QUOTEBENCH: WHEN MATCHED SCORES HIDE COMMAND-PATH FAILURES

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM coding agents issue Bash commands through interfaces that may serialize, wrap, and reparse model output. Matched execution scores alone cannot distinguish command-generation errors from failures introduced after generation. **QuoteBench** separates the two by replay. It holds a model's reply fixed and executes it on two transports, direct and through one deliberately unescaped added parser, and crosses that with two generation contracts, one silent about the boundary and one that discloses it. The matched gap between the two paths then decomposes exactly, per task, into transport damage and contract-conditioned compensation. Across eight same-window configurations on 56 exact-final-state tasks from 14 incident-derived families, replaying the same reply through the added parser loses 55.4–73.2 points; disclosure recovers 30.4–60.7 points for six configurations and zero or slightly negative for the other two, and three public draws preserve both ranges. Raw generation is nearly saturated at the frontier, so boundary adaptation is what separates models: at the frontier the two effects cancel (GPT-5.6-sol's matched gap of -3.6 points hides -64.3 damage and $+60.7$ compensation), while elsewhere the matched score falls with the path, and the deployment configuration reorders models. A real `ssh` boundary reproduces the damage, and correct escaping or a temporary script removes it. Evaluations of command-issuing agents should report the model configuration, generation contract, execution path, operating point, and final-state validator instead of treating a matched score as an intrinsic model property.

1 INTRODUCTION

Bash quoting failures can corrupt literals, break routine agent actions, and trigger repair loops. Even simple tasks such as writing exact bytes, passing a literal argument, editing JSON, or invoking a remote-like wrapper must preserve quotes, dollar signs, backticks, newlines, glob characters, and expansion timing (Free Software Foundation, 2025; Holen, 2012; Wheeler, 2010). Public issue trackers document broken heredocs (the shell's inline multiline-string syntax), over-quoted operators, and repeated repair attempts (Anthropic Claude Code community, 2026b; OpenAI Codex community, 2026; Warp community, 2025). The full survey appears in Appendix A.

A failure also consumes a model generation and tool invocation, while diagnosis and retry remain in the trace. A common recovery is to write and execute a temporary script, adding actions and possibly workspace artifacts. The incident evidence guides family selection and mechanism coverage, not prevalence estimation.

Current benchmarks leave this failure mode entangled with other capabilities. Broad coding and terminal benchmarks combine command construction with planning, repository navigation, and recovery (Liu et al., 2024; Zhou et al., 2024; Xie et al., 2024; Yang et al., 2023; Jimenez et al., 2024; Merrill et al., 2026), whereas command-generation benchmarks score emitted programs under fixed transport (Lin et al., 2018; Agarwal et al., 2021; Westenfelder et al., 2025; Yu et al., 2026). Thus agent success does not reveal whether the first command preserved its payload, and command-generation success does not show whether it survives deployment. Bash quoting permits a focused test because parser rules are public and final state is exactly checkable (Free Software Foundation, 2025).

We introduce **QuoteBench**, a benchmark of one-shot LLM-generated Bash commands. Its 56 tasks cover 14 operation families, each with one benign control and three hazardous payload variants. The

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

tasks exercise multiline text, hostile filenames, regular-expression and glob metacharacters, heredocs, literal argv and environment values, Git metadata, and two local SSH-like simulations. Validators inspect final bytes, argv, JSON, directory state, or Git history, so any command that reaches the correct state receives credit.

Agent interfaces range from direct shell actions to structured action languages (Yang et al., 2024; Wang et al., 2024; Kim et al., 2026). We call a reply that runs as the shell program *raw*, and a command field extracted from a structured tool call *native*. A fixed-commit survey of six public agent systems finds both in use (Appendix Table 5).

Commands targeting remote or containerized machines can cross another boundary and be reparsed inside double quotes, for example through `ssh host "..."`, `docker exec sh -c "..."`, or a CI `run:` step. QuoteBench uses this pattern as the *nested* condition, a controlled intervention that adds one downstream parser. Five of the seventeen retained public incidents contain such a boundary.

We ask how reliability varies across models, operation families, and command paths, how generation contract interacts with execution transport for fixed replies, and what provider-exposed effort ladders reveal about matched and replayed behavior. At the fixed configurations used for the same-window crossover, matched nested success spans 14.3–91.1%. Appendix Table 9 separately selects each model's best observed measured operating point, where three models reach 100.0% and the remaining scores span 14.3–98.2%. Separately, six provider-hosted models using native shell tools score 85.7–98.0%. Effort improves matched success for some models but not others, and the same effort label corresponds to different token budgets across models.

To separate generation errors from transport damage, we replay each fixed raw-conditioned reply with and without one added double-quoted parser. This intervention reduces success by 55.4–73.2 points in every same-window configuration. Under the same transport, contract-conditioned generations recover 30.4–60.7 points for six of eight configurations. This realized contrast is computed over the stored generations in the frozen benchmark. Across the observed trial-0 effort rungs, the unconditional nested-replay pass rate varies by at most 5.4 points within each measured ladder. The same mechanism persists on private payloads and across repeated draws.

Concurrent work already shows that changing the harness reorders model leaderboards (Zhang et al., 2026b), but because it swaps the whole scaffold it measures variance without attributing a reversal to any one mechanism. By fixing the model output and changing a single parser, QuoteBench attributes the reorder to the command path and decomposes the matched score into transport damage and contract-conditioned compensation.

This paper makes three contributions:

1. **A final-state benchmark of command-path reliability.** QuoteBench turns recurring quoting and escaping failures into 56 exact-state tasks from 14 operation families. Controlled payload variants and audited validators isolate literal preservation from planning and recovery.
2. **A crossed design that attributes a known hazard.** Generation contract and execution transport are varied independently, and fixed-reply replay decomposes matched scores into transport damage and contract-conditioned compensation. This reveals when aggregate success masks large opposing effects along the command path.
3. **Robustness and a measured fix.** The transport loss persists across effort settings, repeated draws, userlands (GNU versus BSD coreutils environments), and held-out payloads. Two simple fixes, correct escaping and a temporary script, each remove the effect entirely. Both require the caller to control the boundary, and the harness survey (Table 5) records boundaries applied downstream of the stated contract; a matched score alone cannot tell an evaluator whether a fix is needed. Typed operations are an exploratory alternative.

These results motivate path-matched model and effort selection and require system builders to report both the generation contract and execution transport.

2 Related Work

Agent and terminal benchmarks. General agent benchmarks evaluate web navigation, desktop control, coding, and interactive execution in realistic environments (Liu et al., 2024; Zhou et al., 2024; Xie et al., 2024; Yang et al., 2023; Jimenez et al., 2024). Terminal-focused suites extend this line to command-line workflows and environment setup (Merrill et al., 2026; Eliseeva et al., 2025; Chu et al., 2026). Their realism supports end-to-end evaluation, while the contribution of the command interface remains unresolved. QuoteBench isolates that attribution question.

Shell-command generation and robustness. NL2Bash and NLC2CMD formulate natural-language-to-command translation as semantic parsing or competition-style command generation (Lin et al., 2018; Agarwal et al., 2021). NL2SH-ALFA adds manually verified data and execution-based functional-equivalence checks (Westenfelder et al., 2025). Concurrent work introduces BashBench, a 952-task benchmark of syntax, functionality, and robustness for generated Bash programs (Yu et al., 2026). Static shell analysis and long-standing guidance on hostile filenames document the underlying hazards (Holen, 2012; Wheeler, 2010); run on our replies, ShellCheck flags only 34.6% of the nested-only failures (versus 11.4% of the replies that survive nesting) and misses two-thirds, because each command is individually well-formed and the fault is in the downstream interpolation. These works score the generated program. QuoteBench fixes that program and changes the execution transport, separating an incorrect command from a correct command encoded for the wrong channel. Appendix Table 6 tabulates these differences.

Action representations and boundaries. SWE-agent shows that the agent–computer interface can change coding performance, and OctoBench separates task completion from compliance with scaffold constraints (Yang et al., 2024; Ding et al., 2026). Action Boundary Blindness likewise shows that conventional success can hide errors in action granularity, scope, and completion (Wang et al., 2026). Tool-use benchmarks emphasize tool selection and argument construction, while CodeAct and CODESTRUCT change the action language itself (Patil et al., 2024; Qin et al., 2023; Wang et al., 2024; Kim et al., 2026). QuoteBench makes the model-facing contract and downstream transport explicit and crosses them experimentally. A valid structured call guarantees the envelope, but shell correctness still depends on the bytes delivered to the executor.

Evaluation validity and deployment safeguards. Repeated-sampling studies distinguish one successful trajectory from reliable repeated execution (Yao et al., 2024; Brown et al., 2024). UTBoost shows that permissive validators can accept incorrect coding-agent patches (Yu et al., 2025). QuoteBench therefore audits every validator with initial states, oracles, naive probes, and targeted mutations. CARE studies shell-specific pre-execution verification, a complementary safeguard at the command dispatch boundary (Zhang et al., 2026a). A concurrent practitioner report further documents that shell escaping can reverse the cost and reliability tradeoff between flag-based and JSON-based CLIs (Mastykarz, 2026). Closest in prescription, concurrent work argues that harness variance can exceed model variance and that leaderboards should disclose the harness, reporting rank reversals when the whole scaffold is swapped (Zhang et al., 2026b). Because it replaces the harness wholesale, including context handling, retry, and verification, it measures variance but cannot attribute a reversal to a mechanism; QuoteBench fixes the model output and changes a single parser, so it decomposes the matched score into transport damage and contract-conditioned compensation. Related validity audits target reward hacking and protocol gaming rather than the execution channel (Shao et al., 2026). Input-side prompt-format sensitivity establishes that scores move with a nuisance channel variable (Sclar et al., 2024), but because each format is regenerated, the observed spread cannot be separated into what the channel destroyed and what the model produced differently. Fixed-output replay is what makes that separation possible. Together, these results motivate measuring the command channel as part of the executed system.

3 The QuoteBench Benchmark

3.1 Tasks, scope, and validity

QuoteBench contains 56 one-shot Bash tasks: 14 operation families, each with a benign control and three hazardous payload variants. The families cover literal file content, hostile filenames, regular

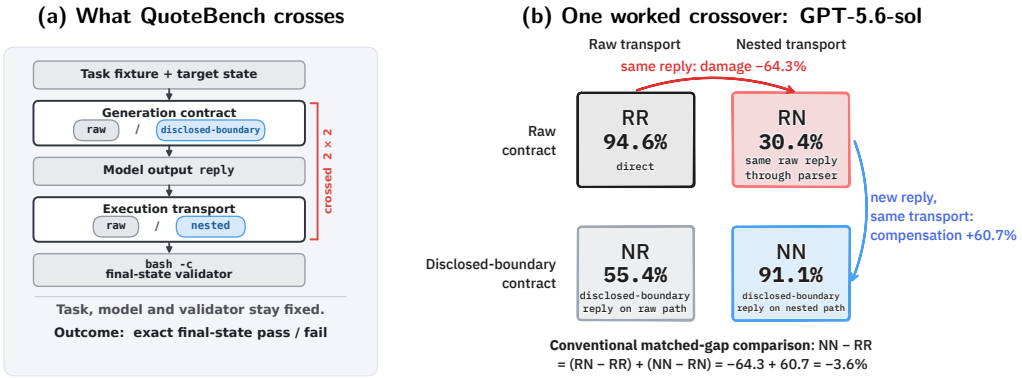


Figure 1: **Crossing generation contract with execution transport.** Task, model configuration, and validator are fixed across cells. The reply is fixed within each transport replay pair. Cell labels are contract-then-transport. On the generation axis, R denotes the raw contract and N the disclosed-boundary contract. On the transport axis, R denotes raw execution and N the nested transport. The RR and NN cells are matched. The RN cell measures fixed-reply damage, and $NN - RN$ is the realized contract-conditioned contrast. Panel (b) works the decomposition for GPT-5.6-sol.

expressions and globbing, heredocs, argument and environment passing, JSON and Git state, and two local simulations of a second shell parser. The hazardous tiers hold the operation fixed while adding quotes, expansion characters, multiline data, leading dashes, or parser-boundary conflicts. The same frozen core is used in every campaign reported here.

Each task provides a fixture, an instruction, and a final-state validator. Fixtures are created without invoking a shell. A model returns one Bash program, which runs in a fresh working directory with a trimmed environment and a 15-second timeout. Validators check exact file bytes, received argument vectors, parsed JSON, directory state, or Git history. They score only the resulting state, so any semantically correct implementation receives credit. Exit codes cannot substitute: across the failing executions, 23.4–47.0% exit zero while leaving the wrong final state (Appendix Table 19), so a benchmark that trusts return codes would silently miss up to half of these failures.

The task families were selected from a pre-release mechanism survey of 86 de-identified incidents in author-owned agent sessions and 412 screened public reports. This evidence supports coverage of repeatedly observed command-construction mechanisms. Prevalence estimation and complete shell coverage require a different study design. QuoteBench focuses on POSIX/Bash command construction. PowerShell, Windows CMD, authentication, network failures, interactive terminal state, and multi-turn recovery remain outside the benchmark. Appendix A reports the survey counts, inclusion rules, and mechanism-to-family mapping.

3.2 Contracts and Transports

Figure 1 previews the crossed design before the result notation: contract selects the stored reply, transport selects how that reply reaches Bash, and final-state validation scores the resulting state.

QuoteBench separates the *generation contract*, which tells the model how to express an action, from the *execution transport*, which determines how that action reaches a shell. It evaluates two observed model-facing contracts and adds one controlled transport intervention:

- **Raw contract:** the model emits one Bash program, executed verbatim as the script argument to `bash -c`.
- **Native contract:** the model fills a provider shell-tool call. QuoteBench extracts its required `command` field and executes that string on the same raw path, isolating the model-facing representation.

