CASEPATH: AN AGENTIC, PROCESS-FIRST ARCHITECTURE FOR DETERMINING EVIDENCE REQUIREMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents in rule-governed workflows must choose actions that apply to the current case. In evidence collection, a single fact can make a receipt required, irrelevant or premature. Direct checklist generation leaves the model to judge that fact and choose the action in the same output, even when the rules are in context. We introduce CASEPATH, a process-first architecture that separates these decisions. The model assesses case conditions and documents; executable process control activates the applicable requirements, derives their evidence needs and holds requests whose conditions remain unresolved. Each request retains its justification. Two synthetic claim studies test this design. On 27 held-out theft-case pairs differing in one fact, CASEPATH makes 15 wrong checklist changes, versus 27 for direct generation and 41 for generation with the process graph in context. On 108 matched tenancy cases with model judgments fixed, checking inherited branch conditions removes 98% of unnecessary requests and more than halves immediate requests, retaining all 635 valid ones. We release both datasets, structured references, predictions and scorers. Executable process state gives agents a control layer between model judgment and evidence requests.

1 INTRODUCTION

Language-model agents operate in workflows such as customer service (Yao et al., 2025) and claim handling, where an action must satisfy the rules that apply at the current process state. A common design asks the model to choose the action directly, sometimes with rules or a process graph in its prompt. The model must interpret the case and decide what the process requires in the same output. A condition in the prompt need not constrain the chosen action. A claimant may then be asked for an irrelevant document, while a necessary request is omitted.

We study one such action: the request for evidence. Suppose a watch is stolen in a household burglary. Whether the insurer needs proof of its value depends on whether it was separately insured above the policy threshold. If so, a receipt or expert valuation can establish value. If its insurance status is unclear, the system should ask before requesting that evidence. A photograph may establish possession but not value; an adequate receipt already on file settles the requirement. The same document can therefore be required, irrelevant, premature or already sufficient, depending on the case.

CASEPATH makes this dependency executable (Figure 1). The LLM interprets the case and assesses the documents. Code determines which requirements apply and derives requests through their links to facts and acceptable evidence. Open conditions lead to holds and clarification questions. Each request keeps the chain that produced it, so an error can be traced to a rule or a case judgment.

We make four contributions:

1. **Architecture.** We formulate evidence collection as execution of conditional requirements, combining LLM assessments with deterministic request derivation (Section 3, Algorithm 1).
2. **Evaluation.** Two studies separate responses to changed case facts from the effect of inherited branch conditions. Comparators include direct generation and generation with process context; a fixed-judgment control isolates the branch check (Section 4).

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.

3. **Datasets.** We release 36 pairs of synthetic theft cases with reference checklist changes, and 150 synthetic tenancy claims with references for processes, obligations, documents and next actions (Figures 3 and 4).
4. **Empirical insight.** Process control reduces wrong checklist changes in Study A and, at fixed judgments, removes unnecessary requests without losing valid ones in Study B. Request traces locate three quarters of Study A's errors in the extracted rules (Section 5).

The released predictions and scorers support inspection and offline reproduction of both studies.[1]

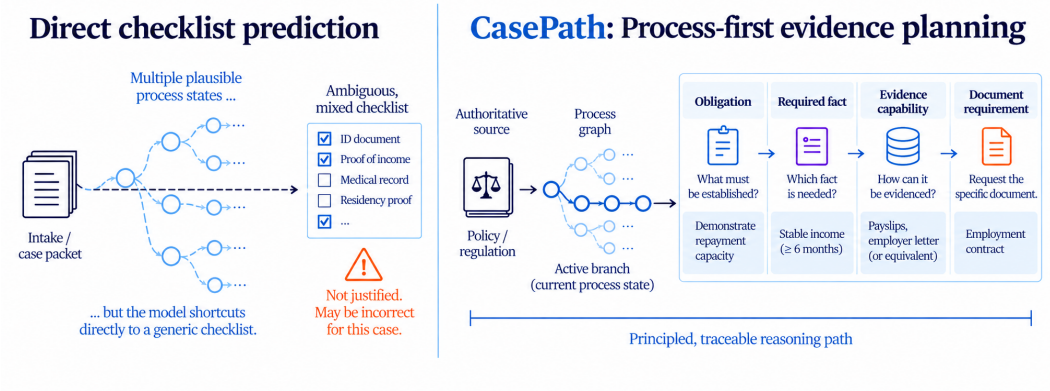


Figure 1: **Requests follow the active process.** Illustrative loan application. The intake packet contains the person's account and supplied files. Dark-blue nodes mark the branch that applies; pale nodes are alternatives. An obligation is a requirement to establish a fact. An evidence capability describes what can establish that fact; its column gives example evidence sources. Documents supply the evidence.

2 Related Work

Rule languages execute structured requirements (Merigoux et al., 2021; Object Management Group, 2024); neuro-symbolic systems use LLM interpretations as inputs to exact solvers (Pan et al., 2023; Guan et al., 2023).

Workflow agents use policies, state machines or procedural graphs to guide action choice (Xiao et al., 2024; Yao et al., 2025; Wu et al., 2024; Lu et al., 2026; Yu et al., 2026). Guardrails check actions already proposed (Chen et al., 2025; Xiang et al., 2025). The closest precedent, an evidence ledger, holds unresolved obligations but creates them per query without specifying inheritance from enclosing branches (Kim, 2026). Study B isolates that inheritance check. Our baselines are custom implementations of the compared design choices. Appendix F compares their inputs, operations and outputs.

3 Method

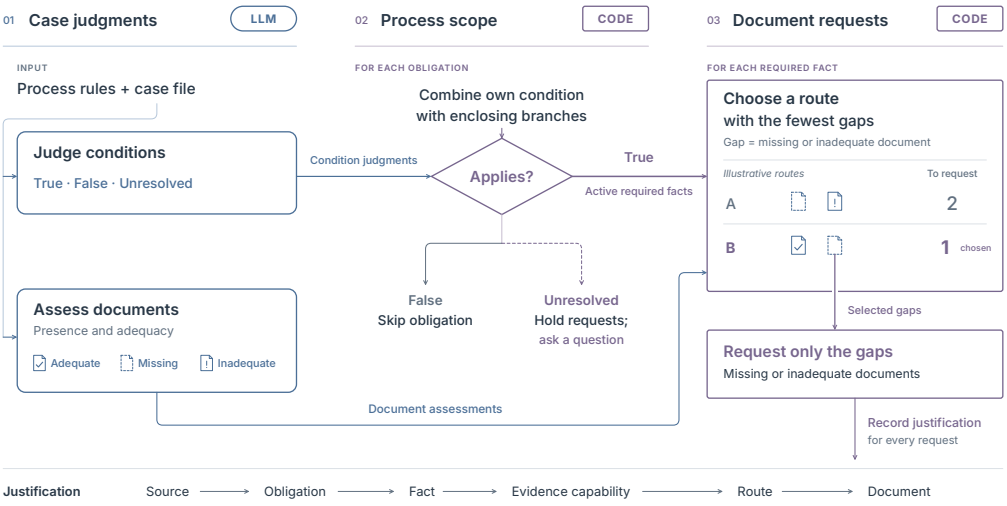
An *obligation* is a conditional requirement to establish a fact. For the separately insured watch, the *required fact* is its value. An *evidence capability* specifies what the evidence must establish. Each *route* is a set of documents sufficient to provide that evidence: a receipt and an expert valuation are alternative routes to proof of value. The process graph links obligations, facts, capabilities and routes to their conditions and sources. LLMs extract rules when formal rules are unavailable.

The model judges each condition true, false or *unresolved*, citing the case text, and assesses each document's presence and adequacy. Code checks an obligation's own condition together with its enclosing branches: its *full process scope*. If the combined condition is false, code excludes the request; if unresolved, it holds the request for clarification. Timing conditions can also hold a request.

[1] </> Code Data.

For an applicable requirement, code uses adequate evidence already on file or chooses the route with the fewest missing or inadequate documents. It requests those gaps and retains the source-to-document justification. Algorithm 1 shows this procedure; Appendix A.1 defines timing and tie-breaking.

Algorithm 1 Process control turns case judgments into justified requests. Blue marks LLM assessments; violet marks code. A gap is a missing or inadequate document. The route example is illustrative.



In Figure 2, the notice email satisfies the notification requirement; evidence of the defect's extent is still missing.

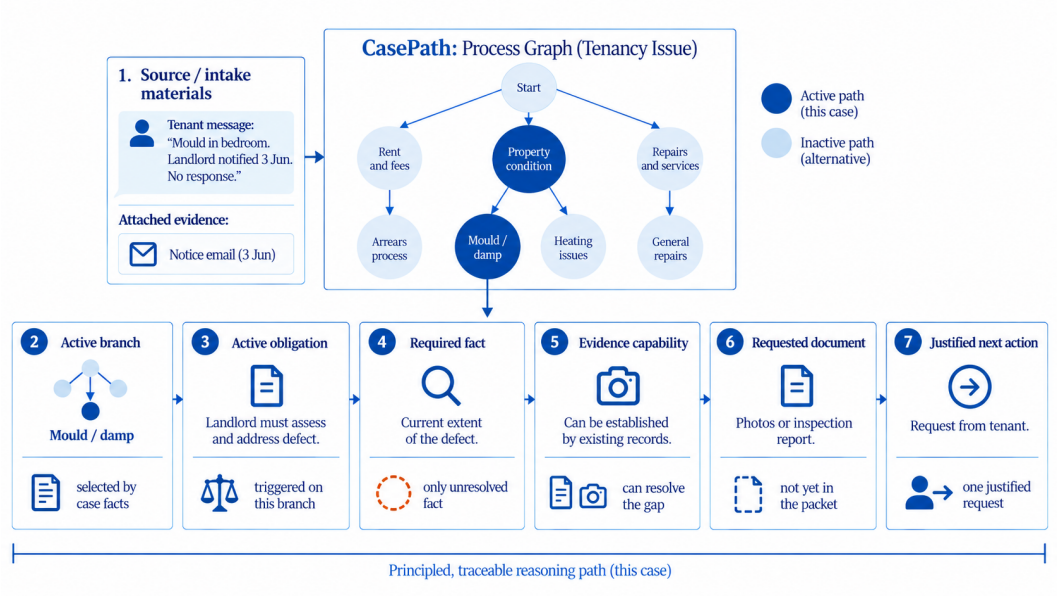


Figure 2: **The unmet requirement determines the request.** Illustrative case with a hypothetical rule. Dark nodes mark the active mould branch; pale nodes are alternatives. The notice email establishes notification. The remaining fact is the extent of the defect; photos or an inspection report can establish it.

The studies test different parts of process control (Table 1). Study A checks individual rule conditions and requests every evidence route. Study B adds the inherited dispute-type condition and has one route per capability.

Table 1: **What each implementation tests.** The studies use separate rules, cases and code. Appendix A gives the full correspondence.

	Study A: theft	Study B: tenancy
Rules	LLMs extract public law and policy rules	Code compiles synthetic templates
Conditions checked	Each rule's own condition	Obligation and inherited dispute type
Evidence routes	Every route requested	One route per capability; no choice
Documents on file	None in these cases	Presence and adequacy assessed
Scored output	Changes between checklists	Complete outputs; separately, saved requests

The workbench opens Study B's claims and replays Study A's method on all 72 cases.[2] Appendix Table 17 maps the evaluated methods to the released software.

4 EXPERIMENTAL DESIGN

We test two consequences of process control. A change in one fact should alter only the requests governed by that fact (Study A). A requirement from an inactive branch should generate no immediate request (Study B). Both studies use synthetic cases with *references*: expected outputs for scoring. A *checklist* is the set of documents requested. Tables show rates from 0 to 1; plots show percentages.

4.1 STUDY A: CHANGE ONE FACT AND COMPARE THE CHECKLISTS

Each pair contains two versions of a theft case that differ in one branch condition. Four stories each yield nine such pairs: one story supplies nine Development pairs, and the other three supply 27 Held-out pairs. The same conditions appear in new stories. Held-out cases were read once after the method, data and analysis were frozen (Appendix B.1).

Each method generates both checklists. We score the documents added or withdrawn, so a system must respond to the changed fact while preserving unrelated requests. Separately insuring the watch should add a receipt or valuation. Either is correct; adding both makes an extra change (Figure 3).

