Do LLM-Generated Skills Make Better AI Data Scientists? A Component Ablation Across Data-Science Workflows

Wei-Jung Huang
Independent Researcher
United States

Abstract

Product data scientists often ask LLM-based agents to help with recurring execution tasks such as cleaning data, writing SQL, choosing statistical tests, and formatting results. Reusable skill files are meant to avoid prompting from scratch by packaging guidance for a task family. Expert-written skills can encode high-quality guidance, but writing and maintaining them across many data-science task families creates a manual bottleneck. We ask whether LLM-generated skills offer a useful low-curation alternative: do they improve performance over the task prompt alone?

We test this question across four lifecycle stages: data preparation, data extraction, statistical analysis, and reporting, using one generated skill per stage. We find no reliable improvement from full generated skills over No-Skill prompting. We then ask whether any part of the skill is useful by ablating different skill components. The main ablation covers 56 tasks, nine model configurations, and three providers, yielding 7,560 runs.

Compared with prompting using the task alone, neither the full generated skill nor any ablated skill variant significantly improves performance; all p-values are at least 0.396, and the total spread across variants is only 1.2 pp. A supplemental token-matched control adds 1,512 runs and finds that Full skills perform similarly to task-irrelevant skill-formatted content. The results caution against using one LLM-generated skill per data-science workflow as a default single-shot prompting strategy.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**.

Keywords

large language models, data-science agents, agent skills, prompt engineering, ablation study, data-science automation

1 Introduction

Product data scientists often ask LLM-based agents to help with recurring execution tasks: cleaning data, writing SQL, choosing statistical tests, computing effect sizes, and formatting reports. We refer to these systems as data-science agents, following recent benchmarks for data-science automation [5, 15]. For these systems, model choice is only part of the design problem. The other part is how to supply domain knowledge.

Recent agent platforms and benchmarks use reusable skill files, such as `SKILL.md`, to package task instructions, examples, and reference notes [1, 8]. For data-science agents, this makes a tempting workflow: write one skill for recurring task families such as data preparation, SQL generation, statistical analysis, and reporting, then prepend it to future tasks.

Expert-written skills can encode high-quality guidance, but they require practitioners to decide what to include, write examples and reference notes, and keep the content current as tools and task conventions change. This manual process scales poorly when teams need guidance for many data-science task families. LLM-generated skills offer a lower-curation alternative: generate task-family guidance once and reuse it across related tasks. The question is whether this low-curation version works in practice.

Existing evidence suggests that this is not guaranteed. SkillsBench [8] (86 tasks, 7,308 trajectories) found that human-curated skills improve performance substantially (+16.2 pp), while LLM-generated skills provide no aggregate benefit. However, SkillsBench does not specifically evaluate reusable skills for data-science workflows.

That null result also raises a component-level question: when an LLM-generated skill fails to improve performance, is every component unhelpful, or are useful sections canceled out by harmful ones? For builders of data-science agents, it is useful to ask both whether LLM-generated skills help on data-science workflows and which failure patterns appear when they do not.

We study these questions across four execution-facing data-science lifecycle stages. Each stage is represented by a task family covering a distinct workflow: **data preparation** (null handling, deduplication, type coercion), **data extraction** (SQL query formulation), **statistical analysis** (hypothesis testing, effect sizes), and **reporting** (structured JSON reporting). These workflows represent common execution tasks for data-science agents, and their outputs can be checked by deterministic verifiers.

Our evaluation has two parts. First, we compare task-only prompting with one full LLM-generated skill per stage. Second, we ablate the same skills to test whether any section helps on its own. We use a simplified single-shot setup so that the comparison focuses on the skill content itself. Section 3 describes this design choice.

We make three contributions:

- We evaluate low-curation LLM-generated skills for data-science agents across four lifecycle stages, 56 tasks, nine model configurations, and 7,560 main-ablation runs.
- We ablate procedures, examples, and reference notes, then add a 1,512-run token-matched control to separate data-science content from prompt-length overhead.
- We observe no reliable improvement from either full skills or ablated variants, and analyze failure patterns that differ across task regimes.

2 Related Work

Our study sits at the intersection of prompt-level knowledge injection, reusable skills, and LLM benchmarks for data-science tasks.

The `SKILL.md` specification is one recent form of reusable domain instruction for LLM prompts [1]. Related ways of supplying task context include retrieval-augmented generation [6] and few-shot prompting [2]. Prior work also shows that prompt components can affect model behavior in non-obvious ways: Min et al. [10] found that label correctness in demonstrations matters less than format and input distribution, and Lu et al. [9] showed that example ordering can significantly affect performance. We move from individual prompt elements to reusable skill files and ask whether their content helps when injected into data-science tasks.

Reusable skills have also been evaluated directly. SkillsBench [8] found that focused human-curated skills improve performance, while LLM-generated skills provide no aggregate benefit. SkillLearnBench [16] studies continual skill-learning methods and finds gains over no-skill baselines, but no method dominates across tasks and models. SkVM [3] treats skills as compiled capabilities across heterogeneous LLM backends. These studies motivate testing reusable skills, but they do not ask which parts of generated skills help or hurt in data-science workflows.

On the benchmark side, DS-1000 [5], DataSciBench [15], and LLM4DS [11] evaluate LLM code generation for data-analysis tasks and show that strong models can solve many data-science tasks from the task prompt alone. That creates a hard test for reusable skills: if task-only prompting already covers many common data-science procedures, skills have less room to help and may still add cost or conflict with task-specific instructions. These benchmarks evaluate data-science capability, but they do not test whether reusable skills improve performance in this domain.

