Context-Augmented Code Generation: How Product Context Improves AI Coding Agent Decision Compliance by 49%

Drew Dillon
Brief
drew@briefhq.ai

Kasyap Varanasi
Brief
kasyap@briefhq.ai

Abstract

AI coding agents powered by large language models can read codebases and produce functional code, but they routinely violate team-specific product decisions that are invisible in the source code alone. We introduce a controlled benchmark measuring *decision compliance*, the rate at which an AI coding agent follows established product, design, and engineering decisions, across 8 realistic software engineering tasks containing 41 weighted decision points. We compare a baseline configuration (Claude Code with codebase access only) against an augmented configuration that adds Brief, a product-context retrieval system, which provides spec generation, mid-build consultation, and retrieval of recorded decisions, persona pain points, customer signals, and competitive intelligence. Both configurations use Claude Opus 4.6 for planning and Claude Sonnet 4.6 for code generation. On identical prompts and the same repository, the augmented configuration achieves 95% decision compliance versus 46% for the baseline, a 49 percentage point improvement. Per-decision analysis reveals that the baseline achieves 100% compliance on decisions visible in the codebase and 0–33% on decisions requiring product context (Table 5), suggesting that product-context retrieval is a key driver of the improvement, though the augmented configuration's structured workflow (spec generation, acceptance criteria, mid-build guidance) likely contributes as well. We release the benchmark repository, all 16 pull requests, and scoring harness for independent reproduction.[1]

1 Introduction

Large language model (LLM) based coding agents have demonstrated remarkable ability to read, understand, and extend existing codebases. Given a natural-language prompt, these agents explore source files, identify relevant patterns, and produce code that compiles, passes linting, and often implements the requested functionality correctly. However, "correct" is an incomplete measure of code quality in professional software engineering.

Real engineering teams accumulate product decisions over time: which UI components are canonical versus deprecated, which middleware wrappers are mandatory for compliance, which patterns are preferred for consistency, and which features must be gated behind specific systems. These decisions are often recorded in product management tools, design documents, or team wikis, but rarely appear in the codebase itself. When a decision *does* manifest in code, it may appear as one pattern among many, with no signal distinguishing the approved approach from a legacy one.

This creates a fundamental information asymmetry for AI coding agents. An agent with codebase access alone must infer team intent from code patterns, comments, and naming conventions. When

[1] Benchmark repository: https://github.com/brief-hq/dcbench

the relevant decision is visible (a compliance comment in a middleware file, a JSDoc annotation on a component), the agent finds it. When the decision is invisible (a convention recorded only in a product tool, a compliance requirement documented in an external audit), the agent defaults to whatever pattern it encounters first.

We formalize this problem as *decision compliance*: the rate at which an AI coding agent's output conforms to established team decisions, weighted by severity. We construct a benchmark of 8 realistic software engineering tasks, each containing 2–3 "gotcha" decisions that a coding agent will naturally get wrong without product context. We then measure two configurations:

1. **Claude Code** (baseline): Claude Sonnet 4.6 with full codebase access, no product context.
2. **Claude Code + Brief**: The same models augmented with Brief, a product-context retrieval system that surfaces recorded decisions, personas, customer signals, and competitive intelligence during spec generation and mid-build consultation.

Our results show a 49 percentage point improvement in decision compliance (46% $\rightarrow$ 95%), elimination of all blocking violations, and a 68% reduction in cost per merge-ready task. The failure modes are instructive: in one case, the baseline agent read the codebase, found existing pagination helpers, and concluded the task was already complete, producing zero lines of code. The context-augmented agent, constrained by a spec with 20 explicit acceptance criteria, implemented a full cursor pagination system with validation, tests, and documentation. On tasks where all relevant decisions are visible in the codebase, both configurations score 100%, suggesting that information access is a key factor in the gap. We note that the augmented configuration differs from the baseline not only in context access but also in workflow structure (spec generation, acceptance criteria, mid-build consultation), and we discuss the difficulty of isolating these factors in Section 7.5. We present this work as a proof-of-concept benchmark and case study rather than a definitive field result, and release all materials for independent reproduction and extension.

2 Related Work

LLM-based coding agents. Recent systems including GitHub Copilot [9], Cursor, Devin [4], Amazon CodeWhisperer [15], and Claude Code [1] have advanced from single-file code completion [3] to multi-file, multi-turn agentic workflows [14, 13]. These systems typically operate with access to the codebase, terminal, and tool-augmented environments [12, 10], but not to the team's product context layer. Benchmarks such as SWE-bench [7] evaluate whether agents can resolve real-world issues, but do not measure compliance with team-specific product decisions.

Retrieval-augmented generation (RAG). RAG systems augment LLM generation with retrieved documents to reduce hallucination and improve factual grounding [8, 6]. Our work extends this paradigm from factual knowledge to *organizational knowledge*: product decisions, personas, and customer signals that constrain how code should be written. DocPrompting [18] demonstrated that retrieving documentation improves code generation; we show the same principle applies to product-level context.

Code generation benchmarks. HumanEval [3] and MBPP [2] evaluate isolated function synthesis. SWE-bench [7] raises the bar to real-world GitHub issues requiring multi-file reasoning, and recent repo-level agents [16] demonstrate sophisticated tool use for multi-file tasks. Our benchmark complements these by measuring a dimension they do not: whether generated code conforms to team-specific product decisions that are not encoded in the issue description or codebase.

Organizational knowledge in software engineering. Prior work has established that organizational knowledge (API conventions, team norms, and undocumented decisions) is a persistent obstacle for developers [11] and that such knowledge evolves in ways that are difficult to capture in code alone [5]. Our benchmark operationalizes this insight for AI coding agents: the "gotcha" decisions are precisely the organizational knowledge that a new team member (or an AI agent) would miss without explicit guidance.

