CAPE: COVERAGE-PRESERVING TOOL-USE OPTIMIZATION FOR AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Unknown receiver interfaces can prevent tool-using agents from executing correct operations. We study train-free compilation for a finite, calibratable receiver family: preserve every configuration supported by an initial playbook, maximize completion, and then minimize execution calls. CAPE builds executable policies from locally closed action effects and uses optimistic bounds to guide joint program and diagnostic acquisition. Native observation partitions determine policy branches. A tighter continuation bound guides the selected acquisition and can stop further queries before the action's row is complete. This selective computation concentrates search effort on the native work it can avoid. The resulting policies preserve the inherited coverage contract and obtain a full-catalog optimality certificate when native and upper values meet. On 64 source-derived requests with DeepSeek-V4-Pro proposals, evaluated through CSV and JSON interfaces, CAPE attains the independent optimum in 502 of 504 executable compilations. Relative to eager exact search, mean native evaluations decrease by 20.8% and compilation lifecycle by 22.0%. Shared-kernel AND/OR comparisons isolate the acquisition scheduler's cost benefit; direct bound ablations identify when tighter computation pays for itself. A 14-deployment panel evaluates proposal correctness and native policy quality across closed services and open-weight models down to 4B.

1 INTRODUCTION

Tool-using agents must match a proposed operation to the receiver's representation (Yao et al., 2022; Wang et al., 2024). Delimiters, date encodings and JSON orientation determine which serialization a receiver can execute. Resolving these choices consumes calls alongside the requested operation. We study guarded workflows where a successful call publishes the requested data product and a rejected call leaves it unchanged.

The compiler receives a playbook for a finite, calibratable receiver family. Its inherited coverage contract requires preserving every configuration that the playbook supports while optimizing completion and calls. At execution, the receiver identity is hidden and native tool observations guide the policy. Figure 1 shows the compilation and deployment workflow.

Compilation must acquire the tool model as well as search it. Each calibration query reveals one action's effect on one configuration. Lazy search allocates these evaluations using optimistic models (Dellin & Srinivasa, 2016; Garrett et al., 2020). Here, missing effects also determine distinguishability: provisionally separate receivers may return the same observation and require a shared continuation. An optimistic tree can therefore be impossible to execute.

Compilation incurs calibration and search costs. Queries outside the eventual policy consume calibration work, while repeated optimistic solves can cost more than the evaluations they avoid. Efficient compilation requires choosing both which effects to acquire and when to refine the bounds used to select them.

CAPE jointly acquires program and probe effects around executable continuations. An action becomes usable once its effects are known on the configurations that reach it. This local closure supports a policy while the rest of that action's row remains unknown. Closed actions compose an executable policy that supplies a feasible lower bound. Unfinished actions contribute optimistic bounds.

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.

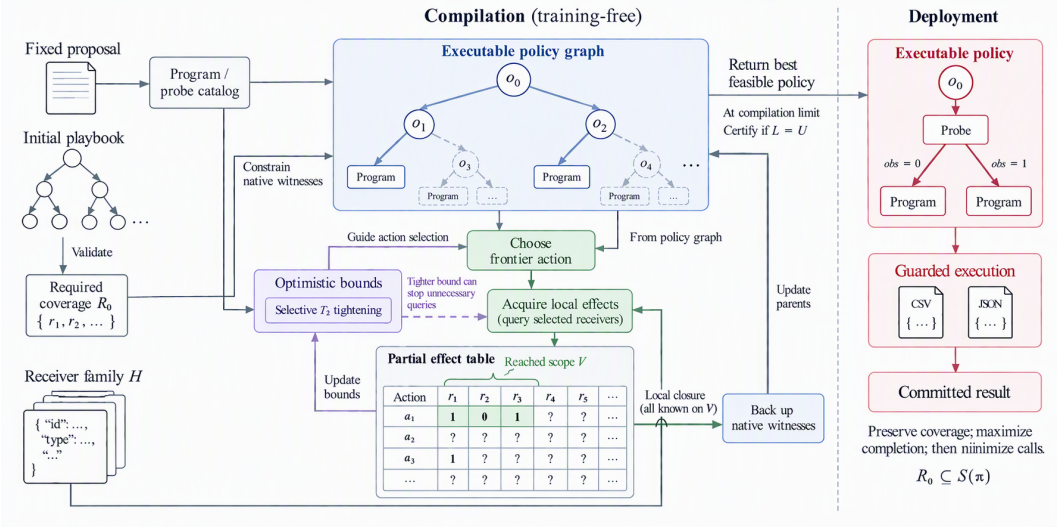

Figure 1: Overview of CAPE. Initial-playbook validation establishes required coverage R_0. Local effect acquisition closes actions on reached receiver scopes; native witness backups improve the executable policy while selective T_2 computation tightens optimistic bounds. Compilation returns the best feasible policy and certifies optimality when native value L equals upper bound U. Deployment branches on tool observations. The matrix and branches are schematic.

The acquisition frontier follows the executable policy and optimistic alternatives, prioritizing missing coverage and improvement per unknown effect. Most unfinished actions use inexpensive support bounds. While acquiring a selected action, CAPE tightens two-call continuations using acquired observation groups and optimistic unknown effects. The sharper bound can stop acquisition before a row is complete. Its computational cost is repaid when it eliminates enough native queries.

The compiler uses a fixed model proposal and native tool evaluations to preserve initial coverage. Matching feasible and upper values certifies the full-catalog optimum. Our CSV and JSON workflows compare CAPE with exact search, generic lazy search, diagnostic lookahead and program retries. A shared-kernel AND/OR control isolates acquisition order; three bound modes measure the cost of tightening. A 14-deployment panel tests coverage across proposal models.

2 Related Work

Agent execution and adaptation. Agent systems combine reasoning and execution (Yao et al., 2022; Wang et al., 2024), improve instructions through documentation and interaction (Hsieh et al., 2023; Fang et al., 2025), and organize calls through parallelization and decomposition (Kim et al., 2024; Prasad et al., 2024; Xu et al., 2023). EnvProbe selects environment queries using belief-state features (Song & Cai, 2026).

Information and coverage constraints. Diagnosis and decision-region determination select informative actions (Javdani et al., 2014; Heckerman et al., 1995); rollout improves base policies (Bhambri et al., 2023). SPIBB controls baseline-relative expected return (Laroche et al., 2019), while shielding and belief-support planning use transition approximations (Carr et al., 2023; Sheng et al., 2023). CAPE protects each initially successful receiver identity.

Search with expensive evaluations. LAO* expands partial solution graphs; bounded dynamic programming maintains lower and upper values (Hansen & Zilberstein, 2001; McMahan et al., 2005). LazySP and PDDLStream allocate expensive evaluations through optimistic planning (Dellin & Srinivasa, 2016; Garrett et al., 2020). CAPE specializes this search structure to effects that determine both success and observation partitions. Locally closed witnesses give feasible policies; its acquisition frontier jointly follows executable and optimistic branches and ranks improvement per missing effect. Our matched AND/OR control shares witnesses, caches and bound operators,

isolating this frontier from best-optimistic-solution-path acquisition. The bound study separately measures the cost of applying the same continuation operator selectively or throughout the expanded graph.

3 JOINT ACQUISITION AND NATIVE POLICY EXPANSION

3.1 THE INHERITED EXECUTION CONTRACT

Let H be a finite receiver family and $A = P \cup Q$ a fixed catalog of program representations and diagnostic probes. For a fixed request, the native effect $f(a,h) = (o(a,h), g(a,h))$ contains an observation and a guard-success bit. Probes have $g = 0$. A successful program publishes the requested effect and terminates; failed calls leave the task state unchanged. Effects are deterministic within the request and receiver family. Calibration can instantiate h, while an executing policy receives tool observations rather than the receiver identity.

For a policy π on $V \subseteq H$, let $S(\pi; V, b)$ denote the identities it completes within b calls and $M(\pi; V, b)$ its total calls, including unsuccessful paths. A supplied playbook π_0 defines $R_0 = S(\pi_0; H, C)$. We maximize

$$J(\pi; H, C) = \big(|S(\pi; H, C)|, -M(\pi; H, C)\big), \qquad R_0 \subseteq S(\pi; H, C), \tag{1}$$

lexicographically, with every execution path bounded by C. The compilation wall allowance and number of calibration queries are separate resources.

The partial table $D \subseteq A \times H$ stores acquired effects immutably. Initial calibration validates π_0 and its required identities. Subsequent acquisition treats program and probe effects jointly. A program's effects outside its reached scope need not be known before it contributes to a policy.

3.2 EXECUTABLE POLICIES FROM CLOSED ACTIONS

A state (V, b) contains the configurations consistent with a native observation history and the remaining calls. Its obligation is $R_0 \cap V$. An action is *closed on* V when every (a,h) with $h \in V$ is in D. Its immediate successes form $G_a(V)$, and failed observations partition the remaining identities into disjoint cells V_o.

A native witness stores a feasible policy, its exact success mask and its call mass. Given closed child witnesses (π_o, S_o, M_o) on $(V_o, b-1)$, a closed action yields

$$S_a = G_a(V) \cup \bigcup_o S_o, \qquad M_a = |V| + \sum_o M_o. \tag{2}$$

Every identity pays the first call, and failures continue according to their native observation. Stop has value $(0,0)$ and is feasible when $R_0 \cap V = \varnothing$.

CAPE stores these witnesses in a persistent native graph. Closed action partitions create executable edges, while unfinished actions contribute bounds. A better child witness updates its native parents through Equation 2. These backups compose feasible child witnesses while their optimal values are still being resolved, allowing native policy expansion to interleave with acquisition. The graph retains its best validated root witness throughout.

Reached effects never change under consistent acquisition, so a witness's paths, successes and calls persist (Figure 6). Each accepted root policy is checked using acquired effects against R_0. Figure 2 measures the additional completion enabled by diagnostics.

3.3 BOUNDS WITH UNKNOWN PROGRAM EFFECTS

Write $E_a(V) = \{h \in V : (a,h) \notin D\}$ and let $G_a(V)$ contain the observed successes. A program's optimistic success set is $\overline{G}_p(V) = G_p(V) \cup E_p(V)$: unknown executions may succeed. Define possible support and the largest optimistic direct success count as

$$\overline{G}(V) = \bigcup_{p \in P} \overline{G}_p(V), \qquad \overline{d}(V) = \max_{p \in P} |\overline{G}_p(V)|. \tag{3}$$

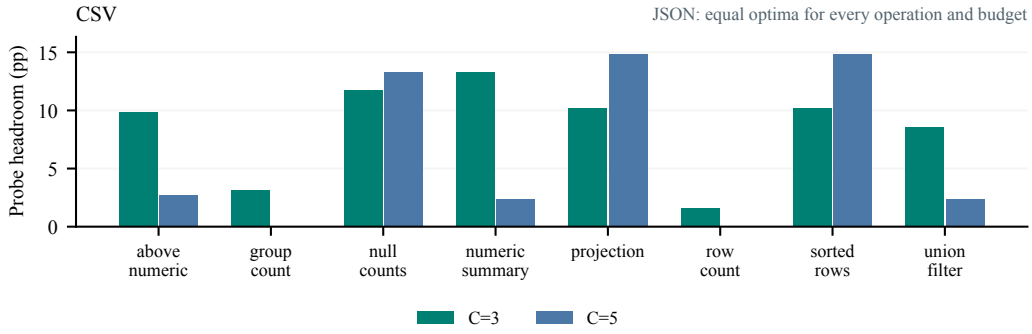

Figure 2: Intrinsic diagnostic headroom on executable inputs: full-catalog optimal completion minus the independently computed program-only optimum. This comparison uses complete native effects and is separate from the 30-second compilation runs.

Required identities outside $\overline{G}(V)$ imply infeasibility. For nonempty support and $b \geq 2$, an admissible value bound is

$$B(V, b) = \left(|\overline{G}(V)|, -\{|V| + |\overline{G}(V)| - \overline{d}(V)\}\right). \tag{4}$$

A policy completing the possible support starts on all of V, and at least $|\overline{G}(V)| - \overline{d}(V)$ successful identities need another call. Policies with fewer successes rank below this bound regardless of cost. At one call, enumerate optimistic terminal programs covering $R_0 \cap V$; at zero calls, only Stop remains. Empty support with no obligation also gives Stop.

For an unfinished action, retain every acquired failed-response group. Unknown program effects optimistically commit. Unknown probe effects receive distinct symbolic observations, one per receiver. Their continuations may therefore choose different terminal programs. A real completion can turn optimistic program successes into failures or merge probe observations, so it cannot outperform this relaxation.

Let $\overline{G}_a$ be the optimistic immediate-success set and $\mathcal{F}_a^+$ its optimistic failure cells. For a child bound B', define

$$\mathcal{U}_a(V, b; B') = \left(|\overline{G}_a(V)| + \sum_{W \in \mathcal{F}_a^+} k_W,\ -|V| + \sum_{W \in \mathcal{F}_a^+} c_W\right), \quad B'(W, b-1) = (k_W, c_W). \tag{5}$$

An infeasible required child makes the action infeasible. Most unfinished actions use the inexpensive bound B. Closed actions use current native child-state upper values. The state takes the best action bound, includes Stop when feasible, and caps the result by Equation 4. Previous valid bounds remain available through the minimum of old and new values.