Recent data-preparation and table-reasoning systems also point to a different form of support than static front-loaded instructions. PrepBench evaluates natural-language-driven data preparation and emphasizes ambiguous intents, imperfect real-world data, interactive disambiguation, code generation, and workflow translation [13]. AutoDCWorkflow generates data-cleaning workflows from a raw table and analysis purpose, then evaluates answer, data, and workflow quality [7]. Chain-of-Query uses multi-agent collaboration and clause-by-clause SQL generation for table understanding [12]. These systems do not evaluate reusable skill files directly, but they suggest one limitation of flat prompt injection: data-science agents may need task-specific orchestration, validation, and feedback.

3 Experimental Design

3.1 Skill Construction

We generated one skill per lifecycle stage (four skills total) using a Gemini 2.5 Pro coding agent with extended thinking enabled. The agent received the target workflow, required section structure, and `SKILL.md` schema, and it could research the target domains and related benchmarks before writing each Full skill. Each skill was produced in one autonomous session with no human editing, candidate selection, or iterative refinement. This is the low-curation workflow we study: generate one skill for a task family, then reuse it as written.

Each skill contains four sections:

- *Routing* (~50 tokens): activation triggers specifying when to apply the skill
- *Core Procedure* (~190 tokens): step-by-step workflow instructions
- *Worked Examples* (~225 tokens): concrete input→output demonstrations
- *Reference Notes* (~225 tokens): supplementary heuristics and conventions

These sections correspond to the content types in Anthropic's Agent Skills documentation [1]. Unlike Anthropic's recommended architecture, which stores reference materials in separate files loaded on demand, we inject all sections as a single flat file. We make this choice to isolate content: the experiment tests whether generated skill content helps when the model sees it, not whether a separate loader can retrieve the right file at the right time. For this reason, the study should not be read as an evaluation of selective loading or progressive disclosure. The format also matches a common low-curation implementation: a single markdown document produced without explicit tooling for multi-file packages.

The ablation variants are produced by mechanically deleting sections from the Full skill, not by separate generation runs. This matters for causal interpretation: the skill variants differ only in which sections are present. Sections are written to be self-contained, so deletion does not leave dangling cross-references. This design lets us compare the full skill against variants that remove supporting content, while keeping the basic task procedure and generation source fixed.

3.2 Ablation Conditions

Our ablation treats Worked Examples and Reference Notes as two independent binary factors, with Routing and Core Procedure held constant. We keep Routing and Core Procedure together because, without activation triggers and procedural steps, the remaining sections lack the context needed to function as a coherent skill. Routing is also redundant in our setup because each skill is only paired with tasks from its matching lifecycle stage. This yields four skill conditions plus a No-Skill baseline:

(1) **No-Skill**: task prompt only
(2) **Core-Only**: Routing + Core Procedure
(3) **Core+Examples**: Routing + Core Procedure + Worked Examples
(4) **Core+Refs**: Routing + Core Procedure + Reference Notes
(5) **Full**: all four sections

Skill content is injected unconditionally as a user-message prefix using a fixed template (`[SKILL START] ... [SKILL END]`, followed by the task prompt) across all conditions and providers.

The No-Skill baseline contains only the task prompt, with no injected skill wrapper or control text. Thus, the comparison is between the same task prompt alone and the same task prompt with generated skill content prepended.

Length-Control tests a simpler alternative explanation: the model may react to a long, skill-formatted prefix rather than to useful data-science guidance. To test this, we add a supplemental **Length-Control** condition. For each lifecycle stage, we create a task-irrelevant office-supply setup skill with the same injection wrapper, markdown-style organization, and tokenizer length within 1% of the corresponding Full skill.

We use coherent office-supply guidance rather than random strings, which could create an unrealistic distraction. The control gives the model plausible procedural instructions, examples, and notes, but none are intended to help with data cleaning, SQL, statistical analysis, or reporting. We treat it as a control, not a harmless placebo, since unrelated instructions can still change model behavior. Comparing Full against Length-Control estimates whether the generated data-science content helps beyond adding similarly long, skill-formatted context.

Generic skill guidance can conflict with the task prompt. If the main problem is priority ambiguity, a short rule telling the model to follow the task over the skill should recover some failures. To test this, we define a supplemental **Full+Priority** condition: it uses the same `full.md` skill files as Full, but inserts a user-message directive after `[SKILL END]` and before the task prompt stating that task instructions override conflicting skill guidance. This is a prompt-level task-over-skill directive, not a system-level instruction hierarchy or retrieval-gated priority mechanism.

These supplemental conditions are narrow checks around the main flat-injection ablation. They do not test selective loading, system-level instruction priority, or schema/tool constraints.

The design also omits an expert-written or task-specific positive-control skill. This keeps the focus on low-curation LLM-generated skills, but it means the study does not test whether this benchmark would detect gains from carefully authored skill content.

3.3 Tasks and Lifecycle Mapping

We evaluate 56 tasks (14 per lifecycle stage), mapped to four data-science workflows:

- **Data Preparation** (14 tasks): CSV cleaning, including null imputation, deduplication, type coercion, and date standardization
- **Data Extraction** (14 tasks): SQL query formulation against relational databases, using schemas from Spider [14] and standard benchmark databases
- **Statistical Analysis** (14 tasks): hypothesis testing, effect-size computation, and statistical inference using standard datasets (Fisher Iris [4], mtcars, PlantGrowth, ToothGrowth)
- **Reporting** (14 tasks): generating structured JSON reports from API responses (GitHub Events, OpenWeatherMap, REST Countries)

These tasks cover the verifiable execution layer of product data-science work rather than the full product decision-making lifecycle. We intentionally exclude upstream product framing and experiment design because those tasks often depend on business context, stakeholder constraints, engineering feasibility, and judgment calls that are not well captured by the deterministic verifiers used here.