Table 2 defines the four methods. All have access to the same cases, sources and document catalogue and use `openai/gpt-5.6-terra` at temperature 0. Graph context also receives CASEPATH's graph; Evidence-first retrieves twelve passages after identifying evidence needs. Compute differs: CASEPATH batches cases by condition, reads fewer tokens than Direct and writes about five times as many (Appendix C.6).

Table 2: **Study A varies how documents are chosen.** Every method has the case, source collection and document catalogue.

Method	How it chooses documents
Direct	Reads the case and sources, then writes the checklist.
Graph context	Also receives CASEPATH's graph and rules, then writes the checklist.
Evidence-first	Identifies evidence needs, retrieves passages, then writes the checklist.
CASEPATH	Judges conditions with the model; code derives the checklist from rules.

References are compiled from a *source contract*: 308 decisions and evidence requirements extracted by LLMs from 197 passages. Appendix Table 8 lists the source documents.

Checklist change precision is the share of predicted changes that are correct; recall is the share of required changes made. *Checklist change* F_1 balances both over pooled Held-out counts. *Pair* F_1

[2] Product source.

scores one pair; an *exact pair* has no wrong or missed change. A wrong document on both lists is invisible to this test. References require additions only, so correct withdrawals are untested. The *reference vocabulary* comprises documents governed by any benchmark branch.

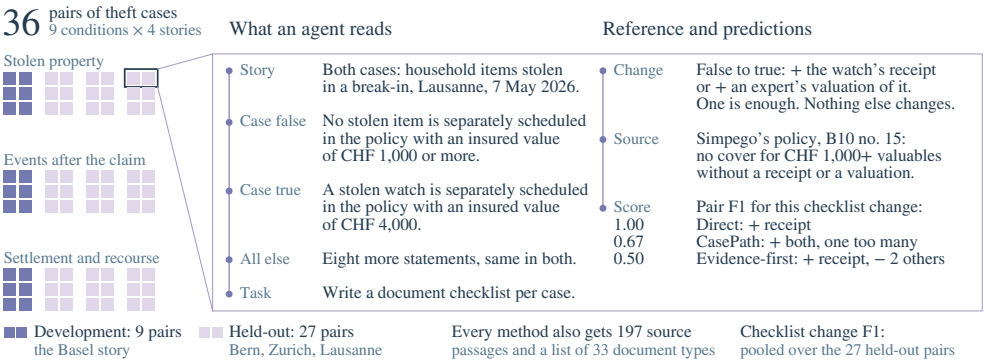


Figure 3: **Study A scores checklist changes.** Rows are conditions, columns stories; two squares form a pair. Dark means Development, light Held-out. The outlined pair changes one condition; + adds a document, − withdraws one. Pair F_1 scores this example; checklist change F_1 pools held-out pairs.

4.2 Study B: compare outputs and isolate branch checks

A requirement can satisfy its own condition yet belong to the wrong process. For example, a defect-process document should not be requested for a rent increase. Study B tests the enclosing dispute category, called the *inherited dispute type*, and compares complete request lists. One comparison holds model judgments fixed to isolate this check; another compares independently generated outputs.

The data comprise 150 tenancy claims in 28 families: 60 Development and 90 Held-out claims, with no family crossing splits. Each claim provides a message and attachments. Code compiles defect, termination and rent-increase templates. Prior product exposure to Held-out inputs makes this a comparison across families, not a fresh generalisation test (Appendix E.2).

Five prediction methods receive the same case and compiled process and return a graph, requests and claim-handling fields. *Direct* generates this output directly; *Document-first* drafts documents first; *Process context* drafts branches first; *Rule-first* drafts and follows condition-and-action rules. CasePath derives the output from model judgments in code. All use `openai/gpt-5.6-terra`, a 2-call budget per claim and the same completion limits (Appendix D.1).

Two controls reuse CasePath's judgments without new LLM calls. *Local scope* checks each obligation and its immediate branch; CasePath also checks the one inherited dispute-type condition. *Compiled control* expresses CasePath's decisions in another compiled form to test implementation agreement.

Study B uses retrospective scoring of saved requests. The original whole-output evaluator accepted none because its reference scenarios lacked the compiler's condition symbols. Request scoring bypasses that acceptance check and evaluates unchanged saved lists (Appendix D.1).

A *valid request* has an obligation-to-document chain matching the reference; *justified-request precision* measures the share valid. *Critical-evidence recall* measures required evidence recovered through complete routes, weighted by importance. *Checklist* F_1 balances wrong and missed documents; *unnecessary request rate* measures the share absent from the reference (Figure 4).

Matched claims have outputs from both planners. All-claim scores also penalise failures: zero when higher is better, one for unnecessary request rate. Empty but available lists are scored normally. Dispute types weigh equally, then families within types and claims within families (Appendix D.2). *Output availability* is the weighted share with a scoreable list; weighted scores differ from ratios of pooled counts.

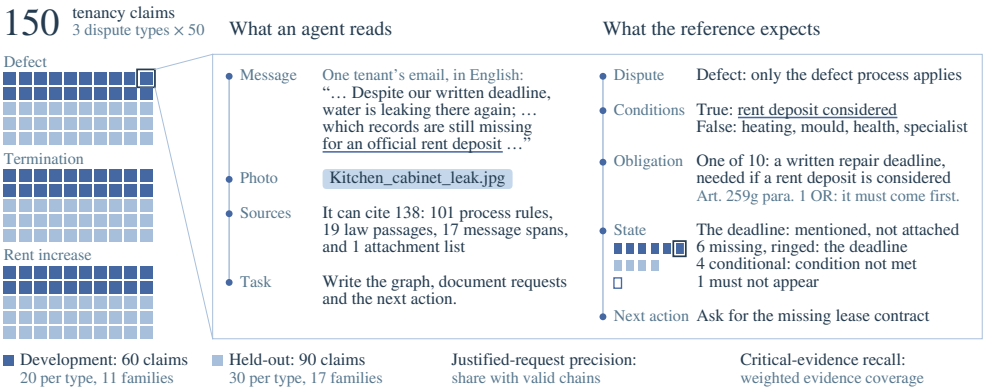


Figure 4: **Study B scores complete request lists.** Squares are claims: dark for Development, light for Held-out. Underlining sets conditions. The outlined deadline document serves one obligation; the next action concerns the whole claim. "Missing" means absent, "conditional" held pending a condition, and "must not appear" forbidden. OR is the Swiss Code of Obligations.

5 RESULTS

5.1 STUDY A: PROCESS CONDITIONS STABILISE CHECKLIST CHANGES

Across 27 Held-out pairs, CASEPATH makes 15 wrong and 24 correct changes, versus Direct's 27 and 25 (Table 3). Graph context makes 41 wrong changes despite receiving the process graph. Direct's errors include 11 unnecessary withdrawals: documents disappear from the checklist even though the changed fact does not remove their requirement. CASEPATH makes no withdrawals (Figure 5).

Table 3: **CASEPATH makes fewer wrong changes; Graph context recovers more required changes.** The 27 held-out pairs require 33 changes, all additions. Exact pairs have no wrong or missed change. Brackets give 95% intervals; arrows show the preferred direction; bold marks the best value.

	Checklist changes				
Method	Correct ↑	Wrong ↓	Missed ↓	Exact pairs ↑	Checklist change F_1 ↑ [95% interval]
CASEPATH	24	**15**	9	**12/27**	**0.667** [0.435, 0.880]
Direct	25	27	8	9/27	0.588 [0.404, 0.759]
Graph context	**31**	41	**2**	4/27	0.590 [0.471, 0.716]
Evidence-first	16	52	17	2/27	0.317 [0.239, 0.400]

All 120 wrong baseline changes concern documents outside the reference vocabulary, versus 9 of CASEPATH's 15 (Appendix Figure 10A). These documents are governed by none of the benchmark's branches. Generating the two lists independently lets unrelated requests appear or disappear; applying the same rules to both cases keeps more of them stable.

The wrong-condition control tests whether changes depend on the specific fact. Matching saved predictions to other conditions reduces CASEPATH's checklist change F_1 from 0.667 to at most 0.083 (Appendix Figure 10B). All 39 predicted changes retain their condition links, allowing this dependency to be checked.



Figure 5: **Most extra baseline errors are unnecessary withdrawals.** All 27 Held-out pairs count. Rows identify methods. (A) Checklist change F_1, with 95% intervals. (B) Counts of correct additions (blue), wrong additions (hatched peach) and wrong withdrawals (dotted lilac). Segment labels give counts. No reference requires a withdrawal.

The gain is clearest when an event changes what evidence the claim requires (Table 4). CASEPATH is exact on the insurer's deadline, its declination (claim rejection) and recovery of stolen items. Across these three conditions, the best baseline mean pair F_1 is 0.40; Direct never requests the declination letter.

Table 4: **CASEPATH is exact on events after the claim and weakest on settlement and recourse.** A scheduled valuable is separately insured. Mean pair F_1 over three held-out stories per condition; 1 means no wrong or missed change. The final row weights conditions equally. Bold marks the best method.

	Branch condition	CASEPATH	Direct	Graph context	Evidence-first
Stolen property	Scheduled valuable	0.67	**1.00**	0.69	0.50
	Bicycle or e-bike	**1.00**	0.61	0.82	0.21
	Misused SIM card	0.67	**0.78**	0.61	0.39
Events after the claim	Insurer's deadline	**1.00**	0.17	0.39	0.23
	Insurer's declination	**1.00**	0.00	0.28	0.22
	Items recovered	**1.00**	0.80	0.52	0.39
Settlement and recourse	Formal expert procedure	0.00	0.49	0.44	**0.56**
	Repair needing consent	0.50	0.78	**0.89**	0.00
	Third party at fault	0.00	**0.89**	0.80	0.33
Mean over the nine conditions		**0.65**	0.61	0.60	0.31

Request traces attribute three quarters of CASEPATH's errors to LLM-generated rules; the rest concern case interpretation and the all-routes policy (Appendix C.5). All nine misses involve documents already on both lists, leaving no addition when their condition changes. The largest losses concern settlement and recourse (recovery from third parties).

Five predeclared traceability and condition-response criteria pass; the three accuracy criteria do not (Table 10). The condition-paired test does not establish an accuracy gain.

Graph context misses 7 fewer changes but makes 26 more wrong ones. If a wrong change costs one unit, CASEPATH has the lowest observed error cost when a miss costs less than $26/7 \approx 3.7$ units; Graph context is cheapest above it (Appendix Figure 11).

5.2 STUDY B: BRANCH CHECKS REMOVE REQUESTS THAT DO NOT APPLY

On 108 matched claims with the same model judgments, checking the inherited dispute type reduces unnecessary requests from 810 to 17, a 98% reduction. Both planners retain 635 of 728 required documents, while immediate requests fall from 1,445 to 652 (Figure 6).

Of the 793 removed requests, 710 are held because the model leaves the dispute type unresolved; 83 are dropped because it judges that type false. The matched set contains 44 Development and 64 Held-out claims. CASEPATH makes no unnecessary request on 63 of the 64 Held-out claims (Figure 6). Compiled control produces the same decisions.

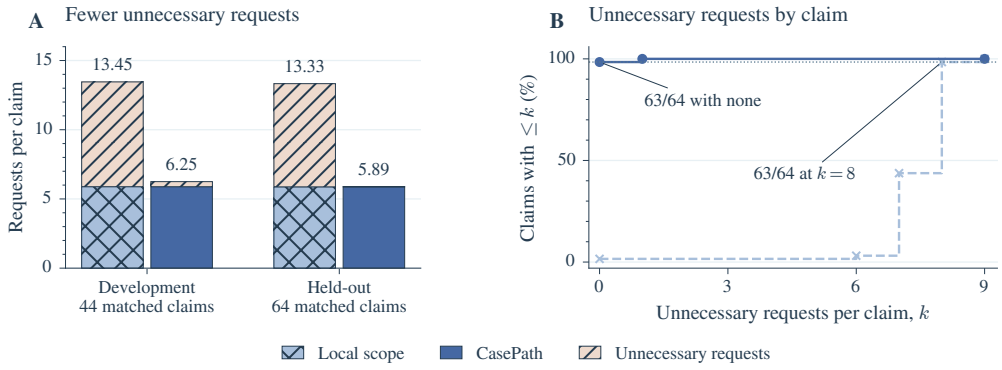


Figure 6: **Branch checks remove unnecessary requests at fixed judgments.** (A) Mean requests per claim: crosshatched light blue is Local scope; solid dark blue is CASEPATH. Blue portions are valid requests; hatched peach portions are unnecessary. Labels give totals. (B) Held-out claims with at most k unnecessary requests; $k = 0$ means none. Dark-blue solid circles mark CASEPATH, light-blue dashed crosses Local scope. Dotted guides mark 63/64.