3.4 Selective continuation bounds

Support alone ignores whether the remaining calls can exploit the acquired observation groups. For a two-call continuation, CAPE enumerates an optimistic first action followed by the best one-call program in each cell:

$$T_2(V) = \max\{\mathcal{U}_a(V, 2; B) : a \in A\}, \tag{6}$$

including Stop when permitted. This is the exact value of the relaxed two-call problem. It is no greater than the support bound and remains an upper bound on every consistent native completion.

CAPE also composes closed actions over two calls to seed executable native witnesses. This calculation uses acquired effects throughout. The optimistic counterpart bounds continuations whose effects are still missing.

CAPE computes the tighter upper value while refining the selected acquisition, where a smaller bound can avoid the next native query. Other frontier rows retain cheap bounds, concentrating detailed bound computation on the selected action. The policy graph uses the full remaining horizon.

Table 1: Main comparisons on the DeepSeek mechanism cohort. Success is pipeline completion (%); calls per receiver and lifecycle seconds condition on executable proposals. Means balance eight sources. Bold marks unique optima and CAPE's tied optima. Quality intervals and complete metrics appear in Appendix D.

		CSV			JSON		
C	Method	Success	Calls	Time	Success	Calls	Time
3	**CAPE**	**85.16**	2.196	10.774	**94.74**	1.476	**11.919**
	Eager exact	85.16	2.196	11.173	94.60	1.474	18.021
	Program-only	76.76	**2.022**	**3.631**	94.60	1.474	17.792
	Protected two-step	83.98	2.474	11.121	94.60	1.493	18.008
	Generic lazy	83.74	2.198	17.089	92.90	1.495	16.200
	Program-first	85.16	2.196	11.402	94.60	1.474	17.878
5	**CAPE**	**98.44**	**2.348**	11.201	**94.74**	1.475	**11.903**
	Eager exact	98.44	2.348	11.417	94.60	1.474	18.079
	Program-only	92.19	2.365	**3.624**	94.60	1.474	17.744
	Protected two-step	98.44	2.838	11.192	94.60	1.493	18.005
	Generic lazy	84.18	2.534	13.700	92.90	1.546	16.008
	Program-first	98.44	2.348	11.572	94.60	1.474	17.919

A frontier contains states reached by the current executable policy and a best optimistic alternative. CAPE activates closed alternatives lazily, creating native children as graph expansion and acquisition proceed. Among unfinished actions on this frontier, states without a feasible required witness receive priority. Other scores rank potential completion gain per missing effect, then call savings per missing effect. Queries within the selected scope target required identities first.

After each effect, CAPE updates the selected action's bound, using T_2 for two-call continuation scopes. Acquisition ends when the action closes on its scope, becomes infeasible, or cannot improve the available native witness. Newly closed actions and improved children update their native parents. Acquisition-aware caches preserve unchanged program and bound calculations. A state whose bounds meet remains solved under consistent acquisition. Once the entire model is known, a compact complete-information solver finishes the same objective within the remaining allowance.

Proposition 1 (Feasible improvement and exact closure)**.** *Under the fixed native-effect assumptions, each accepted root witness preserves R_0 within C calls, and accepted values are nondecreasing. Its value persists under further consistent acquisitions. Every maintained root upper value bounds the full-catalog optimum. Equality of a root lower and upper value certifies optimality for Equation 1.*

Disjoint native cells establish feasible composition; optimistic success and observation refinement establish admissibility. Selective tightening changes acquisition work while retaining these properties. At interruption, CAPE returns the latest complete witness. Interruption before initial validation finishes returns the supplied playbook with certification unset. At most $|A||H|$ distinct effects can be acquired; computation depends on the reachable state graph. Proofs and pseudocode appear in Appendices A and B.

4 Evaluation

We evaluate policy quality and compilation cost, measure the contributions of local closure and continuation bounds, and apply the same compiler across proposal models.

Workflows and proposals. Eight Vega source files (Vega project contributors, 2026) span transport, climate, demography, politics and finance. Source identities, preparation and operations were fixed before retrieval. Retaining at most the first 256 records per file gives 1,850 typed records from 17,681 source records (Appendix C). Each source supplies projection, row count, missing-value counts, grouped count, numeric summary, sorted rows, numeric filtering and union filtering, giving 64 requests.

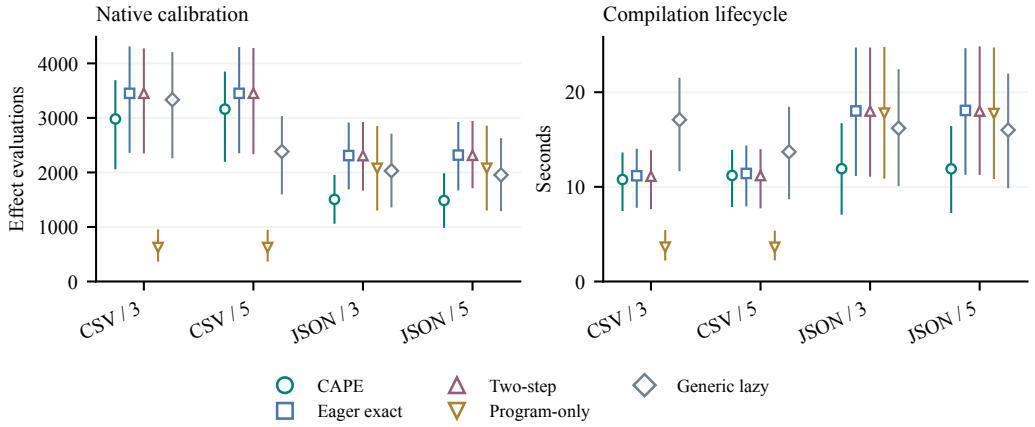

Figure 3: Native calibration and compilation lifecycle under empty caches. Lifecycle covers native setup, initial validation, acquisition, search, serialization and cleanup. Intervals resample the eight sources.

The mechanism cohort uses DeepSeek-V4-Pro at temperature zero with an 8,192-token completion allowance. Each prompt contains the request, typed schema, record count and first five records; each request receives one proposal. Of 64 responses, 63 pass the schema and match the requested program. The provider panel retains these responses and collects one per request from each of 13 other deployments, with settings in Appendix E. Every method receives the same unmodified proposal and requested-output guard.

Native interfaces. CSV/DuckDB and JSON/pandas expose 64 and 88 receiver configurations, respectively (DuckDB Contributors, 2026; The pandas development team, 2026), varying parsing, missing values, numeric/date encodings and JSON orientation. Programs serialize the proposed operation; probes elicit observations without publishing a result. The requested program determines the guard independently of the proposal. CSV configurations combine binary choices of delimiter, quote, escaping, null marker, decimal separator and date format. JSON configurations combine orientation and dtype handling with date conversion, floating-point parsing and date units. Policies observe responses and commit status without receiver identifiers.

Comparisons. All methods inherit the catalog-order trial policy and preserve its required identities. Optimized eager exact uses compact constrained search and shared terminal/support shortcuts, acquiring the remaining table when needed. Generic lazy search refines an optimistic effect model in batches of 32. Program-first expansion acquires program rows before diagnostic effects. Protected two-step retains the initializer whenever a proposed policy loses an initial success or decreases its value. Program-only search removes probes; its quality is checked against the same full-catalog optimum. The initializer tries programs in catalog order, charging its validation to compilation. Protected two-step considers two future probes with at least three calls remaining, one with two calls, and a terminal program with one. Its policy can execute the entire C-call allowance.

Two CAPE ablations retain native witnesses and acquisition priority. Full-row acquisition fills an action's unneeded receiver cells after closing its local scope. No-tightening acquisition replaces selective continuation bounds with the inexpensive support bound.

A separate shared-kernel study compares CAPE's acquisition frontier with matched lazy AND/OR search along the best optimistic solution path. Both share local witnesses, validation, caches, bounds and exact fallback. Each scheduler uses no tightening, selected-acquisition tightening, or always tightening with the same T_2 operator. This 2×3 control isolates acquisition order and bound computation (Appendix F).

Both schedulers prioritize required identities and share the acquisition stopping rule. Appendix F specifies the matched AND/OR path selection and all six configurations.

Table 2: Shared-kernel acquisition queries and lifecycle seconds on executable DeepSeek-V4-Pro proposals. Selective tightening is CAPE's default. The six configurations isolate acquisition order and bound computation. Figure 4 gives paired intervals; Appendix F reports policy quality and certification.

			CSV		JSON	
C	Scheduler	Tightening	Queries	Time	Queries	Time
3	CAPE	None	3330.3	11.489	1505.7	12.007
		Selective	2979.9	10.918	1511.9	11.961
		Always	2546.4	10.177	1505.0	12.012
	AND/OR	None	3278.6	11.315	1980.3	16.130
		Selective	3250.5	11.916	1990.5	16.161
		Always	3249.7	12.654	1975.3	16.235
5	CAPE	None	3161.1	11.374	1494.0	11.975
		Selective	3161.1	11.319	1490.5	11.929
		Always	3148.1	12.464	1473.9	12.005
	AND/OR	None	3163.4	11.148	1941.7	15.784
		Selective	3163.4	11.265	1949.4	15.728
		Always	3163.4	11.381	1932.6	15.699

Execution and accounting. We evaluate the frozen compiler on the source cohort used during method profiling. All studies use both interfaces, $C = 3/5$, empty caches, a soft 30-second compilation allowance and two fresh-process repetitions. The DeepSeek design has 4,032 native measurements and 64 pipeline records from its shared invalid proposal. The provider panel has 4,864 fresh measurements, shared across exactly matching inputs. Completion includes every proposal, with invalid proposals contributing zero. Native costs condition on executable proposals. The shared-kernel study reuses the retained panel inputs for 3648 fresh measurements; its 1824 instrumented profiles form a separate cohort.

Eight single-threaded workers use distinct physical cores, with affinities checked around timing. Each study randomizes the method queue. Lifecycle includes setup, initial validation, acquisition, search, serialization and cleanup, retaining overruns. CPU time and effect counts accompany wall time; proposal time and tokens are recorded separately.

Verification and statistical units. Post-compilation audits acquire complete native tables and use independent dynamic programming and fresh policy replays to check values, receipts, outputs and certificates. Each CAPE history is checked against its acquired cells, required identities and independent optimum. Runtime certification records a proof obtained during compilation; post hoc optimality compares the returned value with the independent complete-model solution. A policy can be optimal before its bounds meet. Full-model acquisition and fresh publication replay are charged to validation.

Each source contributes its mean over operations and repetitions, giving equal weight to the eight sources. Paired 95% intervals resample these source clusters, with configurations, shared inputs and repetitions nested within them. Counts of native inputs, action structures and replay policies describe the evaluated computational structures.

4.1 Execution quality and certified policy values

In the DeepSeek mechanism cohort, CAPE attains the independently checked full-catalog optimum in 502 of its 504 executable measurements, returning runtime certificates in 498. All audited policies preserve the common initializer's successful identities. The two nonoptimal measurements are JSON weather projection at $C = 3$: completion is optimal, but calls remain above the optimum when compilation stops.

Table 1 reports completion, execution calls and lifecycle costs. At $C = 3$, CSV completion reaches 85.16% for CAPE and 76.76% for program-only search. The corresponding JSON values are 94.74% and 94.60%.

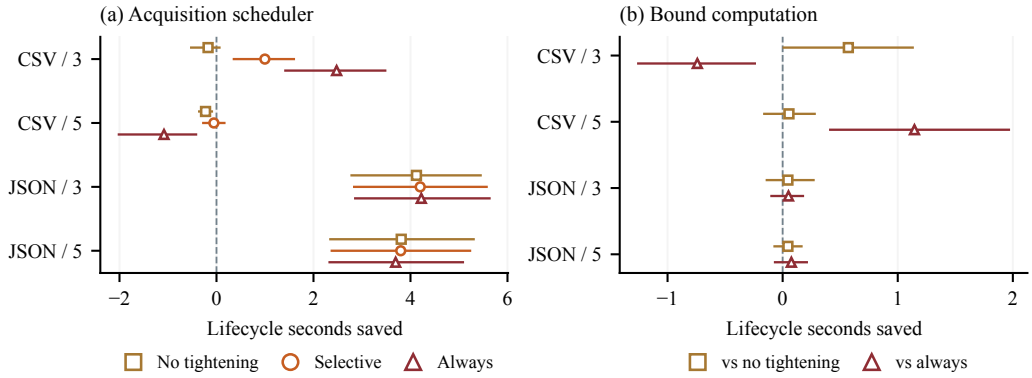

Figure 4: Shared-kernel study with DeepSeek-V4-Pro proposals. (a) Savings over matched AND/OR, holding the bound mode fixed. (b) Selective CAPE versus no tightening and always tightening. Positive values favor CAPE in (a) and selective computation in (b); intervals resample sources.

