errors with large language model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*.

Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, et al. 2025. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features. *arXiv preprint arXiv:2502.14786*.

Karthikeyan Umapathy and Albert D Ritzhaupt. 2017. A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education (TOCE)*, 17(4):1–13.

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

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

Fangsheng Weng, Junhao Chen, Xiang Li, Jie Qin, Hanzhong Guo, ShaochunHao, and Xiaoguang Han. 2026a. GarmentGPT: Compositional garment pattern generation via discrete latent tokenization. In *The Fourteenth International Conference on Learning Representations*.

Jiawei Weng, Saining Zhang, Zhenxin Diao, Peishuo Li, Henghaofan Zhang, Junhao Chen, and Hao Zhao. 2026b. Feedforward 3d editing learns from semantic-part transformation. *Preprint*, arXiv:2605.27351.

Laurie Williams, Robert R Kessler, Ward Cunningham, and Ron Jeffries. 2000. Strengthening the case for pair programming. *IEEE software*, 17(4):19–25.

Chengyue Wu, Yixiao Ge, Qiushan Guo, Jiahao Wang, Zhixuan Liang, Zeyu Lu, Ying Shan, and Ping Luo. 2024. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. *arXiv preprint arXiv:2405.07990*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Ronghuan Wu, Wanchao Su, and Jing Liao. 2025. Chat2svg: Vector graphics generation with large language models and image diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, et al. 2024a. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. *arXiv preprint arXiv:2406.09961*.

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

Yikang Yang, Zhanpeng Hu, Youtian Lin, Mengqi Zhou, Jingxi Xu, Feihu Zhang, Jiaheng Liu, and Yao Yao. 2026. P3d-bench: Benchmarking mllms for parametric 3d generation and structural reasoning. *arXiv preprint arXiv:2606.11152*.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, et al. 2024c. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11789–11804.

Xiaojun Ye, Junhao Chen, Xiang Li, Haidong Xin, Chao Li, Sheng Zhou, and Jiajun Bu. 2024. Mmad: Multimodal movie audio description. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 11415–11428.

Baoli Zhang, Haining Xie, Pengfan Du, Junhao Chen, Pengfei Cao, Yubo Chen, Shengping Liu, Kang Liu, and Jun Zhao. 2023. Zhujiu: A multi-dimensional, multi-faceted chinese benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 479–494.

Ziyao Zhang, Chong Wang, Yanlin Wang, Ensheng Shi, Yuchi Ma, Wanjun Zhong, Jiachi Chen, Mingzhi Mao, and Zibin Zheng. 2025. Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):481–503.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Wanxiang Che, Zhiyuan Liu, and Maosong Sun. 2025. Chartcoder: Advancing multimodal large language model for chart-to-code generation. *arXiv preprint arXiv:2501.06598*.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, et al. 2023. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5673–5684.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*.

A Multi-Round PairCoder Traces on Code-Driven Multimodal Benchmarks

This appendix documents complete PairCoder sessions on the multimodal benchmarks of Sec. 5. Each trace shows the Driver's program, the verification evidence ψ attached to the Navigator's review (execution status and, for image conditioned tasks, the structural similarity of the current rendering to the target), the Navigator's verbatim verdict, and the error triggered role switches. Renders of the rounds accompany each trace so the effect of each review can be inspected visually. Traces use `gpt-5.4-mini` unless stated otherwise; prompts and the review protocol are identical to the main experiments.

A.1 Case 1 (Plot2Code): review, role switch, and cross acceptance

The Driver reproduces a three panel errorbar chart from the target image (Fig. 9). Its first program already runs and is close to the target, but the Navigator, looking at the current rendering next to the target, finds a concrete visual deviation and rejects the round. After the role switch the former Navigator takes the keyboard and applies its own fix, and the former Driver, now reviewing, accepts with `[NOERROR]`. The session costs two generations and two reviews.

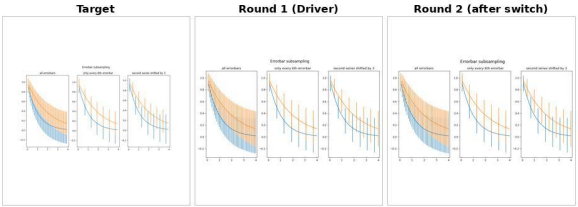


Figure 9: Case 1 rendering progression: target (left), the Driver's first program (middle), the program after the role switch (right).

Round 1, Driver (initial program, excerpt)

```
fig, axs = plt.subplots(1, 3,
    figsize=(12, 6), sharex=True)
fig.suptitle("Errorbar subsampling",
    fontsize=14)
axs[0].errorbar(x, y1, yerr=yerr, ...)
axs[1].errorbar(x, y1, yerr=yerr,
    errorevery=6)
axs[2].errorbar(x, y1, yerr=yerr,
    errorevery=(6, 3))
```

Verification evidence ψ shown to the Navigator

```
runs OK; structural similarity to
the target chart = 0.876
```

Round 1, Navigator review (verbatim)

```
The code has a visual mismatch with
the target figure:
- Line causing the mismatch:
  fig.suptitle("Errorbar subsampling",
               fontsize=14)
- Why: the target image uses a larger,
  more prominent figure title and
  slightly different top spacing. With
  fontsize=14 the title is too small
  compared to the reference.
- Fix: increase the suptitle size and
  adjust its vertical placement.
```

Round 2, after role switch: former Navigator writes the fix (diff)

```
- fig.suptitle("Errorbar subsampling",
-     fontsize=14)
+ fig.suptitle("Errorbar subsampling",
+     fontsize=18, y=0.98)
```

Round 2, former Driver now reviews

```
[NOERROR]
```

A.2 Case 2 (Plot2Code): review oscillation and the quality fallback

A harder session on a two panel contour task. The pair improves the program over three revision rounds (contour levels, then colorbars), but the two reviewers disagree about whether the target contains colorbars, so no round is accepted within the budget. PairCoder then falls back to the quality criterion of Algorithm 1 and returns the round with the highest verified similarity score, which is also the visually closest rendering (Fig. 10). The fallback turns an unresolved review debate into a safe outcome instead of a failure.

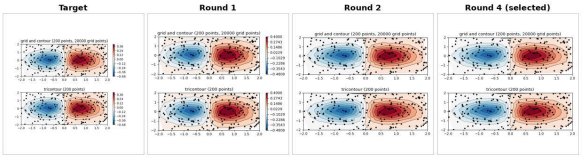


Figure 10: Case 2 rendering progression: target, rounds 1 and 2, and the round selected by the quality fallback after budget exhaustion.

Round 1, Navigator review (verbatim, trimmed)

```
The code has a definite mismatch:
- Problem line:
  levels = np.linspace(-0.48, 0.40, 15)
- Why: the prompt says the plots should
  use 14 levels, and the target image's
  contour/colorbar scaling does not
  match this fixed custom range.
- Concrete fix: use 14 contour levels
  directly in both plots.
```

Round 2, former Driver now reviews (verbatim, trimmed)

```
Missing colorbars.
- The target figure shows a colorbar
  beside each subplot, but the code
  never adds one. The exact missing
  lines are:
  fig.colorbar(cf, ax=ax)
  fig.colorbar(cf2, ax=ax)
```

Round 3, the other agent disagrees (verbatim, trimmed)

```
The code still has a concrete mismatch:
- Problem lines: fig.colorbar(cf, ax=ax)
  and fig.colorbar(cf2, ax=ax)
- Why: the target image shows NO
  colorbars. Adding them changes the
  layout and shrinks the axes.
```

A.3 Case 3 (DaTikZ, `doubao-1.5-lite`): compile error repair on a weak model

The same protocol on the weakest model in our pool, where PairCoder lifts the DaTikZ compile rate from 0.550 to 0.850. The Driver's first Ti*k*Z program fails to compile because its caption references an undefined cross reference macro. The Navigator receives the compiler verdict as evidence, identifies the exact offending construct, and proposes the minimal removal; after the role switch the repaired program compiles and the former Driver accepts (Fig. 11).

Figure 11: Case 3 rendering progression on `doubao-1.5-lite`: target (left), the Driver's first (non-compiling) program (middle), the repaired program after the role switch (right).

Verification evidence ψ shown to the Navigator

```
LaTeX compilation FAILED
```

Round 1, Navigator review (verbatim, trimmed)

```
The error is that the code uses
\cref{claim:paths} which is not defined
anywhere in the provided code. To fix
this, either define the claim:paths
macro appropriately in the preamble or
remove the reference from the caption.
Specific fix: remove the
\cref{claim:paths} part from the
caption.
```

Round 2, former Driver now reviews

```
[NOERROR]
```

A.4 Case 4 (DaTikZ): first round acceptance, the cheap path

For completeness, the most common trace shape: the Driver's first program compiles, the Navigator receives the passing evidence, finds no concrete error, and accepts immediately with `[NOERROR]`. The session costs one generation plus one review, roughly twice a single pass, which is why the average token multiplier in Sec. 4.5 stays in the 3 to 9 range even though failing tasks iterate several times.