Uniform random deletion to the same list lengths would retain fewer than half the valid requests in expectation, versus 100% under the branch check (Figure 7B). This expectation uses each saved list's size and composition. Nodes (concepts) and relations (links) also more than halve while retaining every reference match (Figure 7A).

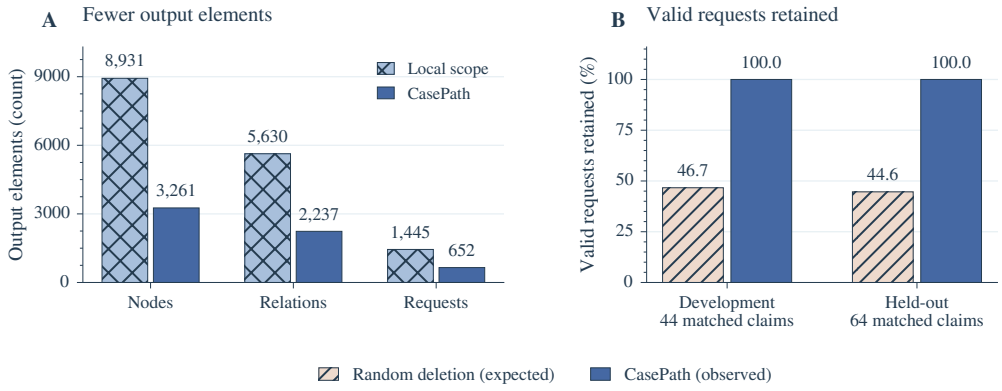


Figure 7: **Smaller outputs retain the correct items already present.** Both panels use 108 matched claims. (A) Counts: crosshatched light blue is Local scope; solid blue is CASEPATH. (B) Retention at CASEPATH's list length: 100% for CASEPATH, versus expected random deletion within each list (hatched peach). Retention concerns previously valid requests, not all required documents.

The gain persists when all 150 claims count, including failed outputs. With Section 4.2's weights and penalties, justified-request precision rises from 0.323 to 0.721 (Table 5). CASEPATH produces 110 outputs and Local scope 108; the two additional outputs account for the recall difference.

Table 5: **Justified-request precision improves across all 150 claims.** Scores use Section 4.2's weights and failure penalties. Recall differs only because output availability differs.

Method	Claims with output	Justified-request precision ↑	Critical-evidence recall ↑
CASEPATH	110 of 150	**0.721**	**0.641**
Local scope	108 of 150	0.323	0.630

The comparison with independently generated outputs includes both output failures and checklist errors. Across all 150 claims, CASEPATH reaches 66.9% checklist F_1, versus at most 44.3% for the prediction baselines (Figure 8). Let A be weighted output availability and Q weighted mean checklist F_1 on available lists. Failures score zero, so overall $F_1 = AQ$. CASEPATH's availability is 73.6%, versus at most 46.7% for the baselines, while its available-list quality lies within their range. Much of the overall advantage comes from producing more scoreable outputs; Q compares each method's own available subset.

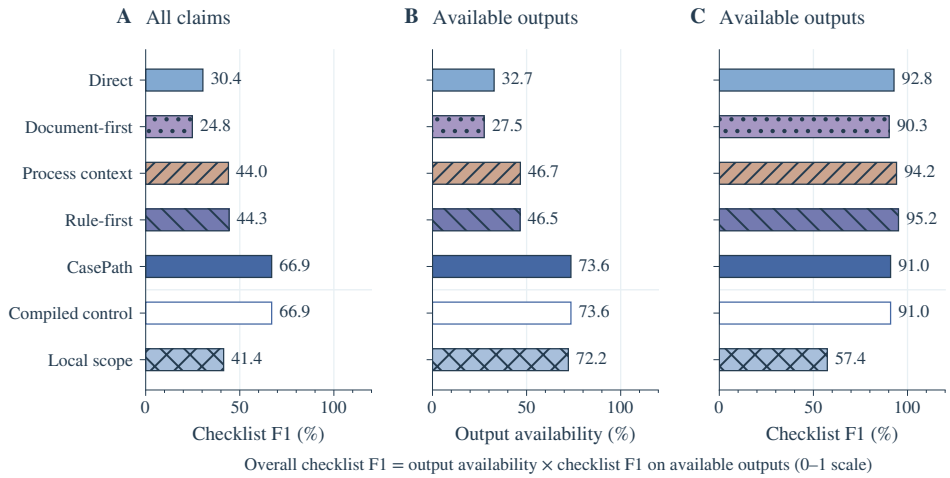


Figure 8: **Availability explains much of the overall score difference.** All 150 claims; rows name methods, with controls below the line. Horizontal axes show (A) overall checklist F_1 with failures scored zero, (B) availability A, and (C) available-list quality Q. Rates are percentages; $F_1 = AQ$ uses 0–1 values. Panel C compares different available subsets, not paired outputs.

Paired contrasts penalising each method's own failures favour CASEPATH. The predeclared rule assigns CASEPATH a loss whenever either output is missing; those contrasts remain negative (Appendix Table 16).

6 DISCUSSION AND CONCLUSION

The studies identify two sources of unnecessary evidence requests: unstable document selection when a case changes, and obligations applied outside their enclosing branch. Executing conditions reduces both. The fixed-judgment tenancy comparison isolates the branch check: it changes request selection while leaving recovered evidence unchanged.

Process control still depends on the rules and case judgments it executes. Study A's request traces locate most errors in the extracted rules, giving a developer a specific place to inspect and correct.

Both studies use self-authored synthetic cases. Legal correctness, expert effort and interaction costs require field evaluation. Study B's 2,006 holds, route choice, timing and next-action accuracy are unscored (Appendix Table 15).

REPRODUCIBILITY STATEMENT

The anonymous supplement reproduces every reported analysis offline, without model or network access. Study A includes frozen sources, rules, the planner, benchmark, predictions, provider receipts and analysis code. Study B includes all 150 claims and references, the compiled process, prediction code, original evaluator, document-request scorer and every run, including failures. Hashes link results to inputs; Appendices C and E.1 describe replay. Both datasets are separately available in `data/`, with licences, datasheets and verification scripts (Appendix B). New model outputs require provider access and are not deterministic.

ETHICS STATEMENT

The studies use public sources and synthetic cases with no personal data. Quoted law, policy terms, claim-form passages and official cantonal form templates remain third-party material and are excluded from the data licence. Study B's evaluated messages named a real insurer's intake mailbox; the release replaces it with a fictional address of the same byte length (Appendix B). CASEPATH plans evidence requests. It gives no legal advice and makes no automated coverage decisions. Real use would require the correct policy and jurisdiction, professional review, privacy controls, and ways to correct or appeal decisions.

AI USE STATEMENT

The authors conceived the research questions, designed CASEPATH and the studies, and directed the research and manuscript development. Under their guidance, generative AI assisted with implementation and testing, experiment execution, synthetic data and source-contract generation, translation, data formatting and pseudonymisation, error analysis, and feedback on the framework, hypotheses, methodology and interpretation. It also assisted with literature search, figures and manuscript drafting and editing. The authors made the scientific decisions and take responsibility for the final text, claims and artifacts. Verification included code tests, recomputation of reported numbers from released files, and checks of references against their sources.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 9112–9141, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/25f7be9694d7b32d5cc670927b8091e1-Paper-Conference.pdf.

AXA Versicherungen AG. Allgemeine Vertragsbedingungen (AVB) Haushaltversicherung: Privathaftpflicht, Hausrat, Cyber Plus, Zusatzversicherungen und Services. Policy terms, 16162DE, 2021. URL https://www.axa.ch/servlets/external/docstoredocument?accesscode=ag58y. Ausgabe 10.2021. Accessed 24 September 2026.

David Basin, Søren Debois, and Thomas Hildebrandt. Proactive enforcement of provisions and obligations. *Journal of Computer Security*, 32(3):247–289, 2024. doi: 10.3233/JCS-210078.

Sheng Bi, Kai Yu, Jiaqi Li, Xiaoyan Wang, Fuhui Sun, Zeyi Miao, Jing Wang, Xin Geng, and Lusheng Wang. CLER: A benchmark for Chinese litigation evidence reasoning. *Information Processing & Management*, 63(5):104667, 2026. doi: 10.1016/j.ipm.2026.104667. URL https://www.sciencedirect.com/science/article/pii/S0306457326000592.

Samuel R. Bowman and George Dahl. What will it take to fix benchmarking in natural language understanding? In *Proceedings of NAACL-HLT*, pp. 4843–4855, 2021. doi: 10.18653/v1/2021.naacl-main.385.

Zhaorun Chen, Mintong Kang, and Bo Li. ShieldAgent: Shielding agents via verifiable safety policy reasoning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix

Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8313–8344. PMLR, 2025. URL https://proceedings.mlr.press/v267/chen25ae.html.

Jiseong Chung, Ronny Ko, Wonchul Yoo, Makoto Onizuka, Sungmok Kim, Tae-Wan Kim, and Won-Yong Shin. GraphCompliance: Aligning policy and context graphs for LLM-based regulatory compliance. *arXiv preprint arXiv:2510.26309*, 2025. URL https://arxiv.org/abs/2510.26309.

CSS Versicherung AG. Hausratversicherung: Allgemeine Versicherungsbedingungen (AVB). Policy terms, Form. 519d-01.24-eff, 2024. URL https://www.css.ch/content/dam/css/de/documents/privatkunden/richtig-versichert/avb/519_d_avb_hausratversicherung.pdf. Ausgabe 02.2024. Accessed 24 September 2026.

CSS Versicherung AG. Schadenanzeige Hausrat / Gebäude. Claim notification form, Form. 15d-08.26-pdf, 2026. URL https://www.css.ch/content/dam/css/de/documents/privatkunden/richtig-versichert/formulare/15_d_formular_schadenanzeige_hausrat.pdf. Accessed 24 September 2026.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3563–3578, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.212. URL https://aclanthology.org/2024.findings-acl.212/.

Weihong Du, Wenrui Liao, Hongru Liang, and Wenqiang Lei. PAGED: A benchmark for procedural graphs extraction from documents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10829–10846, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.583. URL https://aclanthology.org/2024.acl-long.583/.

Robert Geirhos, Jørn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2:665–673, 2020. doi: 10.1038/s42256-020-00257-z.

Generali Allgemeine Versicherungen AG. Allgemeine Versicherungsbedingungen (AVB) Haushaltversicherung. Policy terms, 2026. URL https://www.generali.ch/content/dam/generali/pdf/produktdokumente/avb-ki/hausrat-haftpflicht/avb-haushaltversicherung-de.pdf. Ausgabe 2026. Accessed 24 September 2026.

David Graus. From legal text to executable decision models: Evaluating structured representations for legal decision model generation. *arXiv preprint arXiv:2604.17153*, 2026.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 79081–79094. Curran Associates, Inc., 2023. doi: 10.52202/075280-3459. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/f9f54762cbb4fe4dbffdd4f792c31221-Paper-Conference.pdf.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. Annotation artifacts in natural language inference data. In *Proceedings of NAACL-HLT*, pp. 107–112, 2018. doi: 10.18653/v1/N18-2017.

Dexter Hadley. CANONIC: Governance is compilation. *arXiv preprint arXiv:2607.05410*, 2026.

Divyansh Kaushik and Zachary C. Lipton. How much reading does reading comprehension require? A critical investigation of popular benchmarks. In *Proceedings of EMNLP*, pp. 5010–5015, 2018. doi: 10.18653/v1/D18-1546.