Relative to protected two-step lookahead at $C = 3$, CAPE's paired completion differences are 1.17 and 0.14 percentage points on CSV and JSON. The corresponding call savings are 0.278 and 0.017 calls per receiver. At CSV $C = 5$, both methods reach 98.44% completion, while CAPE saves 0.490 calls per receiver.

Tables 4–7 report policy quality, costs, runtime certificates and independently checked optimality for every method.

4.2 Acquisition and compilation cost

Figure 3 measures the work required to obtain these policies. CAPE's average lifecycle across the four interface/allowance strata is 11.449 seconds, compared with 14.673 for eager exact search and 15.749 for generic lazy search.

Against eager exact, paired lifecycle savings are 0.399 seconds on CSV $C = 3$ [0.041, 0.768], and 6.102/6.176 on JSON $C = 3/5$. CSV $C = 5$ saves 0.216 seconds [-0.125, 0.693], with the interval crossing zero. Figure 9 resolves differences by source.

Program-only can be cheaper when its action space already contains an optimal policy. The complete-model analysis below separates this condition from search interruption.

4.3 Acquisition mechanisms and computation allocation

Local scopes. The original mechanism cohort isolates scope closure and bound tightening (Figure 8). Relative to full-row acquisition, local closure saves 848.1 and 563.3 JSON effects at $C = 3/5$, improving completion by 4.26 and 3.69 percentage points. Unneeded cells consume resources that could establish another executable continuation. Appendix D.3 traces this process.

Development trajectories separate policy improvement from proof completion. On CSV projection, the feasible policy completes 56 receivers before the completion bounds meet. On JSON numeric summary, completion bounds meet before call bounds; subsequent local acquisition closes the call gap (Appendix D.3).

Acquisition order. The shared-kernel study separates scheduler and bound choices (Table 2 and Figure 4). With selective bounds in both schedulers, CAPE saves 0.998 seconds on CSV at $C = 3$, and 4.200/3.799 seconds on JSON at $C = 3/5$; their paired intervals exclude zero. CSV $C = 5$ has equal execution quality and a time interval containing zero. JSON savings persist under all three bound modes. On CSV, acquisition order and tightening interact: without tightening, $C = 3$ has no clear scheduler advantage and CAPE is slower at $C = 5$.

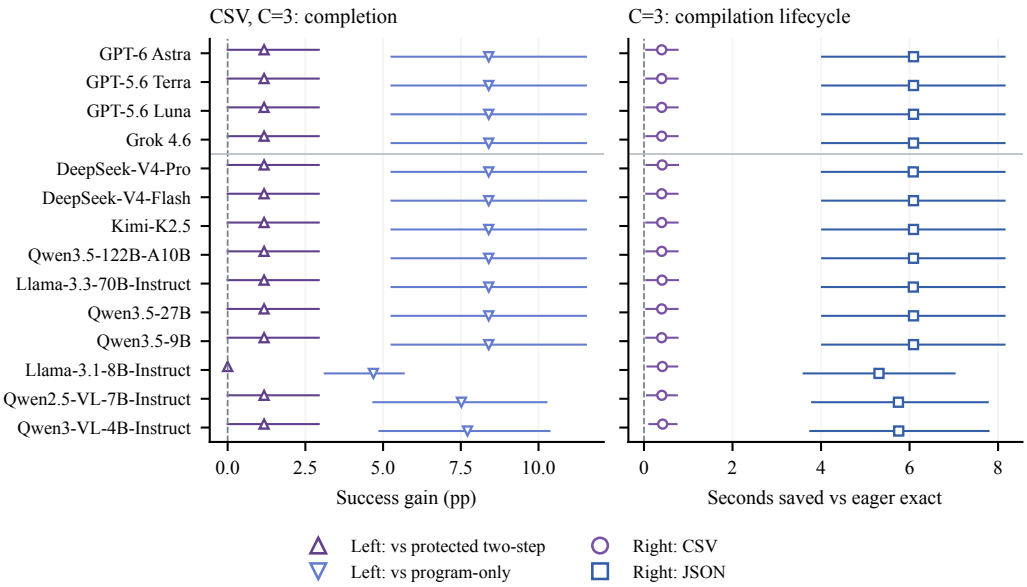

Figure 5: Provider-conditioned comparisons at $C = 3$. Positive values favor CAPE; intervals resample sources. Completion includes all proposals, while native costs condition on executable ones. The divider separates closed services from open-weight models.

Tightening cost. At CSV $C = 3$, always tightening is 0.741 seconds faster than selective computation at equal execution quality. It spends more time on T_2 but avoids 433.6 additional queries. At $C = 5$, selective computation is 1.145 seconds faster; extra tightening saves few queries. Both paired time intervals exclude zero. JSON differences are small, with intervals containing zero. Thus tightening pays when the calibration it avoids exceeds its computational cost. Separate profiles and all six configurations appear in Appendix F.

4.4 When diagnostics expand achievable execution

Independent complete-model optimization compares programs alone with programs and probes under the same required set and budget (Figure 2). Probes raise optimal CSV completion. JSON optima coincide, so timed gains reflect faster discovery of sufficient policies before compilation ends. The artifact also reports call differences at equal completion.

4.5 Across proposal models

Figure 5 compares 14 deployments: four closed services and ten open-weight models, including 4B, 7B and 8B variants. Of 896 proposals, 874 pass the schema and 862 agree with the typed reference. Exact duplicate program inputs share native measurements (Appendix E).

At CSV $C = 3$, 13 deployments gain 1.17 percentage points over protected two-step; Llama-3.1-8B-Instruct ties. At $C = 5$, completion ties while CSV calls decrease by 0.254 to 0.490 per receiver. JSON lifecycle falls by 29.7–34.0% relative to eager exact across both budgets (Appendix E).

5 Conclusion

CAPE compiles coverage-preserving policies through locally closed witnesses and acquisition around executable and optimistic continuations. Shared-kernel comparisons establish scheduling gains on CSV and JSON, while bound ablations expose the tradeoff between computation and avoided queries. The fourteen-model panel extends these results across proposal sources.

AI USE STATEMENT

Generative AI tools were used to improve writing clarity, refine language, and assist with literature search and organization.

REFERENCES

Siddhant Bhambri, Amrita Bhattacharjee, and Dimitri Bertsekas. Playing wordle using an online rollout algorithm for deterministic POMDPs. In *IEEE Conference on Games*, 2023. doi: 10.1109/CoG57401.2023.10333228.

Steven Carr, Nils Jansen, Sebastian Junges, and Ufuk Topcu. Safe reinforcement learning via shielding under partial observability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 14748–14756, 2023. doi: 10.1609/aaai.v37i12.26723. URL https://ojs.aaai.org/index.php/AAAI/article/view/26723.

Christopher M. Dellin and Siddhartha S. Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, 2016. doi: 10.1609/icaps.v26i1.13788. URL https://arxiv.org/abs/1603.03490.

DuckDB Contributors. CSV import documentation, 2026. URL https://duckdb.org/docs/current/data/csv/overview.html. Accessed September 10, 2026.

Wei Fang, Yang Zhang, Kaizhi Qian, James Glass, and Yada Zhu. PLAY2PROMPT: Zero-shot Tool Instruction Optimization for LLM Agents via Tool Play. *arXiv preprint arXiv:2503.14432*, 2025. URL http://arxiv.org/abs/2503.14432v2.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 440–448, 2020. doi: 10.1609/icaps.v30i1.6739. URL https://arxiv.org/abs/1802.08705.

Eric A. Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1–2):35–62, 2001. doi: 10.1016/S0004-3702(01)00106-0.

David Heckerman, John S. Breese, and Koos Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–57, 1995. doi: 10.1145/203330.203341.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool Documentation Enables Zero-Shot Tool-Usage with Large Language Models. *arXiv preprint arXiv:2308.00675*, 2023. URL http://arxiv.org/abs/2308.00675v1.

Shervin Javdani, Yuxin Chen, Amin Karbasi, Andreas Krause, J. Andrew Bagnell, and Siddhartha Srinivasa. Near Optimal Bayesian Active Learning for Decision Making. In *Proceedings of AISTATS*, volume 33 of *Proceedings of Machine Learning Research*, pp. 430–438, 2014. URL https://proceedings.mlr.press/v33/javdani14.html.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM Compiler for Parallel Function Calling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24370–24391, 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

Romain Laroche, Paul Trichelair, and Remi Tachet des Combes. Safe policy improvement with baseline bootstrapping. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3652–3661, 2019. URL https://proceedings.mlr.press/v97/laroche19a.html.

H. Brendan McMahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005. doi: 10.1145/1102351.1102423.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-Needed Decomposition and Planning with Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, 2024. doi: 10.18653/v1/2024.findings-naacl.264. URL https://aclanthology.org/2024.findings-naacl.264/.

Shili Sheng, David Parker, and Lu Feng. Safe POMDP online planning via shielding. *arXiv preprint arXiv:2309.10216*, 2023. URL https://arxiv.org/abs/2309.10216.

Xinyuan Song and Zekun Cai. Ask the World Before Acting: Environment Probing for Calibrated Agent World Models. *arXiv preprint arXiv:2606.31422*, 2026. URL http://arxiv.org/abs/2606.31422v2.

The pandas development team. pandas.read_json: Json ingestion. https://pandas.pydata.org/docs/reference/api/pandas.read_json.html, 2026. Behavior verified against the pandas 3.0.2 implementation.

Vega project contributors. Vega Datasets. https://github.com/vega/vega-datasets, 2026. Source revision 2434f551e0bb12b99a4ce6764fbc0ef39bea145e.

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 ICML*, volume 235 of *Proceedings of Machine Learning Research*, 2024. URL https://proceedings.mlr.press/v235/wang24h.html.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models. *arXiv preprint arXiv:2305.18323*, 2023. URL https://arxiv.org/abs/2305.18323v1.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629*, 2022. URL http://arxiv.org/abs/2210.03629v3.

A NATIVE WITNESSES AND PARTIAL-EFFECT BOUNDS

Model conditions. The receiver family, candidate actions and request are fixed during compilation and execution. Effects are deterministic, probes do not commit, and failed actions leave the state relevant to subsequent actions unchanged. Calibration and execution share observation and guard semantics. Acquired effects are immutable. The surviving receiver set and remaining calls therefore suffice to describe a policy state.

Local feasibility. For a closed action on (V, b), each required identity either belongs to $G_a(V)$ and commits immediately or belongs to exactly one failed-response cell V_o. In the latter case, its child succeeds because that child's obligation is $R_0 \cap V_o$. The parent consumes one call and the child uses at most $b - 1$. Disjoint cells give the success union and call sum in Equation 2, including unsuccessful paths. Stop is the feasible base case exactly when its required set is empty.

Persistence. Fix a native witness and consistent partial tables $D \subseteq D'$. Its first reached effect on any receiver is in D, so its observation and guard result are identical under both tables. Success terminates both paths; failure selects the same native continuation. Induction proves identical paths, success indicators and call counts. A parent's acquired history also fixes which identities reach that witness. Effects acquired outside its scope cannot change those paths.

Native witness propagation. The graph is acyclic in the remaining allowance. A backup composes feasible children through a closed action, producing another feasible witness. Sharing a state across alternatives does not count a receiver twice within one policy. Only strict improvements replace stored witnesses, and previous witnesses remain feasible under acquisition. The best root value therefore cannot decrease. This concerns lexicographic value; the protected identity set remains the original R_0. Lazily activated alternatives retain upper bounds even before their native children are explored.

Possible-support bound. Unknown program cells can add successes, so actual program support is contained in $\overline{G}(V)$ and the largest actual direct success count is at most $\overline{d}(V)$. A policy completing the full possible support starts on every identity and needs another call on at least $|\overline{G}(V)| - \overline{d}(V)$ successful identities. This proves Equation 4. A policy with fewer successes already has a smaller lexicographic value. Empty possible support permits Stop when nothing is required; required identities outside possible support imply infeasibility. At one call, feasible optimistic terminal programs can be enumerated exactly.

Optimistic effects. An unknown program cell is treated as an immediate success. Any policy from a consistent complete model can be simulated by terminating on such cells, preserving its successes and potentially saving calls. Acquired failures retain their native observation groups. For a probe, each unacquired receiver is placed in a distinct symbolic observation cell. Every possible native completion coarsens this partition. A finer observer can emulate a coarser policy by selecting the same continuation in each corresponding cell. The simulation preserves required identities. Combining admissible child bounds consequently yields an admissible action bound.

Two-call tightening. In a relaxed two-call policy, the first action is followed by a terminal program, or by Stop where its obligation is empty. Enumerating first actions and the best terminal program in each optimistic failure cell therefore computes that relaxation's exact value. The optimistic-effects argument shows that every native completion is dominated by some such relaxed policy. Thus T_2 is admissible. The possible-support bound also bounds this relaxed problem, so $T_2(V) \leq B(V, 2)$. Selectively substituting this tighter value for a child bound preserves admissibility at every ancestor.