Verification evidence ψ

```
compiles and renders OK
```

Round 1, Navigator

```
[NOERROR]
```

B Per-Benchmark Qualitative and Quantitative Results

This appendix devotes one full page to each artifact benchmark: a gallery of qualitative comparisons on top and that benchmark's complete measured metric suite below. In every gallery triplet the target or ground truth rendering is on the left, the single model baseline's artifact is in the middle, and PairCoder's artifact is on the right (copper border), with the task id and input quoted beneath; each example is a distinct task. For the chart, plot, and vector benchmarks every example has both arms rendering with PairCoder quantitatively closer to the target, so the comparison isolates artifact quality. For DaTikZ, GenCAD-Code, and 3DCodeBench, where PairCoder's measured advantage is concentrated in executability, the galleries lead with fidelity gains and then show executability repairs, in which the baseline produces no artifact (gray panel) and PairCoder renders successfully. Each table reports every metric we measured for that benchmark, as "single model → PairCoder," with green cells marking improvements and aggregate scores counting non rendering generations as zero.

C Cross-Model Qualitative Comparisons

The per-benchmark galleries above use `gpt-5.4-mini`. To show that the improvement is not specific to one backbone, Figures 19–23 hold the benchmark and the task fixed and vary the base model: each page covers one benchmark across four models (columns) and three shared tasks (stacked blocks, the shared target shown in each strip), with the single model artifact above the PairCoder artifact for the same model and task. The chart, plot, and vector panels use tasks where both arms render, so the comparison isolates quality; the 3DCodeBench panel uses tasks where several backbones fail to render and PairCoder repairs execution (gray "no render" marks the failure).

D P3D-Bench: Detailed Results

Full geometry breakdown. Table 11 gives the full P3D-Bench geometry breakdown, adding the capped Chamfer score (CD) to the F-score, normal-consistency (NC), and volumetric-IoU sub-metrics already reported per model in Table 1. All are the worst-filled mean over the 400-case text-to-3D split (invalid programs score 0, so the metric jointly rewards executability and fidelity). PairCoder improves *every* sub-metric on *every* model. The gains are largest exactly where the single-model baseline leaves headroom: on `gpt-5.4` (71.5% valid) F-score@0.05 rises $0.443 \rightarrow 0.619$, NC $0.435 \rightarrow 0.605$, and IoU $0.226 \rightarrow 0.334$, while on the weak `doubao-1.5-lite` (14.0% valid) every sub-metric roughly triples. Two facts isolate *where* the gain comes from. First, on the cases the baseline already compiles, the mean raw Chamfer distance (unnormalized, lower better) is essentially unchanged (e.g. $0.0042 \rightarrow 0.0042$ on `doubao-seed-2.0-mini`, $0.0029 \rightarrow 0.0029$ on `gpt-5.4-mini`): PairCoder does not perturb solids that already build. Second, the worst-filled sub-metrics nonetheless rise sharply, because PairCoder converts a large fraction of *non-executable* programs into valid solids that recover the target's structure: the coverage gain, not a re-optimization of already-valid geometry, drives the numbers, consistent with the conservative-review design.

Topology breakdown. Table 12 expands the P3D-Bench topology column of Table 1 into its three official sub-metrics, again as worst-filled means: the fraction of solids with no open edge (NoOE), and one minus the inverted-normal (InvN) and non-manifold-edge (NM) ratios; all lie in $[0, 1]$ with higher better. The pattern mirrors geometry: PairCoder improves every topology sub-metric on every model, most strongly where the baseline leaves headroom (`gpt-5.4` NoOE $0.695 \rightarrow 0.970$, `doubao-1.5-lite` NoOE $0.130 \rightarrow 0.400$). Because a program must build a valid solid before it can be watertight or manifold, these gains are again driven by converting non-executable programs into well-formed solids rather than by repairing already-valid meshes.

Qualitative examples: geometry refinement. On P3D-Bench text-to-3D the compile-grounded PairCoder of Table 1 gains by *coverage*: it repairs non-executable programs into valid solids and leaves already-compiling programs unchanged, so on the common set the two solids are identical. We additionally evaluate a vision-grounded geometry-refinement variant: when the single-model program compiles to a valid but geometrically poor

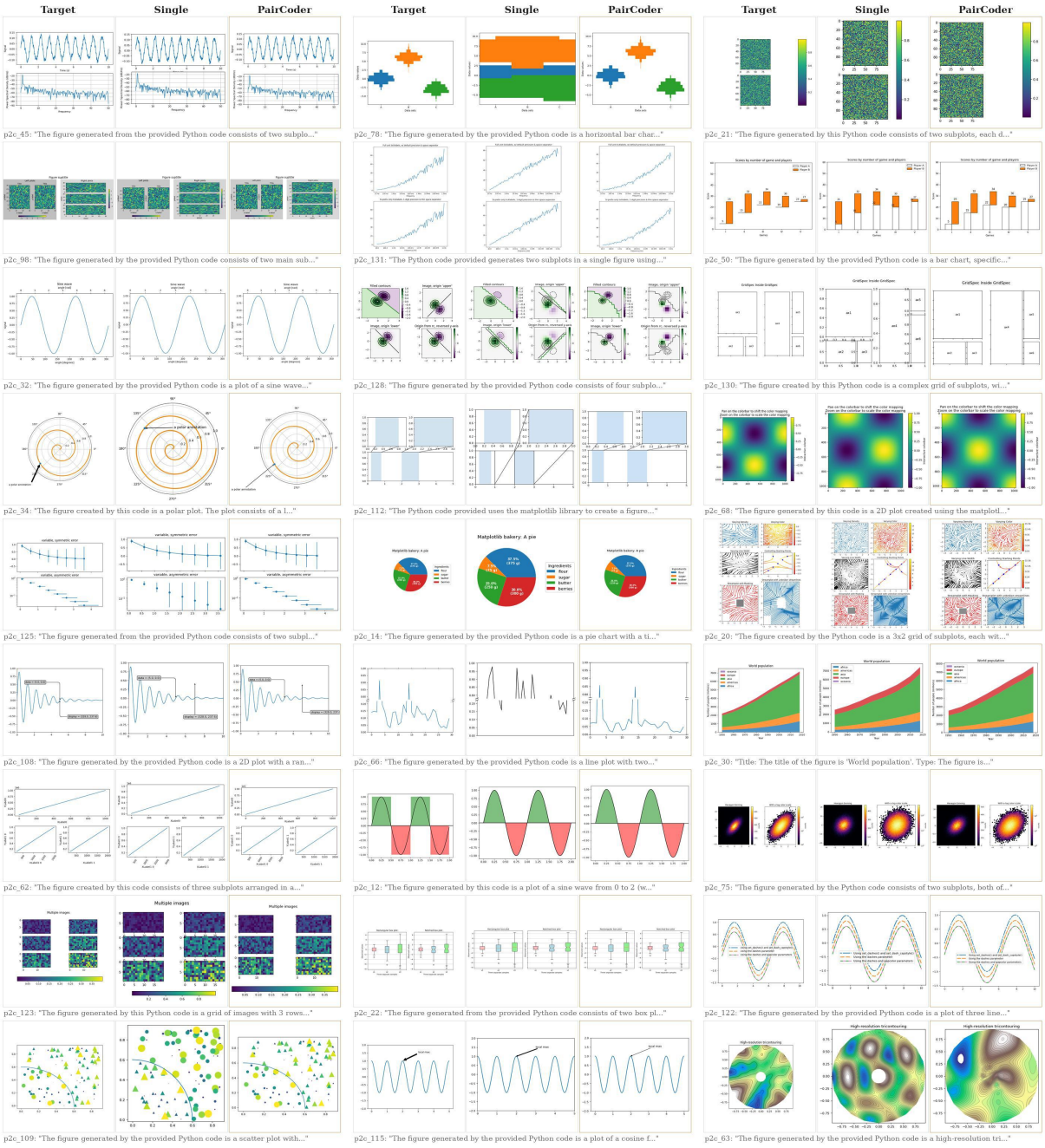


Figure 12: Plot2Code qualitative gallery (27 distinct examples); both arms render and PairCoder is quantitatively closer to the target.

Model	exec ↑	SSIM ↑	CLIP ↑
gpt-5.4-mini	0.841 → **0.962**	0.412 → **0.474**	0.802 → **0.912**
gpt-5.4	0.962 → **0.977**	0.476 → **0.495**	0.916 → **0.920**
gpt-5.5	0.985 → 0.977	0.487 → **0.496**	0.943 → 0.931
doubao-seed-2.0-mini	0.909 → **0.917**	0.410 → **0.443**	0.835 → 0.823

Table 3: Plot2Code: full measured metric suite for every applicable model (single model → **PairCoder**).