- **Disclosed-boundary contract:** the model is told that its reply `R` will be interpolated into `bash -c "R"`. The corresponding **nested transport** then adds that parser. Crossing this contract with raw and nested transports isolates a boundary that can arise downstream in remote, container, or CI commands.

A fixed-commit survey of six public agent systems finds both raw and native model-facing contracts and several downstream transports (Appendix Table 5); Codex, for instance, takes a native command field and then executes it as `shell -c R`, a boundary applied downstream of the stated contract. An extended survey of 24 open systems finds the unescaped second parse deployed in one path, Aider on Windows, and byte rewrites before a single parse in five (Appendix Table 8). We use *nested* as the name of a controlled stress condition: it adds the double-quoted parser boundary found in remote, container, and CI command paths. This boundary is not merely synthetic: replaying each stored raw reply through a real `ssh localhost "R"` remote command reproduces the nested damage to the decimal for seven of eight configurations and within one task for the eighth (Appendix Table 7), so the perturbation stands in for a deployment path a model-authored remote wrapper actually produces. Replaying the disclosed-boundary replies through the same real `ssh` path completes the 2×2. Real-`ssh` compensation matches synthetic nested compensation exactly for five of six replayed configurations; Gemini-3.1-Flash-Lite differs by one task (-1.8 versus -5.4 points; Appendix Table 24). The two non-adapting configurations show no positive compensation on either path. The raw and disclosed-boundary generation contracts differ by one sentence that states this boundary and gives no quoting advice. Appendix A.2 gives the prompts verbatim.

All reported primary executions use a pinned, network-disabled GNU/Linux container. Appendix C.1 repeats the analysis in a BSD/macOS userland. Commands run only in fresh fixtures with timeouts and collateral-file checks.

4 Results

The mechanism analysis rests on eight same-window configurations collected under one randomized schedule with the effort field omitted (Study A). Broader effort ladders, a native-tool campaign replayed in both userlands (Study B), and two private-payload replays extend coverage (Appendix B). Appendix Table 9 collapses the ladders to one best-observed row per base model; model names are provider-public identifiers such as `gpt-5.6-sol` (Appendix Table 13).

4.1 Matched success varies across command paths

At their best observed settings, defined as a within-model maximum over single-trial rungs, three models pass all 56 tasks and the remaining scores range from 14.3% to 98.2% (Appendix Table 9). Complete measured ladders, including lower operating points and Qwen think toggles, appear in Figure 2 and Appendix B.

Measurements use provider-hosted model snapshots identified in Appendix B; the same-window mechanism sweep was queried on 2026-07-31 and the effort ladders earlier in July 2026. Hosted deployments may change under the same identifier, so the query date is part of the result. The frozen 56-task core is versioned as `core-v1`.

Aggregate rank hides distinct failure profiles. Appendix Figure 3 shows that models with similar totals fail on different operation families, while some lower-scoring configurations retain isolated strengths. The heatmap presents the benchmark at the level users encounter in practice: concrete command families.

4.2 Transport damage occurs after correct command generation

To identify the failure mechanism, we vary generation contract and execution transport independently. The task, model configuration, reply, and final-state validator remain fixed for each replay comparison.

Let $G \in \{R, N\}$ denote the generation contract and $T \in \{R, N\}$ the execution transport. On the generation axis, R is the raw contract and N is the disclosed-boundary contract. On the transport axis, R is raw execution and N is the nested transport (Section 3.2). The four cells are RR for a raw reply on raw transport, RN for a raw reply on nested transport, NR for a disclosed-boundary reply

Table 1: All four crossover cells for the eight same-window configurations. Cell notation follows Figure 1: generation contract precedes transport. Cells are pass rates (%). Effects are percentage points. Damage is $RN - RR$, compensation is $NN - RN$, and the matched gap is $NN - RR$. Appendix Table 14 reports the interaction at every measured rung, and Table 15 gives the sensitivity tests.

Model	RR	RN	NR	NN	Damage	Comp.	Matched gap
GPT-5.6-sol	94.6	30.4	55.4	91.1	-64.3	$+60.7$	-3.6
GPT-5.5	100.0	28.6	50.0	89.3	-71.4	$+60.7$	-10.7
Opus-5	96.4	30.4	42.9	89.3	-66.1	$+58.9$	-7.1
Gemini-3.1-Pro	98.2	25.0	33.9	80.4	-73.2	$+55.4$	-17.9
Gemini-3.5-Flash	96.4	28.6	67.9	58.9	-67.9	$+30.4$	-37.5
Opus-4.8	91.1	26.8	62.5	57.1	-64.3	$+30.4$	-33.9
Qwen3.5-27B	85.7	30.4	83.9	30.4	-55.4	0.0	-55.4
Gemini-3.1-Flash-Lite	78.6	19.6	80.4	14.3	-58.9	-5.4	-64.3

on raw transport, and NN for a disclosed-boundary reply on nested transport. For one task, Y_{GT} is the corresponding binary final-state outcome.

Fixed-reply transport damage compares Y_{RN} with Y_{RR}. *Contract-conditioned compensation* compares Y_{NN} with Y_{RN}. It is computed from the two stored generations for each task and describes cancellation in this finite benchmark. The two contrasts sum to the *matched gap* reported by a conventional matched evaluation:

$$Y_{NN} - Y_{RR} = (Y_{RN} - Y_{RR}) + (Y_{NN} - Y_{RN}). \tag{1}$$

Figure 1 defines all four cells and works the decomposition for one configuration. All three quantities are averaged over tasks or operation families and given in percentage points, written *points* in prose and pp in tables.

The replay reuses stored replies and makes no new model calls. For each of the eight same-window configurations, the trial-0 reply for every task is executed through both paths. Effects are averaged over the 14 operation families. Appendix Table 15 reports enumerated family-sign sensitivity analyses with Holm correction. Their scope is the finite, purposively constructed family set. These intervals and p-values quantify variation across the 14 constructed families, not model-call randomness or a sampled task population.

Moving a fixed raw-generated reply from raw to nested transport costs every same-window configuration 55.4–73.2 points. Averaged over three public draws, damage is -54.8 to -73.8 points, and the family-by-draw interval excludes zero for every configuration (Table 2). The loss is not confined to adversarial payloads: the 14 benign control tasks alone lose 28.6–57.1 points, because models emit double-quote-active characters even for ordinary commands. Table 1 gives the signed effects for those fixed configurations. Its diagonal RR and NN cells are not the post-selected settings in Appendix Table 9. All eight effects are negative, and every leave-one-family-out estimate remains negative. Reparsing preserves 123 of the 415 direct-path successes, corresponding to configuration-level retention of 25.0–35.4%. The remaining 292 become failures, and the 33 commands that already fail remain failed. Because each pair reuses the same reply, the added parser accounts for the change in outcome.

The realized generation-by-transport interaction, $(NN - NR) - (RN - RR)$, ranges from -7.1 to $+119.6$ points across the eight configurations (Table 1). The wide range shows that the boundary-aware contract changes command behavior in a transport-specific way. These values characterize the stored generations in this finite benchmark.

At the fixed same-window setting used for this crossover, Gemini-3.5-Flash passes 54/56 tasks in RR but only 8/14 control tasks in NN. These counts describe the fixed mechanism configuration, not the best-observed medium setting in Appendix Table 9.

The damage disappears when the boundary is handled correctly. Escaping the reply at the interpolation point (`bash -c` ⟨quoted input⟩) reproduces the raw-path outcome exactly for all 448 public pairs. Replaying the reply as a temporary script does the same for all 448 public and 126 private-v1 pairs. Neither repair changes raw-path failures: 33 public and 15 private-v1 commands remain failed (Appendix E.1).

Table 2: Three-draw means for the eight same-window configurations (percent of 56 tasks; trial 0 alone is Table 1). Brackets are 95% percentile bootstrap intervals that resample the 14 families with replacement and one draw per task, so they describe the effect for a fresh generation on a benchmark built from the same family process. Resampling draws alone, with families fixed, gives intervals 2–7 points wide (Appendix E.1.6).

Model	RR	RN	NR	NN	Damage	Comp.	Matched gap
GPT-5.6-sol	97.0	31.0	56.0	94.6	−66.1 [-82.1, -48.2]	+63.7 [+48.2, +78.6]	−2.4 [-10.7, +5.4]
GPT-5.5	98.8	29.2	47.6	89.3	−69.6 [-83.9, -53.6]	+60.1 [+44.6, +73.2]	−9.5 [-19.6, -1.8]
Opus-5	97.6	30.4	42.9	91.7	−67.3 [-83.9, -50.0]	+61.3 [+42.9, +78.6]	−6.0 [-14.3, +1.8]
Gemini-3.1-Pro	98.8	25.0	38.1	81.5	−73.8 [-89.3, -57.1]	+56.5 [+33.9, +75.0]	−17.3 [-32.1, -5.4]
Gemini-3.5-Flash	96.4	26.2	70.8	51.2	−70.2 [-83.9, -55.4]	+25.0 [+10.7, +39.3]	−45.2 [-64.3, -25.0]
Opus-4.8	93.5	27.4	67.3	50.6	−66.1 [-80.4, -50.0]	+23.2 [+7.1, +42.9]	−42.9 [-58.9, -26.8]
Qwen3.5-27B	85.1	30.4	83.3	30.4	−54.8 [-69.6, -39.3]	+0.0 [0.0, 0.0]	−54.8 [-69.6, -39.3]
Gemini-3.1-Flash-Lite	79.8	20.2	80.4	13.7	−59.5 [-71.4, -48.2]	−6.5 [-14.3, 0.0]	−66.1 [-76.8, -55.4]

Matched comparisons can obscure cross-path sensitivity. GPT-5.6-sol's matched gap is only -3.6 points, even though fixed-reply transport loses 64.3 points and the realized contract-conditioned contrast restores 60.7. The near-zero matched change is therefore the sum of two large opposing components. The matched NN score accurately describes its declared path, while off-diagonal replay reveals portability when that path changes or adds an undisclosed boundary. Under three draws, the matched gaps of GPT-5.6-sol (-2.4) and Opus-5 (-6.0) lie inside family-by-draw intervals that include zero, while every other configuration's gap is negative beyond its interval (Table 2). Appendix C.2 reports a descriptive cross-configuration threshold analysis.

The deployment configuration reorders models. The RR and NN orderings agree only partially: their Kendall rank correlation is 0.57 (task-cluster bootstrap 95% interval $[0.32, 0.82]$, excluding perfect agreement), and 22 of the 28 pairwise orderings are stable in at least 95% of resamples, so the leaderboard is a bootstrap-supported partial order rather than a fixed ranking. The one reversal that is unambiguous at this resolution is GPT-5.6-sol versus Gemini-3.5-Flash (behind by one task under RR, ahead by eighteen under NN); the count of reversed pairs is itself uncertain (bootstrap mean 4.6, 95% interval $[1, 8]$ of 28 pairs, Appendix E.1.6).

Two further public draws for all eight same-window configurations keep damage negative in every configuration and draw, with per-configuration ranges of 1.8–7.1 points, and change no compensation sign (Appendix E.1.6). On a disjoint private set, both models retain negative transport damage and positive compensation, and three additional draws preserve that sign pattern (Appendix E.1).

4.3 Matched gains can come from contract-conditioned compensation

The $NN - RN$ contrast fixes the nested transport and compares replies generated under two contracts that differ by one disclosure sentence. Six of the eight same-window configurations show 30.4 to 60.7 points of realized compensation, all with family-bootstrap intervals excluding zero (Appendix Table 15). Over three draws, compensation averages $+23.2$ to $+63.7$ points for the six adapting configurations with family-by-draw intervals that exclude zero; Qwen3.5-27B realizes exactly zero in every draw, and Gemini-3.1-Flash-Lite's interval includes zero (Table 2). Qwen3.5-27B shows 0.0 and Gemini-3.1-Flash-Lite -5.4. Similar raw scores can accompany substantially different compensation: Gemini-3.5-Flash and Gemini-3.1-Pro differ by one raw task, yet Pro recovers 25.0 points more. The clause states where the command runs but prescribes no quoting strategy (Appendix A.2).

The compensation is genuine behavioral change, not generic robustness. The same disclosed-boundary replies that recover the nested path lose 28.6–64.3 points when replayed on the raw path (NR versus RR): the six compensating models rewrote their commands for the declared boundary and pay for it where the boundary is absent. The two non-compensating configurations change nothing in either direction (Qwen3.5-27B -1.8, Gemini-3.1-Flash-Lite $+1.8$). Compensation also concentrates where the hazard is explicit. Payload-quoting families such as json-write ($+50.0$) and sed-replace ($+46.9$) recover about half their damage, but implicit hazards remain difficult: find-glob (-12.5), grep-count ($+15.6$), and hostile-filenames ($+18.8$) stay broken even under disclosure.