Bound persistence and caching. Any earlier upper bound applies to every later consistent completion, so taking its minimum with a new bound remains valid. This also justifies retaining an action-pruning bound between search passes. A bound on V depends only on effects restricted to V. Let r_h count acquisitions on receiver h. Along a growing effect table, every r_h is nondecreasing. The sum $\sum_{h \in V} r_h$ is unchanged exactly when no new effect inside V has arrived. Using that sum as a cache generation is therefore sound for a fixed scope and required set. Program support

and closed terminal choices depend only on program effects and the fixed obligation, so changes to probe effects do not invalidate them. Once a native lower witness meets its valid upper bound, consistent later acquisitions cannot alter that optimum.

Native action reductions. Two acquired actions with the same success identities and failed-response partition are interchangeable: attach each continuation to the representative's native label for its cell. Coverage and call mass are preserved. A predictable noncommitting action can be replaced by its continuation, preserving successes and saving a call on every identity in its scope. The unchanged-state assumption makes this deletion valid. Reductions apply to acquired actions; unfinished actions are eliminated through admissible bounds.

Termination, interruption and certification. An acquisition either adds an effect, closes its selected scope, or retains a bound that rules out that action at the current state. A retained pruning bound cannot exclude an improving policy because a native lower witness already attains at least that value. Once all effects are known, the native recurrence is exact by induction on the remaining allowance. Equality of a root lower and upper value certifies the full-catalog optimum even before full acquisition. At the deadline, the compiler returns its latest feasible witness and certifies it when the bounds agree. If initial validation is interrupted, it returns the supplied playbook with certification unset.

Resources. There are at most $|H|(|P|+|Q|)$ distinct acquired effects, less any supplied cells. The policy-state space remains finite and can be exponential in $|H|$. Search and native calibration share the compilation allowance. Full-table audits and publication replay follow compilation as evaluation work. A program-only solution receives a full-catalog certificate only when it independently meets a bound for the full catalog.

B EXECUTABLE-POLICY FRONTIER ALGORITHM

Algorithm 1 CAPE with a native witness frontier

Require: Initial playbook π_0, receivers H, actions $P \cup Q$, path allowance C, partial table D, compilation allowance

Validate π_0, set $R_0 \leftarrow S(\pi_0; H, C)$, and import its native witnesses
Initialize the root lower witness and admissible upper bound
while compilation resources remain and the root bounds differ **do**
  if every effect in $A \times H$ is known **then**
    Solve the complete native model with the remaining allowance
    return The best validated policy and its resulting certification flag
  end if
  Propagate pending native witness and upper-bound updates
  Lazily activate a closed action along a best optimistic alternative
  Form the acquisition frontier from this path and the current executable policy
  Seed newly reached states using closed one-/two-call native witnesses
  Select an unfinished frontier action (a, V, b) by potential gain per missing effect
  if an unfinished action can improve its state's lower witness **then**
    repeat
      Acquire one missing (a, h), with required identities first
      Tighten this action's bound using T_2 for two-call continuation scopes
    until the action closes on V, is infeasible, cannot improve its native lower value, or resources expire
    Activate a newly closed action and update its native parents
  end if
end while
return The best executable root policy; certify it exactly when the root bounds agree

The acquisition priority is lexicographic. A state with no feasible required witness receives first priority, ranked by required identities per missing cell. Otherwise, positive success gain precedes

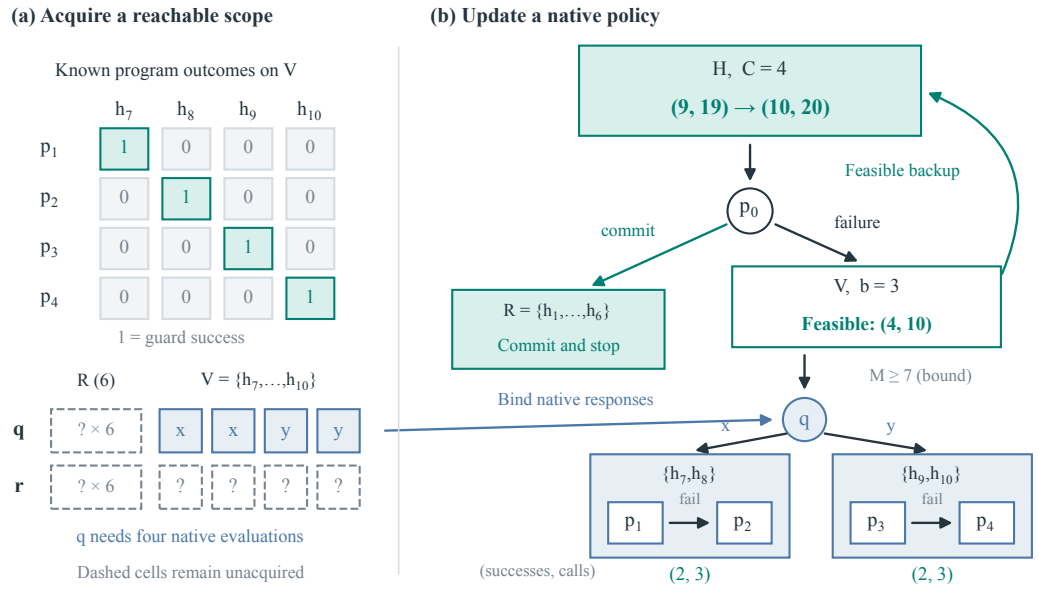

Figure 6: In this ten-receiver illustration ($C = 4$), the one-call initializer p_0 defines the six required identities R. A current feasible trial policy completes nine receivers in 19 calls. On the four remaining identities V, probe q closes after four calibration queries and forms the pairs $\{h_7, h_8\}$ and $\{h_9, h_{10}\}$. Program chains complete both pairs, producing a root with ten successes and 20 calls while unacquired effects keep its upper bound open. Displayed pairs are (successes, total calls); successful programs terminate their paths.

call-only improvement. Completion gain and saved calls are normalized by missing-cell count. Smaller missing scopes break remaining ties, followed by program actions and lower catalog indices.

Worked native policy. Figure 6 illustrates local closure and feasible witness composition on ten receivers, with exact success and call counts.

C Native workflow and proposal details

Receiver families. CSV receivers combine six binary DuckDB reader options: comma or semicolon delimiter, single or double quote, backslash or doubled-quote escaping, `NULL` or `NA` null marker, comma or point decimal separator, and month-first or day-first date format. Their Cartesian product gives 64 configurations. JSON receivers use pandas' records, split, index, columns, values and table orientations. Each has inferred or preserved dtypes except table, which uses its schema. These 11 reading modes combine with date conversion on/off, precise floating-point parsing on/off, and millisecond/nanosecond date units, giving 88 configurations. The gateway sets these options administratively. Deployed policies receive native observations and commit status without receiver IDs.

Sources and preparation. Table 3 reports the application source files and retained prefixes. Source identities were fixed before content retrieval. Weather and finance also appear in earlier native workflow inputs.

CSV decoding infers scalar types over each complete source column before selecting its prefix. Empty fields become missing values. Columns that permit exact signed 64-bit integers retain them; finite numeric conversion also requires preserving any integer values. Leading-zero numeral strings and whitespace remain strings. JSON input uses strict parsing. The common preparer selects the first at most 256 records, expands nested paths and lists of at most 16 elements, and stores remaining

Table 3: Original inputs and the fixed retained prefixes. All eight operations use the same prepared rows of a source.

Source	Original rows	Retained rows	Fields
Airports	3,376	256	7
Bird strikes	10,000	256	14
CO_2 concentration	741	256	3
Disasters	803	256	3
Development indicators	682	256	6
Political contributions	58	58	25
Seattle weather	1,461	256	6
Stock prices	560	256	3

containers as canonical JSON text. The typed schema is determined on the retained records. An internal row index is added and excluded from the reported field counts.

Operations. Projection requests all prepared source columns in their fixed order. Row count returns the batch size; missing-value counts cover each source column. Grouped count uses the first string column, falling back to the first source column, and includes missing groups. Keys are sorted by canonical JSON representation. Numeric summaries return nonmissing count, minimum, maximum and sum for every numeric source column, with zero for an empty sum. Sorted rows return the first 20 rows ordered by the selected field and then original row order, placing missing keys last.

Numeric filtering returns original row identifiers whose selected numeric field exceeds its lower median. Union filtering adds a category predicate equal to the most frequent nonmissing value. Numeric fields with multiple distinct values are preferred; dates are preferred for sorting. Field selection follows schema order, with canonical JSON order breaking category frequency ties. These rules specify the request and reference operation before model sampling.

Proposal failure. The 63 valid DeepSeek-V4-Pro outputs match the requested syntax and typed reference. The remaining output, for numeric filtering on the development-indicator source, combines sorting fields with an unsupported predicate field and fails the declared schema, contributing a pipeline failure for every method. These requests consumed 54,216 reported input and completion tokens and incurred two rate-limit retries. Model settings and proposal counts for the full panel appear in Appendix E.

Control implementation. The controls are defined in Section 4. Protected two-step uses rolling diagnostic lookahead with a program-prefix fallback. Eager exact and program-first expansion share terminal/support shortcuts; program-only certification is checked against a full-catalog bound.

Execution and timing. The environment uses Python 3.12.13, pandas 3.0.2 and DuckDB 1.5.5, with Python hash seed 2070. Native/BLAS query threads are limited to one. Eight fresh-process workers are assigned distinct physical cores through a fixed randomized queue. DuckDB's idle default pool is reduced before measurement, and new connections disable automatic thread pinning and use one query thread from creation. Every live thread's allowed CPUs are checked before and after the timed lifecycle.

Interpreter startup and model generation are outside the timed lifecycle defined in Section 4. Atomic operations and cleanup can overrun the soft 30-second allowance; actual durations are retained.

Verification. Post-compilation audits acquire complete native tables and reexecute the probe libraries. Timed methods begin with empty effect caches. Initial and final policies are replayed with fresh publication state. Observed effects, attempted requests, call counts and committed artifacts must match the recorded policies. An independent complete-model dynamic program checks full-catalog values. Runtime certificates must agree with that optimum. CAPE's saved history additionally checks that every reached effect was already available, that R_0 is preserved, and that the independent optimum lies between its native lower value and recorded upper bound.

Table 4: CSV receivers, $C = 3$. All values are source-balanced.

Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CAPE	85.16	2.196	2980.5	10.774	98.44	100.00
Eager exact	85.16	2.196	3450.9	11.173	98.44	100.00
Program-only	76.76	2.022	624.0	3.631	34.38	35.04
Protected two-step	83.98	2.474	3450.9	11.121	34.38	35.04
Generic lazy	83.74	2.198	3332.5	17.089	93.75	95.31
Program-first expansion	85.16	2.196	3307.3	11.402	98.44	100.00
Full-row acquisition	85.16	2.196	3243.6	11.311	98.44	100.00
No tight bounds	85.16	2.196	3330.3	11.374	98.44	100.00

Table 5: CSV receivers, $C = 5$. Time includes the complete native compilation lifecycle.

Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CAPE	98.44	2.348	3161.1	11.201	98.44	100.00
Eager exact	98.44	2.348	3450.9	11.417	98.44	100.00
Program-only	92.19	2.365	624.0	3.624	34.38	75.00
Protected two-step	98.44	2.838	3450.9	11.192	34.38	70.31
Generic lazy	84.18	2.534	2382.0	13.700	73.44	75.00
Program-first expansion	98.44	2.348	3191.6	11.572	98.44	100.00
Full-row acquisition	98.44	2.348	3161.0	11.177	98.44	100.00
No tight bounds	98.44	2.348	3161.1	11.217	98.44	100.00

Table 6: JSON receivers, $C = 3$. Effect evaluations are physical calibration queries.

Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CAPE	94.74	1.476	1506.2	11.919	95.31	98.44
Eager exact	94.60	1.474	2309.0	18.021	84.38	98.44
Program-only	94.60	1.474	2075.7	17.792	77.34	98.44
Protected two-step	94.60	1.493	2307.2	18.008	77.34	91.96
Generic lazy	92.90	1.495	2029.5	16.200	93.75	95.31
Program-first expansion	94.60	1.474	2170.0	17.878	85.94	98.44
Full-row acquisition	90.48	1.430	2354.3	18.003	85.16	86.72
No tight bounds	94.74	1.476	1516.4	11.921	95.31	98.44

Statistical accounting. Intervals use 5,000 bootstrap resamples of the eight source means, with seed 38117 and the aggregation defined in Section 4. A shared schema failure contributes a zero paired completion difference; unavailable costs remain missing. Quality win/tie/loss counts apply to executable policies.

Reproduction. The artifact retains source bytes, preparation records, model requests and responses, transport outcomes, effect ledgers, policies and numerical evidence. Its offline checks recompute statistics and replay policies from retained proposals. Figures and tables use the audited records; recorded compilation times remain distinct from fresh replay costs.