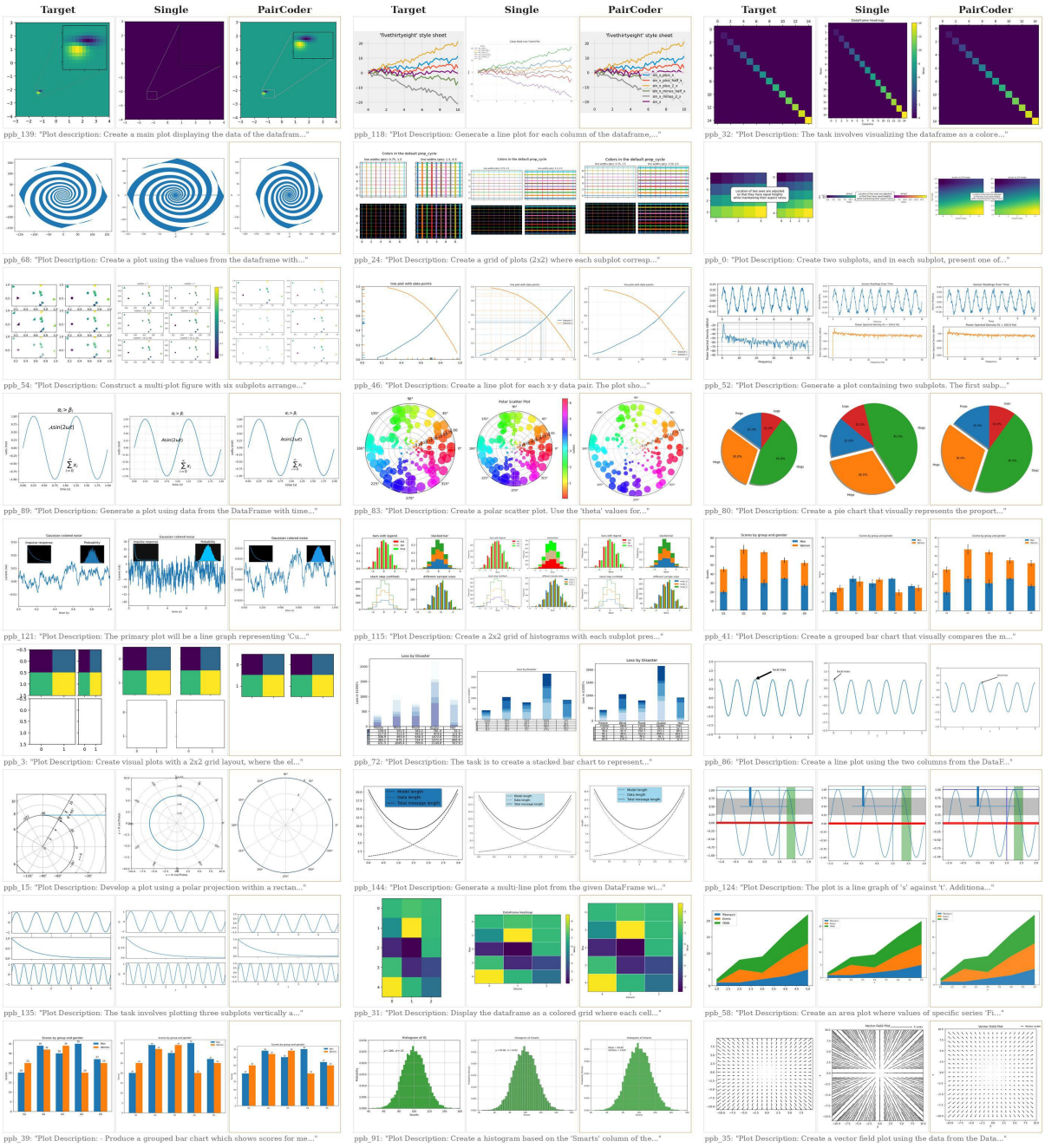


Figure 13: PandasPlotBench qualitative gallery (27 distinct examples); both arms render and PairCoder is quantitatively closer to the target.

Model	exec ↑	SSIM ↑	CLIP ↑
gpt-5.4-mini	0.789 → **0.851**	0.423 → **0.461**	0.728 → **0.794**
gpt-5.4	0.966 → **0.994**	0.567 → **0.610**	0.909 → **0.944**
gpt-5.5	0.960 → **0.977**	0.597 → **0.644**	0.907 → **0.942**
doubao-seed-2.0-mini	0.909 → **0.989**	0.491 → **0.534**	0.846 → **0.922**

Table 4: PandasPlotBench: full measured metric suite for every applicable model (single model → **PairCoder**).

solid, the Navigator renders the solid and reviews it against the specification, and the Driver refines the parametric program (kept only if it still compiles). Across a sample of 40 mid-quality valid solids ($0.2 < \text{F@}0.05 < 0.8$) on `gpt-5.4`, this refinement raises F-score@0.05 on 29 and regresses on 3 (the kept-if-compiles safeguard bounds regressions), for a mean gain of $+0.26$; it is thus a genuine effect rather than a cherry-pick. Figure 24 shows cases where both the single model and this variant build valid solids, but the refined one is much closer to the target: each triplet renders the ground-truth solid (green), the single-model solid (pink; valid but wrong), and the PairCoder-refined solid (blue), with the per-case F-score@0.05 improvement annotated.

A second Text-to-3D format: OpenSCAD. To test whether the P3D-Bench gain is specific to the minimal-JSON format, we repeat the 400-case Text-to-3D experiment with OpenSCAD, a constructive-solid scripting language, on the three models for which both formats were run. Table 13 reports validity, the fraction of programs that compile to a valid solid, for both formats side by side. Two observations stand out. First, PairCoder drives OpenSCAD validity to a perfect 1.000 on all three models, repairing every non-compiling program. Second, the single-model baseline is markedly more fluent in OpenSCAD than in the minimal-JSON format (for example `gpt-5.4` baseline validity $0.715 \rightarrow 0.932$), so the coverage headroom PairCoder recovers is correspondingly smaller: its OpenSCAD validity gain is $+6.8$ points on `gpt-5.4` versus $+27.5$ points in minimal-JSON. This is the same headroom-tracking pattern seen across the rest of the grid. The gain concentrates where the baseline leaves room; OpenSCAD's higher baseline fluency compresses it, while PairCoder still closes the remaining gap to perfect executability in both formats.

Image-to-3D and assembly-3D demonstration. The other two P3D-Bench tracks, image-to-3D and assembly-3D, condition on a rendered image of the target (assembly-3D also on part annotations) and require a vision-capable model; their full evaluation sets build on the licensed Fusion 360 Gallery. We run the three in-repo demo cases per track with `gpt-5.4` across the CadQuery and Three.js program formats (OpenSCAD is already evaluated at scale on text-to-3D, Table 13), scoring the same non-VLM buckets as text-to-3D. Table 14 reports the result. In CadQuery, PairCoder repairs the one non-compiling image-to-3D case ($0.667 \rightarrow 1.000$ valid) and recovers a valid assembly-3D solid from a baseline that builds none ($0 \rightarrow 0.333$), via a sibling candidate. In Three.js, `gpt-5.4` already compiles all three image-to-3D cases, so PairCoder ties, consistent with the headroom pattern seen across the grid. The three-case sample is illustrative rather than conclusive, but it shows the compiler-grounded mechanism reaches the image-conditioned tracks and a web-rendering representation, spanning all four P3D-Bench program formats together with text-to-3D.

E Reproducibility Notes

For every benchmark we use the official task set and scorer where released (ChartMimic, Plot2Code, PandasPlotBench, StarVector, DaTikZ, 3DCodeBench, GenCAD-Code, VerilogEval, RTLLM, WebApp1K, LiveCodeBench, BigCodeBench, DS-1000, HumanEval-X), identical prompts and code extraction in both arms, and provider default sampling with thinking disabled. Visual metrics are computed with SSIM at 256×256 and the public CLIP, DINOv2, and SigLIP-2 checkpoints; Chamfer distance is computed on normalized surface samples, and aggregate variants assign non executing generations the worst observed score so that executability cannot be traded for similarity. Token counts are taken from provider usage fields and logged separately for the baseline arm and the PairCoder arm.

Figure 14: ChartMimic qualitative gallery (23 distinct examples); both arms render and PairCoder is quantitatively closer to the target.

Model	exec ↑	SSIM ↑	CLIP ↑
gpt-5.4-mini	0.967 → 0.967	0.578 → **0.578**	0.871 → 0.861
gpt-5.4	0.917 → **0.983**	0.533 → **0.582**	0.820 → **0.848**
gpt-5.5	0.983 → **1.000**	0.593 → **0.613**	0.874 → **0.887**
doubao-seed-2.0-mini	0.900 → **0.950**	0.443 → **0.476**	0.770 → **0.808**

Table 5: ChartMimic: full measured metric suite for every applicable model (single model → **PairCoder**).