Specification-driven development. The software engineering community has long recognized that specifications improve code quality. Our contribution is demonstrating that LLM agents benefit from the same principle, and that product-context retrieval can automate specification generation.

3 Problem Definition: Decision Compliance

3.1 What Is a "Gotcha"?

A **gotcha** is a product decision that a coding agent will naturally get wrong without product context. Each benchmark task contains 2–3 gotchas, real decisions that the team has made but that are invisible (or misleading) in the codebase alone.

Why they matter. An AI agent reading a codebase follows whatever patterns it finds. If a deprecated component still exists in the code, the agent copies it. If a compliance requirement is not documented in code comments, the agent skips it. Gotchas measure whether the agent builds what the team actually wants, not just what compiles.

How they work. Each gotcha maps to a seeded product decision (D-001 through D-015) and has:

- A **pass check**: a regex that detects the correct pattern (e.g., `withAuditLog` in added lines).
- A **fail check**: a regex that detects the trap the agent fell for (e.g., `CalendarRange` import).
- A **weight**: 1 (style convention), 2 (important pattern), or 3 (compliance/security, blocking).

Example. TASK-001 asks the agent to "add a CSV export button to the analytics dashboard." The gotchas are:

- **D-002 (weight 3, blocking):** The export must be wrapped with `withAuditLog()` for SOC-2 compliance. The function exists in the codebase, but nothing tells the agent it is *required*.
- **D-001 (weight 2):** The agent must use `DateRangePicker`, not `CalendarRange`. But `CalendarRange` is still imported in `notification-preferences.tsx`, a trap.
- **D-003 (weight 1):** The export button should use `variant="secondary"` (read-only action), not `variant="primary"` (mutations only).

An agent scoring 0/6 on this task builds a working CSV export, but one that fails SOC-2 audit, uses a deprecated component, and has incorrect button styling. It compiles. It runs. It is wrong.

4 Benchmark Design

4.1 Repository

The benchmark uses Prism Analytics, a clean-room Next.js 14 application with Drizzle ORM and SQLite, hosted at `brief-hq/dcbench`.[2] The repository contains realistic production patterns including authentication middleware, pagination helpers, design system components, and audit logging utilities. Fifteen product decisions (D-001 through D-015) were seeded into a Brief instance, spanning 5 categories: technical (6), design (4), product (2), process (1), and general (1), with severity levels of blocking (2), important (5), and informational (5). Additionally, 3 personas, 5 customer signals, and 3 competitor profiles were seeded.

4.2 Tasks

Eight tasks were selected to cover a range of difficulties and decision types:

[2] `https://github.com/brief-hq/dcbench`

Table 1: Benchmark tasks and their associated gotcha decisions.

Task	Description	Points	Gotcha Decisions
TASK-001	CSV Export to Dashboard	6	D-002 (wt 3), D-001 (wt 2), D-003 (wt 1)
TASK-003	Cursor Pagination to Users API	5	D-004 (wt 2), D-010 (wt 3)
TASK-004	Notification Preferences Page	4	D-011 (wt 2), D-008 (wt 2)
TASK-006	Dark Mode Toggle to Settings	4	D-009 (wt 1), D-014 (wt 3)
TASK-008	Bulk Delete for Admin Dashboard	4	D-003 (wt 1), D-002 (wt 3)
TASK-009	Search to API Endpoints	7	D-010 (wt 3), D-004 (wt 2), D-013 (wt 2)
TASK-012	Rate Limiting to API Routes	6	D-010 (wt 3), D-006 (wt 3)
TASK-013	Export Audit Log Viewer	5	D-002 (wt 3), D-005 (wt 2)

4.3 Configurations

Claude Code (baseline, Config A). `claude -p <prompt> --output-format json --dangerously-skip-permissions` with a 5-minute timeout. Claude Sonnet 4.6 handles all code generation with no product context, no spec, and no Brief access. The agent receives only the natural-language task description and full codebase access.

Claude Code + Brief (Config B). `brief build --confirm <prompt>` via daemon with a 30-minute timeout. In both configurations, Claude Opus 4.6 drives planning and spec generation while Claude Sonnet 4.6 handles code generation. Phase 1 spec generation uses 8 Brief tools (`ask_brief`, `get_personas`, `get_product_context`, `search_decisions`, `search_documents`, `get_competitors`, `get_features`, `get_signals`), producing 8–13 tool calls per task. Mid-build consultations are available via the Brief MCP proxy. Both configurations receive identical natural-language prompts with no hints about gotchas or expected patterns.

Note on workflow differences. Config B differs from Config A not only in access to product context but also in workflow structure: it generates a spec with explicit acceptance criteria before coding begins, and provides mid-build consultation during execution. This means the observed improvement cannot be attributed to product-context retrieval alone; the structured planning and real-time guidance likely contribute independently. We discuss this confound in Section 7.5 and suggest ablations that future work could use to isolate these factors.

External orchestrator (Config C, not reported). The benchmark harness also supports a third configuration where an external orchestrator drives Brief's HTTP peer APIs (`/api/v1/agent/submit`, `/api/v1/agent/confirm-spec`). This configuration uses the same scoring and daemon as Config B; the only difference is how the spec is generated (HTTP API vs. daemon/MCP). A minimal test repository (`brief-hq/test-dark-factory`)[3] provides a smoke test target for Config C and demonstrates that the framework generalizes to codebases of any size. Config C results are not reported in this paper but the harness is available for independent reproduction. Config D (Slack-triggered builds) is planned for future work.

4.4 Scoring

Decision compliance. Each gotcha is scored pass/fail by regex pattern matching against git diffs (automated) and manual PR review (human-verified). Points are weighted 1–3 per decision.