D COMPLETE NATIVE COMPARISONS

Tables 4–7 report every method under the same native action catalog and initial contract. Success is pipeline completion, including the shared proposal failure. Calls, effect queries and lifecycle time are conditioned on executable proposals. Certification includes every attempted proposal. The last column reports attaining the independently checked full-catalog optimum among executable cases. Runtime certification records a proof obtained during compilation; post hoc optimality records the independent solver's comparison with the returned value.

Independent diagnostic headroom. For each audited native model and initial required set, a second complete-model optimization removes probes while retaining program observations and the

Table 7: JSON receivers, $C = 5$. Runtime certification and independently checked optimality.

Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CAPE	94.74	1.475	1485.9	11.903	96.88	100.00
Eager exact	94.60	1.474	2317.8	18.079	84.38	98.44
Program-only	94.60	1.474	2077.9	17.744	76.56	98.44
Protected two-step	94.60	1.493	2314.1	18.005	77.34	91.96
Generic lazy	92.90	1.546	1954.9	16.008	93.75	95.31
Program-first expansion	94.60	1.474	2174.5	17.919	83.59	98.44
Full-row acquisition	91.05	1.435	2049.3	17.036	86.72	89.84
No tight bounds	94.74	1.475	1486.2	11.862	96.88	100.00

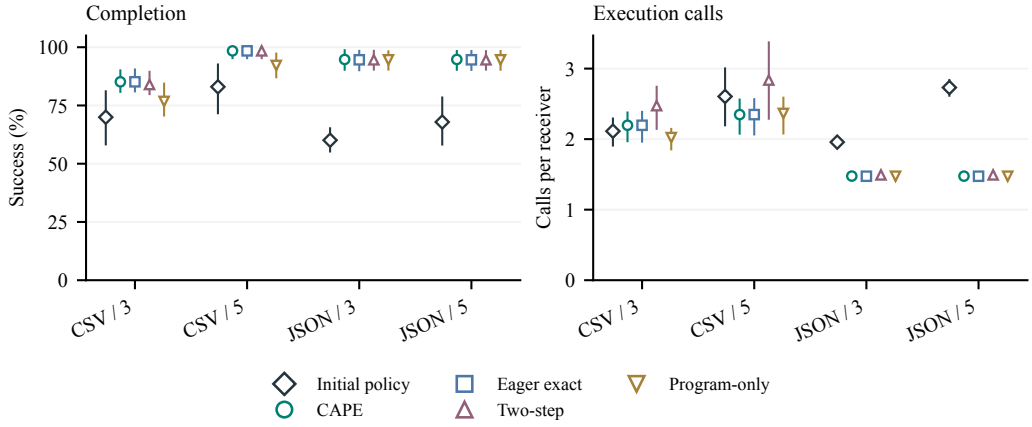

Figure 7: Application policy quality. Completion includes all 64 proposals; calls condition on executable proposals. Labels give the native interface and path allowance. Points and intervals summarize the eight source clusters.

execution allowance. Comparing this restricted optimum with the full-catalog optimum separates an action-space benefit from runtime search interruption. Success headroom is the difference in completed receivers divided by the receiver-family size. Call savings are compared separately when optimal completion is equal.

Acquisition evidence. Local acquisition counts chunks that query at least one effect on a strict descendant scope. In the DeepSeek cohort, 23180 of 41626 acquisition chunks query strict descendant scopes.

D.1 Policy quality and acquisition ablations

Figure 7 shows source-cluster intervals for the completion and call means summarized in Table 1. Figure 8 isolates local closure and bound tightening in the DeepSeek mechanism cohort.

D.2 Source diversity and native execution

The DeepSeek cohort produces 126 complete native inputs and 24 normalized action structures. These counts distinguish new workload contents from repeated observation and success patterns. Figure 9 retains all eight source files and separates their interface and path-allowance conditions.

Independent replay covers 586 distinct initial/returned policy pairs, 83,744 native episodes and 195,798 actual tool calls. Identical paired specifications share replay evidence while retaining their separate timing records. Complete native optimization resolves 252 of 252 model/required-set/allowance objectives. The acquired-cell audit additionally checks 8970 saved native witnesses and their upper values.

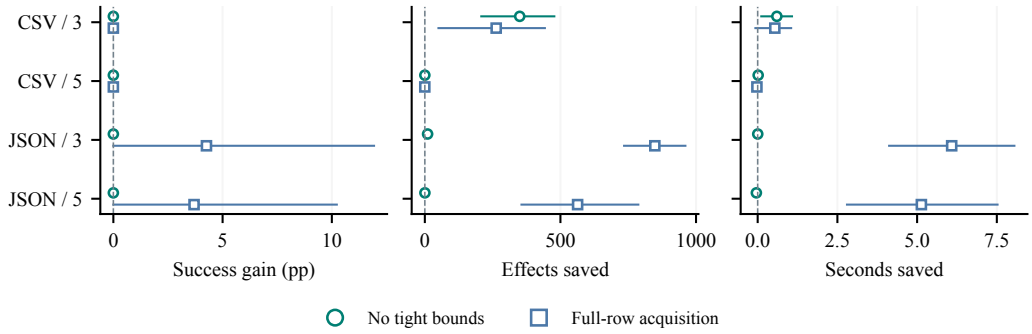

Figure 8: DeepSeek mechanism cohort: matched scope and bound ablations. Positive differences favor CAPE. Completion includes every proposal; query and time differences condition on executable proposals.

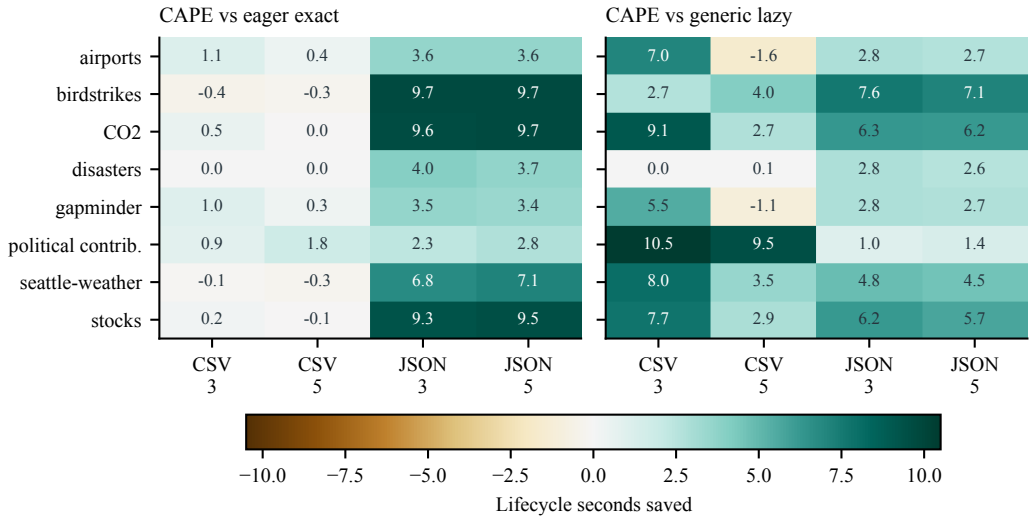

Figure 9: Per-source lifecycle differences against eager exact and generic lazy search. Each cell averages the source's valid operations and two repetitions. Values are signed seconds saved, with positive values favoring CAPE.

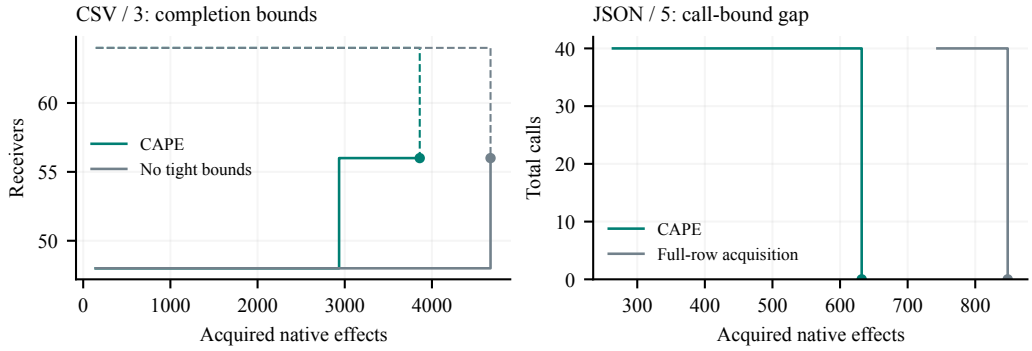

Figure 10: Acquisition traces from development: budgets/projection on CSV at $C = 3$, and flights/numeric summary on JSON at $C = 5$. Left: solid native completion and dashed optimistic completion. Right: call-bound gap, shown where completion bounds agree. Dots indicate exact closure. Application aggregate results use the separate source cohort.

Table 8: Official model names and fixed reasoning settings. The completion allowance is 8,192 tokens. Default denotes an omitted reasoning option; returned model identifiers are retained in the raw records.

Model	Reasoning setting
GPT-6 Astra	low
GPT-5.6 Terra	low
GPT-5.6 Luna	low
Grok 4.6	default
DeepSeek-V4-Pro	default
DeepSeek-V4-Flash	default
Kimi-K2.5	none
Qwen3.5-122B-A10B	nonthinking
Llama-3.3-70B-Instruct	default
Qwen3.5-27B	nonthinking
Qwen3.5-9B	nonthinking
Llama-3.1-8B-Instruct	default
Qwen2.5-VL-7B-Instruct	default
Qwen3-VL-4B-Instruct	default

D.3 Acquisition trajectories

Figure 10 shows two development traces. On CSV, native completion reaches 56 before proof: tightening certifies it after 3860 effects, versus 4672 without tightening. Remaining queries rule out better hypothetical policies. On JSON, completion bounds agree before call bounds; local acquisition closes the call gap after 632 effects, versus 848 with full-row acquisition.

E Provider configurations and complete comparisons

Panel and request configuration. Tables 8 and 9 list all fourteen models, settings and proposal outcomes. Model-specific licenses and public model-card revisions are retained in the artifact; closed-model parameter counts are undisclosed. The VL models receive the same text-only tasks as the other deployments.

Each request uses the program schema, typed fields, record count and first five records, with an 8,192-token completion allowance. GPT deployments use low reasoning and default sampling settings. Qwen3.5 uses its nonthinking option and temperature zero; Kimi-K2.5 uses its nonreasoning option. Other deployments use temperature zero. Payloads and actual returned identifiers are retained. The panel evaluates these fixed deployment configurations.

Table 9: Proposal results for all 14 deployments. Every row has 64 requests. Reference agreement evaluates the proposed program on the typed source data. Tokens include reported input and completion usage.

Model	Access	Valid / 64	Reference / 64	Tokens
GPT-6 Astra	Closed	64	64	52,099
GPT-5.6 Terra	Closed	64	64	52,291
GPT-5.6 Luna	Closed	64	64	53,843
Grok 4.6	Closed	64	64	87,453
DeepSeek-V4-Pro	Open	63	63	54,216
DeepSeek-V4-Flash	Open	64	64	53,673
Kimi-K2.5	Open	64	64	52,926
Qwen3.5-122B-A10B	Open	64	64	57,984
Llama-3.3-70B-Instruct	Open	62	62	54,697
Qwen3.5-27B	Open	64	64	57,461
Qwen3.5-9B	Open	64	64	57,700
Llama-3.1-8B-Instruct	Open	47	41	57,619
Qwen2.5-VL-7B-Instruct	Open	62	59	56,554
Qwen3-VL-4B-Instruct	Open	64	61	56,337

The panel retains the original 64 DeepSeek-V4-Pro responses and adds 832 new model requests. The new requests report 750,637 tokens. Each completed response supplies the final proposal for its request, including format and semantic failures. Transport failures and retries are recorded alongside the model outcomes. The collector retains 18 transport errors and 2 interrupted requests with no observed response; usage on those interrupted requests is unknown.

Exact shared inputs and fresh measurements. Deduplication requires matching task bytes, batch bytes, schema, requested program, proposed program, backend and call allowance. These exact inputs determine native effect acquisition and policy search; provider labels identify the corresponding proposals. Every model/request binding is checked against its representative proposal and native jobs. All distinct inputs receive fresh measurements under the shared configuration in Section 4.

There are 4,864 physical compilations and 57,344 provider-bound records. The expanded records attach model labels to shared compilations and include pipeline failures. The native audit contains 152 input models, 33 normalized action structures and 655 distinct policy pairs. It resolves 304/304 optimal-value problems and checks 10,225 acquired-cell witnesses through separate native replay evidence.

Proposal quality and policy quality. Overall completion measures request fulfillment across every model proposal. Native costs condition on schema-valid programs, and policy optimality is evaluated for each proposed operation. An incorrect operation can have an optimum of zero completed receivers. We also report reference-agreeing subsets, where CAPE attains the independent optimum in 526/528 physical compilations with reference-agreeing proposals. Every source has equal weight within a model's reported condition. Shared compilations and timing repetitions remain within source clusters when constructing intervals.