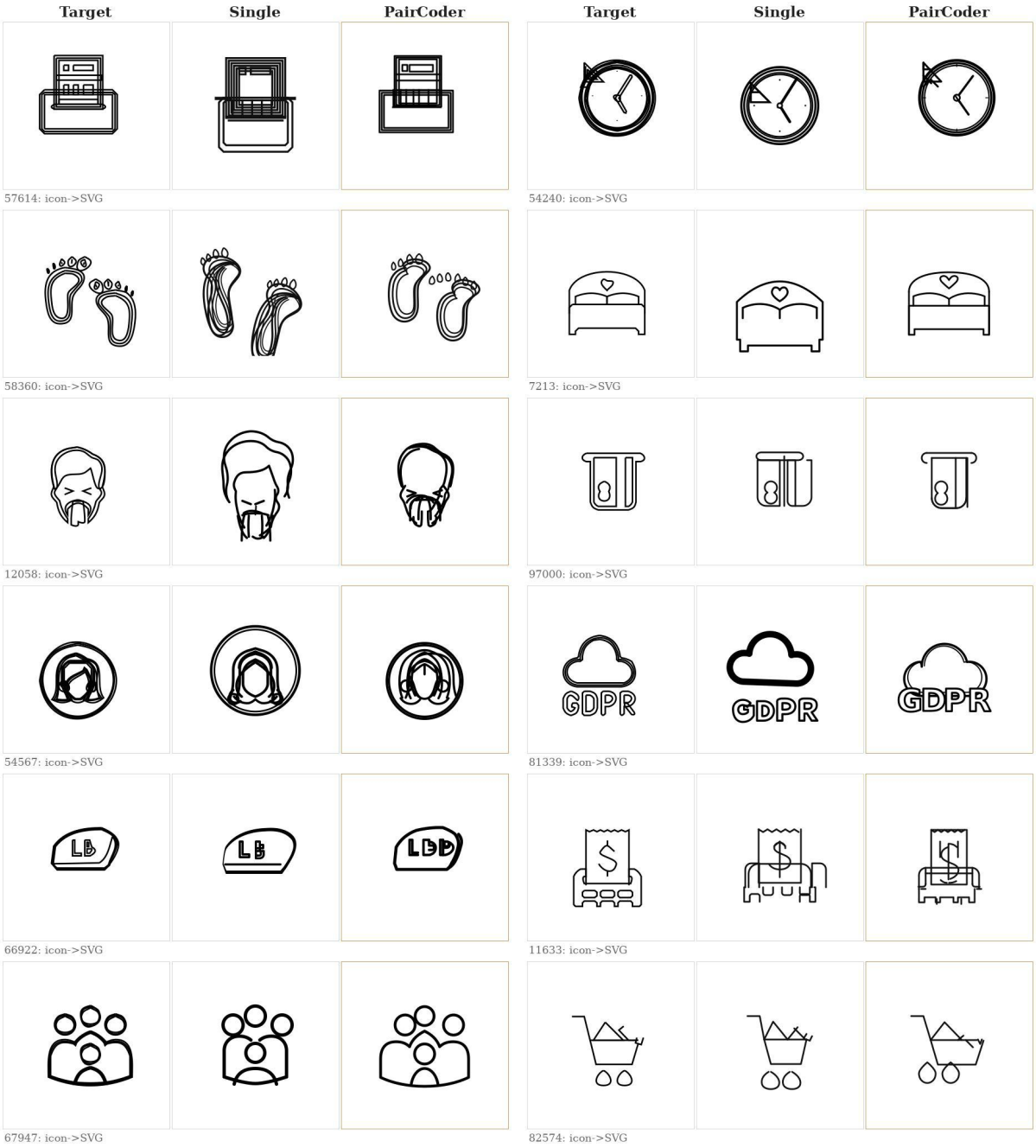


Figure 15: StarVector qualitative gallery (12 distinct examples); both arms render and PairCoder is quantitatively closer to the target.

Model	render rate ↑	SSIM ↑	CLIP ↑	DINO ↑
gpt-5.4-mini	0.983 → **1.000**	0.784 → **0.801**	0.929 → **0.944**	0.874 → **0.885**
gpt-5.4	0.900 → **1.000**	0.776 → **0.873**	0.859 → **0.954**	0.820 → **0.906**
gpt-5.5	1.000 → 1.000	0.777 → **0.787**	0.961 → 0.960	0.938 → 0.934
doubao-seed-2.0-mini	0.983 → **1.000**	0.758 → **0.770**	0.883 → **0.915**	0.793 → **0.819**

Table 6: StarVector: full measured metric suite for every applicable model (single model → **PairCoder**).

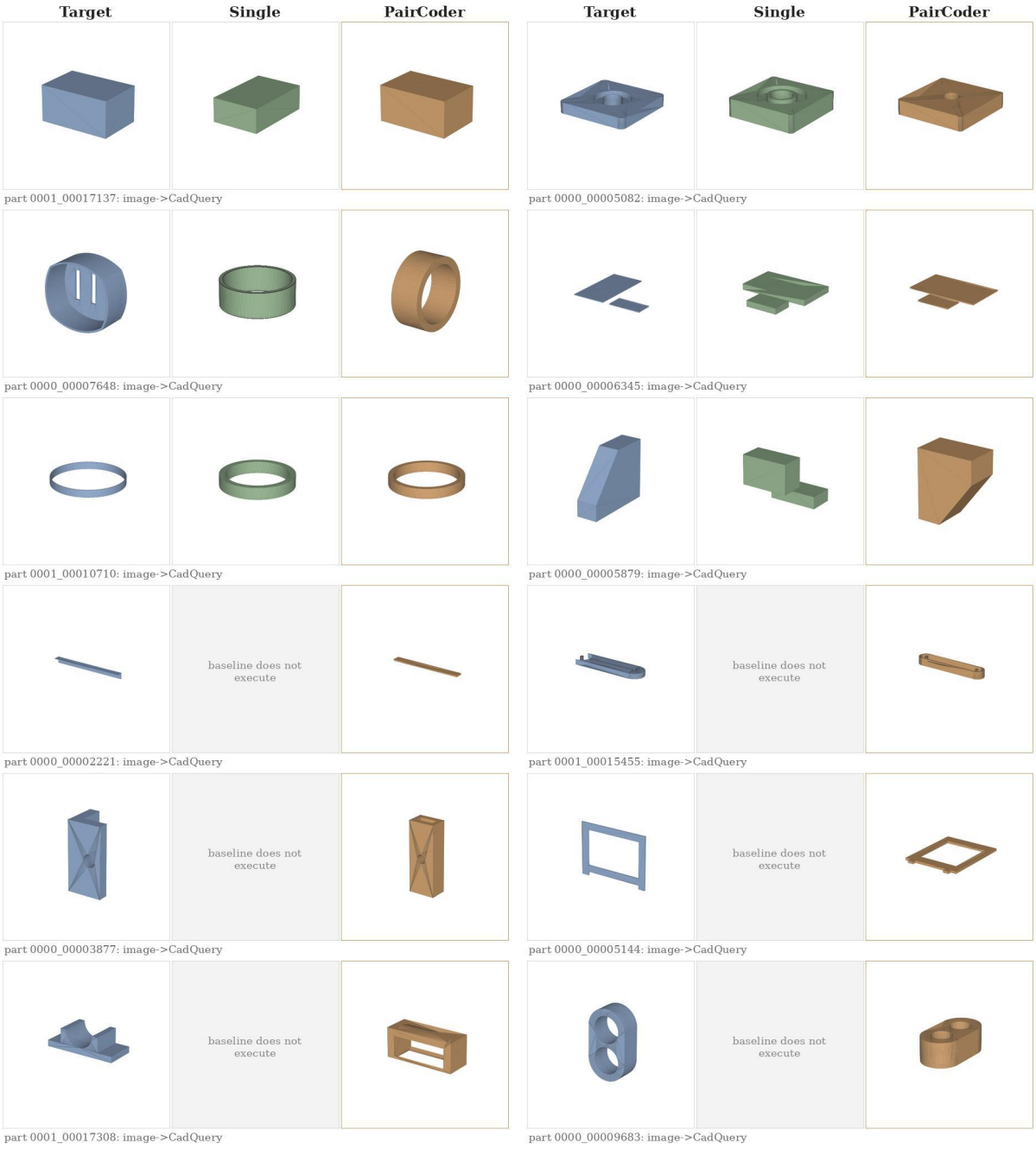


Figure 16: GenCAD-Code qualitative gallery (12 distinct examples) demonstrating PairCoder's advantage; gray panels mark baselines that produce no artifact.

Model	exec ↑	Chamfer (cond.) ↓	Chamfer (agg.) ↓
gpt-5.4-mini	0.900 → **0.983**	0.148 → 0.152	0.233 → **0.166**
gpt-5.4	0.867 → **0.983**	0.145 → **0.140**	0.259 → **0.155**
doubao-seed-2.0-mini	0.833 → **0.967**	0.169 → 0.185	0.308 → **0.212**
gpt-5.5	0.917 → **1.000**	0.143 → **0.126**	0.215 → **0.126**

Table 7: GenCAD-Code: full measured metric suite for every applicable model (single model → **PairCoder**).