Tasks are split evenly between self-authored and externally grounded examples to reduce bias from task authorship. The **self-authored** subset contains 28 tasks (7 per stage). A Gemini 2.5 Pro coding agent with extended thinking enabled drafted the prompts and initial gold outputs for these tasks.

The **externally grounded** subset also contains 28 tasks (7 per stage) and uses public datasets, benchmark schemas, or public APIs rather than purely synthetic inputs.[1]

Gold outputs were then validated by scripts that recompute expected results from source data where applicable. Scoring uses fixed gold files and deterministic verifiers, so Gemini does not act as the evaluator at test time. All task prompts and gold outputs were finalized before skill generation, preventing post-hoc alignment between task requirements and skill content.

3.4 Verification

All outcomes are binary pass/fail, evaluated by fully automated, deterministic verifiers with no human judgment involved. Each lifecycle stage uses a verifier matched to its output type:

- **Data Preparation**: the model's output CSV is parsed into a DataFrame and compared against the gold CSV. Exact-mode tasks check row-level values, ignoring column order but preserving case sensitivity. Structural-mode tasks check expected columns, row counts, absence of nulls, and date-format compliance.
- **Data Extraction**: the model's SQL is executed against an in-memory SQLite database seeded with task-specific test data. The result set is compared to the gold query output with order-insensitive row matching.
- **Statistical Analysis**: numeric outputs (test statistics, p-values, effect sizes) are extracted and compared against gold values with tolerance $\epsilon = 0.01$. Both absolute and relative tolerance are applied to accommodate rounding differences.
- **Reporting**: JSON output is validated against a task-specific schema, including required keys, value types, nested structure, and selected field values where specified.

Exact-match verifiers can conflate reasoning failures with format errors. To check this risk, we programmatically identified all condition-flip cases, where a task passes under one condition but fails under another, and reviewed the associated verifier messages. In these cases, failures reflected substantive errors, such as wrong imputation method or wrong case convention due to competing instructions, rather than trivial formatting mismatches.

3.5 Models

We select nine model configurations spanning three providers (Table 1), varying along three dimensions. First, *model capability*: we include compact models (GPT-4o-mini, Gemini Flash, Claude Haiku) and frontier default models (GPT-4o, Gemini Pro, Claude Sonnet) to test whether stronger models use skills differently. Second, *provider diversity*: sampling from OpenAI, Google, and Anthropic guards against provider-specific artifacts.

Third, *reasoning mode*: Gemini 2.5 Flash and Claude Sonnet 4 are each tested with extended thinking disabled and enabled, and o3-mini provides an always-on reasoning baseline. For stratified analysis, we use three pre-specified groups: compact default models,

[1] SQL tasks use schemas from Spider [14] and Chinook/Northwind-style databases; Data Preparation tasks use the Melbourne Housing, Titanic, UCI Wine Quality, Auto MPG, and student performance datasets; Statistical Analysis tasks use Fisher Iris [4], mtcars, PlantGrowth, ToothGrowth, and the sleep dataset; Reporting tasks use the GitHub Events, OpenWeatherMap, and REST Countries APIs.

frontier default models, and explicit-reasoning configurations. Gemini 2.5 Pro remains in the frontier default group because it was run only in its default configuration, with no paired disabled-thinking condition. Moving it into the explicit-reasoning group would mix model scale with reasoning-mode treatment and break the balanced design.

Model	Provider	Reasoning Mode	Group
GPT-4o-mini	OpenAI	No	Compact
GPT-4o	OpenAI	No	Frontier
o3-mini	OpenAI	Native	Explicit-reasoning
Gemini 2.5 Flash	Google	Disabled	Compact
Gemini 2.5 Flash^{+}	Google	Enabled	Explicit-reasoning
Gemini 2.5 Pro	Google	Default	Frontier
Claude Haiku 4.5	Anthropic	No	Compact
Claude Sonnet 4	Anthropic	Disabled	Frontier
Claude Sonnet 4^{+}	Anthropic	Enabled	Explicit-reasoning

Table 1: Model configurations.

Note. Groups are balanced by design. Explicit-reasoning denotes configurations run through explicit reasoning-mode settings or a reasoning-only API; it does not imply that other models lack internal reasoning. Gemini 2.5 Pro is treated as frontier default because there is no paired disabled-thinking Gemini Pro condition. Temperature is set to 0 where supported; o3-mini and Claude Sonnet 4^{+} use provider-required reasoning settings. Temperature 0 reduces sampling variation but does not make calls deterministic.

Exact API model identifiers are recorded in the evaluation harness. The $^{+}$ variants use the same base identifiers as their non-$^{+}$ counterparts, with extended thinking enabled.

Each of the $56 \times 5 \times 9 = 2{,}520$ main-ablation cells is run three times, yielding 7,560 runs. The supplemental Length-Control and Full+Priority conditions each add $56 \times 1 \times 9 \times 3 = 1{,}512$ runs, for 10,584 total runs. We use three repetitions because API providers can exhibit minor non-determinism even under low-variation decoding settings, due to factors such as batching and quantization. Of the 2,520 main-ablation cells, 93.4% are unanimous (all three repetitions produce the same pass/fail outcome) and only 6.6% show a split verdict. We use majority vote (2-of-3 pass) as the cell-level outcome to avoid treating repetitions as independent observations.