Onur Selim Kilic, Yeti Z. Gurbuz, Cem O. Yaldiz, Afra Nawar, Etrit Haxholli, Oğul Can, and Eli Waxman. Guideline2Graph: Profile-aware multimodal parsing for executable clinical decision graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 6942–6951, June 2026. URL https://openaccess.thecvf.com/content/CVPR2026W/Med-Reasoner/html/Kilic_Guideline2Graph_Profile-Aware_Multimodal_Parsing_for_Executable_Clinical_Decision_Graphs_CVPRW_2026_paper.html.

Myung Ho Kim. Evidence-obligation pool-gated retrieval: Stable multi-cycle retrieval via evidence ledger and warrant judge. Preprints.org, 2026. URL https://www.preprints.org/manuscript/202607.1060. Version 1, posted 15 July 2026; not peer reviewed.

Dawei Li, Renliang Sun, Yue Huang, Ming Zhong, Bohan Jiang, Jiawei Han, Xiangliang Zhang, Wei Wang, and Huan Liu. Preference leakage: A contamination problem in LLM-as-a-judge. In *International Conference on Learning Representations*, 2026. arXiv:2502.01534.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of EMNLP*, pp. 3102–3116, 2023. doi: 10.18653/v1/2023.emnlp-main.187.

Shuyue Stella Li, Vidhisha Balachandran, Shangbin Feng, Jonathan S. Ilgen, Emma Pierson, Pang Wei Koh, and Yulia Tsvetkov. MediQ: Question-asking LLMs and a benchmark for reliable interactive clinical reasoning. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 28858–28888. Curran Associates, Inc., 2024. doi: 10.52202/079017-0908. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/32b80425554e081204e5988ab1c97e9a-Paper-Conference.pdf.

Hugo A. López, Søren Debois, Tijs Slaats, and Thomas T. Hildebrandt. Business process compliance using reference models of law. In *Fundamental Approaches to Software Engineering*, pp. 378–399, 2020. doi: 10.1007/978-3-030-45234-6_19.

Andrew Lou and David Shin. Legal reasoning is not lawyering: Rethinking legal benchmarks for pro se access to justice. *arXiv preprint arXiv:2606.23716*, 2026.

Yuxing Lu, Yicheng Chen, Shanchan Wu, and Sercan Ö. Arık. Procedural graphs: Self-evolving execution structures for LLM agents. *arXiv preprint arXiv:2609.09153*, 2026.

R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of ACL*, pp. 3428–3448, 2019. doi: 10.18653/v1/P19-1334.

Masha Medvedeva and Pauline McBride. Legal judgment prediction: If you are going to do it, do it right. In *Proceedings of the Natural Legal Language Processing Workshop 2023*, pp. 73–84, 2023. doi: 10.18653/v1/2023.nllp-1.9.

Jacob Menick, Maja Trebacz, Vladimir Mikulik, John Aslanides, Francis Song, Martin Chadwick, Mia Glaese, Susannah Young, Lucy Campbell-Gillingham, Geoffrey Irving, and Nat McAleese. Teaching language models to support answers with verified quotes. *arXiv preprint arXiv:2203.11147*, 2022. URL https://arxiv.org/abs/2203.11147.

Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. Catala: A programming language for the law. *Proceedings of the ACM on Programming Languages*, 5(ICFP):1–29, 2021. doi: 10.1145/3473582. URL https://arxiv.org/abs/2103.03198.

OASIS LegalRuleML Technical Committee. LegalRuleML core specification version 1.0. OASIS Standard, 2021. URL https://docs.oasis-open.org/legalruleml/legalruleml-core-spec/v1.0/os/legalruleml-core-spec-v1.0-os.html.

Object Management Group. Case management model and notation (CMMN), version 1.1. OMG Specification, 2016. URL https://www.omg.org/spec/CMMN/1.1.

Object Management Group. Decision model and notation (DMN), version 1.5. OMG Specification, 2024. URL https://www.omg.org/spec/DMN/1.5.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.248. URL https://aclanthology.org/2023.findings-emnlp.248/.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. Hypothesis only baselines in natural language inference. In *Proceedings of *SEM*, pp. 180–191, 2018. doi: 10.18653/v1/S18-2023.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 69679–69703, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/adbe936993aa7cf41e45054d8b72f183-Abstract-Conference.html.

Schweizerischer Versicherungsverband SVV. Allgemeine Bedingungen (AVB) für die Hausratversicherung. Non-binding model policy conditions (unverbindliche Musterbedingungen), Ausgabe 2022, 2022. URL https://svv.ch/sites/default/files/media/documents/2024-02/AVB%20Hausratversicherung%202022.pdf. Version 01.04.2022. Accessed 24 September 2026.

ServiceNow. Add a document list definition to a service definition. Financial Services Operations documentation, 2026. URL https://www.servicenow.com/docs/r/financial-services-operations/insurance-claims/add-document-list-definition-to-service-definition.html. Brazil release; updated September 10, 2026. Accessed 24 September 2026.

Simpego Versicherungen AG. Simpego Haushaltversicherung Home: Allgemeine Versicherungsbedingungen (AVB). Policy terms, 2024. URL https://simpego.ch/dam/AVB_Simpego_Home_de.pdf. Ausgabe Dezember 2024. Accessed 24 September 2026.

Saku Sugawara, Kentaro Inui, Satoshi Sekine, and Akiko Aizawa. What makes reading comprehension questions easier? In *Proceedings of EMNLP*, pp. 4208–4219, 2018. doi: 10.18653/v1/D18-1453.

Swiss Confederation. Bundesgesetz vom 2. April 1908 über den Versicherungsvertrag (Versicherungsvertragsgesetz, VVG), SR 221.229.1. Fedlex, Swiss Federal Chancellery, 2024. URL https://www.fedlex.admin.ch/eli/cc/24/719_735_717/20240101/de. Consolidated version of 1 January 2024 (Stand am 1. Januar 2024), German text; Arts. 38–41a, 45 and 46 used. Accessed 24 September 2026.

Swiss Confederation. Bundesgesetz vom 30. März 1911 betreffend die Ergänzung des Schweizerischen Zivilgesetzbuches (Fünfter Teil: Obligationenrecht), SR 220. Fedlex, Swiss Federal Chancellery, 2026. URL https://www.fedlex.admin.ch/eli/cc/27/317_321_377/20260101/de. Consolidated version of 1 January 2026 (Stand am 1. Januar 2026), German text; tenancy provisions Arts. 257d–273 used. Accessed 24 September 2026.

Maksym Taranukhin, Shuyue Stella Li, Evangelos Milios, Geoff Pleiss, Yulia Tsvetkov, and Vered Shwartz. InfoGatherer: Principled information seeking via evidence retrieval and strategic questioning. *arXiv preprint arXiv:2603.05909*, 2026. URL https://arxiv.org/abs/2603.05909.

Paul Thomas, Nick Craswell, Mark Sanderson, Seth Spielman, Robert Sim, and Ryen W. White. Evaluation validity in information retrieval. In *Proceedings of SIGIR*, pp. 2734–2746, 2026. doi: 10.1145/3805712.3808538.

Roman Vaculín, Richard Hull, Terry Heath, Craig Cochran, Anil Nigam, and Piyawadee Sukaviriya. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *2011 IEEE 15th International Enterprise Distributed Object Computing Conference*, pp. 151–160, 2011. doi: 10.1109/EDOC.2011.36.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. StateFlow: Enhancing LLM task-solving through state-driven workflows. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=3nTbuygoop.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. GuardAgent: Safeguard LLM agents via knowledge-enabled reasoning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68316–68342. PMLR, 2025. URL https://proceedings.mlr.press/v267/xiang25a.html.

Ruixuan Xiao, Wentao Ma, Ke Wang, Yuchuan Wu, Junbo Zhao, Haobo Wang, Fei Huang, and Yongbin Li. FlowBench: Revisiting and benchmarking workflow-guided planning for LLM-based agents. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10883–10900, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.638. URL https://aclanthology.org/2024.findings-emnlp.638/.

Chen Yang, Ruping Xu, Ruizhe Li, Bin Cao, and Jing Fan. Business as rulesual: A benchmark and framework for business rule flow modeling with LLMs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 35198–35220. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1625.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 9965–10017, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b126cc38b8638e07bef37e7b2bb72bf-Paper-Conference.pdf.

Chenglin Yu, Li Yin, Qingxin Fan, Ying Yu, Runyang Ray Zhong, and Ming Li. Compile, then page: Executable SOP programs and a capability-gated runtime for procedural LLM agents, 2026. URL https://arxiv.org/abs/2607.11346v3. Version 3, 23 July 2026.

Yu Yuan, Lili Zhao, Kai Zhang, Guangting Zheng, and Qi Liu. Do LLMs overcome shortcut learning? An evaluation of shortcut challenges in large language models. In *Proceedings of EMNLP*, pp. 12188–12200, 2024. doi: 10.18653/v1/2024.emnlp-main.679.

Xuan Zhang, Yang Deng, Zifeng Ren, See-Kiong Ng, and Tat-Seng Chua. Ask-before-Plan: Proactive language agents for real-world planning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10836–10863, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.636. URL https://aclanthology.org/2024.findings-emnlp.636/.

SUPPLEMENTARY MATERIAL

The sections below define the planner, document the datasets and scoring, and map the experiments to the released software.

Section	What it contains	Page
A	Formal planner and the version executed in each study	15
B	Cases, sources, reference construction and dataset limits	17
C	Study A: frozen protocol, scoring, tests, errors and cost	19
D	Study B: original evaluation, request scoring and full results	24
E	Offline replay, data exposure and software correspondence	28
F	Closest prior work and the scope of our comparisons	29

A FORMAL METHOD AND STUDY IMPLEMENTATIONS

We define the full planner, then specify the version executed in each study. Table 6 compares their rules, inputs and execution.

A.1 THE FULL PLANNER

Applicability and timing answer different questions: does the obligation apply, and may its documents be requested now? In Figure 2, establishing the defect's extent requires both the mould branch and notification of the landlord. The planner checks applicability first, then timing.

The LLM assigns each condition c a value $v(c) \in \{\mathsf{T}, \mathsf{F}, \mathsf{U}\}$: true, false or unresolved. Branch b has condition c_b and immediate parents $\mathrm{pa}(b)$; J_b specifies whether all or any parents must apply. Obligation o belongs to branch $b(o)$, with its own condition c_o and timing condition t_o. Three-valued conjunction $\wedge_3$ returns false if either input is false, unresolved if either remaining input is unresolved, and true otherwise. Code computes

$$\begin{aligned} q_b &= v(c_b) \wedge_3 J_b\big(\{q_p\}_{p \in \mathrm{pa}(b)}\big), \\ a_o &= q_{b(o)} \wedge_3 v(c_o), \qquad r_o = a_o \wedge_3 v(t_o). \end{aligned} \tag{1}$$

Branch state q_b combines the branch's condition with its parents; a root has no parent requirement. Obligation state a_o also checks the obligation's condition, and request state r_o adds timing. Negation $\neg$ swaps true and false; disjunction $\vee$ is the dual of $\wedge_3$.

When $r_o = \mathsf{T}$, derive requests by the route rule below. When $a_o = \mathsf{F}$, drop the obligation's requests. Otherwise hold them, asking about unresolved conditions (Appendix D.3). Algorithm 1 omits timing and shows branches requiring all parents.

Local scope uses the same judgments and sets $q_b = v(c_b)$, omitting inherited conditions. With the ordering $\mathsf{F} < \mathsf{U} < \mathsf{T}$, conjunction $\wedge_3$ returns the lowest input. Adding an inherited condition can therefore only lower a_o and r_o: full process scope can hold or drop a Local scope request, but cannot add one. This is the effect tested in Study B.

Obligation o needs evidence capabilities $\mathcal{E}_o$. Each capability e establishes fact f_e through any acceptable route in $\mathcal{R}_e$. For documents on file H, the LLM assesses presence and adequacy. A route is satisfied, $\rho(H) = \mathsf{T}$, only if its documents are present, adequate and jointly sufficient.