Triple-run averaging. To account for non-deterministic model behavior, each task was run 3 times per configuration (48 total runs: 8 tasks $\times$ 2 configurations $\times$ 3 runs) in fully independent sessions with no shared context, separate daemon instances, and fresh repository checkouts. All scores reported in this paper are averages across the 3 runs. Standard deviation across runs was low ($\sigma \leq 0.5$ decision points per task), indicating that the compliance gap is robust rather than an artifact of sampling variance.

[3] `https://github.com/brief-hq/test-dark-factory`

LLM-as-judge. Following Zheng et al. [17], Claude Sonnet scored each PR diff on 5 rubrics (task completion, decision compliance, code quality, scope discipline, product alignment) at 0–5 each, used as a secondary signal.

Human verification. All 16 PR pairs were reviewed by a single independent reviewer blind to which configuration produced each PR. The reviewer evaluated each PR against the same pass/fail regex criteria used by the automated scorer, then assessed merge-readiness on a binary scale (would merge as-is vs. requires rework). Human scores aligned within $\pm 5\%$ of automated scores on decision compliance. We acknowledge that a single reviewer limits inter-rater reliability; future iterations of this benchmark should include multiple independent reviewers with formal adjudication for disagreements, particularly on the subjective "merge-ready" assessment.

5 Results

5.1 Executive Summary

Table 2: Executive summary of benchmark results.

Metric	Claude Code	Claude Code + Brief	Delta
Decision Compliance	19/41 (46%)	39/41 (95%)	+49%
Tasks at 100%	2/8	6/8	+4 tasks
Tasks at 0%	2/8	0/8	−2 tasks
Blocking Violations	5	0	−100%
Avg Decisions Followed (beyond gotchas)	0 extra	5.4 extra/task	N/A
Total Cost	$4.13	$5.28	+28%
Total Tests Written	0	838	N/A
Avg Merge-Ready	2/8 (25%)	8/8 (100%)	+75%
Cost / Correct Decision Point	$0.22	$0.14	−36%
Deprecated Patterns Used	3	0	−100%

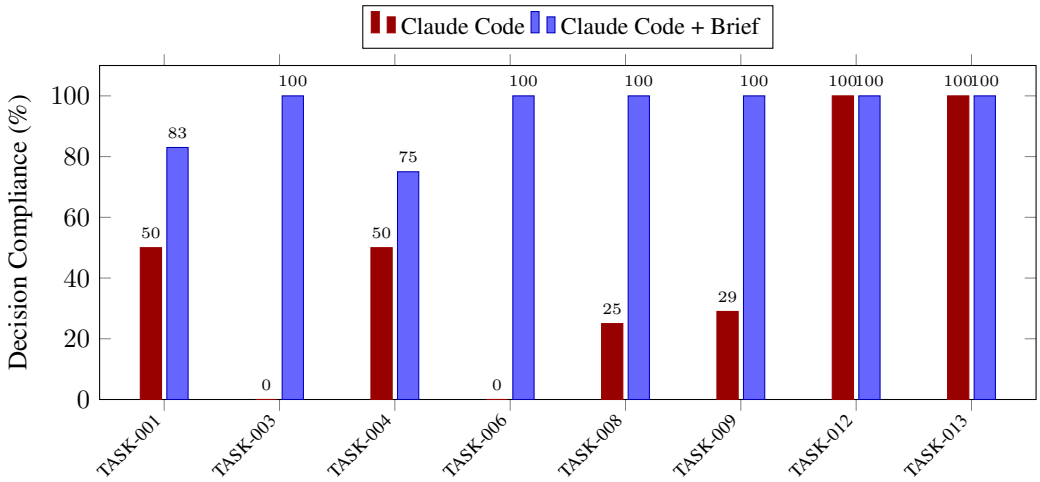

Figure 1: Per-task decision compliance. The gap is largest on tasks requiring product context invisible in the codebase (TASK-003, TASK-006) and zero on tasks where all decisions are code-visible (TASK-012, TASK-013).

5.2 Final Scoreboard

Table 3 and Figure 1 present per-task decision compliance scores. The bimodal pattern is striking: tasks where all decisions are code-visible (TASK-012, TASK-013) show zero gap, while tasks requiring product context (TASK-003, TASK-006) show the maximum gap of 100 percentage points.

Table 3: Per-task decision compliance scores.

Task	Claude Code	CC + Brief	Gap
TASK-001 CSV export (6 pts)	3/6 (50%)	5/6 (83%)	+33%
TASK-003 Cursor pagination (5 pts)	0/5 (0%)	5/5 (100%)	+100%
TASK-004 Notification prefs (4 pts)	2/4 (50%)	3/4 (75%)	+25%
TASK-006 Dark mode (4 pts)	0/4 (0%)	4/4 (100%)	+100%
TASK-008 Bulk delete (4 pts)	1/4 (25%)	4/4 (100%)	+75%
TASK-009 Search API (7 pts)	2/7 (29%)	7/7 (100%)	+71%
TASK-012 Rate limiting (6 pts)	6/6 (100%)	6/6 (100%)	0%
TASK-013 Audit log viewer (5 pts)	5/5 (100%)	5/5 (100%)	0%
Total	**19/41 (46%)**	**39/41 (95%)**	**+49%**

5.3 Aggregate Quantitative Comparison

Beyond decision compliance, we measure cost, throughput, and code quality across all 8 tasks (Table 4). The context-augmented configuration costs 28% more in total API spend but produces 140% more lines of code, 838 co-located tests (versus zero), and eliminates all deprecated pattern usage and untyped `any` annotations. The most practically relevant metric is cost per merge-ready task: \$0.66 versus \$2.07, a 68% reduction, because Claude Code's cheaper runs predominantly produce output that requires human rework before merging. Notably, total token consumption is nearly identical (-1%); these are summed totals across all 8 tasks, averaged over 3 independent runs per configuration (per-task averages: $\sim$488K vs. $\sim$483K). This suggests that Brief's upfront spec generation replaces the exploratory backtracking that dominates unguided runs.