3.6 Analysis

We analyze results with four complementary methods:

- **Mixed-effects model.** We fit a linear mixed-effects model with condition as a fixed effect, task as a random-intercept grouping factor, and model as a random-effect variance component. This estimates condition effects after accounting for task difficulty and model strength. We report the linear model because its coefficients directly express percentage-point differences; a logistic GEE and a non-parametric permutation test give substantively identical conclusions.
- **Group-stratified bootstrap analysis.** We estimate skill effects separately for compact, frontier, and explicit-reasoning groups to test whether effects differ across model configurations.
- **Bootstrap confidence intervals.** We compute percentile bootstrap intervals with 10,000 iterations over the 56 task-level paired differences.
- **McNemar tests.** For each skill condition, we count task-model pairs where skill content flips the outcome from fail to pass versus pass to fail, then test whether these counts differ.

We analyze Length-Control and Full+Priority as supplemental paired controls against No-Skill and Full, rather than as skill components in the main ablation.

4 Results

4.1 LLM-Generated Skills Do Not Improve Performance

Table 2 presents pass rates by model and condition. The aggregate spread across all five conditions is just 1.2 pp (67.3% to 68.5%). The mixed-effects model (Table 3) gives the same conclusion: no condition reaches significance at α=0.05, and all four skill conditions fall well short (all $p \geq 0.396$).

Paired McNemar tests tell a similar story: for each skill condition, skills help and hurt on roughly equal numbers of task×model pairs (e.g., Core+Examples: 20 helped vs. 19 hurt, p=1.000). Full skills consume ∼4.5× the input tokens of No-Skill (mean 1,293 vs. 287 input tokens), so the tested knowledge-injection mechanism adds cost without measurable accuracy gain.

4.2 Task Source Does Not Explain the Null

Because half of the tasks are self-authored and half are externally grounded, we check whether the null result is an artifact of task construction. Table 4 shows that the source split does not change the conclusion. Full skills are slightly below No-Skill for both self-authored tasks (−0.8 pp, 95% CI [−5.2, +3.2]) and externally grounded tasks (−0.8 pp, 95% CI [−5.2, +2.8]). Averaging across all four skill conditions yields the same pattern: −0.7 pp on self-authored tasks and −0.5 pp on externally grounded tasks. Thus, the aggregate null is not driven by one source group.

4.3 Token-Matched Control

We additionally compare Full skills against a token-matched Length-Control condition containing task-irrelevant office-supply guidance. This analysis helps separate the effect of generated data-science content from the effect of adding similarly long, skill-formatted context. Length-Control reaches 66.9% pass rate, compared with 67.5% for Full and 68.3% for No-Skill. The paired comparisons in Table 5 provide no evidence that the data-science skill content improves over the matched control: Full exceeds Length-Control by only +0.6 pp (95% CI [−2.2, +3.4], p=0.775).

Length-Control also does not show strong evidence of harm relative to No-Skill (−1.4 pp, 95% CI [−3.4, +0.6], p=0.311), although the point estimate is negative. Thus, the data do not support a simple prompt-length explanation for the null, nor is there measurable evidence that the generated data-science content adds useful information beyond a similarly sized, skill-formatted control. We treat this as a control analysis, not as proof that irrelevant content has no effect.

	No-Skill	Full	Core-Only	Core+Ex	Core+Refs
GPT-4o-mini	57.1%	57.1%	57.1%	51.8%	**58.9%**
GPT-4o	**64.3%**	**64.3%**	58.9%	60.7%	60.7%
o3-mini	80.4%	80.4%	78.6%	**82.1%**	**82.1%**
Gemini 2.5 Flash	55.4%	58.9%	58.9%	**62.5%**	57.1%
Gemini 2.5 Flash^{+}	58.9%	57.1%	**60.7%**	58.9%	57.1%
Gemini 2.5 Pro	69.6%	67.9%	71.4%	**76.8%**	69.6%
Claude Haiku 4.5	**67.9%**	66.1%	66.1%	66.1%	66.1%
Claude Sonnet 4	**83.9%**	80.4%	82.1%	82.1%	82.1%
Claude Sonnet 4^{+}	**76.8%**	75.0%	73.2%	75.0%	71.4%
Average	*68.3%*	*67.5%*	*67.5%*	***68.5%***	*67.3%*

Table 2: Pass rates by model and condition.

Note. Each cell aggregates 56 tasks with three repetitions by majority vote. Bold indicates the best condition in each row; underline indicates the worst. Across model-condition cells, task-level standard deviations range from 0.37 to 0.50.

Condition	$\hat{\beta}$	p
Full	−0.008	0.497
Core-Only	−0.008	0.497
Core+Ex	+0.002	0.865
Core+Refs	−0.010	0.396

Table 3: Mixed-model condition effects.

Note. n=2,520 majority-vote outcomes; No-Skill is the reference. $\hat{\beta}$ is the effect in pass-rate points. The model includes task random intercepts and a model random-effect variance component. Logistic GEE and permutation tests yield substantively identical conclusions (all p>0.45).

Task Source	Tasks	No-Skill	Full	Avg Skill	Skill Diff.
Self-authored	28	70.2	69.4	69.5	−0.7 [−4.3, +2.6]
Externally grounded	28	66.3	65.5	65.8	−0.5 [−4.3, +2.7]

Table 4: Task-source split.