Six configurations provide raw and disclosed-boundary replies at every effort rung, yielding 26 crossover points from stored generations (Appendix Figure 4 and Appendix Table 14). The nested-

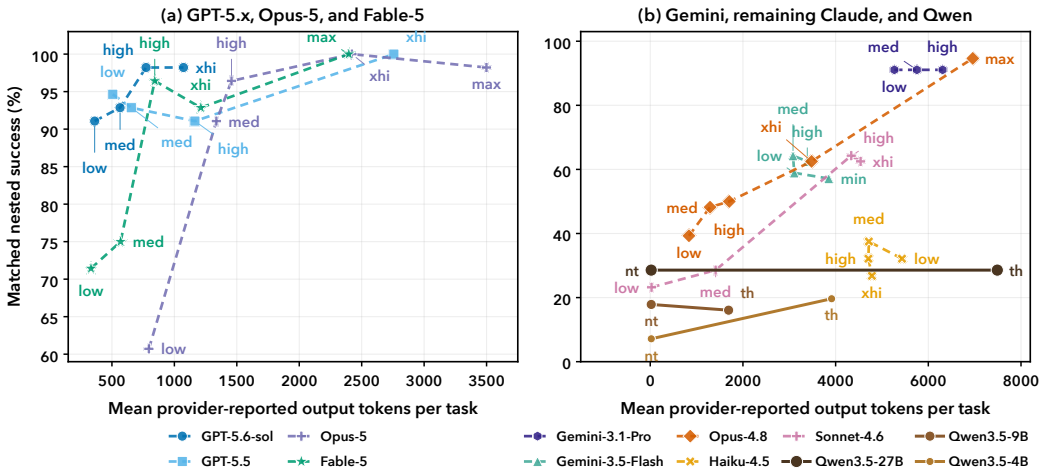


Figure 2: **Effort settings are provider-specific operating points.** Each point shows trial-0 matched-nested success on the 56-task core against mean provider-reported output tokens per task, including reported hidden reasoning. Dashed segments follow the provider's declared order. Backward or dominated segments show measured non-monotonicity. Panels group model families for legibility, and Qwen sizes contribute two-point think-toggle trajectories.

replay pass rate of raw-conditioned replies stays between 23.2% and 33.9%, moves by at most 5.4 points within any one ladder, and accompanies damage of -58.9 to -75.0 points. These trajectories summarize one stored generation at each rung.

Because RN varies little while NN sometimes climbs, the matched gap can narrow without an improved nested-replay pass rate. The pattern varies by model: Gemini-3.1-Pro remains at 91.1% across its three rungs, and Gemini-3.5-Flash moves only 7.1 points.

Each measured effort rung is a deployment-relevant operating point under the frozen benchmark, not an estimate of effort's causal effect. Figure 2 shows that labels map to different token budgets across models and that several ladders are non-monotonic.

4.4 The mechanism transfers across payloads and sampled replies

The mechanism is not a property of the published payloads. The private-v2 crossover repeats the design on 42 unpublished hostile payloads: raw- and disclosed-boundary-contract calls for GPT-5.6-sol and Opus-4.8 were interleaved within one serving window, and each stored reply was replayed through both transports with the public final-state validators.

Both models pass 92.9% of the private tasks on the direct raw path, then lose 73.8 and 76.2 points when the same replies cross the added parser. Compensation remains model dependent: GPT-5.6-sol recovers 78.6 points under the boundary-aware contract, whereas Opus-4.8 recovers 26.2. Every leave-one-family-out slice preserves negative transport damage.

A separate private-v1 replay tests a practical bypass on 42 tasks and three models, executing each stored raw-contract reply through the raw path, the nested wrapper, and a temporary Bash script.

Executing the replies from temporary scripts reproduces the raw-path outcome for every model–task pair. It recovers 87 commands that fail only under the wrapper. The 15 commands that fail directly remain unresolved. Appendix E reports the private design and validator checks, and Appendix E.1 reports the repeated draws, script-bypass records, and typed-operation pilot.

5 Discussion

The results support two reporting practices for command paths that wrap or reparse model output. First, report the generation contract together with the execution path. The crossed design shows that

the generation contract can change the matched score, while off-diagonal replay reveals whether replies remain portable across paths. Second, structured actions remove one quoting boundary while leaving payload-level representation errors possible. In the typed pilot, eleven of 36 programs fail and ten leave the wrong final state, most often because the model copies instruction delimiters into the payload (Appendix E.1).

The command interface is part of the evaluated system, not neutral plumbing. Vendors should report the generation contract, deployed execution path, selected operating point, and family-level failures. Users should compare models and effort settings on that path because provider ladders are non-monotonic and defaults map to different operating points.

Ignoring the path changes which model wins: selecting by raw success picks GPT-5.5 (56/56 raw), which reaches 50/56 on the nested path, whereas the path-aware pick reaches 51/56. The regret is small at the saturated frontier, but the top rank reverses. The scorecard and crossover answer different questions. Appendix Table 9 supports operating-point selection, while Table 1 diagnoses path sensitivity for fixed configurations. Neither is a controlled compute ranking. Deployment reports should publish both the selected point and the measured ladder.

The decomposition extends beyond shell. Any pipeline that transforms generated output before execution defines the same four cells. A JSON tool-call boundary is a second instance. Replaying each stored raw reply through a naive JSON string embedding causes losses from 51.8 to 66.1 points. The serializer re-parses the same double quotes and backslashes as the shell, while a correct round-trip serializer costs exactly zero (Appendix E.1.5). The damage comes from the unescaped transform, and the decomposition transfers to a non-shell boundary.

Where a deployed path adds an interpolating shell layer, the first-line fix is harness-side: escaping the reply at the interpolation point restores every raw-path success in our replays. Where the boundary is not under the caller's control, as in remote or CI patterns, a temporary script preserves the program boundary at the cost of a file lifecycle, and boundary disclosure lets capable models compensate (Appendix E.1).

6 Conclusion

QuoteBench shows that command success depends on the execution path and that a matched score can hide that dependency. Replaying fixed replies through one added parser lowers success by 55.4–73.2 points across all same-window configurations, while contract-conditioned generation recovers 30.4–60.7 points for six configurations. The two effects cancel only at the frontier; elsewhere the matched score falls with the path. Trial-0 ladders shift nested-replay pass rates by at most 5.4 points.

Limitations

QuoteBench isolates one mechanism: one-shot Bash generation under quotation and interpolation hazards. Its 14 constructed families support mechanism attribution, and the nested transport reproduces a real `ssh` remote-execution boundary (Table 7); the results describe this benchmark and stored-reply portability, and they leave deployment prevalence unmeasured: the surveys count code paths at fixed commits, not usage. Causal claims rest on fixed replies in the eight same-window configurations. Effort-ladder rungs use a single stored generation per task, and effort labels do not map to comparable compute budgets. The native-tool campaign is observational. Held-out payloads test transfer to unseen literals without difficulty matching, the typed-operation study covers six naturally typeable families, and other shells and multi-turn recovery remain open.

An agent that sees the failure typically writes a temporary script instead of re-quoting the command; QuoteBench measures that bypass (Appendix Table 21): it restores the raw-path successes at the cost of one extra file and tool call, and the one-shot numbers are the price paid before that switch. The validator reports pass or fail plus a failure class (Appendix Table 19); collateral file changes and privilege effects are unscored.

AI USE STATEMENT

AI assistants supported manuscript editing, data analysis and the associated code, and literature checks. The authors designed the benchmark and experiments, ran and verified every measurement, checked every citation against its source, and take responsibility for all claims.

ETHICS STATEMENT

QuoteBench executes untrusted model output, so the released harness runs each attempt in a fresh fixture inside a timeout-bounded, network-disabled container, and incident evidence is released only as de-identified mechanism classifications: no session text, prompts, file contents, paths, or repository identifiers from those incidents are released. Publishing the frozen core creates a contamination risk; we treat it as a versioned audit set and hold out regenerated private variants. Each released task file also embeds a fixed canary GUID, recorded in the repository, so downstream contamination checks have a known token to search for. The benchmark introduces no shell capability beyond routine coding-agent operations: the hostile payloads exercise documented hazard classes such as word splitting, glob expansion, and option injection that public guidance already catalogs (Holen, 2012; Wheeler, 2010), add no exploitation technique, and run only inside the network-disabled container.

REPRODUCIBILITY STATEMENT

The 56 tasks, validators, verbatim contract prompts, replay harness, analysis scripts, and the complete rollout archive with a SHA-256 manifest are provided in https://anonymous.4open.science/r/quoteBench-31F6. Appendix F.2 lists the commands that rebuild every table and figure from the archive without model access, and Appendix F.1 reports the validator audit.

REFERENCES

Mayank Agarwal, Tathagata Chakraborti, Quchen Fu, David Gros, Xi Victoria Lin, Jaron Maene, Kartik Talamadupula, Zhongwei Teng, and Jules White. NeurIPS 2020 NLC2CMD competition: Translating natural language to bash commands. *arXiv preprint arXiv:2103.02523*, 2021. URL https://arxiv.org/abs/2103.02523.

Aider. aider source: aider/run_cmd.py, commit 5dc9490. https://github.com/Aider-AI/aider, 2026. Inspected 2026-09-09 at the default-branch head.

Anthropic Claude Code community. Avoid shell quoting issues when passing markdown to cli tools like gh. GitHub issue anthropics/claude-code#29619, 2026a. URL https://github.com/anthropics/claude-code/issues/29619. Accessed 2026-07-29.

Anthropic Claude Code community. Claude repeatedly fails heredoc/string escaping when writing files to remote servers, causing multi-attempt delays. GitHub issue anthropics/claude-code#48317, 2026b. URL https://github.com/anthropics/claude-code/issues/48317. Accessed 2026-07-29.

AutoGPT. AutoGPT source: classic/forge/forge/components/code_executor/code_executor.py, commit 98381ab. https://github.com/Significant-Gravitas/AutoGPT, 2026. Inspected 2026-09-09 at the default-branch head.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024. URL https://arxiv.org/abs/2407.21787.

Zhaoyang Chu, Jiarui Hu, Xingyu Jiang, Pengyu Zou, Han Li, Chao Peng, Peter O'Hearn, Earl T. Barr, Mark Harman, Federica Sarro, and He Ye. TerminalWorld: Benchmarking agents on real-world terminal tasks. *arXiv preprint arXiv:2605.22535*, 2026. URL https://arxiv.org/abs/2605.22535.

Cline. cline source: apps/vscode/src/hosts/vscode/terminal/vscodeterminalprocess.ts, commit 1063db4. https://github.com/cline/cline, 2026. Inspected 2026-09-09 at the default-branch head.

Desktop Commander. DesktopCommanderMCP source: src/terminal-manager.ts, commit a781f5a. https://github.com/wonderwhy-er/DesktopCommanderMCP, 2026. Inspected 2026-09-09 at the default-branch head.

Deming Ding, Shichun Liu, Enhui Yang, Jiahang Lin, Ziying Chen, Shihan Dou, Honglin Guo, Weiyu Cheng, Pengyu Zhao, Chengjun Xiao, Qunhong Zeng, Qi Zhang, Xuanjing Huang, Qidi Xu, and Tao Gui. OctoBench: Benchmarking scaffold-aware instruction following in repository-grounded agentic coding. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5958–5978, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.269. URL https://aclanthology.org/2026.acl-long.269/.

E2B. infra source: envd process handler.go, commit 1875237. https://github.com/e2b-dev/infra, 2026. Inspected 2026-09-09 at the default-branch head.

Aleksandra Eliseeva, Alexander Kovrigin, Ilia Kholkin, Egor Bogomolov, and Yaroslav Zharov. EnvBench: A benchmark for automated environment setup. *arXiv preprint arXiv:2503.14443*, 2025. URL https://arxiv.org/abs/2503.14443.

Free Software Foundation. *Bash Reference Manual, Version 5.3*, 2025. URL https://www.gnu.org/software/bash/manual/bash.html.

Gemini CLI. gemini-cli source: packages/core/src/services/shellexecutionservice.ts, commit ed2ac40. https://github.com/google-gemini/gemini-cli, 2026. Inspected 2026-09-09 at the default-branch head.

Google Gemini CLI community. Running commands in cmd.exe shell has escaping issues. GitHub issue google-gemini/gemini-cli#1839, 2025. URL https://github.com/google-gemini/gemini-cli/issues/1839. Accessed 2026-07-29.

Goose. goose source: crates/goose/src/agents/platform_extensions/developer/shell.rs, commit bded379. https://github.com/block/goose, 2026. Inspected 2026-09-09 at the default-branch head.

Vidar Holen. ShellCheck: A static analysis tool for shell scripts, 2012. https://www.shellcheck.net.

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? *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Myeongsoo Kim, Chao-Chun Hsu, Dingmin Wang, Shweta Garg, Varun Kumar, and Murali Krishna Ramanathan. CODESTRUCT: Code agents over structured action spaces. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13290–13306, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.607. URL https://aclanthology.org/2026.acl-long.607/.

LangChain contributors. Langchain shell tool source at commit b3a6d9a. GitHub source repository, 2026. URL https://github.com/langchain-ai/langchain/tree/b3a6d9a012681df8a8e33345c8255ca69ec0e437. Accessed 2026-07-29.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of LREC*, 2018. URL https://arxiv.org/abs/1802.08979. arXiv:1802.08979.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2308.03688. arXiv:2308.03688.

Waldek Mastykarz. Don't rewrite your CLI for agents. Microsoft for Developers, July 2026. https://developer.microsoft.com/blog/dont-rewrite-your-cli-for-agents.

mcp-server-kubernetes. mcp-server-kubernetes source: src/tools/exec_in_pod.ts, commit daf01de. https://github.com/Flux159/mcp-server-kubernetes, 2026. Inspected 2026-09-09 at the default-branch head.

mcp-ssh. mcp-ssh source: src/ssh-client.ts, commit 9c082ac. https://github.com/AiondaDotCom/mcp-ssh, 2026. Inspected 2026-09-09 at the default-branch head.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026. URL https://arxiv.org/abs/2601.11868. The released benchmark is Terminal-Bench 2.0.

MetaGPT. MetaGPT source: metagpt/tools/libs/terminal.py, commit 11cdf46. https://github.com/FoundationAgents/MetaGPT, 2026. Inspected 2026-09-09 at the default-branch head.

Microsoft AutoGen contributors. Autogen docker code executor at commit 027ecf0. GitHub source repository, 2026. URL https://github.com/microsoft/autogen/tree/027ecf0a379bcc1d09956d46d12d44a3ad9cee14. Accessed 2026-07-29.

mini-swe-agent. mini-swe-agent source: src/minisweagent/environments/docker.py, commit 04d809c. https://github.com/SWE-agent/mini-swe-agent, 2026. Inspected 2026-09-09 at the default-branch head.