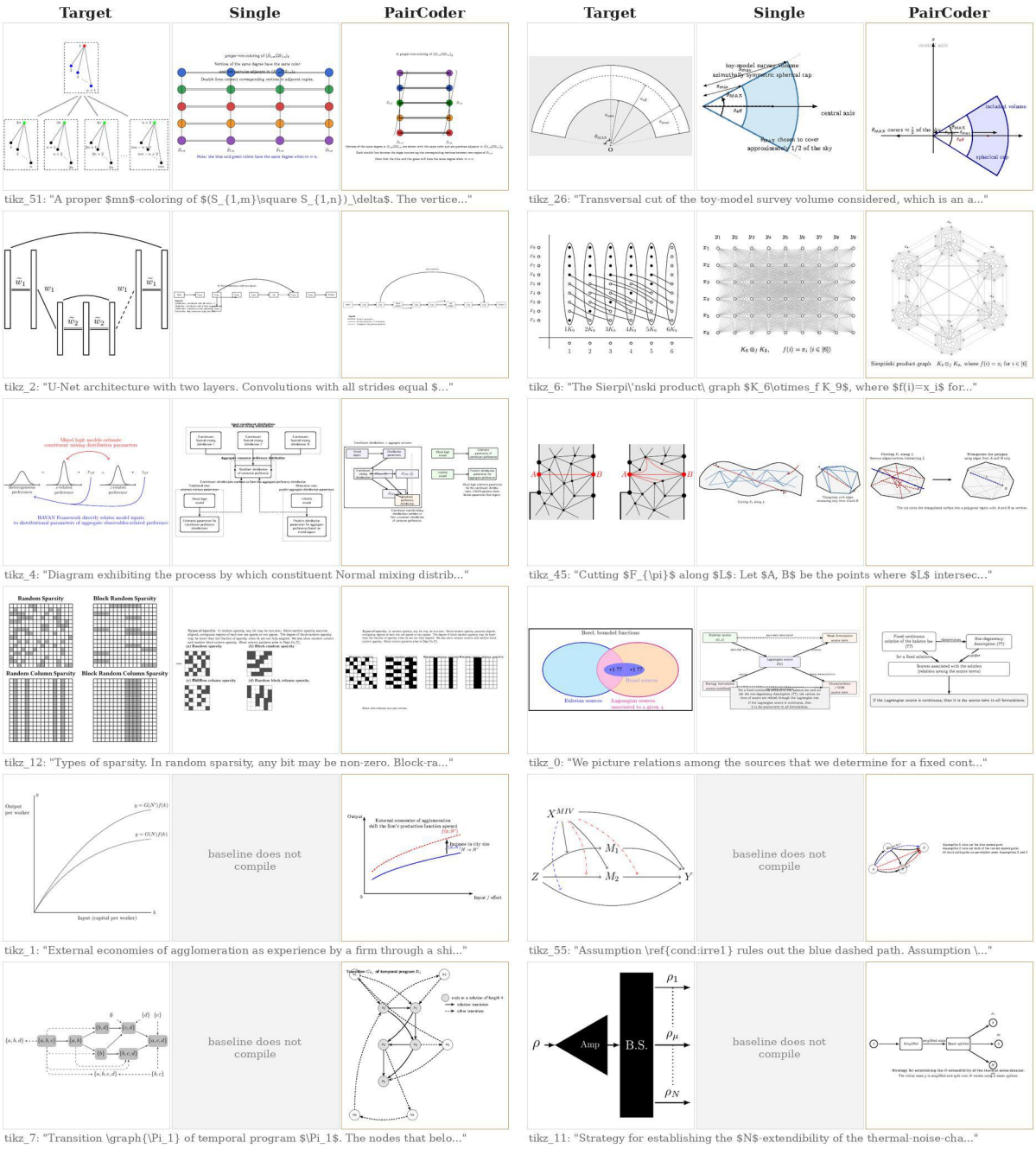


Figure 17: DaTikZ qualitative gallery (12 distinct examples) demonstrating PairCoder's advantage; gray panels mark baselines that produce no artifact.

Model	compile rate ↑	SSIM ↑	CLIP ↑	DINO ↑
gpt-5.4-mini	0.500 → **0.633**	0.215 → **0.256**	0.327 → **0.384**	0.303 → **0.338**
gpt-5.4	0.717 → **0.967**	0.363 → **0.512**	0.566 → **0.770**	0.530 → **0.698**
gpt-5.5	0.783 → **1.000**	0.394 → **0.485**	0.602 → **0.767**	0.554 → **0.730**
doubao-1.5-lite	0.550 → **0.850**	0.406 → **0.601**	0.391 → **0.621**	0.259 → **0.391**
doubao-seed-2.0-mini	0.567 → **0.800**	0.325 → **0.449**	0.441 → **0.606**	0.383 → **0.545**
deepseek-v3.2	0.683 → **0.917**	0.365 → **0.544**	0.508 → **0.702**	0.465 → **0.596**
deepseek-v4-flash	0.633 → **0.733**	0.348 → **0.406**	0.505 → **0.578**	0.445 → **0.519**

Table 8: DaTikZ: full measured metric suite for every applicable model (single model → **PairCoder**).

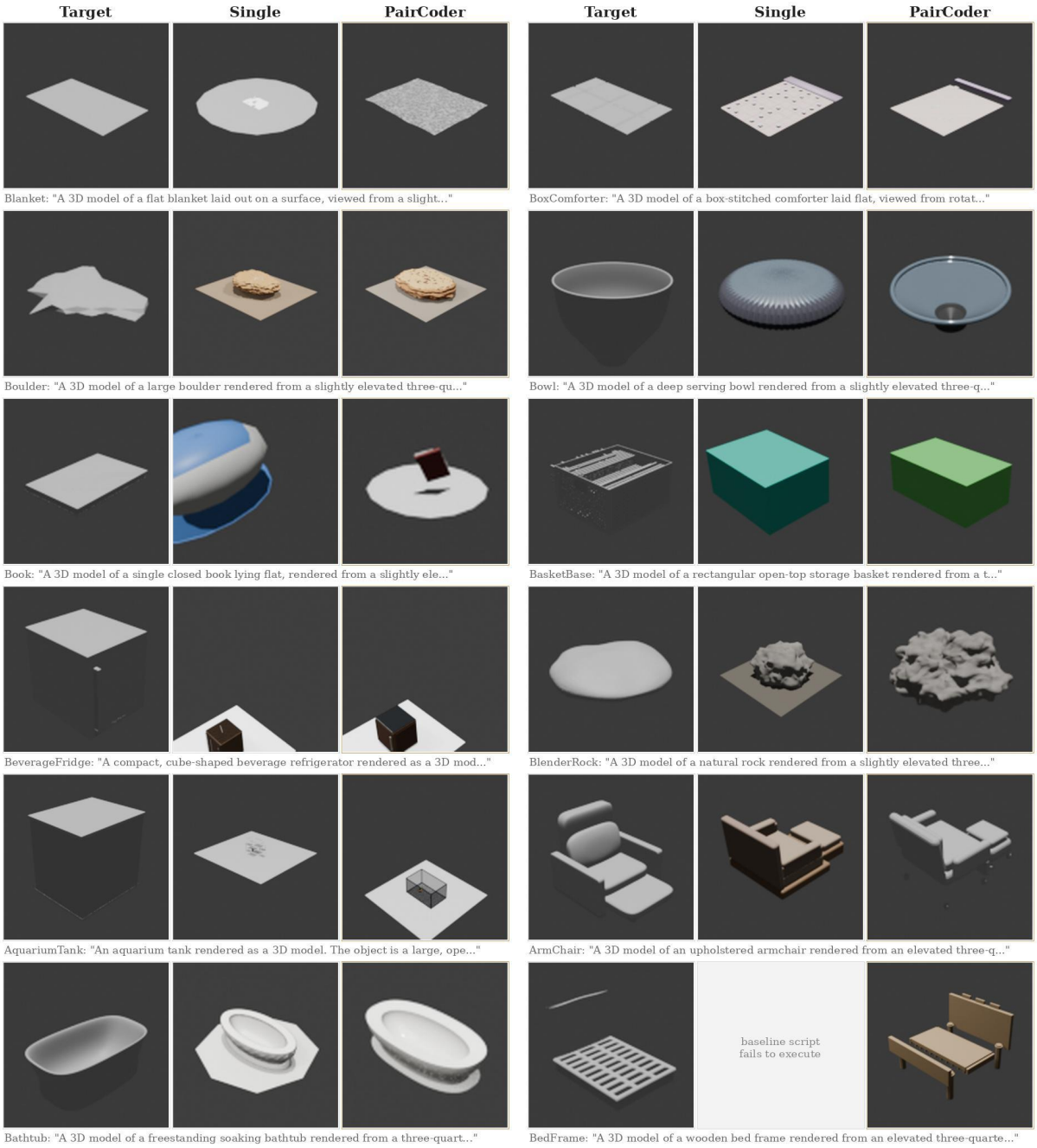


Figure 18: 3DCodeBench qualitative gallery (12 distinct examples) demonstrating PairCoder's advantage; gray panels mark baselines that produce no artifact.

Model	executability ↑
gpt-5.4-mini	0.200 → **0.783**
gpt-5.4	0.433 → **0.783**
gpt-5.5	0.383 → **0.417**
doubao-1.5-lite	0.167 → **0.383**
doubao-seed-2.0-mini	0.200 → **0.533**
deepseek-v3.2	0.333 → **0.633**
deepseek-v4-flash	0.600 → 0.533
Visual / geometric fidelity at `gpt-5.4-mini` *(aggregate, n=60):*	
SigLIP-2 / DINO ↑	.181/.101 → **.702/.376**
Chamfer (aggregate) ↓	4.85 → **2.62**

Table 9: 3DCodeBench: full measured metric suite for every applicable model (single model → **PairCoder**).

Table 10: Per-benchmark token accounting at `gpt-5.4-mini`: single model and PairCoder totals over the full task set, with the cost multiplier; the final row aggregates all runs across all models.