For capability e and route ρ, let $M_{e\rho}$ contain the missing or inadequate documents and $V_{e\rho}$ the items whose presence or adequacy needs review. Code prefers a satisfied route, then fewer missing or inadequate documents, then fewer items to review. Sorted document identifiers and the route identifier break remaining ties. This is a *lexicographic* choice: compare the first criterion, then use the next only to break a tie. For an obligation with $r_o = \mathsf{T}$, choose

$$\rho_e^* = \operatorname*{arg\,min}^{\mathrm{lex}}_{\rho \in \mathcal{R}_e} \big(\mathbf{1}\{\rho(H) \neq \mathsf{T}\},\, |M_{e\rho}|,\, |V_{e\rho}|,\, \mathrm{sort}(M_{e\rho}),\, \mathrm{id}(\rho)\big) \tag{2}$$

and request $\bigcup_{o:\, r_o = \mathsf{T}} \bigcup_{e \in \mathcal{E}_o} M_{e\rho_e^*}$. The indicator $\mathbf{1}\{\cdot\}$ is one when its condition holds and zero otherwise. The union collects the missing or inadequate documents from each chosen route. Algorithm 1's route for a fact is ρ_e^*, the route for the capability establishing f_e.

Code chooses a route separately for each capability and records an evidence gap if no route exists. It then selects the next action in this order: ask a question, review evidence, request documents, review an open gap, or take a permitted case action. The first applicable action wins.

A.2 STUDY B: INHERITED BRANCH CONDITIONS

Study B compiles three tenancy templates. Each obligation inherits only its dispute-type condition, judged by the LLM. Timing always holds ($r_o = a_o$), and each capability has one route containing one document. The study tests inherited applicability, holds and document assessments; it cannot distinguish timing or route-selection policies.

Table 6: **The two studies share the planner's design but no sources, cases or code.** Study A runs the reduced planner of Appendix A.3, and Study B the full planner of Appendix A.1.

	Study A (theft)	Study B (tenancy)
Planner		
Rules come from	Law and policy texts, which LLMs turn into evidence rules mapped to document routes	Three synthetic tenancy templates, which code compiles; every method uses the result
Request activated by	The rule's own condition; an unresolved one becomes a question	The obligation's own condition and the inherited dispute type; an unresolved one becomes a question
Documents on file	Left out without an adequacy check (no case has any)	Each judged for presence and adequacy
Routes requested	Every route of every capability	The chosen route; each capability has one route with one document
When the case changes	The plan is recomputed from the edited case	The plan is recomputed from new judgments
Evaluation		
Cases	9 development and 27 held-out pairs over nine branch conditions; 72 cases	60 development and 90 held-out claims from 11 and 17 families covering defects, terminations and rent increases
Model calls	One call per condition over all cases of a split for CASEPATH; one call per case for Direct and Graph context, two for Evidence-first	Up to two per claim for each prediction method; the controls reuse CASEPATH's judgments
What is scored	Checklist changes between the two cases of a pair	Complete outputs (the benchmark's evaluator) and requested documents (document-request scoring)
Statistics	Bootstrap over conditions, exact paired tests and a wrong-condition control; the three accuracy criteria fail	Weighted descriptive contrasts; no significance claim

A.3 STUDY A: INDEPENDENT RULES AND ALL EVIDENCE ROUTES

Study A treats each rule as an independent obligation: $a_o = r_o = v(c_o)$, or T for an unconditional rule. True activates the rule; unresolved produces a question. The planner requests every route for active rules and every route confirming a true condition. It omits documents on file without checking adequacy and does not check enclosing branches.

Let $O(x)$ be the active rules for case x, and $\mathcal{R}_c^{\text{conf}}$ the routes that confirm condition c. The requested documents are

$$D(x, H) = \Big[\bigcup_{o \in O(x)} \bigcup_{e \in \mathcal{E}_o} \bigcup_{\rho \in \mathcal{R}_e} \rho \;\cup \bigcup_{c:\, v(c)=\mathsf{T}} \bigcup_{\rho \in \mathcal{R}_c^{\text{conf}}} \rho \Big] \setminus H. \tag{3}$$

No Study A case has documents on file, so $D(x) = D(x, \emptyset)$. Each request keeps its rule, required fact, evidence capability and source quotation. All rule corrections were made from sources alone and remain in the record.

Requesting every route has two consequences. An unconditional rule places its documents on both checklists: the experts' appointment cannot be added again when the formal expert procedure activates, causing three misses (Table 11). Splitting receipt and valuation into separate routes would still request both.

B DATASETS, SOURCES AND REFERENCES

The theft dataset describes how a checklist should change; the tenancy dataset describes what a complete case output should contain. Both release case inputs, scoring references and verification scripts. Figures 3 and 4 show the construction, and Tables 7 and 9 list the contents. Datasheets are in `data/`. Code uses Apache 2.0 and data CC BY 4.0, except attributed third-party quotations.

B.1 STUDY A: 36 PAIRS OF THEFT CASES

Each theft case is an English story with nine condition statements. All conditions are off in the first case of a pair; the second turns one on. For example, "No stolen item is separately scheduled in the policy with an insured value of CHF 1,000 or more" becomes "A stolen watch is separately scheduled in the policy with an insured value of CHF 4,000." Methods share the source collection and document catalogue; Evidence-first retrieves a subset for its prompt.

Each pair has a reference set of acceptable changes. When more than one document can meet a requirement, the reference enumerates the allowed alternatives and the scorer selects the closest set (Appendix C.2).

Code derives references from the LLM-generated source contract, which links decisions, conditions, facts, evidence and documents to verbatim quotations. Before held-out evaluation, two amendments replaced "receipt and valuation" with "receipt or valuation" and added the bicycle purchase voucher. The release includes these derivations, generators, cases, references and all four methods' checklists. CASEPATH's frozen representation is a prediction, separate from the references.

Table 7: **The 36 pairs cover nine branch conditions, and each records the change its one fact requires.** Counts over all pairs.

	Count	Detail
Design		
Pairs	36	9 development, 27 held out
Cases	72	40 distinct texts; within a story, the nine cases with every condition off are identical
Branch conditions	9	three groups: stolen property, events after the claim, settlement and recourse
Stories	4	claim settings, the first for development
Words per case	141–155	English
Reference		
Required changes	44	11 development, 33 held out; all additions
Pairs with alternatives	4	one requirement: a receipt or an expert valuation
Condition states	648	nine per case, fixed by construction
Shared inputs		
Source passages	197	from 7 documents (Table 8)
Document catalogue	33	document types, all named in the sources
Release		
Methods recorded	4	each with both checklists of all 36 pairs
Scorer		frozen; recomputes every Study A result offline
Licence		data CC BY 4.0 except the quoted policy terms, claim form and model conditions; code Apache 2.0

People, policies and claims are invented; thresholds and periods come from the attributed excerpts in Table 8. Federal enactments such as the VVG are not copyright-protected. Eight frozen passages cite the wrong document but occur verbatim elsewhere in the corpus. Release notes correct their labels and identify two stale fields unused by the scorer; frozen files remain unchanged.

Table 8: **The 197 passages are verbatim excerpts from seven public documents.** Three passages appear word for word in more than one document, so the counts sum to 201. The release gives the URL and file hash of each document.

Document	Edition	Passages
Federal law		
Insurance Contract Act (VVG), Arts. 38–41a, 45 and 46 (Swiss Confederation, 2024)	1 January 2024	26
Household policy terms		
CSS Versicherung AG (2024), household contents	02.2024	67
AXA Versicherungen AG (2021)	10.2021	17
Generali Allgemeine Versicherungen AG (2026)	06.2026	25
Simpego Versicherungen AG (2024)	12.2024	9
Claim form		
CSS Versicherung AG (2026), household contents and building	08.2026	19
Industry model conditions		
Schweizerischer Versicherungsverband SVV (2022), household contents	01.04.2022	38
Sum over documents		201
Distinct passages		197

The design isolates one changed condition. Each case activates at most one, all reference changes are additions, and one requirement allows alternatives. Interacting conditions and correct withdrawals are therefore outside this test. The source contract has no legal-expert review. Held-out cases use new stories with the same conditions and were read once after the freeze. A future fresh test must exclude these now-public cases from development.

B.2 Study B: 150 tenancy claims

Each tenancy claim contains an *intake packet*: channel, date, language, original message and attachments. Its *source registry* lists the 136 to 140 citable units: 101 process-template entries, 19 Swiss tenancy-law passages (Swiss Confederation, 2026), 15 to 19 message passages and an attachment inventory. Each claim is self-contained.

References come from the generator's state, independently of method predictions. They specify the dispute and its branches, process concepts and typed relations, step order, obligations, document and intake states, and next action. Each grounded concept has three or four *source locators*: pointers to its supporting material. A document marked *conditional* is held until a condition is resolved (Table 9).

The evaluator reports 33 metrics. Acceptance requires eight checks, including valid paths, complete obligations, correct document states and absence of forbidden elements. Its self-tests accept references and reject empty or request-everything outputs. These tests validate the scorer's behaviour; method outcomes and document-request scoring are reported in Appendices D.1 and D.2.

The *state-change track* applies four updates to each of 12 base claims: supply an adequate document, make a branch fact false, leave it unresolved, or resolve all active facts. Nine endpoints measure request status and preservation of unaffected graph elements. References accompany 24 development variants. Those for 24 held-out variants were regenerated from released references and match their precommitted hashes.

Claims, messages, parties, images, templates and references are synthetic. Real sources comprise 19 copyright-exempt legal passages and Canton of Zurich forms in 34 of the 47 PDF attachments.

Raw message headers named a real insurer's mailbox; prediction methods did not read that field. The release substitutes a fictional address of equal byte length and links all 300 changed files to original hashes. These links can be verified once the name is disclosed after review. The 60 development references accompany the benchmark; the 90 held-out references are released separately, byte-identical to their pre-evaluation hashes in `cohort.json`.

Table 9: **The 150 claims cover three dispute types in two languages, and each carries a full structured reference.** Counts over all claims; ranges are per claim.

	Count	Detail
Intake		
Claims	150	60 development, 90 held out
Dispute types	3	defects, lease terminations and rent increases, 50 claims each
Families	28	11 development, 17 held out; no family crosses the split
Languages	2	English 75, Swiss Standard German (de-CH) 75
Channels	4	email 70, portal 46, chat 24, phone note 10
Attachments	57	in 45 claims: 47 PDF notices, demands and forms, 10 photos
Words per message	138–209	median 179
Reference		
Concepts (nodes)	7,800	46–55; includes 600 that an output must not contain
Relations	8,100	47–58; includes 600 that an output must not contain
Branch conditions	750	3–7
Obligation chains	1,397	8–10, each from decision to fact, evidence and document
Order constraints	3,450	20–25
Document states	1,397	missing 961, conditional 395, provided but insufficient 41
Next action	150	process step 100, document request 35, escalation to an external specialist 15
Release		
Evaluator		33 metrics and 8 acceptance gates per claim
State-change track	48	variants: 12 base claims, each updated in four ways
Licence		data CC BY 4.0 except the cantonal form templates; code Apache 2.0

Claims concern one canton at intake. Only the one-update variants test state changes. Each obligation has one acceptable document, images lack region labels, and amounts, liability and insurance coverage are unlabelled. Alternative-route selection is therefore untested.

Next actions are labelled but unscored. Only defect claims vary: they contain all 35 document requests and 15 specialist escalations. Every termination and rent-increase claim expects the step preserving the challenge or extension deadline. Appendix E.2 records prior input exposure. Future studies can reserve the family-disjoint split by excluding it from training and tuning.

C Study A: protocol, scoring and diagnostics

Study A scores changes between paired checklists under a frozen set of rules and references.

C.1 Rules and frozen protocol

LLMs extracted 308 statements from 197 passages (Table 8). The frozen representation covers these extracted statements with 98 evidence rules, 13 statements requiring no evidence, 87 conditions and 33 document types.

Table 4 groups nine conditions. *Stolen property* covers a scheduled valuable above the source threshold, a bicycle or e-bike, and claimed SIM misuse. *Events after the claim* covers an insurer's written demand with forfeiture warning and deadline, declination and recovery of stolen items or news of them. *Settlement and recourse* covers a formal expert procedure, repair above the consent threshold and a liable third party.

The pre-evaluation freeze fixes what generates the requests and what scores them. Hashes bind the sources, benchmark and its two amendments, extracted rules, evidence mapping, LLM condition interpreter, planner, model, provider, temperature and analysis. The freeze also includes the Development split, baseline Development predictions and method tests. All baselines are custom implementations (Appendix F).