OpenAI. Codex shell execution source at commit fa1d4c4. GitHub source repository, 2026. URL https://github.com/openai/codex/tree/fa1d4c40d0e63eef2e0ba8a9e004ccd0a80b77f5. Accessed 2026-07-29.

OpenAI Codex community. Tool-contract ambiguity: exec-command cmd lets models over-quote shell operators. GitHub issue openai/codex#20875, 2026. URL https://github.com/openai/codex/issues/20875. Accessed 2026-07-29.

OpenHands community. Use language server protocol to re-implement code editing. GitHub issue OpenHands/OpenHands#1934, 2024. URL https://github.com/OpenHands/OpenHands/issues/1934. Motivated partly by weird issues from heredoc-plus-Bash editing; accessed 2026-07-29.

OpenHands contributors. Openhands argv command parser at commit 850bd64. GitHub source repository, 2026. URL https://github.com/All-Hands-AI/OpenHands/tree/850bd647b64c9a6b5d2bbf25d4d9e16a3c6f685e. Accessed 2026-07-29.

OpenHands SDK. software-agent-sdk source: openhands-tools terminal and utils/command.py, commit 3fc7b22. https://github.com/OpenHands/software-agent-sdk, 2026. Inspected 2026-09-09 at the default-branch head.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4020. URL

https://proceedings.neurips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.

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 Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.11324.

Jiaqi Shao, Hanck Chen, Wei Zhang, Maxm Pan, and Bing Luo. Do agent benchmarks measure capability? protocol validity in the age of agentic AI. *arXiv preprint arXiv:2607.22368*, 2026. URL https://arxiv.org/abs/2607.22368.

ssh-mcp. ssh-mcp source: src/tools/command-tools.ts, commit d2d7696. https://github.com/tufantunc/ssh-mcp, 2026. Inspected 2026-09-09 at the default-branch head.

SWE-agent contributors. Swe-agent action execution source at commit 3ea751c. GitHub source repository, 2026. URL https://github.com/princeton-nlp/SWE-agent/tree/3ea751c087f32b16e039a2233dd6eefecef325d5. Accessed 2026-07-29.

SWE-ReX. SWE-ReX source: src/swerex/runtime/local.py, commit 5c995c3. https://github.com/SWE-agent/SWE-ReX, 2026. Inspected 2026-09-09 at the default-branch head.

Terminal-Bench contributors. Terminal-bench tmux execution source at commit d28711d. GitHub source repository, 2026. URL https://github.com/laude-institute/terminal-bench/tree/d28711d0da2675d0bb1d56de45ae5df6082438a3. Accessed 2026-07-29.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232. PMLR, 2024. URL https://proceedings.mlr.press/v235/wang24h.html.

Zhangyi Wang, Bingnan Yu, Jiexiang Xu, and Zongze Li. Action boundary blindness: When LLM agents cannot tell where one action ends and another begins. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36883–36899, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1711. URL https://aclanthology.org/2026.acl-long.1711/.

Warp community. Agent mode fails to create files with heredoc syntax—quote escaping issues. GitHub issue warpdotdev/Warp#7735, 2025. URL https://github.com/warpdotdev/Warp/issues/7735. Accessed 2026-07-29.

Finnian Westenfelder, Erik Hemberg, Stephen Moskal, Una-May O'Reilly, and Silviu Chiricescu. LLM-supported natural language to bash translation. In *Proceedings of NAACL*, pp. 11135–11147, 2025. doi: 10.18653/v1/2025.naacl-long.555. URL https://aclanthology.org/2025.naacl-long.555/.

David A. Wheeler. Fixing Unix/Linux/POSIX filenames, 2010. https://dwheeler.com/essays/fixing-unix-linux-filenames.html.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in*

Neural Information Processing Systems, volume 37, 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-1035. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/4b175d846fb008d540d233c188379ff9-Abstract-Datasets_and_Benchmarks.html.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. UTBoost: Rigorous evaluation of coding agents on SWE-bench. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3762–3774, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.189. URL https://aclanthology.org/2025.acl-long.189/.

Lei Yu, Peng Wang, Jia Xu, Jingyuan Zhang, Xin Wang, Jiajia Ma, Li Yang, Changzhi Deng, Zenghua Wang, and Fengjun Zhang. BashCoder-R1: Towards robust and explainable bash code generation with robustness-aware group relative policy optimization. *arXiv preprint arXiv:2606.27733*, 2026. URL https://arxiv.org/abs/2606.27733.

Wenxiao Zhang, Yu Liu, Zhiwei Yang, Zhongyi Zhang, Hanqi Feng, Xinyu Wang, Peng Qiu, Yanbing Liu, Barnabas Poczos, and Jin B. Hong. CARE: Pre-execution command verification for shell-executing LLM agents. *arXiv preprint arXiv:2607.21642*, 2026a. URL https://arxiv.org/abs/2607.21642.

Yunbei Zhang, Janet Wang, Yingqiang Ge, Weijie Xu, Jihun Hamm, and Chandan K. Reddy. Stop comparing LLM agents without disclosing the harness. *arXiv preprint arXiv:2605.23950*, 2026b. URL https://arxiv.org/abs/2605.23950.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2307.13854. arXiv:2307.13854.

A Benchmark Construction, Coverage, and Contracts

Table 3: One worked QuoteBench task (write-file, tier 3). The ⟨⟨·⟩⟩ markers delimit the exact literal payload and are not part of it. The ↩ symbol marks a newline that the payload itself contains. The validator compares exact final bytes, so any command that reaches the target state passes. Appendix Table 4 maps the full family set to the covered mechanisms. The artifact contains all 56 instructions, fixtures, oracles, and validators.

Instruction	Create a file named `out.txt` whose content is exactly the text ⟨⟨`back\slash and` `` `date` `` `here`↩ `'single' "double" $(pwd) 100%`⟩⟩ followed by a single trailing newline.
Hazards	backslash, backtick, command substitution, multiline, mixed quotes
Naive command	`echo "back\slash and` `` `date` `` `here ..." > out.txt`; the backticks execute `date`, `$(pwd)` expands locally, and the payload's inner double quote terminates the string early
Final-state gate	exact bytes of `out.txt`, including one trailing newline, with no collateral files
One passing command	a single-quoted `printf` with an embedded-quote splice (the machine oracle). Any other correct encoding receives equal credit

Two mechanism surveys guided the family design. The internal survey contains 86 de-identified incidents from author-owned coding-agent sessions: 50 Codex incidents and 36 Claude incidents. The public survey screened 412 candidates, read 34 in full, retained 17 model-level POSIX/Bash command-construction incidents, classified 10 harness failures separately, and excluded seven cases outside scope. Every retained model-level incident maps to a mechanism represented in the 14-family core. The surveys document mechanism coverage. Prevalence and complete shell coverage remain outside their purpose. Representative retained reports span Claude Code, Codex, Gemini CLI, Warp, and OpenHands (Anthropic Claude Code community, 2026b;a; OpenAI Codex community, 2026; Google Gemini CLI community, 2025; Warp community, 2025; OpenHands community, 2024).

The released survey record includes the tracker query, inclusion decision, and mechanism code for every candidate. We retain a report as model-level when the model constructs the POSIX command, the failure concerns literal or argument semantics, execution permits final-state scoring, and command-level evidence identifies the mechanism. Product-side rewrites are classified separately as harness regressions. Five of the seventeen retained incidents target a second parser, such as an SSH remote or inner `shell -c`, matching the nested boundary of §3.2.

Table 4: Mechanism groups used to construct the frozen core; the mapping documents coverage of the surveyed failure mechanisms. Benchmark analyses weight the 14 operation families equally.

Mechanism group	Representative failures	QuoteBench families
Literal quote and expansion	Apostrophes, double quotes, dollars, backticks, multiline payloads	write-file, JSON writing, Git commit, environment passing, heredoc writing
Word splitting and path semantics	Spaces, globs, leading dashes, hostile filenames, argument boundaries	argv passing, hostile filenames, find/glob, bulk rename
Embedded-language escaping	Regex versus literal matching, sed replacement, AWK string processing	grep count, sed replace, field lookup, JSON writing
Second parser or remote-like expansion	Local expansion before a second shell, argument joining, heredoc transport	SSH-like nested execution, SSH-like heredoc
Command-boundary representation	Command string, shell stdin, temporary file, argv, provider tool schema	raw/nested crossover, native-tool study, script bypass, typed pilot

A.1 Surveyed command boundaries

The main text uses only the distinction needed for the intervention. Table 5 records the implementation evidence behind that classification. Contract denotes what the system asks the model to produce. Observed boundary denotes what the harness subsequently does with the reply.

The nested transport is a synthetic stand-in for a real remote-execution boundary. Table 7 confirms it behaves like one: each stored raw reply, replayed through an actual `ssh localhost "R"` command in a loopback-`sshd` container (a zero-call execution, the frozen runner image plus a local `sshd`), loses the same points it loses under the synthetic nested transport.

Table 5: Command boundaries in six public agent systems, inspected at fixed commits. *Contract* is what the system asks the model to produce. A nested boundary arises on a separate axis from what the command targets. *Observed boundary* is what the harness then does with the reply. The classification concerns the parser boundary only. Sources: OpenAI (2026); SWE-agent contributors (2026); LangChain contributors (2026); Terminal-Bench contributors (2026); OpenHands contributors (2026); Microsoft AutoGen contributors (2026).

System	Contract	Source anchor	Observed boundary
Codex	native	`core/src/shell.rs:20-30`, commit `fa1d4c4`	command string becomes `shell -c/-lc R`
SWE-agent	raw	`agents.py:936-967` and `swe_env.py:197-222`, commit `3ea751c`	agent action enters a persistent Bash session
LangChain	native	`shell_tool.py:217-232,491-515`, commit `b3a6d9a`	structured `command: string` is written to shell stdin
Terminal-Bench	raw	`tmux_session.py:26-33,75-173`, commit `d28711d`	command/key strings enter an interactive shell through tmux
OpenHands	native	`acp-command.ts:11-45,65-147`, commit `850bd64`	human-readable command is tokenized to argv, and spawn has no shell
AutoGen	native	`_docker_code_executor.py:327-363`, commit `027ecf0`	generated code is written to a temporary file and invoked by argv

The released tasks are generic programmatic distillations carrying no private commands and no personal data. The public core excludes PowerShell, Windows CMD, interactive terminal state, real SSH networking and authentication, and complete multi-turn recovery. Generated variants and future shell tracks are versioned separately so the 56-task core remains auditable.

A.1.1 Extended survey of open agent systems

A second survey, frozen on 2026-09-09, inspects the execution layer of 24 open agent systems and sandboxes at their default-branch heads and classifies 37 model-command paths by what happens to the reply's bytes before a process runs them (Table 8; the full table with pasted code lives in the released repository). *A* passes the reply to one parser: an argv element, a session's stdin, or terminal keystrokes. *B* interpolates the reply into a larger command string that a second parser reads, raw or with quoting. *C* writes the reply to a file and executes it by path. *D* could not be determined from public code. Thirty paths are A, one is B raw (Aider on Windows, where `cmd.exe` parses the composed command before PowerShell does), one is B quoted (a privileged ssh path), one is C, three are D, and one has no shell tool. Five A paths rewrite the reply before the single parse: escape doubling, tokenizer round trips, wrapper stripping, and an `ssh` flag rewrite. The *Disclosure* column records whether the model-facing tool description names the transform; four systems do, most say "shell command" and stop, and AutoGPT describes a shell while running none. The counts are of code paths at fixed commits, not of usage.

A.2 Generation contracts and command boundaries

The task instruction and the common shell rules are byte-identical across all Study-A source arms. The system prompts differ only in the execution-contract clause. Each contract differs from the next by a single clause, and that clause states the execution environment rather than explaining how to quote for it.

The raw clause is:

```
Reply with exactly one bash command and nothing else ---
no explanation, no markdown fences.  Your reply is passed
verbatim as the single argument to bash -c and executed in
the task's working directory.
```

The single-clause disclosed-boundary contract used for every matched nested result in this paper is:

```
Reply with exactly one bash command and nothing else ---
no explanation, no markdown fences.  Your reply R is not
executed directly:  it is interpolated inside double quotes
into an outer command, producing the string bash -c "R", and
that string is executed in the task's working directory.
```

Table 6: Benchmarks closest to QuoteBench, as each describes itself. *Interface varied*: the same task is evaluated under more than one execution interface; *regenerated* means the model produces a fresh output per interface. *Replay*: identical model output is re-executed under a changed interface. *Quoting*: quoting and escaping hazards are the stated target.

Benchmark	Unit scored	Correctness signal	Interface varied	Replay	Quoting
NL2Bash (Lin et al., 2018)	NL to one-line Bash, 9,305 pairs	structure and full-command accuracy against references	no	no	no
NLC2CMD (Agarwal et al., 2021)	NL to Bash	utility and flag match, arguments ignored	no	no	no
InterCode-Bash (Yang et al., 2023)	interactive Bash, 200 tasks	execution: output similarity, file-system diff, file hashes against a gold command	no	no	no
NL2SH-ALFA (Westenfelder et al., 2025)	NL to Bash, 600 test pairs	execution plus LLM-judged functional equivalence	no	no	no
BashBench (Yu et al., 2026)	Bash commands and scripts, 952	execution test suite, `bash -n`, ShellCheck	no	no	no
Terminal-Bench (Merrill et al., 2026)	agent terminal tasks	per-task test script	no	no	no
TerminalWorld (Chu et al., 2026)	agent terminal tasks, 1,530	test assertions on final state	no	no	no
EnvBench (Eliseeva et al., 2025)	environment setup, 994 repositories	static import or compile check	no	no	no
SWE-agent (Yang et al., 2024)	SWE-bench issues	hidden unit tests	ACI versus shell, regenerated	no	no
OctoBench (Ding et al., 2026)	repository-grounded coding, 217 tasks	objective checklist items	three scaffolds, regenerated	no	no
QuoteBench	one-shot Bash, 56 tasks	exact final state, audited validators	contract × transport, crossed; real `ssh`	yes	yes

Table 7: Real-`ssh` grounding. Each stored raw reply is replayed through `bash -c` and through a real `ssh localhost "R"` remote command; *ssh damage* is the second minus the first. It matches the synthetic nested damage to the decimal for seven of eight configurations; the disclosed-reply cells are grounded in Appendix Table 24. Zero model calls.