Benchmark	single model tokens	PairCoder tokens	multiplier
RTLLM	161,708	1,272,758	7.9×
BigCodeBench	428,875	3,514,486	8.2×
DS-1000	550,911	1,995,002	3.6×
HumanEval-X JS	445,887	2,575,842	5.8×
WebApp1K	310,709	1,234,990	4.0×
DaTikZ	260,271	1,802,890	6.9×
StarVector	212,628	717,238	3.4×
ChartMimic	226,356	1,341,183	5.9×
GenCAD-Code	205,964	597,230	2.9×
Plot2Code	216,997	2,004,708	9.2×
PandasPlotBench	227,343	1,439,613	6.3×
All runs, all models	16,373,001	121,722,770	7.4×

Table 11: P3D-Bench text-to-3D **geometry** sub-metrics (single → **PairCoder**, 400 cases, worst-filled means, higher better). CD is the capped-normalized Chamfer score $1 - \min(\mathrm{CD}, \tau)/\tau$ (τ=0.01).

Model	Validity ↑	CD ↑	F@0.05 ↑	F@0.01 ↑	NC ↑	IoU ↑
`gpt-5.4-mini`	0.973→**1.000**	0.760→**0.782**	0.746→**0.767**	0.316→**0.326**	0.678→**0.698**	0.486→**0.501**
`gpt-5.4`	0.715→**0.990**	0.453→**0.623**	0.443→**0.619**	0.153→**0.215**	0.435→**0.605**	0.226→**0.334**
`gpt-5.5`	0.672→**0.943**	0.415→**0.564**	0.419→**0.570**	0.143→**0.188**	0.402→**0.556**	0.207→**0.288**
`doubao-1.5-lite`	0.140→**0.427**	0.086→**0.269**	0.094→**0.275**	0.037→**0.104**	0.087→**0.262**	0.054→**0.157**
`doubao-seed-2.0-mini`	0.812→**0.990**	0.542→**0.641**	0.537→**0.633**	0.196→**0.230**	0.504→**0.609**	0.302→**0.360**
`deepseek-v3.2`	0.950→**0.993**	0.693→**0.724**	0.682→**0.714**	0.282→**0.298**	0.640→**0.671**	0.434→**0.460**
`deepseek-v4-flash`	0.945→**1.000**	0.740→**0.780**	0.727→**0.764**	0.298→**0.311**	0.650→**0.685**	0.474→**0.498**

Table 12: P3D-Bench text-to-3D **topology** sub-metrics (single → **PairCoder**, 400 cases, worst-filled means, higher better): no-open-edge fraction (NoOE), and one minus the inverted-normal (InvN) and non-manifold-edge (NM) ratios.

Model	NoOE ↑	InvN ↑	NM ↑
`gpt-5.4-mini`	0.963→**0.990**	0.973→**1.000**	0.968→**0.996**
`gpt-5.4`	0.695→**0.970**	0.715→**0.990**	0.714→**0.988**
`gpt-5.5`	0.662→**0.930**	0.672→**0.943**	0.672→**0.940**
`doubao-1.5-lite`	0.130→**0.400**	0.140→**0.427**	0.139→**0.419**
`doubao-seed-2.0-mini`	0.800→**0.978**	0.812→**0.990**	0.811→**0.988**
`deepseek-v3.2`	0.940→**0.983**	0.950→**0.993**	0.949→**0.992**
`deepseek-v4-flash`	0.945→**1.000**	0.945→**1.000**	0.939→**0.994**

Table 13: P3D-Bench text-to-3D validity across two program formats (single → **PairCoder**, 400 cases, thinking off).

	Validity ↑ (single → **PC**)	
Model	minimal-JSON	OpenSCAD
`gpt-5.4`	0.715→**0.990**	0.932→**1.000**
`doubao-seed-2.0-mini`	0.812→**0.990**	0.925→**1.000**
`deepseek-v3.2`	0.950→**0.993**	0.990→**1.000**

Table 14: P3D-Bench image-to-3D and assembly-3D **three-case demonstration** (single → **PairCoder**, `gpt-5.4`) in the CadQuery and Three.js formats; full evaluation requires the licensed Fusion 360 Gallery.

Track	Format	Validity ↑	Geometry ↑	Topology ↑
Image-to-3D	CadQuery	0.667→**1.000**	0.100→**0.320**	0.667→**1.000**
Image-to-3D	Three.js	1.000→1.000	0.417→0.417	0.778→0.778
Assembly-3D	CadQuery	0.000→**0.333**	0.000→**0.068**	0.000→**0.333**

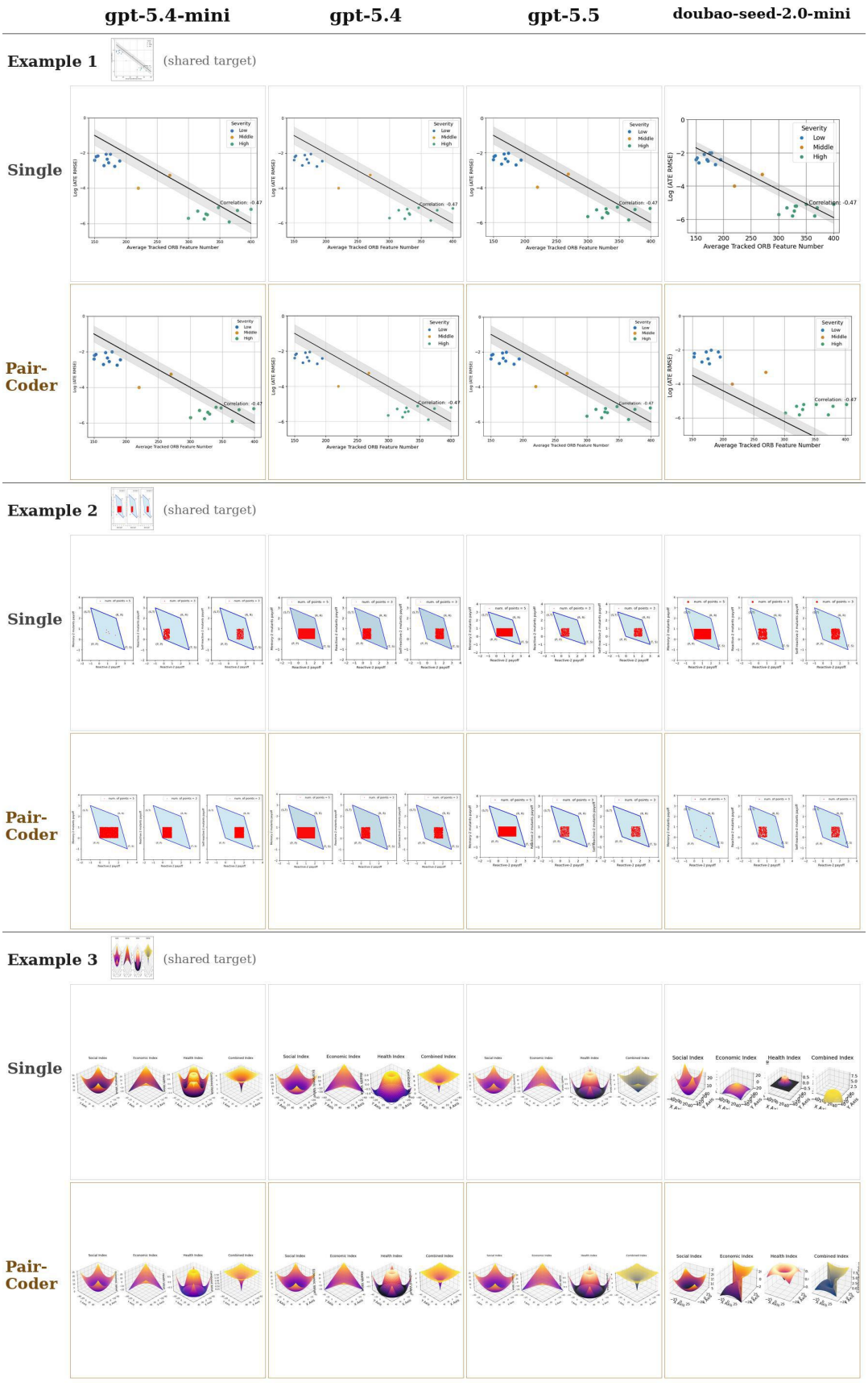


Figure 19: ChartMimic across four models and three shared tasks (single vs. PairCoder per cell).

