Astra and Sol in Galley: Evaluation Methods and Results

This resource supports Switching from GPT-5.6 Sol to GPT-6 Astra: Start with Medium Effort, with the comparison's criteria, findings, and usage figures.

Scores by phase

Phase	Astra low	Astra medium	Astra high	Sol high
Repository analysis	83	85	91	77
Implementation	81	84	92	69
Code review	86	88	90	86

Scores are out of 100. Astra high produced the strongest implementation; medium found a startup failure that high missed in review. Medium also had the lowest total cost. These results support starting with medium, choosing high for work across retries and persisted state, and keeping a separate review pass.

Experiment and scope

Four conditions worked on Galley, a tool for unattended development: GPT-6 Astra at low, medium, and high effort, and GPT-5.6 Sol at high effort. Each performed repository analysis, implementation, and code review once.

Implementation and review used Codex CLI 0.153.4 on macOS, with Galley's repository instructions, quality profile, and skills from codex-workflows.

- **Analysis:** investigate the repository and select worthwhile improvements.
- **Implementation:** execute the same plan in separate worktrees. The plan came from Astra findings; Sol-only findings did not expand it.
- **Review:** use fresh sessions to review identical copies of Sol's implementation.

The implementation plan covered Git change capture, input reuse and cleanup, corrupt-task recovery and process ownership, scheduling and timeouts, YAML maintenance, verification

evidence, preparation reuse, and installer checksums. Agents chose the design and verification while preserving existing operator controls, including manual requeue of running tasks.

Why these evaluation criteria

I wanted agents to find and fix consequential problems without adding more machinery than the product needed. Engineering judgment therefore counted alongside technical correctness. Overengineering meant lasting complexity out of proportion to user value, rather than a large diff or attention to rare cases.

Analysis and review weighted technical validity most heavily because a finding can lead to changes in working software:

Analysis and review axis	Weight	What was evaluated
Discovery	30	Important problems found across relevant responsibilities and paths; breadth and impact rather than count.
Technical validity	40	Whether conditions, causes, consequences, existing contracts, and proposed remedies agree with the code.
Engineering judgment	30	Selection, priority, scope, and whether the proposed work earns its maintenance and operational cost.

Implementation assessed the finished behavior and the agent's ability to deliver it:

Implementation axis	Weight	What was evaluated
Correctness and completeness	40	Required behavior works through related paths and preserves existing contracts.
Design and maintainability	30	Causes are addressed at suitable boundaries, with justified complexity and appropriate reuse.
Autonomous execution	30	The agent interprets scope, investigates, implements, verifies, recovers from failures, and reports completion accurately.

Scores reflect judgment within each axis, not points per finding. An omission counted as deliberate restraint only when visible evidence supported it. Because implementation uses

different axes, the phase scores are not combined into one quality score.

Repository analysis

Condition	Discovery / 30	Technical validity / 40	Engineering judgment / 30	Total / 100
Astra low	25	34	24	83
Astra medium	27	35	23	85
Astra high	27	37	27	91
Sol high	26	33	18	77

Low found useful ownership and input-resolution problems, but also proposed restricting manual requeue and expanding cleanup changes into archive behavior.

Medium matched high's discovery score. Its main judgment weakness was treating manual requeue as a defect even though documentation and tests established it as an operator override.

High found evidence-loss problems and checked the intended behavior before excluding the requeue restriction. That supported its stronger judgment score.

Sol high found valid issues, but several proposed fixes added broader fingerprint, rereview, artifact-routing, or pruning mechanisms than the demonstrated need justified. Its marker-lock proposal also left crash recovery unresolved.

Implementation

Condition	Correctness / 40	Design and maintainability / 30	Autonomous execution / 30	Total / 100
Astra low	31	24	26	81
Astra medium	33	25	26	84
Astra high	37	27	28	92

Sol high	27	20	22	69

All four addressed the plan. The differences appeared where operations interacted across runs.

Low made mostly local changes, but left preparation invalidating its own reuse key, large staging operations exceeding argument limits, and gaps in run attribution and corrupt-task isolation.

Medium repaired reuse for the skeleton stage, but generated verification text still changed the setup key. It handled run attribution more completely than low while retaining the large-staging and corrupt-history gaps.

High separated generated verification text from the user contract used for reuse keys, extended an existing stdin pathspec mechanism for large staging operations, and passed the recovery checks below. Its broader fingerprints and process-ownership migration still carried maintenance costs.

Sol high left repeat cleanup broken and could adopt an unrelated file as a reused input. It retained both old and new scheduling paths, increasing maintenance work. Its custom PID/file locking added rules where the Astra implementations used OS locking.

Focused implementation checks

Evaluator checks supplemented the agents' own tests:

Check	Low	Medium	High	Sol high
Repeat cleanup after the parent directory has been pruned	Pass	Pass	Pass	Fail
Prior input evidence cannot adopt an unrelated file in another worktree	Pass	Pass	Pass	Fail
Setup reuse key remains stable after applying a real generated skeleton	Fail	Fail	Pass	Fail
Status-to-stage operation with 16,000 paths	Argument limit	Argument limit	Pass	Argument limit
Rename handling when filenames contain `->`	Pass	Pass	Pass	Pass

Corrupt running task and same-name failed history are both isolated	Incomplete	Incomplete	Pass	Pass
Run attribution for a task name ending in a hyphen	Fail	Pass	Pass	Pass

All four fixed basic Git status-output truncation; the 16,000-path check exposed a further staging limit on macOS. “Incomplete” isolation means low and medium preserved both records but did not fully isolate them; the queue was not blocked.