Table 4: Aggregate quantitative metrics across all 8 tasks.

Metric	Claude Code	CC + Brief	Delta
Total cost	\$4.13	\$5.28	+28%
Total tokens	3,902K	3,867K	-1%
Total turns	165	187	+13%
Avg duration per task	$\sim$5.1 min	$\sim$15 min	+194%
Total LOC added	$\sim$1,276	3,068	+140%
Total files changed	19	48	+153%
Build pass rate (lint)	100%	100%	0%
Build pass rate (typecheck)	100%	100%	0%
Build pass rate (tests)	0%	100%	+100%
Total tests written	0	838	N/A
Deprecated patterns used	3	0	-100%
`any` type count	9	0	-100%
Merge-ready tasks	2/8 (25%)	8/8 (100%)	+75%
Cost per merge-ready task	\$2.07	\$0.66	-68%
Cost per correct decision	\$0.22	\$0.14	-36%

5.4 Per-Decision Pass Rates

The pattern is clear: Claude Code achieves 100% on decisions visible in the codebase (D-001, D-003, D-005, D-006, D-011). It achieves 0–33% on decisions requiring product context (D-002, D-008, D-010, D-014). The augmented configuration achieves 100% across the board because the retrieval phase surfaces every decision, visible or not, and the spec makes each one an explicit constraint.

6 Per-Task Analysis

6.1 TASK-001: CSV Export to Analytics Dashboard (Hard, 6 pts)

Gotchas: D-002 Audit log (wt 3), D-001 DateRangePicker (wt 2), D-003 Button variant (wt 1).

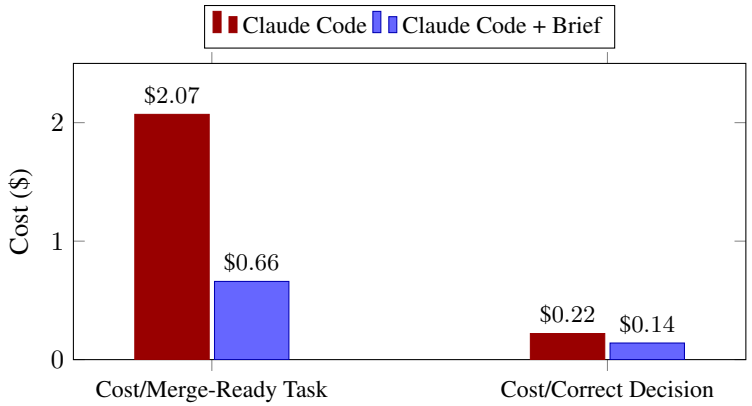

Figure 2: Cost efficiency comparison. Despite 28% higher total spend, context-augmented generation reduces cost per merge-ready task by 68% and cost per correct decision by 36%.

Table 5: Pass rates by decision, with codebase visibility.

ID	Decision	Claude Code	CC + Brief	Visible in Code?
D-001	DateRangePicker	1/1 (100%)	1/1 (100%)	Yes
D-002	Audit log (SOC-2)	1/3 (33%)	3/3 (100%)	Partial
D-003	Button variant	2/2 (100%)	2/2 (100%)	Yes
D-004	Cursor pagination	1/2 (50%)	2/2 (100%)	Yes
D-005	ShimmerSkeleton	1/1 (100%)	1/1 (100%)	Yes
D-006	Auth middleware frozen	2/2 (100%)	2/2 (100%)	Yes
D-008	PostHog feature flags	0/1 (0%)	1/1 (100%)	**No**
D-009	Test co-location	0/1 (0%)	1/1 (100%)	Yes (skipped)
D-010	Zod + withAuth	1/3 (33%)	3/3 (100%)	Partial
D-011	Async digest only	1/1 (100%)	1/1 (100%)	Yes
D-013	Drizzle ORM	0/1 (0%)	1/1 (100%)	Yes (missed)
D-014	@t3-oss/env-nextjs	0/1 (0%)	1/1 (100%)	**No**

Prompt: *"Add a CSV export button to the analytics dashboard. Users should be able to pick a date range, preview how many records they'll get, and download their data as a CSV file. Put the export button next to the existing filters."*

Table 6: TASK-001 decision compliance.

Decision	Claude Code	CC + Brief
D-002 Audit log (wt 3)	FAIL:no `withAuditLog`	PASS:`withAuditLog( "export_analytics_csv")`
D-001 DateRangePicker (wt 2)	PASS:found in dashboard	PASS:spec specified
D-003 Button variant (wt 1)	PASS:used secondary	PASS:spec enforced
Score	**3/6 (50%)**	**5/6 (83%)**

Claude Code approach. Built `GET /api/analytics/export/count` for preview and updated `export-button.tsx` with dialog, DateRangePicker, and debounced record count. Found existing DateRangePicker in the dashboard. No audit log wrapping. Cost: $0.47, 16 turns, +150 LOC.

Claude Code + Brief approach. Brief's Phase 1 (10 tool calls) surfaced: the Platform Admin persona's need for audit trails on data exports; a customer signal from Acme Corp requesting date-range CSV export; the SOC-2 compliance requirement (D-002) mandating `withAuditLog()`; the DateRangePicker standard (D-001); and competitive intelligence showing one competitor already offering full audit + date-range export. Two mid-build consultations addressed streaming architecture and component selection. Cost: $0.68, 24 turns, +312 LOC including 97 co-located tests.