Configuration	Raw	`ssh`	ssh damage	Nested damage
GPT-5.6-sol	94.6	30.4	−64.3	−64.3
GPT-5.5	100.0	28.6	−71.4	−71.4
Opus-5	96.4	30.4	−66.1	−66.1
Gemini-3.1-Pro	98.2	26.8	−71.4	−73.2
Gemini-3.5-Flash	96.4	28.6	−67.9	−67.9
Opus-4.8	91.1	26.8	−64.3	−64.3
Qwen3.5-27B	85.7	30.4	−55.4	−55.4
Gemini-3.1-Flash-Lite	78.6	19.6	−58.9	−58.9

A character-level diff of the two full system prompts shows that the sentence beginning “Your reply” is the only text that changes. Both arms then continue with the same shared rules. The reply may use pipes, `&&`, `;`, or quoted embedded newlines, but it must remain one `bash -c` payload. Interactive

Table 8: Extended command-boundary survey (2026-09-09). *Class*: A single parse, B second parse, C file transport; *rewrite* marks A paths that change the reply's bytes first. *Disclosure*: whether the tool description tells the model about the transform.

System	Path	Source	Construction	Class	Discl.
Aider (Aider, 2026)	Windows, PowerShell parent	`run_cmd.py: 51-54,62-68`	reply interpolated into `powershell -Command R`, run with `shell=True`: `cmd.exe` parses, then PowerShell, no escaping	B raw	no
ssh-mcp (ssh-mcp, 2026)	`privileged-command`	`command-tools.ts: 102`	`sudo -S sh -c '<R>'` sent over ssh with single-quote escaping; remote login shell, then `sh -c`	B quoted	partial
OpenHands SDK (OpenHands SDK, 2026)	TerminalTool, tmux and PTY	`tmux_terminal.py: 177-179` `utils/command.py: 71-111`	reply typed into a bash session after a tree-sitter pass doubles backslash-escaped `; & | < >` outside quotes	A rewrite	partial
SWE-ReX (SWE-ReX, 2026)	BashSession (local, Docker, Modal)	`runtime/local.py: 159-165,297-306`	reply split with bashlex and rejoined with `;`, then sent to a persistent bash	A rewrite	n/a
MetaGPT (MetaGPT, 2026)	`Terminal.run_command`	`terminal.py: 49-57,86-96`	regex split on `&&`, each part to the stdin of a persistent bash	A rewrite	partial
Gemini CLI (Gemini CLI, 2026)	`run_shell_command`	`shellExecutionService.ts: 455-462`	argv `[bash, -c, R]`, `shell:false`; a model-written `bash -c` wrapper is stripped and re-tokenized; `$()` and backticks are rejected	A rewrite	yes
Desktop Commander (Desktop Commander, 2026)	`start_process`	`terminal-manager.ts:219-233,256`	`spawn(R)` with `shell:true`; a leading `ssh` is rewritten to `ssh -t`	A rewrite	partial
AutoGPT classic (AutoGPT, 2026)	`execute_shell`	`code_executor.py: 300-307,368-372`	`shlex.split(R)` with no shell by default; the tool description says "Execute a Shell Command"	A tokenizer	no
Goose (Goose, 2026)	`shell`, Unix	`developer/shell.rs:90-92,715-740`	argv `[-c, R]` to the user's shell	A	yes
mini-swe-agent (mini-swe-agent, 2026)	Docker environment	`environments/docker.py: 38,107-124`	argv `[..., bash, -lc, R]`	A	partial
E2B envd (E2B, 2026)	`commands.run`	`handler.go: 201-203`	`sh -c '..' - R`: the reply is an argument, parsed once by the sandbox shell	A	n/a
mcp-ssh (mcp-ssh, 2026)	`runRemoteCommand`	`ssh-client.ts: 278-299`	`spawn(ssh, [.., host, R])` with `shell:false`; the remote login shell parses once, its dialect undisclosed	A	partial
Cline (Cline, 2026)	VS Code terminal	`VscodeTerminalProcess.ts: 96-110,491`	VS Code shell integration `executeCommand` or `sendText` into the user's terminal shell	A	not located
kubernetes MCP (mcp-server-kubernetes, 2026)	`exec_in_pod`	`exec_in_pod.ts: 105-123`	`execFile(kubectl, [.., -, R])`: no shell unless the model itself passes `sh -c`	A no-shell	yes

programs and helper script files are excluded, and text between the $\langle\langle\cdot\rangle\rangle$ markers is exact literal text. Compensation therefore measures the total effect of stating the parsing environment, with no format constraint and no escaping instruction attached.

Other evaluated contracts. The typed-operation pilot must state the interface semantics, but its request to preserve every literal character, including trailing newlines, provides mild coaching. The pilot is therefore exploratory. The native contract also mentions a Bash script payload, a difference included in Study B's declared total-effect estimand. The marker note and shared rules are common to all arms.

Appendix F.2 lists the packaged evidence and includes the exact contract string literals used by the harness, allowing direct verification against the source.

B Model Configurations and Effort Details

Table 9: Best-observed QuoteBench scorecard. Each base model contributes the measured setting with the highest matched-nested All-56 score. Ties prefer `default`, then lower mean provider-reported output tokens. `default` means that the request omitted the effort field. The number and names of measured settings differ by provider and appear in Figure 2 and Appendix B. This table is descriptive: each cell is one stored trial-0 generation and the selection is a within-model maximum over rungs; Appendix E.1.6 reports draw-to-draw spread. The Qwen rows expose only a think toggle rather than an effort ladder, so their best-observed setting is taken from the same-window sweep; for Qwen3.5-27B, the only Qwen size with a ladder row in Table 12, the resulting one-task difference is serving-window drift. Fixed same-window configurations support the mechanism analysis in Table 1.

Model	Best observed setting	Control	Hostile	All 56 (%)	Hostile LOFO range (%)
GPT-5.5	xhigh	14/14	**42/42**	**100.0**	[100.0, 100.0]
Opus-5	xhigh	14/14	**42/42**	**100.0**	[100.0, 100.0]
Fable-5	max	14/14	**42/42**	**100.0**	[100.0, 100.0]
GPT-5.6-sol	high	14/14	41/42	98.2	[97.4, 100.0]
Opus-4.8	max	12/14	41/42	94.6	[97.4, 100.0]
Gemini-3.1-Pro	low	14/14	37/42	91.1	[87.2, 89.7]
Sonnet-4.6	high	9/14	27/42	64.3	[61.5, 69.2]
Gemini-3.5-Flash	medium	10/14	26/42	64.3	[59.0, 66.7]
Haiku-4.5	medium	8/14	13/42	37.5	[28.2, 33.3]
Qwen3.5-27B	non-think	8/14	9/42	30.4	[15.4, 23.1]
Qwen3.5-4B	think	5/14	7/42	21.4	[10.3, 17.9]
Qwen3.5-9B	non-think	5/14	5/42	17.9	[5.1, 12.8]
Gemini-3.1-Flash-Lite	default	6/14	2/42	14.3	[2.6, 5.1]

Table 9 partitions matched-nested success into 14 benign Control tasks and 42 Hostile tasks. The Hostile LOFO column is a family-jackknife range (minimum–maximum hostile success over the 14 leave-one-family-out slices), not a confidence interval. Raw generation itself is close to saturated at the frontier: the six frontier configurations pass 91.1–100% of tasks on the direct path, so raw scores carry almost no discriminative signal. The entire signal lives on the nested side, which is also the precondition for masking. What still separates models is how they handle the command path. Matched nested scores range from 14.3 to 91.1 at the fixed configurations, and realized compensation ranges from -5.4 to $+60.7$; the next two subsections isolate these effects. The complete ladders remain visible in Figure 2, and all mechanism estimates use the fixed same-window configurations in Table 1.

Controls help separate basic operation competence from literal preservation. At its best-observed medium setting, Gemini-3.5-Flash passes 10/14 controls and 26/42 hostile tasks. At the lower end, Gemini-3.1-Flash-Lite passes only two of 42 hostile payloads. Rows are ordered by best-observed matched-nested score. The three perfect rows remain perfect on every hostile leave-one-family-out slice. GPT-5.6-sol and Opus-4.8 each miss one hostile task at their selected settings. Lower-scoring models retain distinct family profiles rather than a single shared failure order.

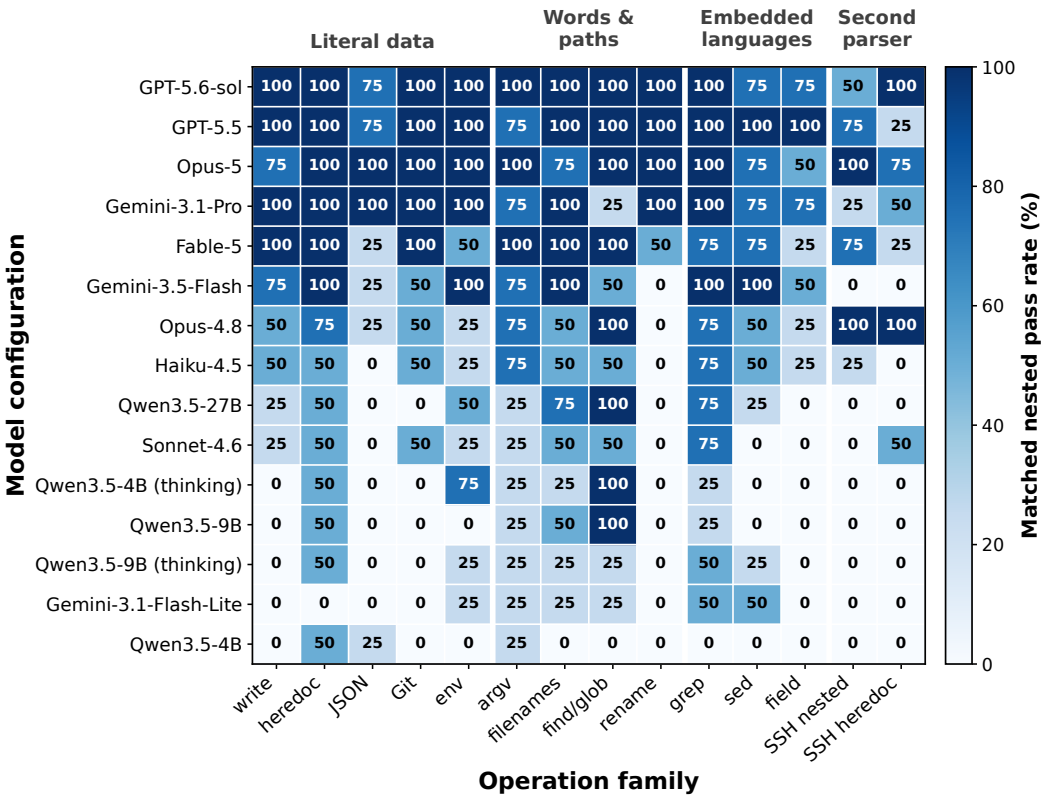


Figure 3: **Matched-nested success varies by operation family across the frozen configuration sweep.** Each cell is the percentage of a family's four tasks passed by one stored generation. Rows retain the 15 frozen configurations used for broad coverage; Qwen rows marked (thinking) enable the think toggle and unmarked Qwen rows are non-thinking, and Appendix Table 13 lists the queried effort settings and request parameters. Table 9 separately summarizes the best observed setting per base model. Columns follow the mechanism groups in Appendix Table 4. SSH-like tasks use the local two-shell simulation. Rows are ordered by matched-nested score in this frozen sweep, which differs from the best-observed order in Table 9.

Table 10 lists every measurement campaign the paper draws on and which results it feeds.

Table 10: Campaign map. All replays are zero-call executions of stored replies in the pinned container.

Campaign	Generations	Design	Feeds
Study A same-window sweep	8 configs × 56 × 2 contracts	one randomized window, effort unset	Tables 1, 15
Study A ladder sweep	44 rungs, 11 configs	per-provider windows	Tables 9, 12
Study A rung crossover	30 rungs, 7 configs, 26 crossover pts	replay both transports	Table 14, Fig. 4
Public three-draw repeat	8 configs × 56 × 2 × 2 draws	same design as the sweep	Appendix E.1.6
Study B native tool	8,736 generations, 17,472 replays	observational, both userlands	Tables 17, 18, 19
Private-v2 crossover	2 models × 42 hostile payloads	one serving window	Table 20
Private-v1 replay	3 models × 42 tasks	earlier generations; script bypass	Table 21
BSD-live legacy	6 configurations	BSD-elicited, GNU-replayed	Table 16
Real-ssh grounding	8 configs × 56	`ssh localhost` replay	Table 7
Advice arm	8 configs × 56 × 2	same-window paired advice contrast	Table 22
Grammar crossover	8 configs × 56 × 2 disclosed	replay-only wrapper 2×2	Table 23
Real-ssh full crossover	6 configs × 56 × 2	disclosed replies on real ssh	Table 24
JSON boundary	6 configs × 56	serializer replay	Table 25

Mechanism analysis is restricted to the eight same-window configurations. The broader matched-outcome ladders remain useful for descriptive operating-point comparison, but no off-diagonal cell is reconstructed across serving windows. Table 11 shows why an omitted effort field cannot be interpreted as a common neutral rung.