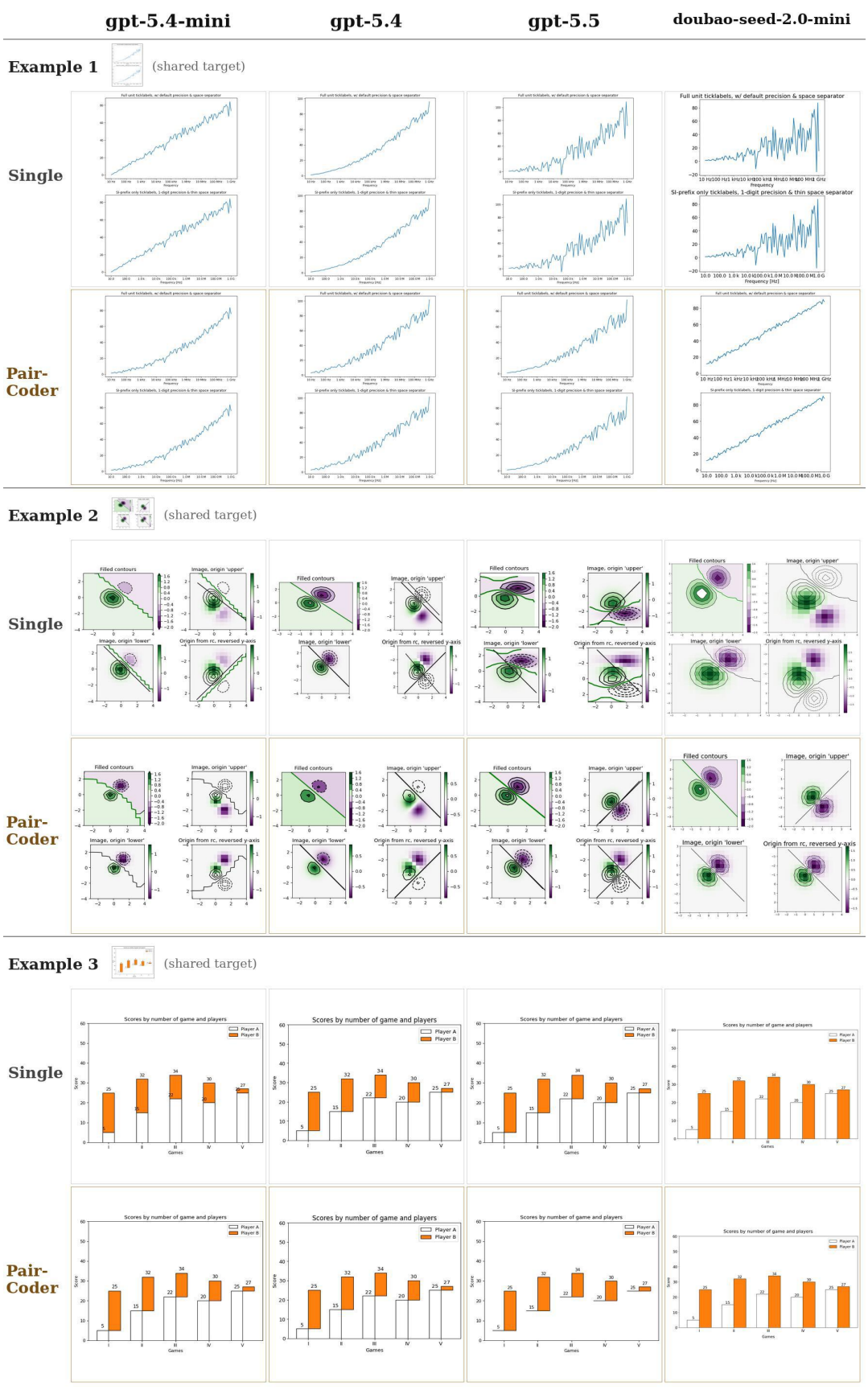


Figure 20: Plot2Code across four models and three shared tasks.

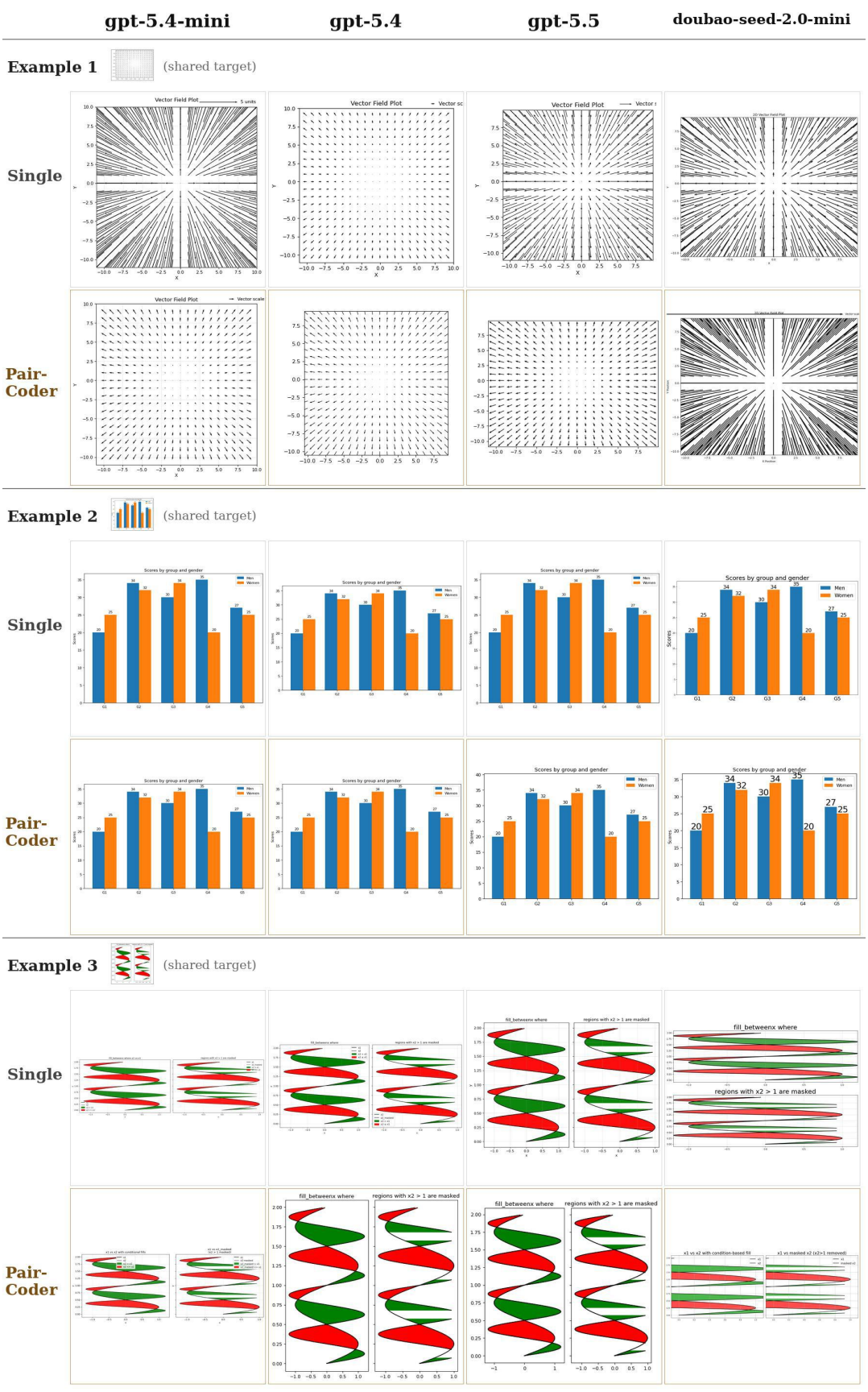


Figure 21: PandasPlotBench across four models and three shared tasks.

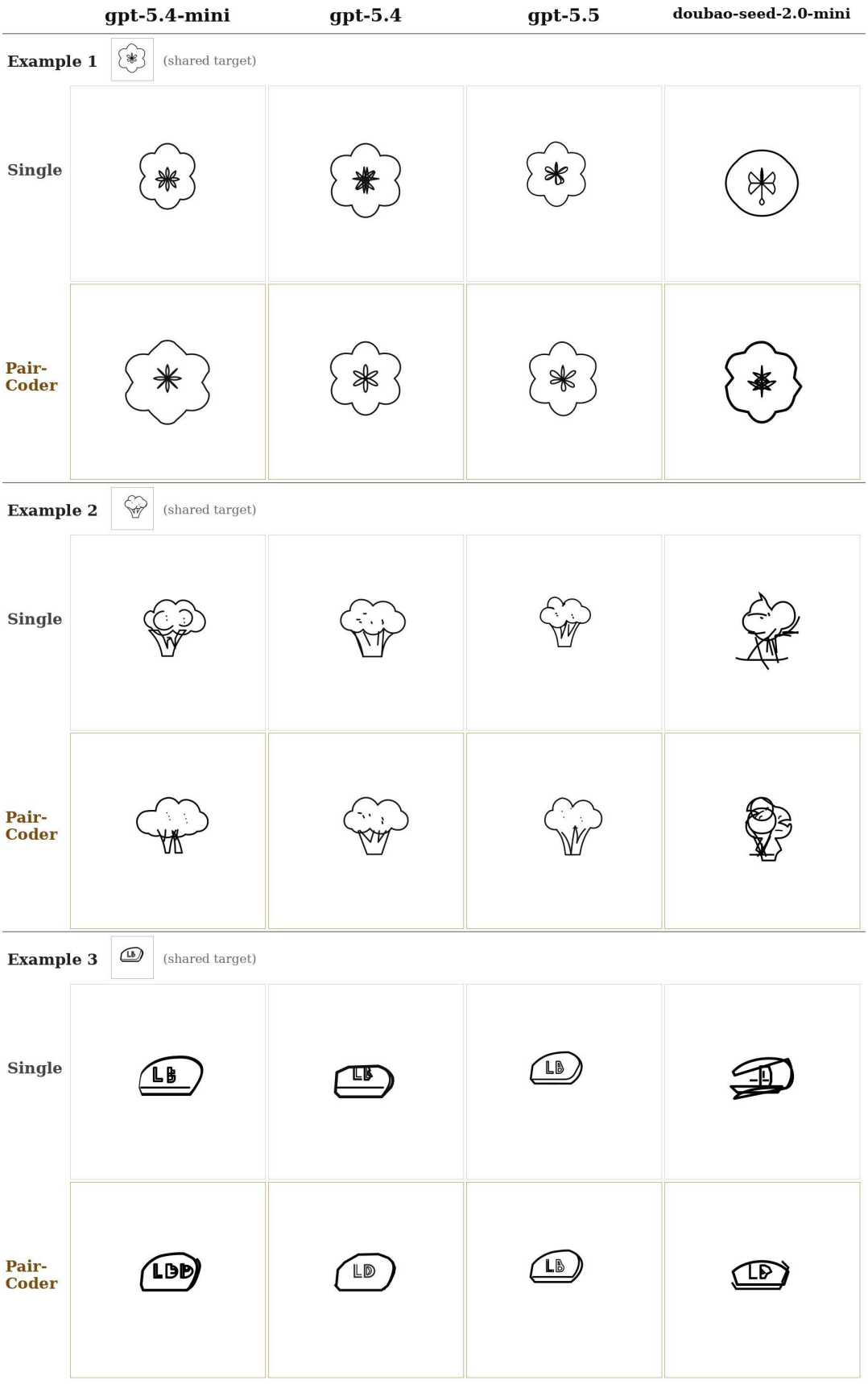


Figure 22: StarVector across four models and three shared tasks.