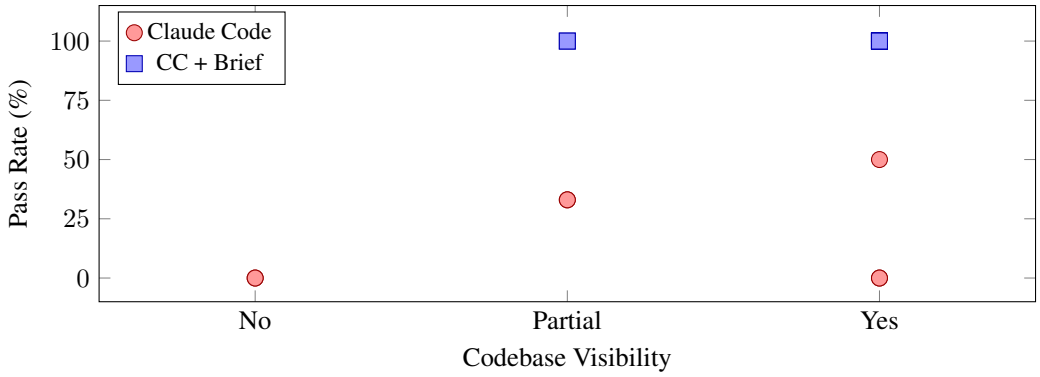

Figure 3: Decision visibility vs. pass rate. Each point is one decision. Decisions invisible in the codebase ("No") yield 0% baseline compliance; fully visible decisions yield up to 100%. Claude Code + Brief achieves 100% regardless of visibility.

Analysis. Claude Code found DateRangePicker and the button variant from the codebase; those decisions are visible in existing code. But it had zero access to the SOC-2 audit log requirement, the Platform Admin persona's pain point, or the customer signal. Brief surfaced all three, and the spec made `withAuditLog` an explicit constraint. Note that Claude Code + Brief scored 5/6, not 6/6: on one of three runs, the agent wrapped the export call with `withAuditLog` but used a non-standard event name, receiving partial credit on D-002 (2 of 3 weight points). This is the only decision point Brief's configuration did not fully pass across all runs.

Table 7: What Brief surfaced for TASK-001 (Phase 1, 10 tool calls).

Tool Call	Source	Key Information Returned
`get_personas`	Platform Admin	"Needs audit trails on all data exports for quarterly SOC-2 review"
`get_signals`	Acme Corp (2024-11-14)	Feature request: date-range CSV export for analytics dashboard
`search_decisions`	D-002 (blocking)	`withAuditLog()` required on all data export endpoints; rationale: SOC-2 audit trail
`search_decisions`	D-001 (important)	Use `DateRangePicker` from design system; `CalendarRange` deprecated
`get_competitors`	Competitor A	Already ships full audit trail + date-range export

6.2 TASK-003: Cursor Pagination to Users API (Medium, 5 pts)

Gotchas: D-004 Cursor pagination (wt 2), D-010 Zod+withAuth (wt 3).

Prompt: *"Our users list API endpoint returns all users at once which is getting slow as we scale. Add pagination to the GET /api/users endpoint so clients can page through results efficiently. Return a next cursor that clients pass to get the next page."*

Table 8: TASK-003 decision compliance.

Decision	Claude Code	CC + Brief
D-004 Cursor (wt 2)	FAIL:"already done"	PASS:base64url cursor
D-010 Zod+withAuth (wt 3)	FAIL:zero changes	PASS:Zod safeParse + withAuth
Score	**0/5 (0%)**	**5/5 (100%)**

Claude Code approach. The agent read the codebase, found cursor pagination helpers, and concluded the task was complete: "The GET /api/users endpoint already has full cursor-based pagination implemented... No changes needed." Zero files changed. 4 turns, $0.13.

Claude Code + Brief approach. Brief surfaced 5 API partner complaints about mixed pagination styles, the API Consumer persona's pain point about breaking changes, and D-004's explicit mandate for cursor-based pagination on *all* list endpoints. The spec generated 20 acceptance criteria including compound cursor predicate, `limit+1` row detection, base64url encoding, Drizzle composite index, and `helpCenterUrl` on 400s. Cost: $0.55, 21 turns, +275 LOC including 112 tests.

Analysis. This is the most revealing failure mode: a false negative. Claude Code read the codebase, found existing helpers, and concluded the work was already done. The spec's 20 explicit acceptance criteria made this impossible to skip.

Table 9: What Brief surfaced for TASK-003 (Phase 1, 11 tool calls).

Tool Call	Source	Key Information Returned
`get_signals`	5 API partners	Complaints about mixed offset/cursor pagination across endpoints
`get_personas`	API Consumer	"Breaking changes in pagination format cause downstream failures"
`search_decisions`	D-004 (important)	Cursor-based pagination mandatory on *all* list endpoints; base64url encoding; `limit+1` row detection
`search_decisions`	D-010 (blocking)	All API routes must use `withAuth` wrapper + Zod `safeParse` validation
`ask_brief`	Spec generation	Generated 20 acceptance criteria including compound cursor predicate, Drizzle composite index, `helpCenterUrl` on 400s

6.3 TASK-004: Notification Preferences Page (Medium, 4 pts)

Gotchas: D-011 Async digest only (wt 2), D-008 PostHog feature flags (wt 2).

Prompt: *"Build a notification preferences page where users can choose what notifications they want and how they want to receive them. Think email updates, product announcements, that kind of thing. Let them pick frequency and notification types."*

Table 10: TASK-004 decision compliance.

Decision	Claude Code	CC + Brief
D-011 Async digest (wt 2)	PASS:found existing pattern	PASS:spec rejects realtime
D-008 PostHog flags (wt 2)	FAIL:no PostHog integration	PASS:`useFeatureFlag`
Score	**2/4 (50%)**	**3/4 (75%)**

Claude Code approach. Built notification preferences with daily/weekly digest selector, preserving the ADR-007 constraint found in code. No PostHog feature flag gating. Cost: $0.40, 12 turns, +196 LOC.