Development used 9 pairs (18 cases). After its predeclared checks passed, 27 Held-out pairs (54 cases) were read once. All eight criteria in Table 10 preceded that read; frozen files remained unchanged. Hashes document this internal freeze.

C.2 Scoring checklist changes

A pair contains case x^- with a condition off and case x^+ with it on. Let $D(x)$ be the predicted checklist. A document added in x^+ has sign $+d$; one withdrawn has sign $-d$. The predicted changes are

$$\Delta(D) = \{+d : d \in D(x^+) \setminus D(x^-)\} \cup \{-d : d \in D(x^-) \setminus D(x^+)\}. \tag{4}$$

Pair i may have several acceptable references $G \in \mathcal{G}_i$, one for each allowed combination of documents. The scorer compares the predicted changes $P_i = \Delta(D)$ with each reference and keeps the one that maximises the following criteria in order:

$$\big(F_1(P_i, G),\ |P_i \cap G|,\ \mathrm{Prec}(P_i, G),\ \mathrm{Rec}(P_i, G),\ -|G|,\ \mathrm{sort}(G)\big);$$

the final sorted list breaks any remaining tie. Here precision, Prec, is the share of predicted changes that are correct; recall, Rec, is the share of required changes made. F_1 balances both. For a requirement with alternative documents, requesting both when one suffices adds a wrong change.

Precision and recall are zero for empty denominators; F_1 is one when prediction and reference are both empty. *Checklist change* F_1 pools counts over the 27 held-out pairs (Table 3). *Pair* F_1 scores a pair and is averaged within each condition (Table 4). All reference changes are additions. The withdrawal score therefore measures avoidance of unnecessary removals.

C.3 Uncertainty, paired tests and condition controls

Table 10: **Five predeclared criteria pass; all three accuracy criteria fail.** Traceability asks whether changes keep their source and condition; specificity asks whether they follow the changed condition. "Met" compares the observed value with the fixed target; pct. means percentile.

	Criterion	Target	CasePath	Met
Accuracy	Smallest F_1 margin over a baseline	≥ 0.10	0.076	no
	Largest Holm-corrected paired p	≤ 0.05	0.797	no
	Checklist change precision	≥ 0.70	0.62	no
Traceability	Changes traced to a changed condition	≥ 0.90	1.00	yes
	Required changes reachable by a source chain	≥ 0.80	0.82	yes
	Requested documents with a source chain	all	all	yes
Specificity	Changes per required change	0.75–1.50	1.18	yes
	Correct- vs. wrong-condition F_1	>97.5th pct.	0.667 vs. $\leq$0.083	yes

The nine conditions are the units of uncertainty. For 95% intervals, we bootstrap 10,000 samples (seed 20260917), keeping each condition's three Held-out stories together. The exact one-sided paired test enumerates 512 sign assignments of per-condition differences. Against Direct, Graph context and Evidence-first, raw p-values are 0.436, 0.398 and 0.043; Holm-corrected values are 0.797, 0.797 and 0.129. A gain must therefore persist across conditions to pass the test.

The *input-free control* fits the best constant change on eight conditions and tests it on the ninth. Its highest F_1 is 0.0: a fixed answer learned from other conditions scores nothing. The *wrong-condition control* reassigns saved changes to different conditions, using 8 cyclic shifts and 5,000 random derangements, which leave no condition in its original position. Its add-one p-value is $(1 + N_{\geq})/(1 + 5{,}008)$, where $N_{\geq}$ counts reassignments with F_1 at least as high as the observed score. None reaches 0.667, so $N_{\geq} = 0$; their maximum is 0.083 (Figure 10B). A lookup from conditions to documents would pass both controls, so Tables 4 and 11 also inspect the changes themselves.

C.4 WHY THE BASELINES CHANGE MORE DOCUMENTS

Figure 9 separates changes inside and outside the reference vocabulary defined in Section 4.1. Baselines generate the two checklists independently; CASEPATH derives them through the same rules.

Baselines often change unrelated documents such as payment details or a claim form. CASEPATH's nine outside-vocabulary changes are repair consents and the SIM bank statement (Table 11); its other six wrong changes are inside. This diagnostic uses the selected reference without rescoring. A document inside the vocabulary can still have the wrong branch or change sign.

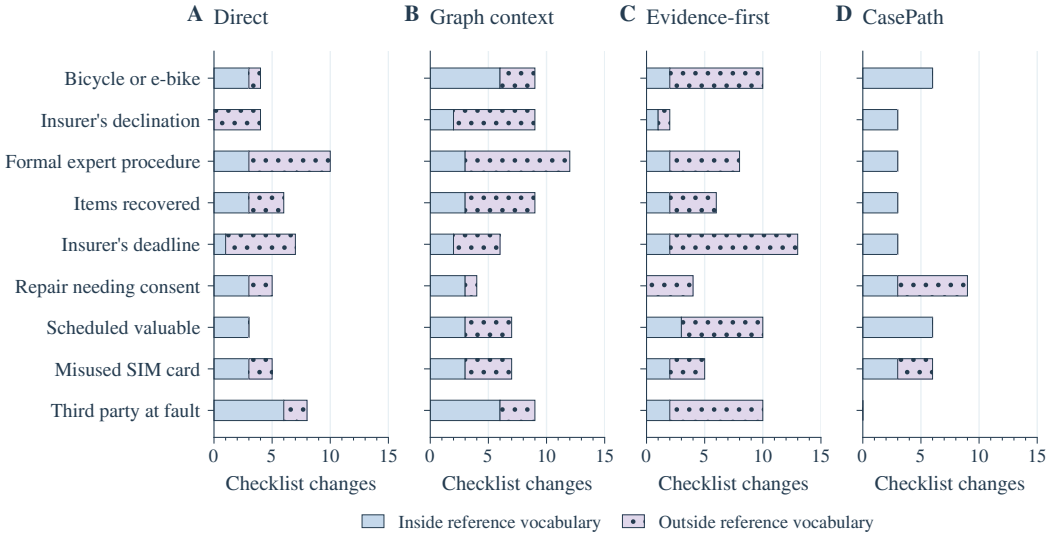


Figure 9: **Baseline changes leave the reference vocabulary on almost every condition.** Panels identify methods; rows identify the nine conditions in Table 4. The horizontal axis counts changes across three held-out stories per condition. Solid blue counts documents inside the reference vocabulary; dotted lilac counts those outside. Inside does not mean correct: the sign or branch can still be wrong.

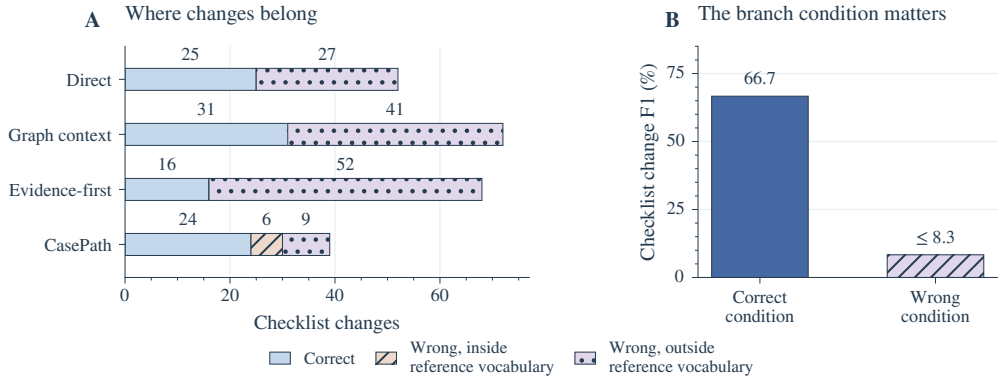


Figure 10: **Many wrong changes concern documents no benchmark branch governs.** (A) Counts by method: blue is correct; hatched peach is wrong but inside the reference vocabulary; dotted lilac is wrong and outside it. (B) Checklist change F_1 with the original references and the best of 5,008 wrong-condition reassignments of saved predictions. The second bar is an upper bound, not a new model run.

C.5 WHERE CASEPATH'S ERRORS COME FROM

CASEPATH makes 24 errors: 15 wrong changes and 9 misses, all in five conditions. The bicycle condition and three events after the claim have none. The rule and quoted case text behind each request locate 18 errors in LLM-generated rules: nine in overly broad conditions and nine in excessive routes or requests (Table 11). No rule was changed for this analysis.

The remaining six errors split between case interpretation and the all-routes policy (Appendix A.3). All nine misses occur because the required document appears on both checklists, leaving no addition when its condition changes.

Table 11: **Three quarters of CASEPATH's errors in Study A trace to LLM-generated rules.** Wrong and missed changes over the three held-out stories of each condition, grouped by where the error arose; the number after each group is its count of errors.

	Changes		
Condition	Wrong	Missed	What happened
In the rules: a condition broader than the reference (9)			
Repair needing consent	6	0	The condition on preserving evidence covers any planned alteration of damaged property, so a repair adds the insurer's and the police's consent
Third party at fault	0	3	The condition on recourse covers any third party, so the unidentified thief keeps the assignment of claims on both checklists
In the rules: a route or rule that asks for too much (9)			
Scheduled valuable	3	0	One route asks for the receipt and the expert valuation together, where the sources accept either
Misused SIM card	3	0	The rule's only route adds a bank statement to the theft report
Third party at fault	0	3	The claim-form section on the liable person comes from a rule without a condition, so it is on both checklists
In reading the case (3)			
Formal expert procedure	3	0	A written request for the procedure is read as one to appoint an expert, which adds the insurer's written demand
In the planner: every route requested (3)			
Formal expert procedure	0	3	Requesting every route keeps the experts' appointment on both checklists
Total	15	9	

Set the cost of a wrong change to 1 and the cost of a missed change to $\lambda \geq 0$. Then each method's total error cost is its wrong count plus λ times its missed count. Table 3 gives

$$L_{\mathrm{CasePath}} = 15 + 9\lambda, \qquad L_{\mathrm{Direct}} = 27 + 8\lambda,$$
$$L_{\mathrm{Graph}} = 41 + 2\lambda, \qquad L_{\mathrm{Evidence}} = 52 + 17\lambda.$$

CASEPATH has the lowest cost for $\lambda < (41 - 15)/(9 - 2) = 26/7$; Graph context has the lowest above it. Direct and Evidence-first are never cheapest (Figure 11). Here L_{Graph} and L_{Evidence} denote Graph context and Evidence-first.

The trade-off is especially sensitive to the third-party condition. Six of CASEPATH's nine misses occur there because both required documents are already on both lists. Removing only those six misses would move the cost crossing from 3.7 to 26. This is a counterfactual calculation from observed counts, with equal prices for documents and no uncertainty estimate, rather than a measurement of customer costs.

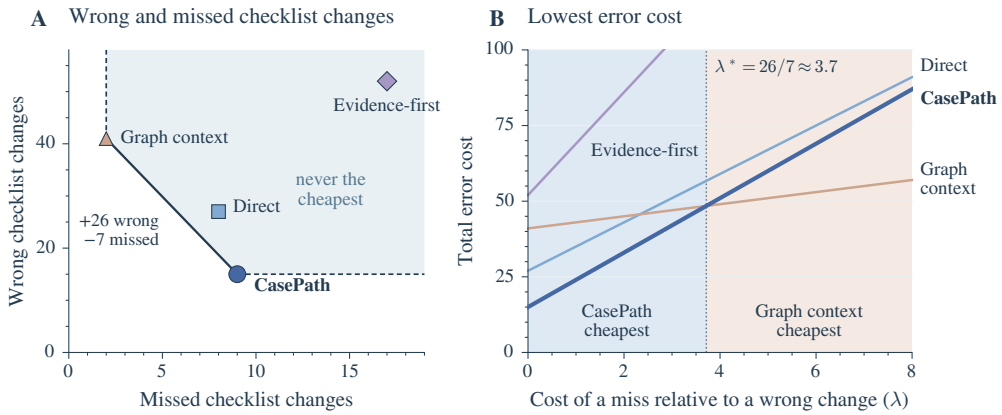


Figure 11: **The preferred method depends on the cost assigned to a missed change.** (A) Horizontal position counts missed checklist changes; vertical position counts wrong ones. The arrow from CASEPATH to Graph context trades 7 fewer misses for 26 more wrong changes. Shading marks methods that are never cheapest. (B) λ is the cost of one miss relative to one wrong change; total error cost is wrong changes plus λ times misses. Labeled curves identify methods. The vertical divider at $\lambda^* = 26/7 \approx 3.7$ marks equal costs for CASEPATH and Graph context.