Table 11: Calibration of the unset-effort sweep arm against each configuration's labelled ladder. Distance is unset success minus the lowest-rung success, in percentage points. The two measurements come from different serving windows, so small differences are descriptive only.

Model	Unset (%)	Nearest rung	Distance to lowest rung
GPT-5.6-sol	91.1	low	0.0
GPT-5.5	89.3	high	−5.4
Opus-5	89.3	medium	+28.6
Gemini-3.1-Pro	80.4	none within the ladder	−10.7
Gemini-3.5-Flash	58.9	low	+1.8
Opus-4.8	57.1	xhigh	+17.9
Gemini-3.1-Flash-Lite	14.3	all four rungs tie	0.0

B.1 Effort ladders and model configurations

An unset effort field maps to different parts of each provider's ladder. Across the seven same-window configurations with both measurements, Opus-4.8's unset arm resembles xhigh, Opus-5's resembles medium, and Gemini-3.1-Pro's falls below its entire ladder. Users and evaluators should compare operating points through measured behavior. Appendix B.1 reports the exact rung values and calibration table.

Tables 12 and 13 document the measured operating points. The first reports each model's exposed settings and outcomes; the second records exact model identifiers and request parameters. Provider labels are within-model controls, not common compute units. Qwen exposes a think toggle rather than a multi-rung effort parameter. For Haiku-4.5, the output-token means are strongly right-skewed; the corresponding medians are 1,206, 1,560, 936, and 1,283 tokens, so the non-monotonic budget ordering persists under a robust summary.

B.2 Crossover at every effort rung

Most matched-score movement comes from the contract-conditioned contrast. It rises from +10.7 at low to +64.3 at max for Opus-4.8 and from +32.1 at low to +66.1 at max for Opus-5. Interior rungs are not monotone (Figure 4). Gemini-3.1-Flash-Lite returns byte-identical replies at all four settings and therefore contributes a single observed operating point.

Opus-4.8 shows the masking effect when matched success does climb. Its matched gap moves from −48.2 points at low to −3.6 at max, while damage grows from −58.9 to −67.9. The raw arm also improves by 10.7 points, but the nested-replay pass rate ends near where it began. A matched

Table 12: Matched-nested effort ladders under the disclosed-boundary contract. Within each row, success rates and mean provider-reported output tokens follow the setting order in the second column. Each point uses trial 0 over 56 tasks.

Model	Settings (in order)	Success (%)	Mean output tokens
GPT-5.6-sol	low / medium / high / xhigh	91.1 / 92.9 / 98.2 / 98.2	362 / 565 / 773 / 1,073
GPT-5.5	low / medium / high / xhigh	94.6 / 92.9 / 91.1 / 100.0	507 / 655 / 1,164 / 2,757
Opus-5	low / medium / high / xhigh / max	60.7 / 91.1 / 96.4 / 100.0 / 98.2	796 / 1,336 / 1,458 / 2,421 / 3,499
Fable-5	low / medium / high / xhigh / max	71.4 / 75.0 / 96.4 / 92.9 / 100.0	332 / 569 / 843 / 1,212 / 2,396
Opus-4.8	low / medium / high / xhigh / max	39.3 / 48.2 / 50.0 / 62.5 / 94.6	835 / 1,291 / 1,706 / 3,481 / 6,960
Gemini-3.1-Pro	low / medium / high	91.1 / 91.1 / 91.1	5,267 / 5,753 / 6,308
Sonnet-4.6	low / medium / high / xhigh	23.2 / 28.6 / 64.3 / 62.5	28 / 1,411 / 4,337 / 4,539
Gemini-3.5-Flash	minimal / low / medium / high	57.1 / 58.9 / 64.3 / 62.5	3,851 / 3,108 / 3,081 / 3,389
Haiku-4.5	low / medium / high / xhigh	32.1 / 37.5 / 32.1 / 26.8	5,432 / 4,717 / 4,708 / 4,783
Qwen3.5-27B	non-thinking / thinking	28.6 / 28.6	20 / 7,489
Gemini-3.1-Flash-Lite	minimal / low / medium / high	14.3 / 14.3 / 14.3 / 14.3	20 / 20 / 20 / 20

Table 13: Study-A model identifiers and request parameters. The effort column lists exactly the settings queried; sweep arms omitted the effort field. Temperature is reported only where the interface accepts it.

Display name	Model identifier	Effort settings queried	Decoding parameters
GPT-5.6-sol	`gpt-5.6-sol`	low, medium, high, xhigh	max output tokens 16,000, temperature not sent
GPT-5.5	`gpt-5.5`	low, medium, high, xhigh	max output tokens 16,000, temperature not sent
Opus-5	`claude-opus-5`	low, medium, high, xhigh, max	provider defaults, no sampling or length control sent
Opus-4.8	`claude-opus-4-8`	low, medium, high, xhigh, max	provider defaults, no sampling or length control sent
Fable-5	`claude-fable-5`	low, medium, high, xhigh, max	provider defaults, no sampling or length control sent
Sonnet-4.6	`claude-sonnet-4-6`	low, medium, high, xhigh	provider defaults, no sampling or length control sent
Haiku-4.5	`claude-haiku-4-5`	low, medium, high, xhigh	provider defaults, no sampling or length control sent
Gemini-3.1-Pro	`gemini-3.1-pro-preview`	low, medium, high	temperature 0, max tokens 4,096 in the sweep, omitted in the ladder
Gemini-3.5-Flash	`gemini-3.5-flash`	minimal, low, medium, high	temperature 0, max tokens 4,096 in the sweep, omitted in the ladder
Gemini-3.1-Flash-Lite	`gemini-3.1-flash-lite-preview`	minimal, low, medium, high	temperature 0, max tokens 4,096 in the sweep, omitted in the ladder
Qwen3.5-27B	`Qwen/Qwen3.5-27B`	non-thinking, thinking	temperature 0, max tokens 4,096 non-thinking, omitted thinking
Qwen3.5-9B	`Qwen/Qwen3.5-9B`	non-thinking, thinking	temperature 0, max tokens 4,096 non-thinking, omitted thinking
Qwen3.5-4B	`Qwen/Qwen3.5-4B`	non-thinking, thinking	temperature 0, max tokens 4,096 non-thinking, omitted thinking

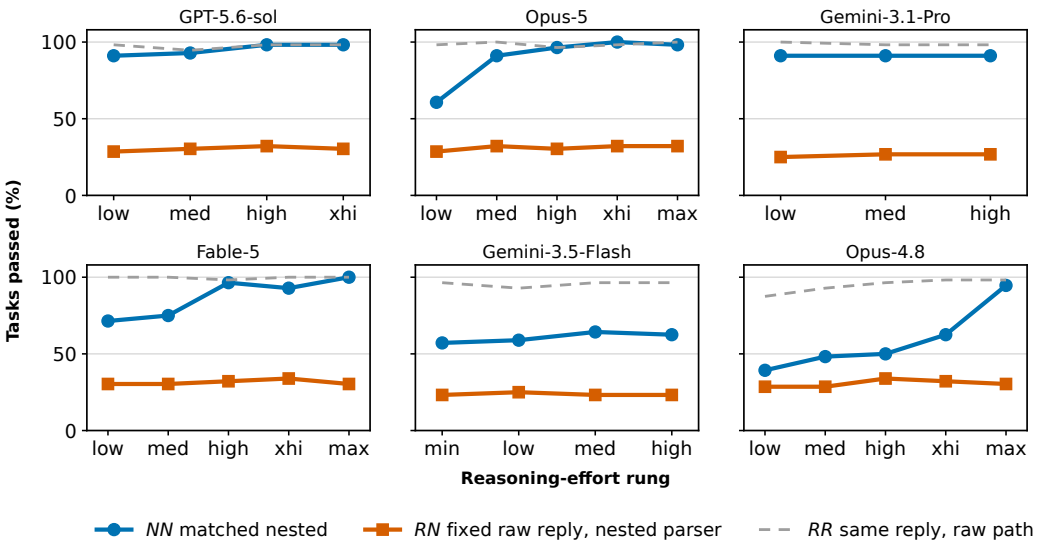


Figure 4: **Where effort raises matched success, the nested-replay pass rate changes little.** Each point uses 56 tasks, trial 0, and the GNU replay. The RR value is raw success, RN is the nested-replay pass rate of raw-conditioned replies, and NN is matched nested success. The $NN - RN$ gap is compensation. Appendix Table 14 reports every rung, including the byte-identical Gemini-3.1-Flash-Lite settings.

evaluation would attribute the improvement to repair, whereas fixed-reply replay shows that cross-path portability remains essentially unchanged.

For six configurations, stored raw and disclosed-boundary replies are available at every acted-on rung. Table 14 additionally lists Gemini-3.1-Flash-Lite, whose four byte-identical rungs are excluded from the six-configuration crossover count. Replaying each through both transports yields the cells plotted in Figure 4. No additional model call is made.

Table 14: Crossover at every measured rung. Cells are percentages and effects are percentage points, defined as in Table 1. An asterisk marks the descriptive masked-fragility rule. Gemini-3.1-Flash-Lite returns byte-identical replies at all four settings and is retained only to document that the provider did not expose a usable ladder.

Configuration	Rung	RR	RN	NR	NN	Damage	Compensation	Matched gap
GPT-5.6-sol	low	98.2	28.6	53.6	91.1	−69.6	+62.5	−7.1
GPT-5.6-sol	medium	94.6	30.4	48.2	92.9	−64.3	+62.5	−1.8*
GPT-5.6-sol	high	98.2	32.1	48.2	98.2	−66.1	+66.1	+0.0*
GPT-5.6-sol	xhigh	98.2	30.4	51.8	98.2	−67.9	+67.9	+0.0*
Opus-5	low	98.2	28.6	57.1	60.7	−69.6	+32.1	−37.5
Opus-5	medium	100.0	32.1	41.1	91.1	−67.9	+58.9	−8.9
Opus-5	high	96.4	30.4	42.9	96.4	−66.1	+66.1	+0.0*
Opus-5	xhigh	98.2	32.1	42.9	100.0	−66.1	+67.9	+1.8*
Opus-5	max	100.0	32.1	46.4	98.2	−67.9	+66.1	−1.8*
Gemini-3.1-Pro	low	100.0	25.0	32.1	91.1	−75.0	+66.1	−8.9
Gemini-3.1-Pro	medium	98.2	26.8	41.1	91.1	−71.4	+64.3	−7.1
Gemini-3.1-Pro	high	98.2	26.8	35.7	91.1	−71.4	+64.3	−7.1
Fable-5	low	100.0	30.4	48.2	71.4	−69.6	+41.1	−28.6
Fable-5	medium	100.0	30.4	48.2	75.0	−69.6	+44.6	−25.0
Fable-5	high	98.2	32.1	42.9	96.4	−66.1	+64.3	−1.8*
Fable-5	xhigh	100.0	33.9	46.4	92.9	−66.1	+58.9	−7.1
Fable-5	max	100.0	30.4	35.7	100.0	−69.6	+69.6	+0.0*
Gemini-3.5-Flash	minimal	96.4	23.2	75.0	57.1	−73.2	+33.9	−39.3
Gemini-3.5-Flash	low	92.9	25.0	67.9	58.9	−67.9	+33.9	−33.9
Gemini-3.5-Flash	medium	96.4	23.2	76.8	64.3	−73.2	+41.1	−32.1
Gemini-3.5-Flash	high	96.4	23.2	71.4	62.5	−73.2	+39.3	−33.9
Opus-4.8	low	87.5	28.6	69.6	39.3	−58.9	+10.7	−48.2
Opus-4.8	medium	92.9	28.6	67.9	48.2	−64.3	+19.6	−44.6
Opus-4.8	high	96.4	33.9	64.3	50.0	−62.5	+16.1	−46.4
Opus-4.8	xhigh	98.2	32.1	60.7	62.5	−66.1	+30.4	−35.7
Opus-4.8	max	98.2	30.4	44.6	94.6	−67.9	+64.3	−3.6*
Gemini-3.1-Flash-Lite	minimal	78.6	19.6	80.4	14.3	−58.9	−5.4	−64.3
Gemini-3.1-Flash-Lite	low	78.6	19.6	80.4	14.3	−58.9	−5.4	−64.3
Gemini-3.1-Flash-Lite	medium	78.6	19.6	80.4	14.3	−58.9	−5.4	−64.3
Gemini-3.1-Flash-Lite	high	78.6	19.6	80.4	14.3	−58.9	−5.4	−64.3

C Statistical Details

The main text treats the 14 operation families as inferential units. For each of the two primary Study-A components reported here, we apply Holm's step-down procedure across the eight model-specific enumerated sign-flip tests. Because the families are purposively constructed, the p values use a family-sign symmetry null: conditional on the observed effect magnitudes, positive and negative signs are exchangeable. Enumeration is exact for this finite family set. Table 15 reports the two primary components with 95% intervals from a scenario-family percentile bootstrap of the mean (10,000 replicates, families resampled as units). The transport-damage result also has a direct finite-benchmark reading: every model-specific effect is negative and every leave-one-family-out range remains negative. The largest adjusted p is .001465.

A configuration joins the supported positive-compensation set when its effect is positive, its Holm-adjusted p is at most .05, and its leave-one-family-out estimates stay positive. Six qualify: GPT-5.6-sol, GPT-5.5, Opus-5, Gemini-3.1-Pro, Opus-4.8, and Gemini-3.5-Flash. The supported set is defined over the eight same-window rows alone.