Note. Entries are percentages based on majority-vote outcomes. Avg Skill averages Full, Core-Only, Core+Examples, and Core+Refs. Skill Diff. reports Avg Skill minus No-Skill, with 95% bootstrap CIs over task-level paired differences.

Comparison	Diff.	95% CI	Help/Hurt	p
Length-Control vs. No-Skill	−1.4 pp	[−3.4, +0.6]	14/21	0.311
Full vs. Length-Control	+0.6 pp	[−2.2, +3.4]	26/23	0.775
Full vs. No-Skill	−0.8 pp	[−3.8, +2.0]	19/23	0.644

Table 5: Paired length-control comparisons.

Note. Each condition has 504 majority-vote outcomes. Differences are percentage-point changes for the left condition relative to the right. Help/Hurt counts task×model pairs where the left condition changes a failure to a pass versus a pass to a failure. Confidence intervals bootstrap over task-level paired differences; p values use two-sided McNemar tests.

4.4 Prompt-Level Priority Control

We next test whether a prompt-level task-over-skill directive mitigates the suspected skill-task conflicts. Full+Priority reaches 68.7% pass rate, compared with 67.5% for Full and 68.3% for No-Skill. The paired difference is small and not statistically reliable: Full+Priority exceeds Full by +1.2 pp (95% CI [−1.0, +3.6], p=0.362) and exceeds No-Skill by +0.4 pp (95% CI [−2.4, +3.0], p=0.875). In the mixed-effects model, the Full+Priority coefficient relative to No-Skill is +0.004 (p=0.744).

The cell transitions suggest a tradeoff rather than a clean repair. Full+Priority recovers 12 of the 23 task×model cells where No-Skill passes but Full fails, consistent with the directive resolving some skill-task conflicts. But it also introduces 8 new failures among cells where both No-Skill and Full pass, and loses 4 cells where Full alone helped. The added failures are mostly format or output-discipline errors, suggesting that the priority rule can pull the model back toward the task while weakening useful discipline supplied by the skill.

The directive also does not materially repair the main Data Preparation harm. Data Preparation rises from 50.8% under Full to 51.6% under Full+Priority, still below the 57.1% No-Skill baseline. A simple user-message priority sentence is therefore not enough to remove the conflict-heavy failure mode. Stronger mechanisms, such as system-message placement, retrieval-gated skill snippets, or schema/tool-level constraints, remain open.

4.5 No Component Shows a Reliable Benefit

The aggregate null could still hide component-level cancellation: one section might help while another hurts. The component-level comparisons are small and non-significant, but they help characterize why the aggregate effect is flat:

- **Reference Notes.** Core+Refs (67.3%) falls below No-Skill (68.3%) by −1.0 pp, and below Core-Only (67.5%) by −0.2 pp.
- **Worked Examples.** Core+Examples (68.5%) is closest to No-Skill (68.3%), differing by just +0.2 pp.
- **Full Skill.** Full (67.5%) matches Core-Only, suggesting that adding both Examples and Reference Notes to the base procedure produces no net change.

Because the design crosses Worked Examples and Reference Notes, we can also estimate their interaction. Adding Reference Notes on top of Core+Examples changes performance by −1.0 pp, close to the −0.2 pp change from adding Reference Notes on top of Core-Only, yielding an interaction of −0.8 pp. This small value gives little evidence that a large positive effect from examples is being canceled by a large negative effect from reference notes. It

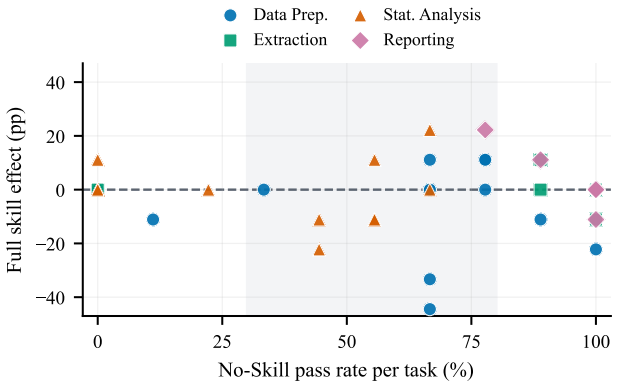

Figure 1: Full-skill effects by task.

Note. Each point is one task, averaged over nine models after majority vote. The shaded region marks the 30 to 80% No-Skill range used for the informative-task analysis.

does not, however, reveal whether models internally ignore, give less weight to, or reconcile competing sections.

These near-zero differences are not just cancellation across models: 4 of 9 models achieve their best pass rate under No-Skill, and no single skill condition is best for more than 3 models. The component ablation therefore gives little evidence that any section helps on its own.

4.6 Failures Vary by Workflow Difficulty

The ablation results provide little support for the simplest cancellation story, but they do not by themselves explain why skill content fails in different workflows. Table 6 shows that the aggregate null is not a single uniform failure. Baseline difficulty differs sharply across lifecycle stages, and each regime points to a different plausible reason why flat, LLM-generated skill injection may fail to help.

Lifecycle Stage	No-Skill	Full	L-Control	C-Only	C+Ex	C+Refs	Range (SD)
Data Extraction	88.9	91.3	89.7	89.7	90.5	89.7	0 to 100 (26.1)
Reporting	97.6	99.2	97.6	99.2	98.4	99.2	78 to 100 (6.4)
Data Preparation	57.1	50.8	52.4	52.4	54.8	52.4	0 to 100 (32.6)
Stat. Analysis	29.4	28.6	27.8	28.6	30.2	27.8	0 to 67 (28.4)

Table 6: Pass rates by lifecycle stage.