C.6 INFERENCE COST

CASEPATH batches all 54 Held-out cases for each condition; baselines process one case per call. Table 12 reports charges, and Figure 12 relates them to checklist change F_1. CASEPATH reads fewer prompt tokens than Direct but writes about five times as many completion tokens. Evidence-first reads twelve retrieved passages; Graph context also reads the graph and rules, producing the largest prompt.

Table 12: **CASEPATH costs more than Direct and less than Graph context, with more calls than either.** Held-out run; every method uses `openai/gpt-5.6-terra` at temperature 0 with one logged attempt per call. Evidence-first allows 18,000 completion tokens for extraction and 14,000 for planning. Charges exclude source preparation.

		Tokens			
Method	Calls	Prompt	Completion	Completion limit per call	Cost (USD)
CASEPATH	87	1,005,211	243,891	10,000	5.44
Direct	54	2,900,625	45,678	14,000	4.83
Graph context	54	5,345,421	42,663	14,000	8.41
Evidence-first	108	259,890	66,450	18,000/14,000	1.06

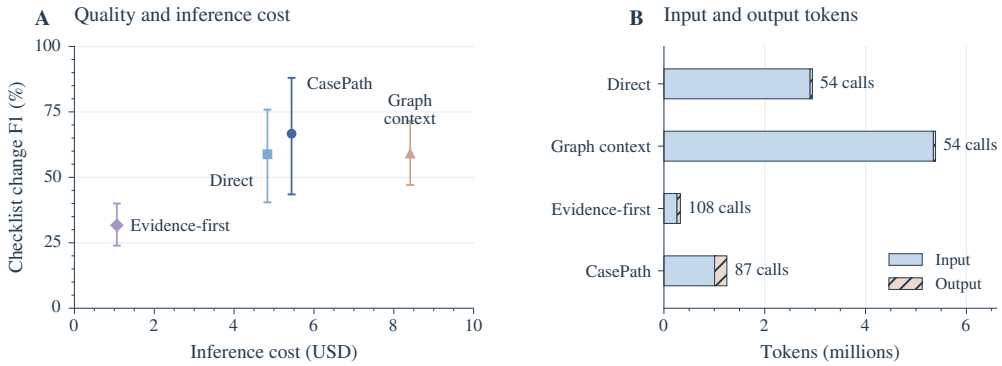


Figure 12: **CASEPATH has the highest checklist change F_1 at an intermediate inference cost.** (A) Horizontal position is the held-out run's provider charge in USD; vertical position is checklist change F_1, with 95% intervals. Labels identify methods. (B) Horizontal bar length is millions of tokens: blue Input is prompt tokens, peach hatched Output is completion tokens. Annotations count model calls. Charges exclude source preparation.

D STUDY B: PROTOCOL, SCORING AND RESULTS

Study B separates output availability from request quality. We distinguish the original whole-output evaluation from retrospective scoring of saved requests.

D.1 RUNS AND THE ORIGINAL EVALUATOR

Each method–claim combination is a *run*. Five prediction methods and two controls give 1,050 runs over 150 claims (Figure 13). Prediction methods use `openai/gpt-5.6-terra` through OpenAI, with a 2-call budget per claim, medium reasoning effort and at most 4,096 completion tokens per call. Temperature is not sent; its effective value is unverified. There are no tools or repeated samples. Controls reuse CASEPATH's judgments.

Three statuses describe output production. *Generated* means the output passes its method's structural checks; *Failed* means it does not; *Blocked* means a control lacks its prerequisite CASEPATH output. Each run preserves the output or provider response and failure reason. The original evaluator applies a separate acceptance check.

Failures include dangling relations, truncated or absent final answers, unknown concept identifiers, inconsistent document states, incomplete judgments and documents outside judged routes. Controls can also lack their prerequisite output. The four baselines attempt complete outputs within this two-call budget and fail on more than half of claims. Code-assembled CASEPATH outputs fail on 40 (Figure 13A). The comparison cost USD 89.06, including failed attempts; repaired outputs do not replace failures.

The original evaluator checks conditions against reference scenarios, which assign no values to the compiler's 50 condition symbols. All 558 generated outputs fail this check. Every one of the 1,050 runs receives failure penalties; none of the 12 held-out targets is met.

A target-free counterexample shows why substitution would change semantics: an absent condition is unresolved for the planner but raises a missing-variable error in the evaluator. Replacing it with null makes the evaluator return false, while the planner still returns unresolved. Neither retrospective analysis renames symbols, supplies defaults or injects predicted states into references.

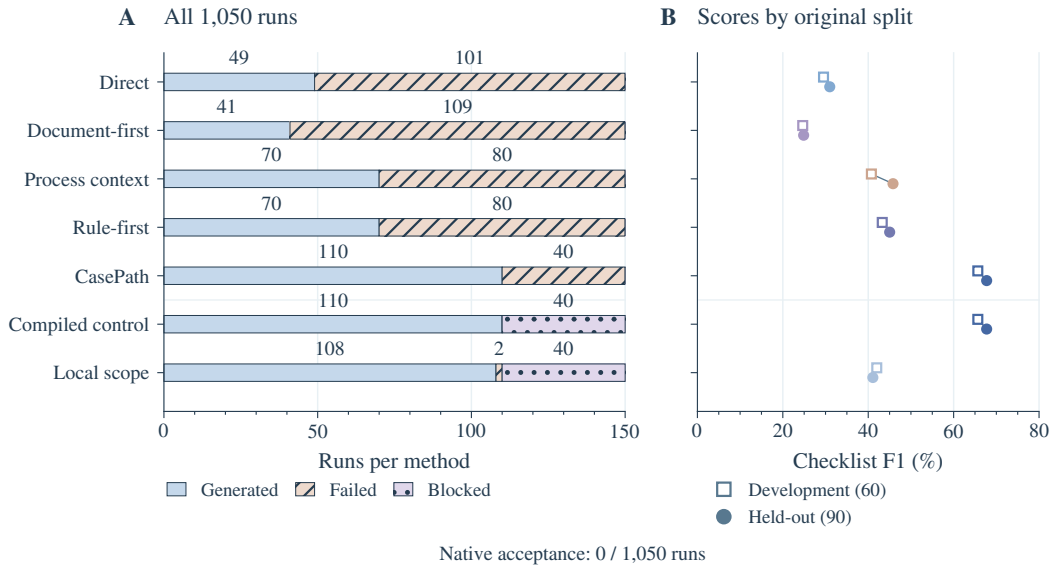


Figure 13: **Generation failures and original evaluator acceptance are separate outcomes.** Rows identify methods; the line separates prediction methods from controls. (A) Horizontal bars count runs: blue Generated passed structural checks; hatched peach Failed did not; dotted lilac Blocked lacked the required CASEPATH output. "Native acceptance" means passing the original evaluator, which accepted none. (B) Horizontal position is checklist F_1 from document-request scoring, with failures scored zero. Open squares are Development (60 claims), filled circles Held-out (90); connecting lines pair the two scores for each method.

D.2 DOCUMENT-REQUEST SCORING

D.2.1 WHAT EACH METRIC MEASURES

The metrics distinguish justified requests from recovered evidence. *Justified-request precision* is the fraction of requests whose obligation–fact–evidence–document chains match the reference. An available empty list scores one. *Critical-evidence recall* supplies the complementary completeness measure: it weights requirements by criticality and counts one as recovered only if its obligation is active and a complete evidence route is present.

Checklist F_1 and the *unnecessary request rate* compare document sets. Let P be the distinct documents requested now and G the reference documents. Document precision is $|P \cap G|/|P|$, the share of requests required by the reference. Document recall is $|P \cap G|/|G|$, the share of required documents recovered. Checklist $F_1 = 2|P \cap G|/(|P| + |G|)$ balances both; the unnecessary request rate is $|P \setminus G|/|P|$.

When both sets are empty, F_1 is one. An empty prediction has precision one and unnecessary request rate zero; an empty reference has recall one. These are scores for an available empty list. A failed output instead receives the penalties below.

D.2.2 WHICH OUTPUTS ARE SCORED

Document-request scoring uses the unchanged saved requests, public document mapping and original request scorer. It omits whole-output acceptance and does not evaluate compiler conditions against reference scenarios.

All 150 references supply an eligible document set. The 558 generated lists receive measured scores; 492 failed or blocked outputs receive the frozen worst-case penalties: zero when higher is better, one when lower is better. Failures add no invented requests, and missing reference labels never reduce denominators.

D.2.3 WEIGHTS AND PAIRED COMPARISONS

For claim i, let m_i be the number of families in its dispute type and n_i the number of claims in its family. Its weight is $\omega_i = 1/(3\, m_i\, n_i)$. Thus the three dispute types count equally, families within a type count equally, and claims within a family count equally.

As in Figure 8, the all-claim score is AQ: weighted output availability times weighted mean checklist F_1 conditional on an available list. Failed outputs contribute zero.

Let s_{ik}^{CP} and s_{ik}^{j} be CASEPATH's and baseline j's scores on claim i for metric k. The sign $\sigma_k \in \{-1, 1\}$ is positive when higher is better and negative when lower is better. A conventional paired contribution is $\sigma_k(s_{ik}^{\mathrm{CP}} - s_{ik}^{j})$. Positive values favour CASEPATH. The conservative contribution is the same when both outputs exist and -1 otherwise, including when only the baseline failed. The predeclared success rule requires the conservative contrasts in checklist F_1, critical-evidence recall and unnecessary request rate all to exceed their thresholds over all held-out claims.

D.3 BRANCH CHECKS AT FIXED MODEL JUDGMENTS

To isolate process control, both planners reuse the same CASEPATH judgments. Full process scope checks enclosing branches; Local scope omits that check (Appendix A.1). Each resulting *snapshot* contains the currently active nodes and relations. Unresolved elements remain recorded but inactive. Snapshots are scored without the original whole-output acceptance requirement.

Reference graphs and matching rules are unchanged. Each control retains its own graph and requests for the same claim. All 110 Compiled control snapshots and scores match CASEPATH.

The 108 matched claims underlying Figure 6 retain 635 of 728 required documents while unnecessary requests fall from 810 to 17. The following decomposition accounts for every removal.

D.3.1 WHAT HAPPENS TO EXCLUDED REQUESTS

Every removal follows the inherited dispute-type condition. Unresolved types hold 710 requests on 95 claims. False types drop 83 requested and 103 previously held documents on 11 claims. Holds rise from 1,387 to 1,994 on the matched claims, and total 2,006 across all 110 available CASEPATH outputs.

Development requests fall from 592 to 275; Held-out requests from 853 to 377. The same 259 and 376 valid requests remain. Across 108 matched claims, nodes fall from 8,931 to 3,261 and relations from 5,630 to 2,237, retaining all 2,331 matching nodes, 2,031 relations and 635 requests (Figure 7A). These are unweighted item counts; Table 13 instead applies the stated weights to all 150 claims.

D.3.2 COMPARISON WITH RANDOM DELETION

Random deletion provides a length-matched reference for selectivity. On claim i, Local scope makes ℓ_i requests, with ℓ_i^{valid} valid; full process scope makes ℓ_i^{full}. A uniformly random subset of size ℓ_i^{full}, sampled without replacement, retains $\ell_i^{\mathrm{valid}}\, \ell_i^{\mathrm{full}}/\ell_i$ valid requests in expectation. The expected fraction retained within a split is

$$\frac{\sum_i \ell_i^{\mathrm{valid}}\, \ell_i^{\mathrm{full}}/\ell_i}{\sum_i \ell_i^{\mathrm{valid}}},$$

with the two empty-list claims contributing zero to both sums. Calculated from the saved list sizes, this expectation is below one half (Figure 7B), compared with complete retention under the branch check.

In this dataset, valid-chain documents coincide with necessary documents, so justified-request precision equals one minus unnecessary request rate. With equal criticality weights, critical-evidence recall also equals *obligation completeness*: the fraction of active obligations with complete evidence.