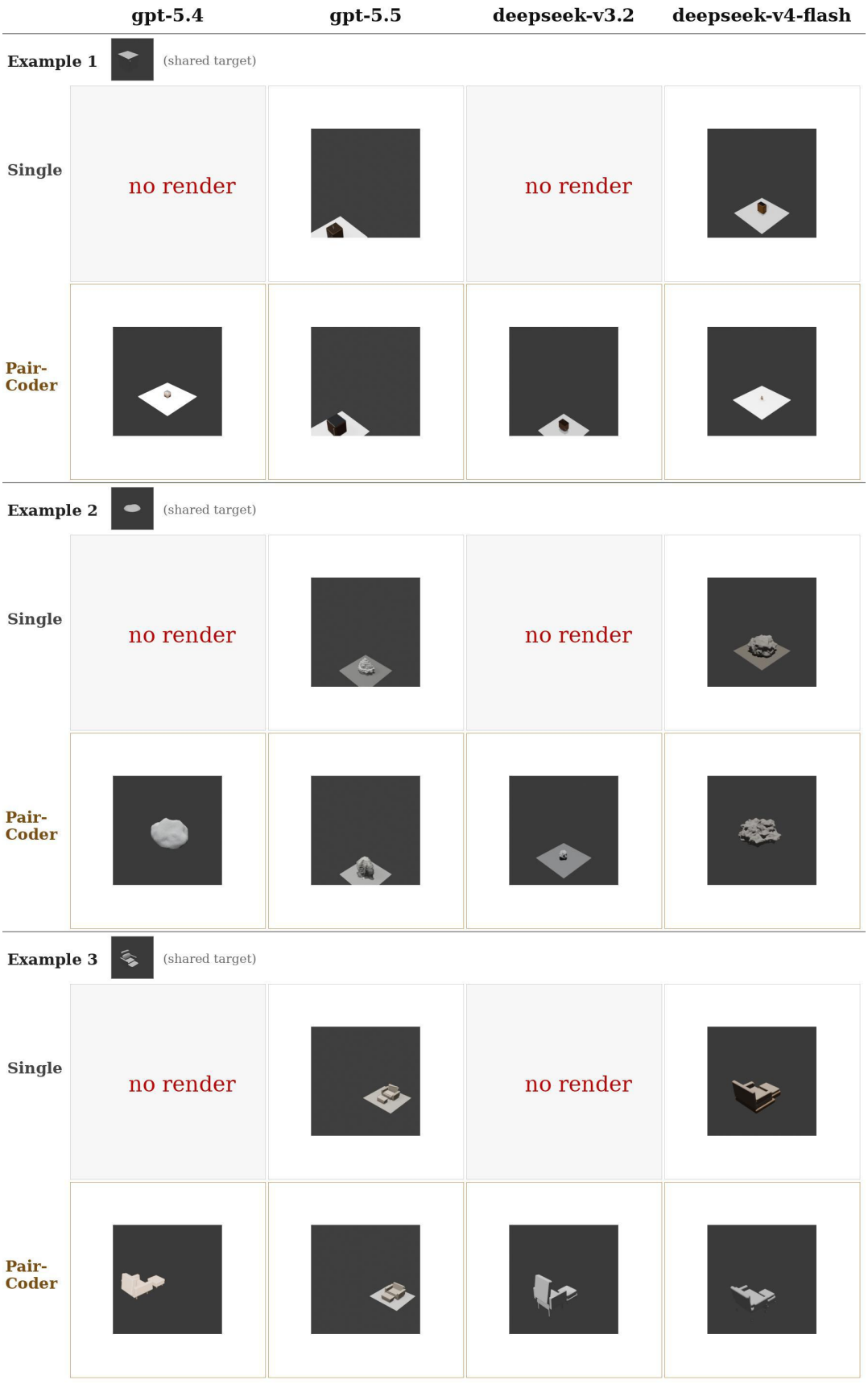


Figure 23: 3DCodeBench across four models and three shared tasks; gray marks baselines that fail to render.

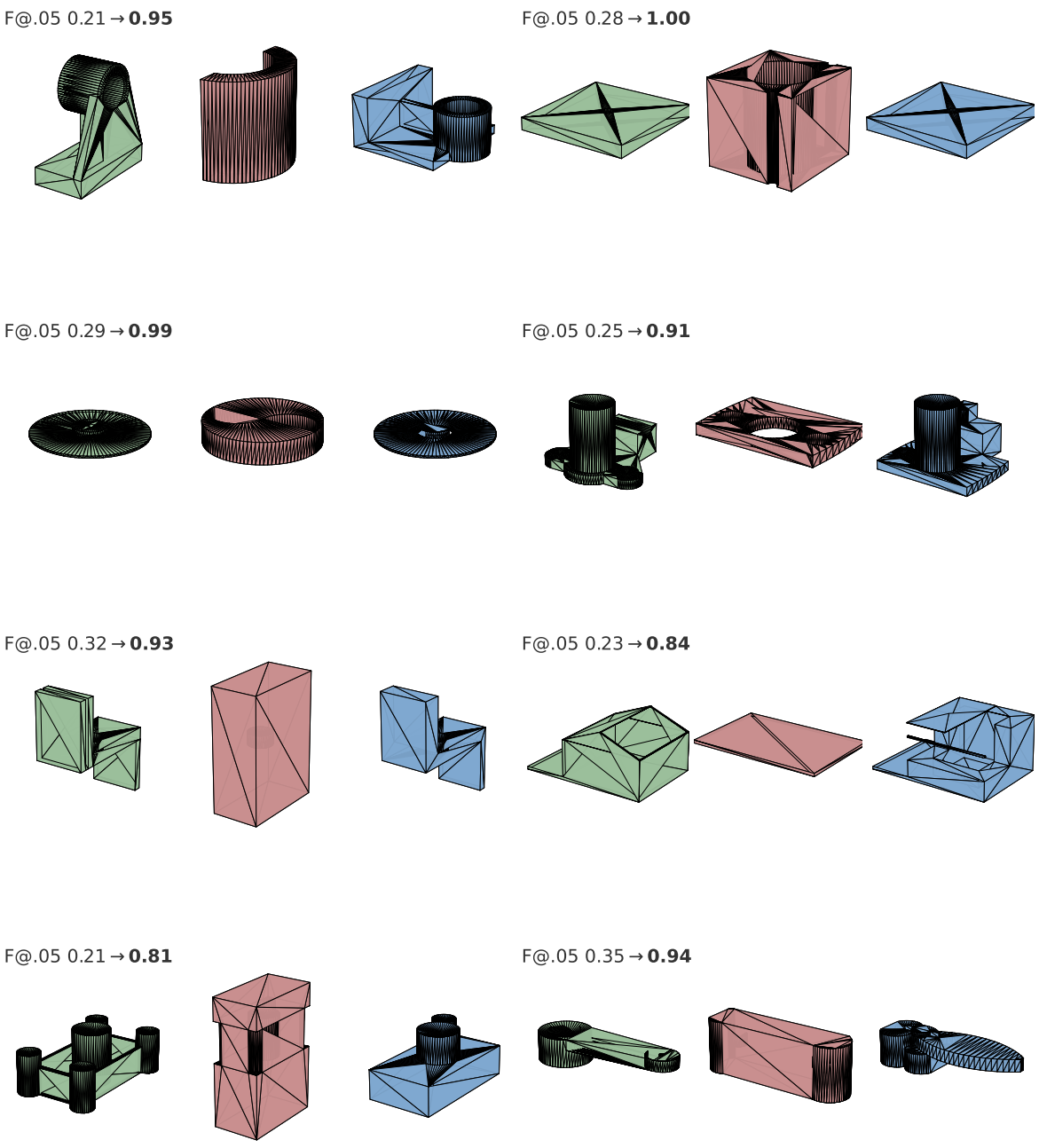


Figure 24: P3D-Bench **geometry-refinement** gallery (`gpt-5.4`): ground truth (green), single model (pink; valid but geometrically poor), PairCoder (blue); per-case F-score@0.05 annotated.