Table 15: Enumerated and Holm-adjusted two-sided family-sign p values for the two primary Study-A components under the single-clause disclosed-boundary contract. Effect sizes are percentage points.

	Fixed-reply transport			Contract-conditioned compensation		
Model	Effect [95% CI]	Enum. p	Holm p	Effect [95% CI]	Enum. p	Holm p
GPT-5.6-sol	-64.3 $[-80.4, -46.4]$	.000244	.001465	$+60.7$ $[+46.4, +75.0]$	.000244	.001953
GPT-5.5	-71.4 $[-85.7, -55.4]$	.000244	.001465	$+60.7$ $[+44.6, +75.0]$	.000244	.001953
Opus-5	-66.1 $[-82.1, -50.0]$	.000244	.001465	$+58.9$ $[+41.1, +75.0]$	.000488	.002930
Gemini-3.1-Pro	-73.2 $[-87.5, -58.9]$	.000122	.000977	$+55.4$ $[+33.9, +73.2]$	.001221	.006104
Gemini-3.5-Flash	-67.9 $[-82.1, -51.8]$	.000244	.001465	$+30.4$ $[+12.5, +48.2]$	.013672	.041016
Opus-4.8	-64.3 $[-80.4, -46.4]$	.000244	.001465	$+30.4$ $[+14.3, +48.2]$	.003906	.015625
Qwen3.5-27B	-55.4 $[-69.6, -41.1]$	.000244	.001465	0.0 $[0.0, 0.0]$	1.000000	1.000000
Gemini-3.1-Flash-Lite	-58.9 $[-71.4, -48.2]$	.000122	.000977	-5.4 $[-10.7, 0.0]$	.250000	.500000

C.1 USERLAND ROBUSTNESS

Userland changes are smaller than the added-parser effect, but they are not always negligible. In the current Study-A campaign, fixed-reply damage remains negative in both BSD/macOS and GNU/Linux. Corresponding crossover cells differ by at most 3.6 points for six evaluated provider-hosted configurations and by 7.1–12.5 points for the other two.

An earlier frozen BSD-live campaign reveals model-specific dialect affinity. The identical stored commands improve on GNU for both Qwen3.5-27B settings and Gemini-3.1-Pro, while Fable-5 and Gemini-3.5-Flash retain higher success on BSD. Opus-4.8 is higher on BSD in the raw arm and tied in the nested arm. Shifts reach 8.9 points and partially reorder the models under both generation contracts. Because all commands were elicited in BSD/macOS sessions, this analysis measures cross-userland transfer, not what a model would generate when explicitly targeting GNU. Study B hints at a contract-by-userland interaction. Gemini-3.1-Pro and Fable-5 flip from positive native-minus-raw changes on BSD to negative on GNU, though the shift does not survive the sensitivity analysis (Appendix D). Command benchmarks should therefore report their userland and replay stored commands across the environments they claim to support (Appendix C.1).

We report the pinned GNU/Linux replay; the BSD/macOS execution changes only the utility environment, not the stored reply. For the six evaluated provider-hosted configurations, corresponding crossover cells differ by at most 3.6 points. The two remaining configurations differ by 7.1–12.5 points. Fixed-reply damage remains negative for all eight configurations in both userlands, with largest Holm-adjusted p values of .001465 (GNU) and .001709 (BSD), and the masked set is unchanged. Four of 48 ladder comparisons change their internal rung order, so we report one primary userland.

The earlier BSD-live campaign provides a separate cross-userland transfer analysis. Its raw and nested commands were elicited in BSD/macOS sessions and replayed unchanged in the pinned GNU container. Table 16 therefore measures how BSD-elicited commands transfer across utility dialects; it does not estimate what the same models would generate if explicitly prompted for GNU. The direction is model-specific, the largest shift is 8.9 points, and both raw and nested rankings change across userlands.

Table 16: Cross-userland transfer in the earlier BSD-live campaign. Each cell reports pass rate on BSD live execution and GNU replay of the identical stored command. Arms retain their original campaign multiplicity: raw cells contain 56 records, while nested cells contain 56 or 168 depending on the configuration. We interpret only the within-arm BSD→GNU change. The final column summarizes the higher-transfer userland; it is not a counterfactual GNU-targeted generation result.

Model	Raw BSD→GNU	Nested BSD→GNU	Transfers better to
Fable-5	$96.4 \to 91.1$	$92.9 \to 87.5$	BSD
Qwen3.5-27B (non-thinking)	$78.6 \to 87.5$	$25.0 \to 32.1$	GNU
Qwen3.5-27B (thinking)	$73.2 \to 82.1$	$44.6 \to 50.0$	GNU
Gemini-3.1-Pro	$92.9 \to 100.0$	$89.3 \to 96.4$	GNU
Gemini-3.5-Flash	$100.0 \to 96.4$	$69.6 \to 67.9$	BSD
Opus-4.8	$91.1 \to 87.5$	$73.8 \to 73.8$	BSD (raw); tie (nested)

The raw ordering changes from Gemini-3.5-Flash/Fable-5/Gemini-3.1-Pro on BSD to Gemini-3.1-Pro/Gemini-3.5-Flash/Fable-5 on GNU; the nested ordering likewise swaps Fable-5 and Gemini-3.1-Pro at the top. Utility dialect is therefore a second systems axis for measured shell competence, distinct from quoting reliability.

Study B contains the only effect-sign changes: Gemini-3.1-Pro and Fable-5 move from small positive native-minus-raw effects on BSD to small negative effects under GNU (Table 18). This comparison also transfers the same BSD-elicited commands to GNU, keeping generation fixed.

C.2 Masked fragility and the diagonal identity

As one illustrative reading aid (not a fitted criterion), a configuration may be called descriptively masked when the matched gap is small while both components are large, for instance

$$|Y_{NN} - Y_{RR}| \leq 5 \text{ pp}, \quad Y_{RN} - Y_{RR} \leq -40 \text{ pp}, \quad Y_{NN} - Y_{RN} \geq 30 \text{ pp}.$$

At the reported rungs GPT-5.6-sol meets this reading ($-3.6 = -64.3 + 60.7$), and ten of the 30 rung-level crossovers meet the same cut (asterisks in Table 14), all at ladder tops. Masking depends on the operating point and the chosen cut, not on a model-level invariant.

This descriptive threshold identifies cancellation between two large components. The decomposition is an identity verified for every task before aggregation:

$$Y_{NN} - Y_{RR} = (Y_{RN} - Y_{RR}) + (Y_{NN} - Y_{RN}).$$

D Native-Contract Diagnostics

Table 17: Absolute GNU replay pass rates in the separate Study-B campaign, pooled over each model's measured effort ladder and three trials per cell.

Model	Attempts	Raw (%)	Native (%)	Δ (pp)
Opus-4.8	840	95.4	98.0	+2.6
Opus-5	840	98.2	97.4	−0.8
Fable-5	840	99.3	97.1	−2.1
Gemini-3.1-Pro	504	98.8	95.0	−3.8
GPT-5.6-sol	672	96.9	94.3	−2.5
Gemini-3.5-Flash	672	95.7	85.7	−10.0

The native campaign provides a separate comparison for six evaluated provider-hosted models. Their provider-native shell-tool success ranges from 85.7% to 98.0%, compared with 95.4–99.3% on the raw path (Table 17). These values average over every reported effort rung and three trials, whereas Table 9 selects one trial from the best observed setting. Native-tool performance is substantially closer to raw execution than performance under the controlled nested boundary, although the native effect remains model dependent.

For each model, Table 17 pools Study B's measured effort rungs and three trials per cell. The Attempts column is the per-arm denominator.

Across pooled effort rungs, the native-minus-raw change ranges from $+2.6$ to -10.0 points and is smaller than the controlled nested loss for every model. Appendix Table 18 reports paired effects, leave-one-family-out ranges, transitions, BSD comparison, and the failure taxonomy.

Study B contains 8,736 generated arm records: 4,368 raw and 4,368 native. Each stored generation is replayed in both BSD and GNU userlands, yielding 17,472 execution outcomes. The main text reports aggregate GNU pass rates. This appendix adds the paired effects, robustness comparisons, and failure diagnostics.

Table 18 reports the paired contract effects. Δ is the campaign-average native-minus-raw change in percentage points. LOFO gives the range when each operation family is omitted in turn. PF/FP counts paired pass→fail and fail→pass transitions. The BSD columns provide the userland robustness comparison of Appendix C.1.

Table 18: Exploratory native-minus-raw effects over each measured effort ladder, ordered by GNU replay effect.

	BSD live			GNU replay		
Model	Δ	LOFO	PF/FP	Δ	LOFO	PF/FP
Opus-4.8	$+3.10$	$[+2.18, +3.46]$	5/31	$+2.62$	$[+1.67, +2.95]$	11/33
Opus-5	-0.36	$[-0.51, 0.00]$	12/9	-0.83	$[-1.03, -0.51]$	16/9
Fable-5	$+0.95$	$[0.00, +1.15]$	3/11	-2.14	$[-2.44, -0.64]$	22/4
GPT-5.6-sol	-1.19	$[-1.92, -0.48]$	25/17	-2.53	$[-3.37, -0.64]$	26/9
Gemini-3.1-Pro	$+1.98$	$[-0.64, +3.21]$	8/18	-3.77	$[-4.91, -0.64]$	23/4
Gemini-3.5-Flash	-4.02	$[-4.33, -3.04]$	37/10	-9.97	$[-10.74, -6.89]$	77/10

Process exit status misses a substantial share of failures. Across the four contract–userland conditions, 41–62 executions exit zero while leaving the wrong state, accounting for 23.4–47.0% of that condition's failures (Table 19). Native one-call schema adherence is 98.2–100%.

Table 19: Study-B execution outcomes, aggregated across six models. Each row partitions 4,368 executions. Adherence denotes an invalid one-call tool invocation. Syntax includes parser and command-usage errors. Exit-0 wrong is a silent final-state failure.

Userland	Contract	Pass	Adherence	Syntax	Nonzero	Exit-0 wrong
BSD	raw	4230	0	31	59	**48**
BSD	native	4236	20	9	41	**62**
GNU	raw	4252	0	39	36	**41**
GNU	native	4146	20	22	128	**52**

D.1 Study-B robustness and failure analysis

The family-sign sensitivity analysis treats the 14 operation families as the inferential units and applies Holm adjustment across the six models separately in each userland. All twelve adjusted values exceed .05. The minimum is .18750 for Gemini-3.5-Flash in both userlands. Study B is therefore descriptive and exploratory. Table 18 reports the effect sizes, leave-one-family-out ranges, and paired transitions that support that interpretation.

D.1.1 Effects by effort rung

Figure 5 decomposes the aggregate effects from the main text. Each cell compares raw and native arms within one provider-specific effort rung.

Wire integrity. Three model configurations from two providers retained raw native-tool arguments, allowing byte-level verification between the decoded `command` field and executor input: 660 records for Gemini-3.5-Flash, 497 for Gemini-3.1-Pro, and 672 for GPT-5.6-sol. Nineteen adherence failures contain no usable argument. For Opus-4.8, Opus-5, and Fable-5, the stored artifact begins at the decoded command string, so their analysis starts at that boundary.

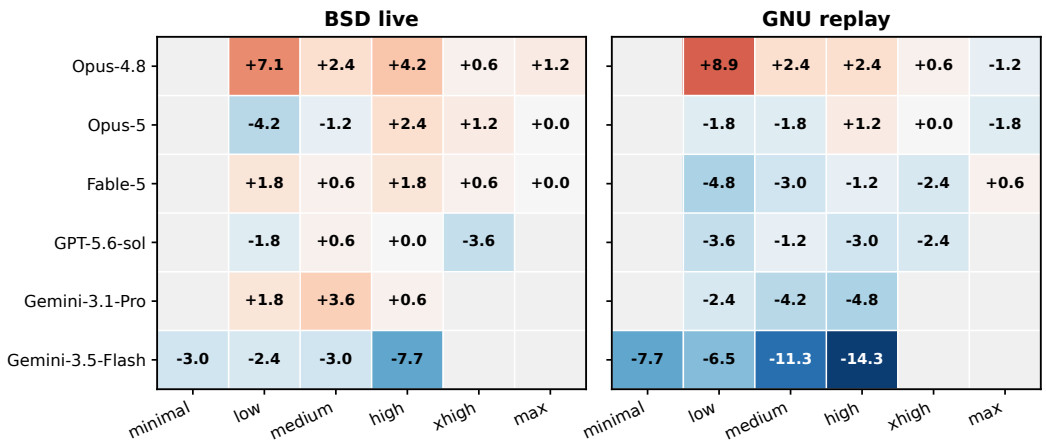


Figure 5: Native-minus-raw pass-rate change by provider effort rung, in percentage points on a zero-centered scale. Blank cells mark rungs the provider does not expose. Models follow descending GNU replay campaign effect. Labels are not comparable compute budgets across providers.

E Replication and Transport Robustness

Table 20: Private-v2 crossover on 42 hostile payloads under the single-clause disclosed-boundary contract. The private set is hostile-only and not difficulty-matched to the public core, so absolute rates are interpreted within this set. Cells are pass rates. Effects are percentage points. Damage is $RN - RR$, compensation is $NN - RN$, interaction is $(NN - NR) - (RN - RR)$, and the matched gap is $NN - RR$.

Model	RR	RN	NR	NN	Damage	Comp.	Interaction	Matched gap
GPT-5.6-sol	92.9	19.0	50.0	97.6	−73.8	+78.6	+121.4	+4.8
Opus-4.8	92.9	16.7	59.5	42.9	−76.2	+26.2	+59.5	−50.0

Table 21: Private-v1 fixed raw replies under three transports on 42 tasks. This campaign predates the private-v2 crossover in Table 20. The two tables therefore use different generations. The first three columns report tasks passed out of 42. Script gain is the temporary-script rate minus the nested-wrapper rate, in percentage points.