Note. Entries are percentages. L-Control is the token-matched task-irrelevant control. Range and SD summarize within-stage task difficulty under No-Skill.

Figure 1 shows the same pattern at task level. Many tasks sit near ceiling or floor under No-Skill, limiting the room for measurable gains. Within the informative 30 to 80% band, full-skill effects remain mixed and often negative, especially for Data Preparation and Statistical Analysis tasks.

Pattern 1: Redundancy at ceiling. Data extraction (88.9%) and reporting (97.6%) exhibit near-ceiling baselines, leaving little room for improvement. This may reflect both the relative simplicity of many tasks in these stages and the extensive representation of SQL and JSON generation in LLM training corpora. In these stages, skill content appears mostly redundant with behavior the model already has in this benchmark.

Pattern 2: Conflicting heuristics at moderate baselines. Data preparation (57.1%) sits in the informative difficulty range, yet skills show a slight negative direction (−2.3 to −6.3 pp). Examining condition-flip cases suggests a plausible explanation: among the 14 task×model pairs where Core-Only passes but Core+Refs fails, data preparation tasks account for 8 of 14 flips. The remaining flips span Statistical Analysis (3), SQL (2), and Reporting (1), so the example below illustrates the largest category rather than all failures. We treat this analysis as diagnostic evidence rather than a causal mechanism claim. One representative case illustrates the problem:

> **Task instruction**: "fill missing ages with *median*, missing ports with 'Unknown'." **Reference Notes**: "For numeric columns with <30% missing, apply *forward-fill*. For string columns, fill with 'UNKNOWN'." Under Core-Only, the output follows the task and passes. Under Core+Refs, the output instead follows the skill's generic heuristics, producing output that fails verification.

When LLM-generated skills contain generic heuristics that conflict with task-specific instructions, outputs sometimes follow the skill content, introducing errors that would not occur without it. The risk is not just low skill quality: plausible but over-general rules can compete with the actual task.

Pattern 3: Capability gaps at floor. Statistical analysis tasks (29.4%) require multi-step reasoning over hypothesis tests and effect-size computations, and the low baseline suggests that models often struggle regardless of skill content. The best skill condition adds only +0.8 pp, which is consistent with a computational or statistical reliability bottleneck that prompt-level knowledge injection alone does not fix.

Difficulty also varies substantially within each lifecycle stage. Of the 56 tasks, 17 fall in the informative 30 to 80% No-Skill baseline range (9 Data Preparation and 7 Statistical Analysis tasks, plus 1 Reporting task). The remaining 39 tasks are near ceiling (28 tasks above 80%) or near floor (11 tasks below 30%), so the aggregate average should be read together with the stage-level results.

On the 17 informative tasks, all four skill conditions show a slight negative direction, but all intervals include zero: Full −2.6 pp (95% CI [−11.1, +5.2]), Core-Only −5.2 pp ([−15.7, +4.6]), Core+Examples −2.0 pp ([−7.8, +3.3]), and Core+Refs −4.6 pp ([−12.4, +2.6]). This qualifies the negative pattern: the informative subset is the right place to look for gains, but the current sample supports only a diagnostic conclusion, not a precise small-effect estimate.

4.7 Skills Do Not Interact with Model Group

One natural hypothesis is that the aggregate null conceals opposing effects across model groups: perhaps compact models benefit from structured guidance while frontier models are harmed. Figure 2 shows stratified bootstrap results by group: compact (GPT-4o-mini, Gemini Flash, Claude Haiku), frontier default (GPT-4o, Gemini Pro, Claude Sonnet), and explicit-reasoning (o3-mini, Gemini Flash^{+}, Claude Sonnet^{+}). The intervals include zero across all three groups, providing no reliable evidence of a group-specific benefit at the current sample size.

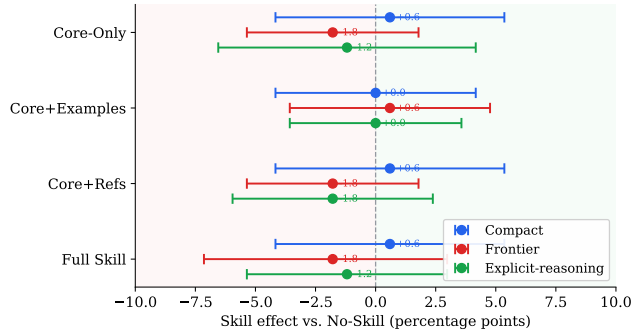

Figure 2: Skill effects by model group.

Note. Effects are relative to No-Skill with 95% bootstrap CIs. No interval excludes zero.

Compact models show effects centered near zero, ranging from 0.0 to +0.6 pp across conditions. Frontier default models show effects from −1.8 to +0.6 pp, with Full, Core-Only, and Core+Refs tied at −1.8 pp. Explicit-reasoning configurations show a similar pattern, with the largest point estimate at −1.8 pp for Core+Refs. All intervals include zero.

Individual models within each group vary substantially: Gemini Flash gains +7.1 pp from Core+Examples while GPT-4o-mini loses −5.4 pp from the same condition. This heterogeneity does not form a systematic group-level pattern. Skills are not reliably more useful for weaker, stronger, or explicit-reasoning configurations. Appendix B reports a stage-by-group diagnostic table; it does not reveal a hidden explicit-reasoning gain on data preparation or statistical analysis.