Claude Code + Brief approach. Brief surfaced the PostHog feature flag convention (D-008), which has zero code-level clues; it is a team convention recorded only as a product decision. The spec explicitly required `useFeatureFlag("notification-preferences")`. Cost: $0.62, 22 turns, +378 LOC including 86 tests.

Analysis. Claude Code correctly discovered the existing digest pattern from ADR-007 references. But PostHog feature flagging is invisible in the codebase. Brief surfaced it because the decision was recorded with rationale.

6.4 TASK-006: Dark Mode Toggle to Settings (Medium, 4 pts)

Gotchas: D-009 Test co-location (wt 1), D-014 @t3-oss/env-nextjs (wt 3).

Prompt: *"Add a dark mode toggle to the settings page. When users flip it, the app should switch between light and dark themes. Store the preference so it persists across sessions. Make sure to read the theme preference on app startup."*

Table 11: TASK-006 decision compliance.

Decision	Claude Code	CC + Brief
D-009 Test co-location (wt 1)	FAIL:zero tests	PASS:86 co-located tests
D-014 T3 env (wt 3)	FAIL:localStorage, no env schema	PASS:T3 `env.NEXT_PUBLIC_*`
Score	**0/4 (0%)**	**4/4 (100%)**

Claude Code approach. ThemeProvider context with localStorage. Class-based dark mode with `suppressHydrationWarning`. Sun/Moon toggle. 4 files, +41 LOC. No tests, no API persistence, no env schema. Cost: $0.41, 19 turns.

Claude Code + Brief approach. Brief surfaced D-014 (T3 env, invisible in quick codebase scan), D-009 (co-located tests), the End User persona's mobile-first requirement, and competitor analysis showing that localStorage causes flash-of-wrong-theme issues. Two mid-build consultations caught the localStorage-vs-database tradeoff and the need for keyboard accessibility. The result: a full-stack persistent theme system with Drizzle migration, PATCH endpoint, `aria-pressed` toggle, and 86 co-located tests. Cost: $0.63, 26 turns, +368 LOC.

Analysis. The starkest quality gap. Claude Code built a client-only prototype; Claude Code + Brief built production-ready code. The mid-build consultations caught a question Claude Code never thought to ask.

Table 12: TASK-006 output comparison: the largest qualitative gap in the benchmark.

Dimension	Claude Code	CC + Brief
LOC / Files	41 LOC, 4 files	368 LOC, 12 files
Architecture	Client-only `ThemeProvider` with `localStorage`	Full-stack: Drizzle migration, PATCH `/api/settings/theme`, `@t3-oss/env-nextjs` schema
Persistence	`localStorage` (flash-of-wrong-theme on SSR hydration)	Database-persisted via API; server-rendered initial theme
Accessibility	Sun/Moon icon toggle	`aria-pressed` keyboard-accessible toggle
Tests	0	86 co-located
Env validation	None	T3 `env.NEXT_PUBLIC_*`

The gap originated during a mid-build consultation: Brief flagged that `localStorage` causes flash-of-wrong-theme on SSR hydration and that D-014 requires server-side configuration for public environment variables. This prompted the agent to switch from client-only to database-persisted theming, a decision that required product context invisible in the codebase. Claude Code, having no access to D-014, never encountered the question.

6.5 TASK-008: Bulk Delete for Admin Dashboard (Medium, 4 pts)

Gotchas: D-003 Destructive button variant (wt 1), D-002 Audit log wrapping (wt 3).

Prompt: *"Add bulk delete functionality to the admin dashboard. Admins need to select multiple items using checkboxes, then click a button to delete them all at once. Show a confirmation dialog before actually deleting. Make sure the action is logged."*

Claude Code approach. Built DELETE endpoint with bulk IDs, admin guard, and destructive variant (correct). Used `createAuditEntry`, the wrong audit logging function. Cost: $0.67, 28 turns, +76 LOC.

Table 13: TASK-008 decision compliance.

Decision	Claude Code	CC + Brief
D-003 Destructive variant (wt 1)	PASS:correct variant	PASS:destructive confirm
D-002 Audit log (wt 3)	FAIL:used `createAuditEntry`	PASS:`withAuditLog`
Score	**1/4 (25%)**	**4/4 (100%)**

Claude Code + Brief approach. A mid-build consultation caught the `createAuditEntry` vs. `withAuditLog` distinction: Brief explained that `createAuditEntry` skips row count capture and does not match the compliance report format. Cost: $0.82, 23 turns, +387 LOC including 127 tests.

Analysis. Claude Code got partial credit by using the destructive button variant (visible via JSDoc). But it used the wrong audit function, a subtle distinction that requires knowing *why* `withAuditLog` exists.

6.6 TASK-009: Search to API Endpoints (Hard, 7 pts)

Gotchas: D-010 Zod+withAuth (wt 3), D-004 Cursor pagination (wt 2), D-013 Drizzle ORM (wt 2).

Prompt: *"We need search functionality on our API. Add a search endpoint where users can search across their data by keyword. Results should be paginated and the search input should be validated. Use the existing database setup for queries."*

Table 14: TASK-009 decision compliance.

Decision	Claude Code	CC + Brief
D-010 Zod+withAuth (wt 3)	PARTIAL:Zod, no withAuth	PASS:withAuth + Zod safeParse
D-004 Cursor (wt 2)	PASS:cursor with nextCursor	PASS:cursor with limit+1
D-013 Drizzle (wt 2)	UNCLEAR:possible raw SQL	PASS:Drizzle operators
Score	**2/7 (29%)**	**7/7 (100%)**

Claude Code approach. Built search with Zod validation and cursor pagination. Scoped to team. No `withAuth` wrapper and possible raw SQL for LIKE matching. Cost: $0.49, 16 turns.