Model	Raw	Nested wrapper	Temporary script	Script gain (pp)
GPT-5.6-sol	41/42	8/42	41/42	**+78.6**
Opus-4.8	40/42	7/42	40/42	**+78.6**
Qwen3.5-27B	30/42	9/42	30/42	**+50.0**

Design. The private-v2 single-clause crossover uses 42 unpublished hostile payloads: three variants for each of the 14 operation families. The task manifest, contract order, and sample hash were fixed before inference. GPT-5.6-sol and Opus-4.8 each contribute one accepted reply per task under the raw and single-clause disclosed-boundary contracts, for 168 accepted generations. Raw- and disclosed-boundary-contract calls were interleaved within one serving window. Every stored reply was replayed through both transports and scored by the same final-state validators used for the public core. Table 20 reports the resulting effects.

Here and in the private-v1 campaign, an accepted reply is the first usable response returned for a scheduled model–task–contract cell. Semantic failure, truncation, refusal, malformed output, and contract error are retained as outcomes rather than retried or filtered. At the campaign-invocation level, private-v2 made 174 attempts: 168 returned replies were retained, six provider or transport errors that yielded no usable response were retried, and no returned replies were rejected. Private-v1 made 253 attempts across three configurations and two contracts: 252 returned replies were retained,

one no-response error was retried, and no returned replies were rejected. Lower-level retries inside provider adapters are not separately observable.

Validator checks. The 42 private tasks accept their oracles under both transports and reject every untouched initial state. Naive probes produce no false accepts, and mutation testing rejects all 148 applicable corruptions. These checks cover the declared corruption classes.

Scope. The private set contains hostile payloads only and was designed to test whether the mechanism transfers to unseen literals. Its difficulty was not matched to the 56-task public core, so absolute rates should be read within the private set. The result establishes transfer of fixed-reply transport damage across payload samples. Repeated-generation stability, temporary-script bypass, and typed-operation evidence are reported in Appendix E.1.

E.1 Bypass, repeated draws, and typed operations

E.1.1 Temporary-script bypass

A public-core replication executes every stored trial-0 raw reply of the eight same-window configurations through raw `bash -c`, the nested wrapper, and a temporary Bash script. The script transport reproduces the raw-path outcome for all 448 configuration–task pairs, recovering all 292 nested-only failures and rescuing no raw-path failure. An escaped variant of the nested transport, which interpolates each stored reply with standard shell quoting instead of verbatim substitution, reproduces the raw-path outcome for the same 448 pairs: correct escaping at the boundary removes the entire effect. The released records reproduce both replays without private data.

Table 21 uses the earlier private-v1 raw-generation campaign, which includes Qwen3.5-27B and is distinct from the private-v2 generations in Table 20. Each accepted raw-contract reply runs through three paths: raw `bash -c`, the nested wrapper, and a temporary Bash script. The fixed reply makes the comparison a direct test of transport behavior. Table 21 reports the resulting pass counts.

E.1.2 Advice arm

Disclosure, not instruction, carries the effect for capable models. A paired arm regenerates the advice-free and advice-bearing disclosed contracts in one serving window, so the contrast has no window confound. At the top of the ladder, the added escaping advice barely moves matched nested success: GPT-5.6-sol -8.9, GPT-5.5 $+7.1$, and Opus-5 $+3.6$ points. Disclosure alone already elicits the adaptation. In the middle it makes the largest difference: Sonnet-4.6 gains $+25.0$, Haiku-4.5 $+12.5$, and Opus-4.8 $+7.1$ points from the advice. At the bottom neither contract helps (Qwen3.5-27B and Gemini-3.1-Flash-Lite $+1.8$). Boundary advice thus barely moves the top, changes middle-tier outcomes the most, and does not move the bottom (Appendix Table 22).

Table 22 reports matched nested success under the advice-free and advice-bearing disclosed contracts, collected for each configuration in one serving window and replayed with the public validators. Δ is the paired same-window difference, free of the cross-window drift the repeated draws bound (Appendix E.1.6).

Table 22: Matched nested success (of 56) under the disclosed contract without and with escaping advice, paired within one serving window. Advice barely moves the top of the ladder, moves the middle of the ladder the most, and does not reach the bottom.

Configuration	Advice-free	With advice	Δ (pp)
GPT-5.6-sol	54	49	-8.9
GPT-5.5	49	53	$+7.1$
Opus-5	52	54	$+3.6$
Sonnet-4.6	32	46	$+25.0$
Haiku-4.5	25	32	$+12.5$
Opus-4.8	30	34	$+7.1$
Qwen3.5-27B	17	18	$+1.8$
Gemini-3.1-Flash-Lite	7	8	$+1.8$

E.1.3 GRAMMAR CROSSOVER

The adaptation is conditioned on the declared grammar rather than applying a fixed defense. A crossed arm discloses either a double-quote or a single-quote wrapper and replays each stored reply through both, forming a 2×2 of disclosed against executed grammar. Capable models pass far more on the grammar they were told than on the other: GPT-5.6-sol passes 53/56 of its single-disclosed replies on the single-quote wrapper but only 10/56 on the double-quote one, and its diagonal (matched) advantage over the anti-diagonal is $+80.4$ points. The advantage separates the same top, middle, and bottom groups as the matched-nested scores: $+80.4$, $+77.7$, and $+65.2$ at the top, $+18.8$ to $+25.0$ in the middle, and $+0.0$ (Qwen3.5-27B) to -19.6 (Gemini-3.1-Flash-Lite) at the bottom (Appendix Table 23). This is contract-conditioned behavioral adaptation to the declared grammar, not a memorized double-quote fix.

Table 23 crosses the disclosed grammar (double- or single-quote wrapper) with the executed wrapper. Each disclosed reply is replayed on both the wrapper it was told about and the other one; a model that adapts to the stated grammar passes more on the matched diagonal than the mismatched anti-diagonal. All executions are zero-call replays.

Table 23: Grammar crossover (tasks passed of 56). Rows are the disclosed grammar, columns the executed wrapper; the diagonal is matched, the anti-diagonal mismatched. The diagonal advantage separates the same top, middle, and bottom groups as the matched-nested scores.

	Double disclosed		Single disclosed		Diag−anti
Configuration	on double	on single	on single	on double	(pp)
GPT-5.6-sol	54	7	53	10	+80.4
GPT-5.5	49	8	54	8	+77.7
Opus-5	52	8	45	16	+65.2
Sonnet-4.6	32	11	23	16	+25.0
Opus-4.8	30	11	23	18	+21.4
Haiku-4.5	25	13	20	11	+18.8
Qwen3.5-27B	17	11	11	17	+0.0
Gemini-3.1-Flash-Lite	7	19	9	19	−19.6

E.1.4 REAL-SSH FULL CROSSOVER

Table 24 completes the real-`ssh` 2×2 by replaying the disclosed-boundary replies as well, giving NR and NN on the real path. Compensation ($NN - RN$) on the real boundary matches the synthetic nested compensation exactly for five of six replayed configurations; Gemini-3.1-Flash-Lite differs by one task (-1.8 versus -5.4 points). The two non-adapting configurations show no positive compensation on either path. Zero model calls.

Table 24: Real-`ssh` full crossover (tasks passed of 56). Damage is $RN - RR$ and compensation is $NN - RN$, both on the real `ssh` path. The final column repeats the synthetic nested compensation for comparison.

Configuration	RR	RN	NR	NN	`ssh` comp.	Nested comp.
GPT-5.6-sol	53	17	31	51	+60.7	+60.7
GPT-5.5	56	16	28	50	+60.7	+60.7
Opus-5	54	17	24	50	+58.9	+58.9
Opus-4.8	51	15	35	32	+30.4	+30.4
Qwen3.5-27B	48	17	47	17	+0.0	+0.0
Gemini-3.1-Flash-Lite	44	11	45	10	−1.8	−5.4

E.1.5 JSON SERIALIZER BOUNDARY

Table 25 replays each stored raw reply through a JSON tool-call boundary two ways: a correct serializer (`json.dumps` then `json.loads`) that round-trips the reply, and a naive embedding that pastes the reply into a JSON string field without escaping. The naive boundary re-parses the

reply's double quotes and backslashes as JSON syntax and breaks on the same characters as the shell boundary. Many replies do not even parse, so its damage is comparable to the shell nested transport, while the correct serializer costs nothing. Zero model calls.

Table 25: JSON serializer boundary (tasks passed of 56). Correct-serializer damage is ≈ 0; naive-embedding damage is comparable to the shell nested transport. *Unparseable* counts replies whose naive JSON embedding fails to parse.

Configuration	Raw	Correct	Naive	Naive damage
GPT-5.6-sol	53	53	21	-57.1
GPT-5.5	56	56	19	-66.1
Opus-5	54	54	22	-57.1
Opus-4.8	51	51	18	-58.9
Qwen3.5-27B	48	48	19	-51.8
Gemini-3.1-Flash-Lite	44	44	8	-64.3

E.1.6 Repeated generations

Additional sampling preserves the same interpretation. Three more generations for eight private tasks produce different reply text in 19 of 32 task–contract cells, yet all four draws retain negative damage and positive compensation for both models.

A public-core replication adds two generations per contract for all eight same-window configurations. Across the three draws, damage stays negative for every configuration and draw, with per-configuration ranges of 1.8–7.1 points. No compensation changes sign: the largest spread is 12.5 points (Opus-4.8), and Qwen3.5-27B realizes exactly zero compensation in every draw. Scoring each configuration by the tasks it passes in all three draws still yields five strict rank reversals between the matched contracts. In the trial-0 draws, 26 of 28 pairs are strictly comparable (non-tied under both matched contracts) and five reverse: GPT-5.6-sol versus Gemini-3.5-Flash, Gemini-3.1-Pro, GPT-5.5, and Opus-5, and Opus-5 versus Gemini-3.1-Pro. Consistent with the main text, the one reversal that is unambiguous at this resolution is GPT-5.6-sol versus Gemini-3.5-Flash (behind by one task under RR, ahead by eighteen under NN); the other four rest on a single-task margin on at least one side. The same five reverse under the all-draw criterion.

On the private payloads, we collect three additional generations for eight tasks under both generation contracts and both models. Across the four draws, GPT-5.6-sol shows mean damage of -78.1 points and mean compensation of $+81.3$. The corresponding values for Opus-4.8 are -84.4 and $+46.9$. Reply text varies in 9/16 task–contract cells for GPT-5.6-sol and 10/16 for Opus-4.8, yet every draw preserves negative damage and positive compensation. The mechanism therefore persists across distinct sampled replies.

E.1.7 Typed operations

A representation study replaces shell-string construction with structured operations on 18 private tasks from six naturally typeable families. Across two models, typed operations pass 25/36 tasks (69.4%), compared with 35/36 (97.2%) for raw `bash -c` and temporary scripts. Of the eleven typed failures, ten reach the wrong final state and one fails during execution. Structured actions remove one quoting surface, while literal-preservation errors remain in arguments and payload fields. The two configurations probe representation sensitivity.

F Validation and Reproduction

F.1 Validator mutation audit

We validate both task solvability and scoring strictness. A machine-constructed oracle solves every task with one command. Benign naive probes pass, whereas their hazardous counterparts fail on the raw path. We then mutate oracle-produced states by deleting or altering required artifacts, adding collateral files, restoring files that should be removed, or changing Git-only state. The validators accept every oracle and benign probe, and reject every untouched fixture, hazardous probe, and all

197 applicable mutations. As a solvability control, three configurations pass all 56 tasks under the nested transport (Table 9), so every task has a feasible nested solution and the nested arm is not degenerate. These checks cover the specified invalid states. Other validator blind spots may remain.

Validator checks begin from each oracle-produced valid state, apply every applicable mutation class, and rerun the validator:

Mutation class	Rejected
Delete a changed/required file	60/60
Flip one byte in a changed file	60/60
Insert an unexpected collateral file	56/56
Restore a file that should be removed	17/17
Amend a Git-only final state	4/4
Total	197/197

The validators accept every oracle and benign tier-0 probe. They reject every untouched fixture, hostile probe, and all 197 mutated states. These checks cover the enumerated invalid states. Unenumerated false positives remain possible.

F.2 Execution and reproduction

Each command runs in a fresh fixture with a trimmed environment and a 15-second timeout. The local runner invokes `bash -c` through `execve`. The reported GNU replay uses a pinned, network-disabled container. All 56 oracles must pass in that container before crossover replay, and replay never queries a model.

The public code artifact is available at https://anonymous.4open.science/r/quoteBench-31F6. It contains the provider-agnostic harness, all 56 public tasks, validators, contract prompts, and offline rollout verification and descriptive-analysis commands. Together with the separately hosted rollout archive and its SHA-256 manifest, these files reproduce the released campaign/model/contract/effort/trial/toolchain rates and the public GNU crossover table. `REPRODUCE.md` documents the commands, package layout, and campaign-level count reconciliation. Serving-path, authentication, private-payload, and internal adapter metadata are excluded. Each released rollout records prompt, completion, and reasoning token counts: the archive's 12,999 generations total 23.27M provider-reported tokens (10.03M prompt, 13.24M completion, of which 6.23M reasoning), so per-campaign token cost can be recomputed from it; Table 10 lists generation counts per campaign.

Private-payload records are withheld to preserve held-out evaluation and are not part of the public release. They are used only for the private replication and mitigation analyses reported in this paper; neither the payloads nor replies are included in the arXiv source package or ancillary files.

Rollout archive. The sanitized public rollout archive at the rollout archive carried in https://anonymous.4open.science/r/quoteBench-31F6 contains 12,999 records across 33 arm files in the `quotebench-rollout-v1` schema; each record carries one generation with its replays, prompt, reply, identifiers, usage, and final-state outcomes.