4.8 Token Cost Without Accuracy Gain

Full-skill injection increases input length by about 4.5× across providers (Table 7). Because providers charge for input tokens, this overhead increases cost before any change in output length. Input-token counts are deterministic for a fixed prompt and tokenizer, so they provide a stable measure of prompt-side overhead. We report this table for non-thinking configurations because explicit reasoning modes add provider-specific reasoning tokens, which would conflate skill-injection overhead with reasoning-budget overhead.

Provider	Condition	Mean Input Tok	P50 Output Tok	P50 Total Tok
OpenAI	No-Skill	263	337	598
	Full	1,192	353	1,502
Google	No-Skill	303	174	500
	Full	1,335	203	1,687
Anthropic	No-Skill	296	432	736
	Full	1,354	783	2,101

Table 7: Token footprint under No-Skill and Full for non-thinking configurations.

Note. Mean input tokens average deterministic prompt token counts over tasks and listed models. Output and total tokens use medians over runs. OpenAI pools GPT-4o and GPT-4o-mini; Google uses Gemini 2.5 Flash with thinking disabled; Anthropic pools Claude Haiku 4.5 and Claude Sonnet 4 with thinking disabled.

Output length changes are less uniform. Median output changes only modestly for OpenAI and Google, but rises more sharply for Anthropic in these non-thinking configurations. Thus, the total token cost of skill injection depends on both the provider and the model's response behavior.

Because the aggregate accuracy effect is null, this additional token cost comes without a measured accuracy gain in our setting. We do not report latency as a precise overhead estimate because runs were not interleaved or controlled for time-of-day and API-load variation.

5 Implications for Data-Science Agent Design

For the setting we test, the design message is limited but useful: single-shot, flat-file, LLM-generated skill packages are not a reliable default. Different lifecycle stages appear to need different kinds of support.

Lifecycle-dependent strategy. The three failure patterns suggest that a one-size-fits-all knowledge injection strategy may be brittle across the data-science lifecycle. For high-baseline stages (data extraction, reporting), skills add cost (~4.5× input-token overhead) with little observed room for improvement. For moderate-baseline stages (data preparation), generic skills can harm performance by introducing conflicting heuristics. For low-baseline stages (statistical analysis), the results are consistent with a computation and statistical-reasoning bottleneck, so prompt-level instructions alone may be insufficient.

Task specificity over family coverage. Our skills target entire lifecycle stages rather than individual tasks. SkillsBench [8] found that focused skills with two to three modules outperform comprehensive documentation; our family-level skills are closer to the comprehensive end. The conflicting-heuristics failure at the data-preparation stage is consistent with this breadth: a single skill cannot anticipate the particular imputation method each task requires. Task-specific skills that match exact requirements may avoid this failure mode, though at the cost of scalability.

Selective loading and priority. Our experiment injects all skill content as a single flat file. Anthropic's recommended architecture [1] instead stores reference materials in separate files loaded only on demand. Because our conflicting-heuristics failures are disproportionately associated with Reference Notes (−1.0 pp vs. No-Skill), reference material should be gated rather than always prepended when systems support selective loading. Full+Priority also suggests that a user-message priority sentence is too weak; stronger priority mechanisms belong in system instructions, retrieval policy, or schema and tool constraints.

Feedback over front-loaded guidance. For workflows that require statistical computation or complex data cleaning, execution feedback may matter more than front-loaded generic guidance. The failure patterns point toward agents that retrieve narrower guidance, execute code, validate outputs, and revise, rather than agents that paste a full family-level skill into the first prompt.

6 Limitations

Our study has three main limitations. First, the scope is narrow. We evaluate 56 single-turn tasks in one domain, and only 17 tasks fall in the informative 30 to 80% baseline range. The 95% bootstrap CIs span ±4 to 6 pp, so the study can rule out moderate average effects

but not small effects (1 to 3 pp). We also do not include an expert-written or task-specific positive-control skill; that choice matches the low-curation workflow we test, but leaves the benchmark's sensitivity to carefully authored skills unmeasured.

Second, the setting is not a full data-science agent. It omits multi-turn planning, tool use, code execution, memory, iterative revision, and upstream product-framing tasks that require business context or stakeholder judgment. Skill placement is also fixed: skills are prepended as user-message content, with no system-level priority rule, retrieval gate, or schema/tool constraint. Our findings therefore apply most directly to single-shot, flat-file, LLM-generated, family-level skills on deterministically verifiable execution tasks.

Third, several confounds remain. Skill conditions increase prompt length (from a mean of 287 input tokens for No-Skill to 1,293 for Full). The Length-Control condition reduces concern that length alone explains the null, but it uses one irrelevant domain, office-supply setup, so it cannot characterize all forms of unrelated context. The use of one generated skill per lifecycle stage also ties section effects to four skill instances. Sampling multiple generated skills per stage would estimate inter-generation variance, but would no longer match the one-candidate low-curation workflow we study.

There is also an authorship overlap. Gemini 2.5 Pro assisted with skill writing and with drafting the self-authored task prompts and initial gold outputs, and Gemini 2.5 Pro is one of the evaluated model configurations. Deterministic validation, fixed gold files, and within-task condition comparisons reduce the risk that scoring directly rewards Gemini outputs. Still, the overlap could affect task style, formatting conventions, or absolute difficulty, especially for exact-match tasks. A cleaner design would separate task authoring, skill authoring, and model evaluation.