Complete tables without duplicate measurements. Nine models produce the same executable inputs for all 64 requests and therefore share every native measurement. Their complete results are shown once in Table 10, with all nine model names in its caption. Each remaining model has a separate table. Every table retains all eight methods, both interfaces, both call allowances and all reported metrics. The artifact also retains each model's individual table and proposal records.

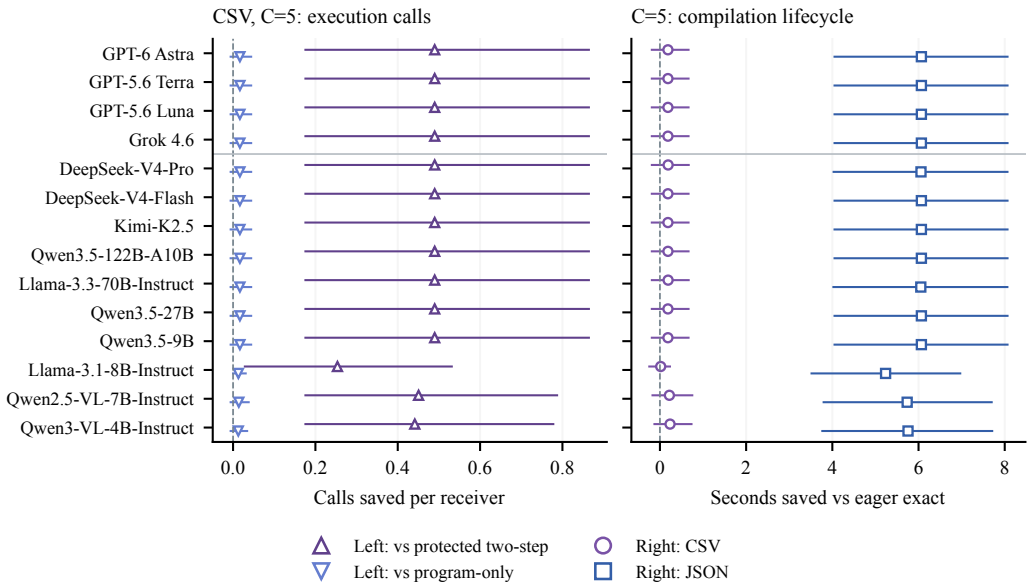

Figure 11: Execution-call and compilation-time comparisons at $C = 5$. Positive differences favor CAPE. Call differences are interpreted alongside completion in the complete tables, since completing more receivers can also consume more calls. Intervals resample the source clusters.

Table 10: Shared native results for GPT-6 Astra, GPT-5.6 Terra, GPT-5.6 Luna, Grok 4.6, DeepSeek-V4-Flash, Kimi-K2.5, Qwen3.5-122B-A10B, Qwen3.5-27B, Qwen3.5-9B. All methods, interfaces and path allowances; these models share exact inputs and physical measurements.

Interface, C	Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CSV, 3	CAPE	86.72	2.188	2948.4	10.655	100.00	100.00
CSV, 3	Eager exact	86.72	2.188	3412.0	11.056	100.00	100.00
CSV, 3	Program-only	78.32	2.016	624.0	3.611	35.94	35.94
CSV, 3	Protected two-step	85.55	2.461	3412.0	11.004	35.94	35.94
CSV, 3	Generic lazy	85.21	2.188	3293.3	16.868	94.53	94.53
CSV, 3	Program-first expansion	86.72	2.188	3268.5	11.268	100.00	100.00
CSV, 3	Full-row acquisition	86.72	2.188	3204.5	11.131	100.00	100.00
CSV, 3	No tight bounds	86.72	2.188	3291.0	11.191	100.00	100.00
CSV, 5	CAPE	100.00	2.338	3124.6	11.060	100.00	100.00
CSV, 5	Eager exact	100.00	2.338	3412.0	11.246	100.00	100.00
CSV, 5	Program-only	93.75	2.354	624.0	3.618	35.94	75.00
CSV, 5	Protected two-step	100.00	2.828	3412.0	11.051	35.94	70.31
CSV, 5	Generic lazy	85.74	2.524	2363.8	13.618	75.00	75.00
CSV, 5	Program-first expansion	100.00	2.338	3155.6	11.361	100.00	100.00
CSV, 5	Full-row acquisition	100.00	2.338	3124.4	11.019	100.00	100.00
CSV, 5	No tight bounds	100.00	2.338	3124.6	11.062	100.00	100.00
JSON, 3	CAPE	96.31	1.475	1502.5	11.800	96.88	98.44
JSON, 3	Eager exact	96.16	1.473	2319.9	17.889	85.16	98.44
JSON, 3	Program-only	96.16	1.473	2079.5	17.639	78.91	98.44
JSON, 3	Protected two-step	96.09	1.491	2311.7	17.910	78.91	91.41
JSON, 3	Generic lazy	94.46	1.494	2026.2	16.010	95.31	95.31
JSON, 3	Program-first expansion	96.16	1.473	2165.1	17.764	85.94	98.44
JSON, 3	Full-row acquisition	92.05	1.428	2363.2	17.851	88.28	88.28
JSON, 3	No tight bounds	96.31	1.475	1510.9	11.845	96.88	98.44
JSON, 5	CAPE	96.31	1.474	1489.0	11.837	99.22	100.00
JSON, 5	Eager exact	96.16	1.473	2316.4	17.904	85.16	98.44
JSON, 5	Program-only	96.16	1.473	2081.3	17.633	78.12	98.44
JSON, 5	Protected two-step	96.09	1.491	2311.7	17.899	79.69	91.41
JSON, 5	Generic lazy	94.46	1.545	1952.8	15.787	95.31	95.31
JSON, 5	Program-first expansion	96.16	1.473	2174.6	17.769	86.72	98.44
JSON, 5	Full-row acquisition	92.61	1.434	2044.8	16.916	88.28	89.84
JSON, 5	No tight bounds	96.31	1.474	1493.0	11.803	100.00	100.00

Table 11: DeepSeek-V4-Pro: all methods and both interfaces and path allowances.

Interface, C	Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CSV, 3	CAPE	85.16	2.196	2980.5	10.749	98.44	100.00
CSV, 3	Eager exact	85.16	2.196	3450.9	11.159	98.44	100.00
CSV, 3	Program-only	76.76	2.022	624.0	3.614	34.38	35.04
CSV, 3	Protected two-step	83.98	2.474	3450.9	11.110	34.38	35.04
CSV, 3	Generic lazy	83.64	2.197	3332.7	17.040	92.97	94.53
CSV, 3	Program-first expansion	85.16	2.196	3307.3	11.380	98.44	100.00
CSV, 3	Full-row acquisition	85.16	2.196	3243.6	11.238	98.44	100.00
CSV, 3	No tight bounds	85.16	2.196	3330.3	11.302	98.44	100.00
CSV, 5	CAPE	98.44	2.348	3161.1	11.160	98.44	100.00
CSV, 5	Eager exact	98.44	2.348	3450.9	11.350	98.44	100.00
CSV, 5	Program-only	92.19	2.365	624.0	3.619	34.38	75.00
CSV, 5	Protected two-step	98.44	2.838	3450.9	11.155	34.38	70.31
CSV, 5	Generic lazy	84.18	2.534	2384.9	13.702	73.44	75.00
CSV, 5	Program-first expansion	98.44	2.348	3191.6	11.463	98.44	100.00
CSV, 5	Full-row acquisition	98.44	2.348	3161.0	11.122	98.44	100.00
CSV, 5	No tight bounds	98.44	2.348	3161.1	11.164	98.44	100.00
JSON, 3	CAPE	94.74	1.476	1507.5	11.815	95.31	98.44
JSON, 3	Eager exact	94.60	1.474	2326.9	17.899	83.59	98.44
JSON, 3	Program-only	94.60	1.474	2079.5	17.642	77.34	98.44
JSON, 3	Protected two-step	94.53	1.492	2318.8	17.922	77.34	91.18
JSON, 3	Generic lazy	92.90	1.495	2031.7	16.026	93.75	95.31
JSON, 3	Program-first expansion	94.60	1.474	2169.4	17.769	84.38	98.44
JSON, 3	Full-row acquisition	90.48	1.429	2367.0	17.859	86.72	88.28
JSON, 3	No tight bounds	94.74	1.476	1515.9	11.863	95.31	98.44
JSON, 5	CAPE	94.74	1.475	1494.8	11.854	97.66	100.00
JSON, 5	Eager exact	94.60	1.474	2323.5	17.911	83.59	98.44
JSON, 5	Program-only	94.60	1.474	2081.3	17.636	76.56	98.44
JSON, 5	Protected two-step	94.53	1.492	2318.8	17.910	78.12	91.18
JSON, 5	Generic lazy	92.90	1.546	1958.4	15.805	93.75	95.31
JSON, 5	Program-first expansion	94.60	1.474	2179.3	17.779	85.16	98.44
JSON, 5	Full-row acquisition	91.05	1.435	2052.7	16.929	86.72	89.84
JSON, 5	No tight bounds	94.74	1.475	1498.8	11.811	98.44	100.00

Table 12: Llama-3.3-70B-Instruct: all methods and both interfaces and path allowances.

Interface, C	Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CSV, 3	CAPE	83.59	2.196	2980.5	10.750	96.88	100.00
CSV, 3	Eager exact	83.59	2.196	3450.9	11.158	96.88	100.00
CSV, 3	Program-only	75.20	2.022	624.0	3.614	32.81	35.04
CSV, 3	Protected two-step	82.42	2.474	3450.9	11.111	32.81	35.04
CSV, 3	Generic lazy	82.08	2.197	3332.7	17.040	91.41	94.53
CSV, 3	Program-first expansion	83.59	2.196	3307.3	11.382	96.88	100.00
CSV, 3	Full-row acquisition	83.59	2.196	3243.6	11.238	96.88	100.00
CSV, 3	No tight bounds	83.59	2.196	3330.3	11.304	96.88	100.00
CSV, 5	CAPE	96.88	2.348	3161.1	11.160	96.88	100.00
CSV, 5	Eager exact	96.88	2.348	3450.9	11.350	96.88	100.00
CSV, 5	Program-only	90.62	2.365	624.0	3.620	32.81	75.00
CSV, 5	Protected two-step	96.88	2.838	3450.9	11.154	32.81	70.31
CSV, 5	Generic lazy	82.62	2.534	2384.9	13.702	71.88	75.00
CSV, 5	Program-first expansion	96.88	2.348	3191.6	11.463	96.88	100.00
CSV, 5	Full-row acquisition	96.88	2.348	3161.0	11.123	96.88	100.00
CSV, 5	No tight bounds	96.88	2.348	3161.1	11.165	96.88	100.00
JSON, 3	CAPE	93.18	1.476	1507.2	11.812	93.75	98.44
JSON, 3	Eager exact	93.04	1.474	2326.9	17.896	82.03	98.44
JSON, 3	Program-only	93.04	1.474	2079.5	17.641	75.78	98.44
JSON, 3	Protected two-step	92.97	1.492	2318.8	17.919	75.78	91.18
JSON, 3	Generic lazy	91.34	1.495	2031.7	16.024	92.19	95.31
JSON, 3	Program-first expansion	93.04	1.474	2169.4	17.769	82.81	98.44
JSON, 3	Full-row acquisition	88.92	1.429	2367.0	17.856	85.16	88.28
JSON, 3	No tight bounds	93.18	1.476	1515.6	11.860	93.75	98.44
JSON, 5	CAPE	93.18	1.475	1494.7	11.855	96.09	100.00
JSON, 5	Eager exact	93.04	1.474	2323.5	17.907	82.03	98.44
JSON, 5	Program-only	93.04	1.474	2081.3	17.635	75.00	98.44
JSON, 5	Protected two-step	92.97	1.492	2318.8	17.908	76.56	91.18
JSON, 5	Generic lazy	91.34	1.546	1958.4	15.802	92.19	95.31
JSON, 5	Program-first expansion	93.04	1.474	2179.3	17.776	83.59	98.44
JSON, 5	Full-row acquisition	89.49	1.435	2052.7	16.930	85.16	89.84
JSON, 5	No tight bounds	93.18	1.475	1498.8	11.812	96.88	100.00

Table 13: Llama-3.1-8B-Instruct: all methods and both interfaces and path allowances.