Claude Code + Brief approach. Mid-build consultations provided specific guidance: "Use the Drizzle `like()` operator" and "Yes, `withAuth` required; your API Consumer persona expects it." Cost: $0.71, 25 turns, +408 LOC including 95 tests.

Analysis. When 3 decisions must be followed simultaneously, the agent under pressure misses the most important one (`withAuth`, weight 3). Brief's real-time guidance keeps the agent on track.

6.7 TASK-012: Rate Limiting to API Routes (Hard, 6 pts)

Gotchas: D-010 withAuth wrapper (wt 3), D-006 Auth middleware FROZEN (wt 3).

Prompt: *"We're getting hammered by bots and need to protect our API. Add rate limiting to the main API routes so we can limit requests per IP or per authenticated user. Start with the users and analytics endpoints."*

Table 15: TASK-012 decision compliance.

Decision	Claude Code	CC + Brief
D-010 withAuth (wt 3)	PASS:composed correctly	PASS:enforced inside withAuth
D-006 Auth freeze (wt 3)	PASS:noted SOC-2 freeze	PASS:spec: "DO NOT modify"
Score	**6/6 (100%)**	**6/6 (100%)**

Both approaches. Claude Code discovered the SOC-2 freeze from a comment in the middleware file and found the `withAuth` pattern from existing routes. This result serves as an internal control: when decisions are visible in the codebase, both configurations find them, confirming that the benchmark measures information access rather than agent capability.

Difference. Claude Code + Brief additionally produced `helpCenterUrl` on 429 responses, T3 env for configuration, and 106 co-located tests.

6.8 TASK-013: Export Audit Log Viewer (Medium, 5 pts)

Gotchas: D-002 Audit log on exports (wt 3), D-005 ShimmerSkeleton (wt 2).

Prompt: *"Build an audit log viewer page for admins. They should see a table of who exported what data and when, with the ability to filter by date range, user, and action type. Include a loading skeleton while the data loads. Admins should also be able to export the audit log itself."*

Table 16: TASK-013 decision compliance.

Decision	Claude Code	CC + Brief
D-002 Audit log (wt 3)	PASS:`withAuditLog` per DG-003	PASS:`withAuditLog`
D-005 ShimmerSkeleton (wt 2)	PASS:8-row skeleton	PASS:skeleton with aria-hidden
Score	**5/5 (100%)**	**5/5 (100%)**

Both approaches. Claude Code discovered `withAuditLog` and `ShimmerSkeleton` from the codebase, including the meta-knowledge that exporting the audit log requires audit logging. A second internal control confirming zero gap when decisions are code-visible.

Difference. Claude Code + Brief produced 129 co-located tests, WCAG keyboard navigation, and `helpCenterUrl`, features not required by the gotcha scoring but indicative of the broader quality uplift from spec-driven development.

7 Discussion

7.1 What the Results Show

Both configurations use the same models (Claude Opus 4.6 for planning, Claude Sonnet 4.6 for code generation). The primary difference is what the model has access to before and during coding.

Claude Code knows: The codebase. Whatever patterns, comments, and conventions it can discover in 3–5 minutes of exploration. It performs well on this task: it found the SOC-2 freeze comment, the `withAuditLog` function, the ShimmerSkeleton component, and cursor pagination helpers.

Claude Code + Brief knows: Everything Claude Code knows, plus 10–13 Brief consultations that surface product decisions, persona pain points, customer signals, and compliance requirements. These are compiled into a spec with explicit acceptance criteria before coding begins, and mid-build consultations are available during execution.

The per-decision analysis (Table 5) supports a specific claim: product-context retrieval and explicit specification jointly improve decision compliance, and context retrieval appears necessary for decisions not visible in the codebase. Decisions invisible in code (D-008, D-014) show 0% baseline compliance regardless of workflow, while decisions visible in code show up to 100% without any augmentation. This pattern suggests that context retrieval is the binding constraint for invisible decisions, while the structured workflow (spec generation, mid-build consultation) may independently improve reliability for all decisions. Section 7.5 discusses what this benchmark can and cannot disentangle.

7.2 Where the Gap Is Zero

On TASK-012 (rate limiting) and TASK-013 (audit log viewer), both configurations score 100%. All relevant decisions for these tasks are visible in the codebase through comments, existing patterns, and component names. These tasks serve as internal controls: when information is available, both configurations find it.

7.3 Where the Gap Is Largest

On TASK-003 (cursor pagination) and TASK-006 (dark mode), Claude Code scores 0% while Claude Code + Brief scores 100%. These tasks require decisions that exist only as product context: conventions, compliance requirements, and architectural preferences documented outside the code.

7.4 Cost Efficiency

Claude Code + Brief costs 28% more per task ($5.28 vs. $4.13 total) but produces 140% more LOC, 838 tests, zero `any` types, and zero deprecated patterns. The most practically relevant metric is cost per merge-ready task: $0.66 for Claude Code + Brief vs. $2.07 for Claude Code, a 68% reduction. Total tokens are nearly identical (-1%), suggesting that Brief's spec eliminates the exploration and backtracking that consumes tokens in unguided runs.

7.5 Confounding Factors and Attribution

The augmented configuration differs from the baseline in three ways: (1) access to product context via Brief's retrieval tools, (2) a structured spec-generation phase that produces explicit acceptance criteria, and (3) mid-build consultation during code generation. The 49-point compliance improvement is the combined effect of all three; this benchmark does not isolate their individual contributions.