Finally, the condition-flip review gives only partial evidence about mechanism. It can identify overt cases where the output follows a generic skill heuristic over the task instruction, but it cannot count all cases where the skill was ignored, partially used, or internally reconciled with the task. We therefore report the flip analysis as diagnostic rather than as a complete instruction-adherence taxonomy.

7 Conclusion

We evaluated whether low-curation, LLM-generated skills improve data-science agent performance across four data-science lifecycle stages when skills are injected directly as flat prompt content. Across 56 tasks, nine model configurations, and three providers, neither the full generated skill nor any ablated skill variant significantly outperforms the task-only baseline. A token-matched control also performs similarly to the full skill, and a prompt-level task-over-skill directive does not reliably improve performance. The null result is consistent across skill components and model groups, and it aligns with three diagnostic patterns: redundancy at high baselines, conflicting heuristics at moderate baselines, and statistical-reasoning bottlenecks at low baselines.

For practitioners building data-science agents, these results support a cautious default: LLM-generated, family-level skill injection did not improve performance in this single-shot setting. The next tests should compare task-specific skills, selective loading, system-level priority rules, and execution-feedback agents.

References

[1] Anthropic. 2025. Agent Skills: Overview. https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview Accessed: 2026-04-28.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., Red Hook, NY, USA, 1877–1901.

[3] Le Chen, Erhu Feng, Yubin Xia, and Haibo Chen. 2026. SkVM: Revisiting Language VM for Skills across Heterogenous LLMs and Harnesses. arXiv:2604.03088

[4] Ronald A. Fisher. 1936. The use of multiple measurements in taxonomic problems. *Annals of Eugenics* 7, 2 (1936), 179–188.

[5] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation. arXiv:2211.11501

[6] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[7] Lan Li, Liri Fang, Bertram Ludäscher, and Vetle I. Torvik. 2025. AutoDCWorkflow: LLM-based Data Cleaning Workflow Auto-Generation and Benchmark. arXiv:2412.06724 EMNLP Findings 2025.

[8] Xiangyi Li et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv:2602.12670 https://arxiv.org/abs/2602.12670

[9] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity. arXiv:2104.08786

[10] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? arXiv:2202.12837

[11] Nathalia Nascimento, Everton Guimaraes, Sai Sanjna Chintakunta, and Santhosh Anitha Boominathan. 2024. LLM4DS: Evaluating Large Language Models for Data Science Code Generation. arXiv:2411.11908

[12] Songyuan Sui, Hongyi Liu, Serena Liu, Li Li, Soo-Hyun Choi, Rui Chen, and Xia Hu. 2025. Chain-of-Query: Unleashing the Power of LLMs in SQL-Aided Table Understanding via Multi-Agent Collaboration. arXiv:2508.15809 AACL 2025 Main Conference (Oral).

[13] Jingzhe Xu, Rui Wang, Jiannan Wang, and Guoliang Li. 2026. PrepBench: How Far Are We from Natural-Language-Driven Data Preparation? arXiv:2605.08687

[14] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 3911–3921.

[15] Dan Zhang, Sining Zhoubian, Min Cai, et al. 2025. DataSciBench: An LLM Agent Benchmark for Data Science. arXiv:2502.13897

[16] Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. 2026. SkillLearnBench: Benchmarking Continual Learning Methods for Agent Skill Generation on Real-World Tasks. arXiv:2604.20087

A Length-Control Token Matching

The Length-Control condition uses a task-irrelevant office-supply setup skill for each lifecycle stage. The control skills contain coherent procedural content about arranging a shared supply station, including activation guidance, steps, examples, and reference notes. They preserve the same flat injection wrapper and skill-like markdown organization as the Full skills, but remove data-science content. Token counts were computed with `tiktoken` v0.13.0 under both `o200k_base` and `cl100k_base`. Delta is computed as (Length-Control − Full)/Full.

	`o200k_base`			`cl100k_base`		
Lifecycle Stage	**Full**	**Control**	**Delta**	**Full**	**Control**	**Delta**
Data Preparation	725	721	−0.55%	718	724	+0.84%
Data Extraction	910	902	−0.88%	912	908	−0.44%
Statistical Analysis	1094	1090	−0.37%	1095	1098	+0.27%
Reporting	947	942	−0.53%	947	953	+0.63%

Table 8: Token matching for the Length-Control skills.

Note. The control skills are token-matched to the corresponding Full skills within 1% under both tokenizers.

B Stage-by-Group Diagnostic Effects

Table 9 reports Full-minus-No-Skill effects within each lifecycle stage and model group. These are descriptive diagnostics, not a formal search over all stage, group, and component interactions. The table does not show evidence that explicit-reasoning configurations uniquely benefit from Full skills on statistical analysis or data preparation; those two rows are near zero or negative across groups, with wide confidence intervals.

Lifecycle Stage	**Compact**	**Frontier**	**Explicit-reasoning**
Data Preparation	−4.8 [−16.7, +7.1]	−7.1 [−19.0, +2.4]	−7.1 [−19.0, +2.4]
Data Extraction	+4.8 [0.0, +11.9]	+0.0 [−7.1, +7.1]	+2.4 [0.0, +7.1]
Stat. Analysis	+0.0 [−9.5, +9.5]	+0.0 [−14.3, +14.3]	−2.4 [−9.5, +4.8]
Reporting	+2.4 [−7.1, +14.3]	+0.0 [0.0, 0.0]	+2.4 [0.0, +7.1]

Table 9: Full-skill effects by lifecycle stage and model group.

Note. Entries are percentage-point differences relative to No-Skill with 95% bootstrap CIs over task-level paired differences.