Interface, C	Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CSV, 3	CAPE	57.81	1.798	2327.5	8.728	73.44	100.00
CSV, 3	Eager exact	57.81	1.798	2695.7	9.144	73.44	100.00
CSV, 3	Program-only	53.12	1.666	624.0	3.647	39.06	52.40
CSV, 3	Protected two-step	57.81	2.001	2695.7	9.123	39.06	52.40
CSV, 3	Generic lazy	56.84	1.801	2581.0	13.167	70.31	96.43
CSV, 3	Program-first expansion	57.81	1.798	2680.4	9.596	73.44	100.00
CSV, 3	Full-row acquisition	57.81	1.798	2609.7	9.380	73.44	100.00
CSV, 3	No tight bounds	57.81	1.798	2645.5	9.405	73.44	100.00
CSV, 5	CAPE	64.06	1.894	2547.4	9.265	73.44	100.00
CSV, 5	Eager exact	64.06	1.894	2695.7	9.282	73.44	100.00
CSV, 5	Program-only	61.52	1.906	624.0	3.651	39.06	86.01
CSV, 5	Protected two-step	64.06	2.147	2695.7	9.156	39.06	84.23
CSV, 5	Generic lazy	59.18	2.017	1875.8	10.467	65.62	86.01
CSV, 5	Program-first expansion	64.06	1.894	2581.4	9.526	73.44	100.00
CSV, 5	Full-row acquisition	64.06	1.894	2547.3	9.272	73.44	100.00
CSV, 5	No tight bounds	64.06	1.894	2547.4	9.264	73.44	100.00
JSON, 3	CAPE	62.43	1.248	1550.0	12.329	73.44	100.00
JSON, 3	Eager exact	62.29	1.247	2266.0	17.638	63.28	98.21
JSON, 3	Program-only	62.29	1.247	2080.8	17.494	60.16	98.21
JSON, 3	Protected two-step	62.29	1.268	2263.2	17.651	60.16	92.49
JSON, 3	Generic lazy	61.72	1.256	2007.8	15.904	71.88	98.21
JSON, 3	Program-first expansion	62.29	1.247	2112.3	17.492	63.28	98.21
JSON, 3	Full-row acquisition	59.16	1.211	2268.1	17.657	64.84	90.18
JSON, 3	No tight bounds	62.43	1.248	1551.0	12.390	73.44	100.00
JSON, 5	CAPE	62.43	1.248	1507.7	12.402	73.44	100.00
JSON, 5	Eager exact	62.29	1.247	2259.8	17.638	63.28	98.21
JSON, 5	Program-only	62.29	1.247	2082.1	17.466	59.38	98.21
JSON, 5	Protected two-step	62.29	1.268	2262.7	17.624	60.94	92.49
JSON, 5	Generic lazy	61.72	1.273	1942.6	15.782	71.88	98.21
JSON, 5	Program-first expansion	62.29	1.247	2115.2	17.496	64.84	98.21
JSON, 5	Full-row acquisition	59.73	1.217	1986.2	16.801	66.41	91.96
JSON, 5	No tight bounds	62.43	1.248	1507.7	12.422	73.44	100.00

Table 14: Qwen2.5-VL-7B-Instruct: all methods and both interfaces and path allowances.

Interface, C	Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CSV, 3	CAPE	80.47	2.060	2752.8	10.046	96.88	100.00
CSV, 3	Eager exact	80.47	2.060	3188.6	10.449	96.88	100.00
CSV, 3	Program-only	72.95	1.903	624.0	3.596	39.06	41.07
CSV, 3	Protected two-step	79.30	2.314	3188.6	10.401	39.06	41.07
CSV, 3	Generic lazy	79.39	2.061	3085.9	15.851	92.97	95.98
CSV, 3	Program-first expansion	80.47	2.060	3045.0	10.612	96.88	100.00
CSV, 3	Full-row acquisition	80.47	2.060	2981.7	10.486	96.88	100.00
CSV, 3	No tight bounds	80.47	2.060	3067.1	10.549	96.88	100.00
CSV, 5	CAPE	92.19	2.199	2909.1	10.430	96.88	100.00
CSV, 5	Eager exact	92.19	2.199	3188.6	10.649	96.88	100.00
CSV, 5	Program-only	86.82	2.213	624.0	3.613	39.06	77.46
CSV, 5	Protected two-step	92.19	2.650	3188.6	10.452	39.06	72.77
CSV, 5	Generic lazy	79.98	2.360	2229.5	12.839	75.00	77.46
CSV, 5	Program-first expansion	92.19	2.199	2940.5	10.665	96.88	100.00
CSV, 5	Full-row acquisition	92.19	2.199	2908.9	10.388	96.88	100.00
CSV, 5	No tight bounds	92.19	2.199	2909.1	10.434	96.88	100.00
JSON, 3	CAPE	88.78	1.412	1538.7	12.130	92.19	98.21
JSON, 3	Eager exact	88.64	1.409	2324.0	17.879	82.03	98.44
JSON, 3	Program-only	88.64	1.409	2080.2	17.635	75.78	98.44
JSON, 3	Protected two-step	88.64	1.428	2317.9	17.896	75.78	92.19
JSON, 3	Generic lazy	86.93	1.479	2025.2	16.060	90.62	93.30
JSON, 3	Program-first expansion	88.64	1.409	2170.2	17.749	82.81	98.44
JSON, 3	Full-row acquisition	85.09	1.369	2360.2	17.842	85.16	89.62
JSON, 3	No tight bounds	88.78	1.412	1547.2	12.175	92.19	98.21
JSON, 5	CAPE	88.78	1.411	1530.6	12.158	94.53	100.00
JSON, 5	Eager exact	88.64	1.409	2321.7	17.895	82.03	98.44
JSON, 5	Program-only	88.64	1.409	2081.6	17.616	75.00	98.44
JSON, 5	Protected two-step	88.64	1.428	2318.0	17.893	76.56	92.19
JSON, 5	Generic lazy	86.93	1.567	1957.1	15.878	90.62	93.30
JSON, 5	Program-first expansion	88.64	1.409	2179.2	17.749	83.59	98.44
JSON, 5	Full-row acquisition	85.65	1.375	2053.1	16.948	85.16	91.18
JSON, 5	No tight bounds	88.78	1.411	1534.9	12.131	95.31	100.00

Table 15: Qwen3-VL-4B-Instruct: all methods and both interfaces and path allowances.

Interface, C	Method	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CSV, 3	CAPE	83.40	2.062	2763.2	10.062	100.00	100.00
CSV, 3	Eager exact	83.40	2.062	3208.0	10.484	100.00	100.00
CSV, 3	Program-only	75.68	1.904	624.0	3.583	40.62	40.62
CSV, 3	Protected two-step	82.23	2.320	3208.0	10.436	40.62	40.62
CSV, 3	Generic lazy	82.32	2.063	3106.0	15.878	96.09	96.09
CSV, 3	Program-first expansion	83.40	2.062	3064.5	10.650	100.00	100.00
CSV, 3	Full-row acquisition	83.40	2.062	3015.0	10.545	100.00	100.00
CSV, 3	No tight bounds	83.40	2.062	3087.0	10.580	100.00	100.00
CSV, 5	CAPE	95.31	2.197	2925.5	10.426	100.00	100.00
CSV, 5	Eager exact	95.31	2.197	3208.0	10.661	100.00	100.00
CSV, 5	Program-only	89.94	2.210	624.0	3.594	40.62	78.12
CSV, 5	Protected two-step	95.31	2.639	3208.0	10.502	40.62	73.44
CSV, 5	Generic lazy	83.11	2.354	2228.0	12.745	78.12	78.12
CSV, 5	Program-first expansion	95.31	2.197	2956.5	10.663	100.00	100.00
CSV, 5	Full-row acquisition	95.31	2.197	2925.3	10.398	100.00	100.00
CSV, 5	No tight bounds	95.31	2.197	2925.5	10.432	100.00	100.00
JSON, 3	CAPE	92.33	1.407	1527.4	12.063	95.31	98.44
JSON, 3	Eager exact	92.19	1.405	2271.4	17.818	85.16	98.44
JSON, 3	Program-only	92.19	1.405	2081.0	17.645	80.47	98.44
JSON, 3	Protected two-step	92.19	1.416	2266.3	17.875	80.47	93.75
JSON, 3	Generic lazy	90.48	1.473	2024.2	16.015	93.75	93.75
JSON, 3	Program-first expansion	92.19	1.405	2168.5	17.766	85.94	98.44
JSON, 3	Full-row acquisition	88.64	1.366	2355.0	17.840	88.28	89.84
JSON, 3	No tight bounds	92.33	1.407	1535.3	12.101	95.31	98.44
JSON, 5	CAPE	92.33	1.406	1514.3	12.072	97.66	100.00
JSON, 5	Eager exact	92.19	1.405	2267.9	17.828	85.16	98.44
JSON, 5	Program-only	92.19	1.405	2083.1	17.621	79.69	98.44
JSON, 5	Protected two-step	92.19	1.416	2265.1	17.829	81.25	93.75
JSON, 5	Generic lazy	90.48	1.555	1952.7	15.836	93.75	93.75
JSON, 5	Program-first expansion	92.19	1.405	2175.8	17.761	86.72	98.44
JSON, 5	Full-row acquisition	89.20	1.371	2046.7	16.926	88.28	91.41
JSON, 5	No tight bounds	92.33	1.406	1518.4	12.058	98.44	100.00

F MATCHED ACQUISITION SCHEDULES AND CONTINUATION BOUNDS

F.1 SHARED IMPLEMENTATION AND FACTORIAL PROTOCOL

The shared-kernel study holds CAPE fixed and compares two acquisition schedules under three bound modes. CAPE follows the incumbent and optimistic frontier described in Section 3. The matched lazy AND/OR scheduler follows the action with the largest upper value on the current optimistic solution path. When that action is closed, it activates its native children and descends into the unresolved child with the largest value gap; an absent required witness takes priority. At the next unfinished action it acquires effects on that action's reached scope.

Both schedules share the state representation, catalogs, guard, objective, local validation, witness backups and seeds, caches, action reductions, required-first query order, stopping rule and exact fallback of CAPE. The architectural control follows partial solution expansion in AND/OR search (Hansen & Zilberstein, 2001), changing acquisition order while retaining these shared operators and the feasible witness returned on interruption.

No tightening uses the support bound. Selective tightening uses T_2 inside the selected acquisition, as in CAPE. Always tightening uses that same operator and receiver-generation cache on every eligible two-call bound request throughout the expanded graph. Probe facts invalidate all affected continuation bounds so that the always mode incorporates their information. All modes retain the full execution horizon and the same acquisition stopping condition.

The primary comparisons are selective CAPE versus selective AND/OR and selective CAPE versus its other bound modes. All six configurations and their within-mode scheduler and within-scheduler bound contrasts were fixed before measurement.

The study uses the retained fourteen-model proposals from Appendix E, with no additional proposal calls. Every one of the 304 exact executable task/program/interface/budget groups receives all six methods and two fresh-process repetitions: 3648 primary measurements. The execution configuration and lifecycle accounting follow Section 4, on an AMD EPYC 7742 with Python hash seed 2070.

One additional instrumented run per group and method yields 1824 profiles, after primary timing completes. Exclusive nested wall-clock accounting separates native calibration, T_2, other bounds, graph search, complete-model solving and remaining work. Descendant bound computation within T_2 is charged to T_2. Profiling overhead can change deadline termination, so profiles describe the instrumented run separately from primary timing and policy quality.

Paired intervals use 5,000 resamples of the same eight source clusters. Independent full-model optimization resolves all 304 objectives across 152 native models and 33 normalized action structures. Native replay checks 414 initial/returned policy pairs, 61,440 episodes and 133,066 tool calls; acquired-cell checks cover 32,182 historical witnesses. Primary CAPE/selective attains 606/608 optima; matched AND/OR/selective attains 595/608. CAPE's two misses preserve optimal completion but use extra calls on JSON weather projection at $C = 3$.

F.2 COMPLETE PRIMARY COMPARISONS

Tables 16–19 report the DeepSeek-V4-Pro bindings for every configuration. Completion and runtime certification include failed proposals. Calls, queries, lifecycle and independent optimality condition on executable proposals. Runtime certification and independently attained optimality follow the definitions in Section 4. These measurements form a separate cohort from the mechanism and provider studies.

Table 16: Shared-kernel primary measurements: CSV, $C = 3$.

Scheduler	Bounds	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CAPE	No tightening	85.16	2.196	3330.3	11.489	98.44	100.00
CAPE	Selective	85.16	2.196	2979.9	10.918	97.66	100.00
CAPE	Always	85.16	2.196	2546.4	10.177	98.44	100.00
AND/OR	No tightening	85.16	2.196	3278.6	11.315	98.44	100.00
AND/OR	Selective	85.16	2.196	3250.5	11.916	98.44	100.00
AND/OR	Always	85.16	2.196	3249.7	12.654	98.44	100.00

Table 17: Shared-kernel primary measurements: CSV, $C = 5$.

Scheduler	Bounds	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CAPE	No tightening	98.44	2.348	3161.1	11.374	98.44	100.00
CAPE	Selective	98.44	2.348	3161.1	11.319	98.44	100.00
CAPE	Always	98.44	2.348	3148.1	12.464	93.75	100.00
AND/OR	No tightening	98.44	2.348	3163.4	11.148	98.44	100.00
AND/OR	Selective	98.44	2.348	3163.4	11.265	98.44	100.00
AND/OR	Always	98.44	2.348	3163.4	11.381	98.44	100.00

Table 18: Shared-kernel primary measurements: JSON, $C = 3$.