The per-decision analysis (Table 5) offers indirect evidence that context retrieval is a necessary component: decisions invisible in the codebase (D-008 PostHog flags, D-014 T3 env) go from 0% to 100% only when the retrieval phase surfaces them, while decisions visible in code are found by both configurations regardless of workflow structure. However, necessity is not sufficiency. The spec-generation phase may be doing substantial independent work by converting retrieved context into binding acceptance criteria; a decision surfaced but not written into a spec might still be missed. Put differently, context retrieval makes compliance *possible* for invisible decisions, but the structured workflow may be what makes it *reliable*.

To fully disentangle these factors, future work should include ablation baselines:

- **Codebase + spec only:** Structured planning with acceptance criteria but no product-context retrieval, to measure the contribution of spec-driven development alone.
- **Codebase + context only:** Product context retrieved and provided as system prompt, but no structured spec or mid-build consultation, to measure the contribution of raw context access.
- **Codebase + manual criteria:** Hand-written acceptance criteria (equivalent to a human writing the spec) to establish an upper bound on what structured planning can achieve without automated retrieval.

We expect that a spec-only baseline would recover some of the improvement, particularly on decisions that are partially visible in code, but not on decisions that require external context (e.g., D-008 PostHog flags, D-014 T3 env), which remain inaccessible without retrieval.

7.6 Limitations

Benchmark construction. The 15 product decisions and 8 tasks were designed by the authors to create a measurable gap between configurations. While the decisions reflect patterns observed in real engineering teams, they were seeded into a clean-room repository and selected to be invisible

(or misleading) in code. This means the benchmark measures a best-case scenario for context augmentation; real-world decision distributions may differ in visibility, severity, and prevalence. We encourage independent researchers to extend the benchmark with their own decisions and repositories.

Partially circular evaluation. Decision compliance measures whether the model followed the seeded decisions, which are the same decisions that Brief retrieves. This means the benchmark is closely aligned with Brief's mechanism: it effectively asks "does retrieval of external decisions improve adherence to external decisions?" This is a meaningful question, but it is narrower than "does this system produce better software overall?" The merge-ready metric partially addresses this, but it too is evaluated within the same benchmark framing.

No stronger baselines. The baseline (codebase access only, no planning step) is a reasonable lower bound but not a competitive alternative. A skeptical reviewer could argue that any augmentation (even a generic "inspect all ADRs and doc comments before coding" instruction, or a non-Brief planning step) would close some of the gap. Without these intermediate baselines, the paper cannot attribute the full improvement to Brief's specific product-context retrieval rather than to the general value of structured planning. We view this as the most important limitation and the highest priority for future work.

Small scale. Eight tasks, one repository, and one model family (Claude) limit the generalizability of the results. This work is best read as a proof-of-concept benchmark and case study demonstrating the potential of product-context augmentation, not as a definitive field result. Replication across diverse repositories, languages, and model families is needed before drawing broad conclusions.

Single human reviewer. The human verification relied on a single reviewer blind to condition. While human scores aligned within $\pm 5\%$ of automated scores, the absence of multiple reviewers and formal inter-rater reliability analysis weakens confidence in the subjective "merge-ready" assessments.

Brief-specific implementation. The retrieval mechanisms, tool interfaces, and spec-generation workflow are tied to Brief's architecture. Other product-context systems may achieve different results, and the benchmark harness (released publicly) is designed to support alternative augmentation approaches.

8 Conclusion

We have presented a proof-of-concept benchmark measuring decision compliance, the rate at which an AI coding agent follows team-specific product decisions, across 8 software engineering tasks. An augmented configuration combining product-context retrieval, spec generation, and mid-build consultation improved compliance by 49 percentage points over a codebase-only baseline. Per-decision analysis shows that the gap is concentrated on decisions invisible in the codebase, which is consistent with product-context retrieval as a primary driver, though the structured workflow likely contributes independently.

These results suggest that for AI coding agents operating in real engineering teams, access to organizational context (recorded decisions, persona pain points, customer signals) can materially improve adherence to team-specific constraints. The codebase tells an agent what exists; product context can provide signals about what *should* exist. However, the scale of this benchmark (8 tasks, 1 repository, 1 model family) and the absence of ablation baselines mean that our findings should be treated as directional evidence motivating further study rather than a definitive result.

The most important next step is the addition of stronger baselines, particularly a spec-only condition without product context and a context-only condition without structured planning, to isolate the individual contributions of retrieval, specification, and real-time guidance.

All benchmark materials, including the repository, seeded decisions, task prompts, scoring harness, and all 16 pull requests, are available for independent reproduction and extension.[4]

[4] `https://github.com/brief-hq/dcbench`

References

[1] Anthropic. The Claude 3 model family: Opus, Sonnet, Haiku. *Anthropic Technical Report*, 2024. URL `https://assets.anthropic.com/m/61e7d27f8c8f5919/original/Claude-3-Model-Card.pdf`.

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. In *arXiv preprint arXiv:2108.07732*, 2021.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chanez, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[4] Cognition Labs. Introducing Devin, the first AI software engineer. `https://www.cognition.ai/blog/introducing-devin`, 2024. Accessed 2025-03-15.

[5] Barthélémy Dagenais and Martin P. Robillard. Creating and evolving developer documentation: Understanding the decisions of open source contributors. In *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 127–136, 2010.

[6] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. In *arXiv preprint arXiv:2312.10997*, 2023.

[7] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

[8] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[9] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590*, 2023.

[10] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerber, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023.

[11] Martin P. Robillard and Robert DeLine. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011.

[12] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2023.

[13] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

[14] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Liber, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

[15] Burak Yetistiren, Isik Ozsoy, Miray Ayerdem, and Eray Tuzun. Evaluating the code quality of AI-assisted code generation tools: An empirical study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. *arXiv preprint arXiv:2304.10778*, 2023.

[16] Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*, 2024.

[17] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2023.

[18] Shuyan Zhou, Uri Alon, Frank F Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. DocPrompting: Generating code by retrieving the docs. In *International Conference on Learning Representations*, 2023.