Code review

All reviewers examined the same Sol implementation.

Condition	Discovery / 30	Technical validity / 40	Engineering judgment / 30	Total / 100
Astra low	22	38	26	86
Astra medium	24	38	26	88
Astra high	26	38	26	90
Sol high	26	36	24	86

Finding in the common implementation	Low	Medium	High	Sol high
Failure during a later save can roll back and delete a reused, edited input	Yes	—	Yes	Yes
Repeat cleanup fails after the parent directory is gone	Yes	Yes	Yes	—
Previous input evidence can adopt an unrelated file in another worktree	—	—	Yes	Yes
Corrupt owner data stops startup before task recovery can run	—	Yes	—	—

Age-based lock recovery can treat a live holder as stale; deletion has an ownership race	Yes	Yes	Yes	Yes
Generated verification text changes the preparation fingerprint	—	Yes	Yes	—
Requeue's review-iteration counter changes the fingerprint	—	—	Yes	—
A learned environment profile changes the next setup reuse key	—	—	—	Yes
Trimming committed paths changes whitespace-bearing filenames	Yes	—	Yes	—
Skeleton Git snapshot has no command timeout	Yes	—	Yes	Yes
Legacy process-registry fallback can select the wrong owner	—	Yes	—	Yes
Scheduling can claim another task after cancellation	—	—	—	Yes
macOS case aliases for the same repository are rejected	—	—	—	Yes

A dash means the issue was not reported. Finding more issues alone did not earn a higher engineering-judgment score.

Low found rollback deletion and several local failures, but covered fewer connected paths.

Medium uniquely found that corrupt owner data could stop daemon startup before the new recovery code ran. The evaluator reproduced this through the daemon's startup path.

High found the broadest set of input-ownership and fingerprint interactions among the Astra runs, but missed medium's startup issue. This is the strongest reason to retain an independent review even after using high for implementation.

Sol high was stronger at review than implementation. Its setup-profile, cancellation, and macOS path findings were useful. Its score was reduced for extending the macOS claim to Windows without support and proposing persistent ownership metadata before justifying the additional state.

No reviewer reported the duplicate scheduler or the large staging argument limit.

Usage, cost, and time

Costs are API-equivalent estimates using the rates recorded on September 5, 2026, rather than charges from a Codex subscription.

Model	Uncached input / million	Cached input / million	Output / million
Astra	$10	$1	$50
Sol	$4	$0.40	$20

Rates came from the official Astra model page and Sol model page.

```
cost = ((input - cached_input) × input_rate
        + cached_input × cached_rate
        + output × output_rate) / 1,000,000
```

Cached input is included in input; reasoning output is included in output. Input totals accumulate across requests. Evaluator work and subsequent repairs are excluded.

Time below subtracts estimated approval-wait overhead. The sessions ran concurrently on one machine, so these figures describe this experiment rather than standardized latency benchmarks.

Analysis usage

Condition	Input tokens	Cached input	Output tokens	Requests	Cost	Time (min)
Astra low	5,538,450	5,321,728	20,759	39	$8.526898	12.52
Astra medium	4,022,437	3,837,696	17,996	35	$6.584906	11.36
Astra high	7,178,211	6,884,224	32,553	51	$11.451744	19.54
Sol high	13,952,295	13,403,392	50,116	103	$8.559289	21.65

Implementation usage

Condition	Input tokens	Cached input	Output tokens	Requests	Cost	Time (min
Astra low	10,759,274	10,537,984	47,072	78	$15.104484	29.6
Astra medium	11,110,682	10,888,960	50,028	80	$15.607580	31.3
Astra high	14,044,933	13,697,280	77,144	107	$21.031010	47.8
Sol high	37,767,141	37,186,944	97,653	238	$19.148626	43.5

Code review usage

Condition	Input tokens	Cached input	Output tokens	Requests	Cost	Time (min)
Astra low	2,025,895	1,925,760	8,235	26	$3.338860	7.12
Astra medium	2,024,681	1,913,984	9,138	26	$3.477854	7.82
Astra high	2,847,908	2,703,104	11,968	28	$4.749544	9.78
Sol high	7,197,482	6,987,008	22,512	48	$4.086939	10.28

Totals across the three tasks

Condition	Total tokens	Requests	Estimated cost	Adjusted time (minutes)
Astra low	18,399,685	143	$26.97	49.31
Astra medium	17,234,962	141	$25.67	50.56
Astra high	24,192,717	186	$37.23	77.13

Sol high	59,087,199	389	$31.79	75.44

These totals sum separate analysis, implementation, and review tasks, rather than an end-to-end pipeline. Requests count model usage events.

Medium cost about 19% less than Sol high. Astra high cost about 17% more than Sol high and 45% more than medium. For implementation alone, high cost about 35% more than medium and took about 52% longer: roughly 48 versus 31 minutes.

Limits and retained implementation

This was one repository and one run per condition per phase. Evaluation was model-assisted and the final scoring was not blind. Read small score differences alongside the findings, rather than as a general ranking of models. The evaluation did not include native Windows execution.

The Astra high implementation was retained in Galley PR #137. For the practical interpretation, return to the article.