Scheduler	Bounds	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CAPE	No tightening	94.74	1.477	1505.7	12.007	95.31	97.66
CAPE	Selective	94.74	1.476	1511.9	11.961	96.88	98.44
CAPE	Always	94.74	1.476	1505.0	12.012	95.31	98.44
AND/OR	No tightening	94.39	1.481	1980.3	16.130	92.97	94.53
AND/OR	Selective	94.39	1.478	1990.5	16.161	92.97	94.53
AND/OR	Always	94.46	1.483	1975.3	16.235	93.75	95.31

Table 19: Shared-kernel primary measurements: JSON, $C = 5$.

Scheduler	Bounds	Success (%)	Calls	Queries	Time (s)	Cert. (%)	Opt. (%)
CAPE	No tightening	94.74	1.475	1494.0	11.975	97.66	100.00
CAPE	Selective	94.74	1.475	1490.5	11.929	96.88	100.00
CAPE	Always	94.74	1.475	1473.9	12.005	96.09	100.00
AND/OR	No tightening	94.50	1.483	1941.7	15.784	93.75	95.31
AND/OR	Selective	94.53	1.483	1949.4	15.728	93.75	95.31
AND/OR	Always	94.42	1.480	1932.6	15.699	93.75	95.31

Tables 20 and 21 retain every planned within-mode scheduler and within-scheduler bound contrast. C and A denote CAPE and matched AND/OR; N, S and T denote no tightening, selective tightening and always tightening, respectively. Positive differences favor the named candidate. Success gains are in percentage points; call costs must be read with completion. For selective CAPE versus selective AND/OR, the JSON completion-gain intervals are [0.000, 0.781] at $C = 3$ and [0.000, 0.497] at $C = 5$; both include zero.

Table 20: CSV paired contrasts. Time intervals resample the eight sources.

C	Candidate	Control	Gain (pp)	Queries saved	Seconds saved	95% interval
3	C-N	A-N	0.000	-51.7	-0.173	[-0.522, 0.061]
3	C-S	A-S	0.000	270.6	0.998	[0.358, 1.596]
3	C-T	A-T	0.000	703.4	2.477	[1.420, 3.479]
3	C-S	C-N	0.000	350.4	0.571	[0.009, 1.130]
3	C-S	C-T	0.000	-433.6	-0.741	[-1.255, -0.242]
3	A-S	A-N	0.000	28.1	-0.600	[-0.869, -0.291]
3	A-S	A-T	0.000	-0.8	0.738	[0.512, 0.925]
5	C-N	A-N	0.000	2.2	-0.226	[-0.357, -0.095]
5	C-S	A-S	0.000	2.2	-0.054	[-0.273, 0.163]
5	C-T	A-T	0.000	15.2	-1.082	[-2.013, -0.421]
5	C-S	C-N	0.000	0.0	0.055	[-0.160, 0.278]
5	C-S	C-T	0.000	-13.0	1.145	[0.413, 1.966]
5	A-S	A-N	0.000	0.0	-0.117	[-0.275, 0.039]
5	A-S	A-T	0.000	0.0	0.117	[-0.134, 0.391]

Table 21: JSON paired contrasts. Gain denotes completion gain in percentage points.

C	Candidate	Control	Gain (pp)	Queries saved	Seconds saved	95% interval
3	C-N	A-N	0.355	474.6	4.123	[2.785, 5.446]
3	C-S	A-S	0.355	478.6	4.200	[2.839, 5.570]
3	C-T	A-T	0.284	470.3	4.223	[2.860, 5.630]
3	C-S	C-N	0.000	-6.2	0.046	[-0.138, 0.268]
3	C-S	C-T	0.000	-6.9	0.051	[-0.097, 0.176]
3	A-S	A-N	0.000	-10.2	-0.031	[-0.203, 0.145]
3	A-S	A-T	-0.071	-15.3	0.074	[-0.103, 0.239]
5	C-N	A-N	0.249	447.8	3.809	[2.348, 5.302]
5	C-S	A-S	0.213	458.9	3.799	[2.377, 5.228]
5	C-T	A-T	0.320	458.7	3.694	[2.332, 5.081]
5	C-S	C-N	0.000	3.5	0.046	[-0.070, 0.164]
5	C-S	C-T	0.000	-16.6	0.076	[-0.066, 0.210]
5	A-S	A-N	0.036	-7.7	0.056	[-0.100, 0.207]
5	A-S	A-T	0.107	-16.9	-0.029	[-0.217, 0.151]

F.3 Diagnostic cost decomposition

Figure 12 shows how the three CAPE bound modes allocate computation on CSV. At $C = 3$, always tightening spends 1.468 seconds on T_2, compared with 0.621 for selective computation, while native calibration falls from 8.814 to 7.658 seconds. At $C = 5$, the corresponding T_2 times are 1.192 and 0.019, with similar calibration work. The primary timings in Figure 4 measure the resulting net cost differences.

Table 22 includes both schedulers and both interfaces. Of all 1824 instrumented compilations, 1805 return the same success identities and call mass as every primary repetition on the corresponding input and method. The agreement and query-difference columns describe the changed execution under instrumentation. The diagnostic data remain separate from the primary performance estimates.

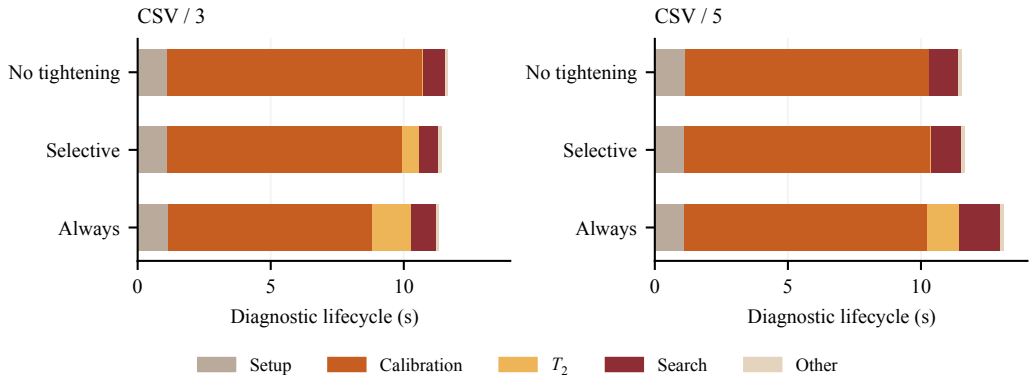

Figure 12: Separate diagnostic runs, DeepSeek-V4-Pro bindings. Stacks show exclusive mean lifecycle components, including all diagnostic records. Search groups graph search, other bounds and complete-model solving; Other groups bookkeeping, encoding and cleanup.

Table 22: All diagnostic configurations. Component times are seconds. Agree compares the returned value with all corresponding primary repetitions; Δ queries is diagnostic minus primary mean acquisition.

Setting	Method	Setup	Calibration	T_2	Search	Other	Agree (%)	Δ queries
CSV/3	C-N	1.103	9.612	0.000	0.844	0.110	100.00	0.0
CSV/3	C-S	1.122	8.814	0.621	0.747	0.114	100.00	0.5
CSV/3	C-T	1.149	7.658	1.468	0.925	0.108	100.00	-1.8
CSV/3	A-N	1.114	9.492	0.000	0.696	0.108	100.00	0.0
CSV/3	A-S	1.139	9.571	1.290	0.617	0.118	100.00	0.0
CSV/3	A-T	1.157	9.620	2.266	0.653	0.114	96.88	-18.9
CSV/5	C-N	1.152	9.155	0.000	1.109	0.134	100.00	0.0
CSV/5	C-S	1.101	9.259	0.019	1.115	0.134	100.00	0.0
CSV/5	C-T	1.101	9.148	1.192	1.539	0.133	100.00	-39.3
CSV/5	A-N	1.121	9.265	0.000	1.025	0.133	100.00	0.0
CSV/5	A-S	1.110	9.243	0.000	1.043	0.137	100.00	0.0
CSV/5	A-T	1.111	9.315	0.265	1.040	0.136	96.88	-8.0
JSON/3	C-N	0.431	11.307	0.000	0.349	0.093	98.44	3.0
JSON/3	C-S	0.454	11.139	0.070	0.276	0.094	100.00	-11.6
JSON/3	C-T	0.435	11.183	0.100	0.277	0.094	98.44	-20.4
JSON/3	A-N	0.446	15.561	0.000	0.312	0.096	93.75	10.8
JSON/3	A-S	0.424	15.541	0.146	0.212	0.097	95.31	-10.1
JSON/3	A-T	0.418	15.581	0.154	0.214	0.097	96.88	3.0
JSON/5	C-N	0.414	11.282	0.000	0.326	0.096	100.00	-8.9
JSON/5	C-S	0.421	11.364	0.004	0.326	0.097	100.00	-5.4
JSON/5	C-T	0.428	11.336	0.006	0.342	0.096	100.00	8.0
JSON/5	A-N	0.455	14.989	0.000	0.295	0.098	98.44	-29.0
JSON/5	A-S	0.420	15.188	0.000	0.303	0.098	98.44	-6.4
JSON/5	A-T	0.427	15.025	0.000	0.309	0.098	96.88	15.2

F.4 Acquisition-order comparison across proposal sources

Tables 23 and 24 compare selective CAPE with selective AND/OR under every retained proposal source. Across model bindings, lifecycle reductions are 7.53–10.83% for CSV $C = 3$, 22.51–26.04% for JSON $C = 3$, and 19.52–24.18% for JSON $C = 5$. CSV $C = 5$ intervals contain zero for every model. Exact duplicate inputs share physical measurements, so the provider rows are correlated views of the source workload. The artifact retains the complete six-configuration summaries and pairings for every binding.

Table 23: CSV lifecycle savings for selective CAPE over selective AND/OR, by proposal source. Model configurations are in Table 8.

Proposal source	$C = 3$, s	95% interval	$C = 5$, s	95% interval
DeepSeek-V4-Flash	0.973	[0.345, 1.561]	-0.053	[-0.273, 0.164]
DeepSeek-V4-Pro	0.998	[0.358, 1.596]	-0.054	[-0.273, 0.163]
Kimi-K2.5	0.973	[0.345, 1.561]	-0.053	[-0.273, 0.164]
Llama-3.3-70B-Instruct	0.996	[0.354, 1.594]	-0.055	[-0.274, 0.162]
Qwen2.5-VL-7B-Instruct	0.833	[0.227, 1.437]	-0.065	[-0.277, 0.139]
Qwen3-VL-4B-Instruct	0.894	[0.290, 1.490]	-0.099	[-0.327, 0.122]
Qwen3.5-122B-A10B	0.973	[0.345, 1.561]	-0.053	[-0.273, 0.164]
Qwen3.5-27B	0.973	[0.345, 1.561]	-0.053	[-0.273, 0.164]
Qwen3.5-9B	0.973	[0.345, 1.561]	-0.053	[-0.273, 0.164]
Llama-3.1-8B-Instruct	1.080	[0.525, 1.580]	0.057	[-0.122, 0.223]
GPT-5.6 Luna	0.973	[0.345, 1.561]	-0.053	[-0.273, 0.164]
GPT-5.6 Terra	0.973	[0.345, 1.561]	-0.053	[-0.273, 0.164]
GPT-6 Astra	0.973	[0.345, 1.561]	-0.053	[-0.273, 0.164]
Grok 4.6	0.973	[0.345, 1.561]	-0.053	[-0.273, 0.164]

Table 24: JSON lifecycle savings for selective CAPE over selective AND/OR, by proposal source.

Proposal source	$C = 3$, s	95% interval	$C = 5$, s	95% interval
DeepSeek-V4-Flash	4.204	[2.848, 5.572]	3.795	[2.372, 5.224]
DeepSeek-V4-Pro	4.200	[2.839, 5.570]	3.799	[2.377, 5.228]
Kimi-K2.5	4.204	[2.848, 5.572]	3.795	[2.372, 5.224]
Llama-3.3-70B-Instruct	4.201	[2.840, 5.570]	3.803	[2.381, 5.228]
Qwen2.5-VL-7B-Instruct	3.961	[2.687, 5.168]	3.458	[2.136, 4.772]
Qwen3-VL-4B-Instruct	3.982	[2.711, 5.192]	3.556	[2.270, 4.812]
Qwen3.5-122B-A10B	4.204	[2.848, 5.572]	3.795	[2.372, 5.224]
Qwen3.5-27B	4.204	[2.848, 5.572]	3.795	[2.372, 5.224]
Qwen3.5-9B	4.204	[2.848, 5.572]	3.795	[2.372, 5.224]
Llama-3.1-8B-Instruct	3.627	[2.371, 4.955]	3.039	[1.783, 4.323]
GPT-5.6 Luna	4.204	[2.848, 5.572]	3.795	[2.372, 5.224]
GPT-5.6 Terra	4.204	[2.848, 5.572]	3.795	[2.372, 5.224]
GPT-6 Astra	4.204	[2.848, 5.572]	3.795	[2.372, 5.224]
Grok 4.6	4.204	[2.848, 5.572]	3.795	[2.372, 5.224]