The evaluator's *provenance-chain inheritance rate* checks inherited source locations. It is zero under both scopes. With a loss assigned whenever either scope lacks an output, justified-request precision still gains $+0.112$, $+0.103$ and $+0.107$ on Development, Held-out and All claims. Critical-evidence recall changes by -0.261, -0.289 and -0.278. Those recall differences come entirely from missing-output penalties; recall is identical wherever both outputs exist.

Table 13: **Full process scope improves precision with little change in recall.** Node and relation precision measure the share of predictions matched to the reference; recall measures the share of reference elements recovered. Request metrics are defined in Appendix D.2. All 150 claims count, including failed-output penalties, with equal dispute-type weight. Rates are 0–1; "Valid / all requests" gives raw counts. Bold marks higher precision. Compiled control matches CASEPATH and is omitted.

	Development		Held-out		All claims	
	CASEPATH	Local scope	CASEPATH	Local scope	CASEPATH	Local scope
Claims with output	44	44	66	64	110	108
Output graph						
Node precision	**0.523**	0.263	**0.536**	0.256	**0.531**	0.258
Node recall	0.542	0.542	0.544	0.526	0.543	0.532
Relation precision	**0.676**	0.378	**0.679**	0.368	**0.678**	0.372
Relation recall	0.374	0.374	0.381	0.370	0.379	0.372
Requests						
Justified-request precision	**0.704**	0.331	**0.732**	0.318	**0.721**	0.323
Critical-evidence recall	0.635	0.635	0.645	0.627	0.641	0.630
Valid / all requests	**259/275**	259/592	**384/385**	376/853	**643/660**	635/1,445

D.4 COMPLETE COMPARISON ACROSS METHODS

Figure 8 shows higher availability for code-assembled CASEPATH outputs, with available-list quality inside the prediction baselines' range. Local scope also assembles outputs in code and has similar availability, but its extra requests reduce available-list checklist F_1 to 57.4%.

Tables 14 and 15 report split scores and counts. A Development trace selected before target scoring illustrates a hold: judged arrears justify requesting the missing cure notice, while unresolved family-home status holds the spouse notice. The plan requests seven documents, holds four and asks the unresolved question.

More than half of Held-out comparisons lack at least one output. Each contributes -1 under the conservative rule, even when only the baseline failed, making all conservative contrasts negative (Table 16).

Table 14: **CASEPATH leads the prediction methods in every column on both splits.** Document-request scoring over all claims of each split; a failed output scores 0 for F_1 and 1 for the unnecessary request rate. Dispute types weigh equally, then families within a type and claims within a family (Appendix D.2). The controls reuse CASEPATH's judgments. Bold marks the best prediction method in each column.

	Claims with output		Checklist F_1 ↑		Unnecessary request rate ↓	
Method	Development	Held-out	Development	Held-out	Development	Held-out
Prediction methods						
CASEPATH	**44**	**66**	**0.657**	**0.677**	**0.296**	**0.268**
Direct	19	30	0.295	0.310	0.687	0.688
Document-first	17	24	0.247	0.249	0.732	0.737
Process context	27	43	0.407	0.458	0.570	0.538
Rule-first	28	42	0.433	0.450	0.545	0.540
Controls						
Compiled control	44	66	0.657	0.677	0.296	0.268
Local scope	44	64	0.420	0.411	0.669	0.682

Table 15: **CASEPATH holds more documents; Local scope requests more than twice as many.** Requested counts documents requested now; Correct counts those in the reference; Missed counts required documents absent from the list; Held counts documents waiting for a condition. Counts sum across available lists. Precision and recall use all claims, failure penalties and the stated weights, so they are not ratios of these totals. For CASEPATH and Local scope, they equal justified-request precision and critical-evidence recall. Bold marks the best prediction method.

	Documents, summed over lists				Over all claims	
Method	Requested	Correct	Missed	Held	Precision ↑	Recall ↑
Prediction methods						
CASEPATH	660	643	95	2,006	**0.721**	**0.641**
Direct	316	301	30	65	0.312	0.298
Document-first	247	237	39	139	0.265	0.236
Process context	449	433	36	31	0.451	0.434
Rule-first	439	432	36	36	0.458	0.432
Controls						
Compiled control	660	643	95	2,006	0.721	0.641
Local scope	1,445	635	93	1,387	0.323	0.630

Table 16: **Conventional contrasts favour CASEPATH; conservative contrasts, which count every missing output against it, do not.** Paired contrasts over the held-out claims (Appendix D.2); positive values favour CASEPATH, with the unnecessary request rate's sign reversed. A conservative contribution is -1 whenever either list is missing.

	Conventional		Conservative	
CASEPATH against	Checklist F_1	Unnecessary request rate	Checklist F_1	Unnecessary request rate
Direct	$+0.367$	$+0.419$	-0.758	-0.744
Document-first	$+0.429$	$+0.469$	-0.788	-0.785
Process context	$+0.219$	$+0.270$	-0.673	-0.657
Rule-first	$+0.227$	$+0.272$	-0.671	-0.650

E REPRODUCIBILITY AND SOFTWARE CORRESPONDENCE

The release supports offline score reproduction and inspection of each request. Table 17 distinguishes evaluated methods from the workbench and teaching examples.

E.1 OFFLINE REPLAY

All 1,050 prediction and failure records are released with the 150 references, source commitments, original evaluator and request scorer. Offline replay regenerates both phase reports, the combined predeclared report, all request scores and the descriptive report. The canonical JSON matches exactly; replay needs neither model calls nor numerical tolerance.

E.2 DATA EXPOSURE AND SCORING RECORDS

Held-out inputs were exposed during product work. An incomplete input/output audit also prevents proving that reference labels were never accessed. Scoring authenticated both prediction sets before reading references. Original scoring, symbol-mismatch diagnosis and request-only scoring have separate records preserving every failure. Released references match their recorded hashes and licence.

E.3 THE RELEASED SOFTWARE

The local workbench opens each of Study B's 150 synthetic claims with its message, attachments, process step, evidence requirements and next action. A user can inspect, correct, export and replay

the case. The single-user service records updates in a hash-chained journal; it sends nothing to a claimant.

A separate deterministic review selects one of three templates and executes six roles in order; only the process role changes state. This product feature makes no LLM calls and is outside the experiments.

Table 17: **What the release implements and how it was checked.** Method replay, benchmark tools and product features have distinct evidence.

Part	What it is	Evidence
Method		
Study A	Condition interpreter and planner of Equation 3, installed in the service as a separate endpoint	Offline replay of all 72 cases matches
Study B	Planner of Equations 1 and 2, shown on an authored teaching example	Section 5.2; 45 teaching states re-executed; no parity claim
Benchmark tools		
Study A	Scorer and explorer	Appendix C
Study B	Evaluator, document-request scorer and state-change track	Appendix D
Product		
Workbench	Interface, journal, agent review, correction and replay over the 150 claims	Demonstrates handling mechanics only

Study A has a direct implementation check. The verifier replays all 72 cases through the installed method and service using saved judgments. It matches condition values, document sets, next actions, all 36 paired changes and source quotations.

Study B's workbench is a separate implementation, with no parity claim. Its teaching example uses an invented repair policy; `replay_teaching.py` re-executes and checks 45 authored states. These demonstrate the planner, not benchmark performance.

Fresh LLM inference requires provider access and supports theft claims only. The anonymous mirror's string substitutions break the workbench launcher's integrity checks. Its download therefore supports source inspection, dataset verification and Study A replay, but does not start the workbench.

F Detailed comparison with prior work

Table 18 compares the inputs, operations and outputs of the closest systems.

F.1 Published systems and our custom baselines

Our task maps a case and its process to document requests. The cited workflow systems usually produce tool calls, environment actions or database states. Applying them to this task would require choices about inputs, state and actions, listed in Table 18.

StateFlow needs a manually designed state machine and prompts (Wu et al., 2024). Compile-then-Page requires machine-readable constraints and was not tested on natural-language procedures. At the recorded check, neither Compile-then-Page nor the evidence ledger released code; Procedural Graphs linked none.

Our custom baselines compare request-selection designs under shared inputs. They test supplied rules, process context and evidence-first generation; they do not reproduce the published systems in Table 18.

F.2 The closest precedents

The evidence ledger is the closest precedent (Kim, 2026). An LLM plans knowledge obligations for each query and judges support; code manages closure. The ledger holds obligations until a tool

Table 18: **No cited process-aware system runs unchanged on our cases.** In the last column, (A) and (B) mark Study A and Study B.

		On our cases		
System	Returns	Would need	Why not run	Closest test
Workflow benchmarks				
FlowBench (Xiao et al., 2024)	Tool calls in a dialogue	A workflow as text, code or flowchart, and tools	A benchmark	Graph context (A); every B baseline has the process in its prompt
τ-bench (Yao et al., 2025)	Tool calls and replies	The policy as system prompt, tools and a simulated user	A benchmark	Direct (A), with the policy text in its prompt
Process-aware agents				
StateFlow (Wu et al., 2024)	SQL, shell or household actions	A state machine and prompts built by hand for each task	A state machine for our processes would be our own design	Process context (B), which resolves the process before choosing
Procedural Graphs (Lu et al., 2026)	The next action, with advice from the active node	A tool call to find the active node (a one-step task stays at the Start node); scored runs and a validation split to evolve the graph	One list per case, no tool calls; no official code	Graph context (A): the graph in the prompt without the guidance model
Compile-then-Page (Yu et al., 2026)	Tool calls on SOPBench, scored on the call sequence and final database state	Machine-readable dependency constraints for each user goal	Code not released	Every B baseline has the compiled process in its prompt; the paging runtime is not tested
Evidence planning				
Evidence ledger (Kim, 2026)	An answer to a knowledge-base query	Obligations a model plans for each query, and a retrieval tool	No code or benchmark	Evidence-first (A), Document-first (B): evidence needs first, no code-kept states
Guardrails				
ShieldAgent, GuardAgent (Chen et al., 2025; Xiang et al., 2025)	Allow or deny for another agent's action	Another agent's actions to check	Could only filter a baseline's list	None
Process builders				
PAGED, BREX, Guideline2Graph, Graus (Du et al., 2024; Yang et al., 2026; Kilic et al., 2026; Graus, 2026)	A procedural graph, rule flow or decision model	Source documents. Generation takes no case; Graus runs the generated models on preset test scenarios	They build a process, the step LLMs perform in Study A	Rule-first (B), which derives and runs its own rules
Rule engines and products				
DMN or Catala rule engine (Object Management Group, 2024; Merigoux et al., 2021)	Decisions on typed inputs	Typed facts, while our cases are free text	On the model's judgments it would compute what Compiled control computes	Compiled control (B): CASEPATH's decisions as one flattened formula, matched exactly
ServiceNow document lists (ServiceNow, 2026)	A document list per claim service definition, set when a case is opened	The commercial platform	Closed software; reads only the service definition	None

resolves their branches and requires every condition to resolve before answering. CASEPATH takes obligations from a process shared across cases and can ask the claimant about unresolved conditions. Study B tests inherited branch conditions, which the ledger does not describe.

The ledger's Warrant Judge assesses whether retrieved passages answer a query, a role similar to document-adequacy assessment. Its evidence comes from a knowledge base. It has no claimant document requests or alternative routes, and its trace analysis uses one university knowledge base without a released benchmark.

Yu et al. (2026) compile standard-operating-procedure constraints into pseudo-code and expose the active frame to the LLM. Rule routines return verdicts and evidence, but the LLM supplies each verdict and chooses each tool call; enforcement is "deliberately soft". CASEPATH lets code combine case judgments and derive requests from obligations.

The representation also supports delayed requests, inadequate documents, joint evidence and alternative routes. Study A exposes the cost of requesting every route, but neither study evaluates route selection, timing or joint assessment (Appendix A).

F.3 RULES, PROCESSES AND PROCEDURE EXTRACTION

LegalRuleML records norms and sources (OASIS LegalRuleML Technical Committee, 2021); Catala executes legislative computations (Merigoux et al., 2021). DMN formalises decisions, and CMMN represents case work activated by conditions (Object Management Group, 2024; 2016). Guard-Stage-Milestone models describe decision- and knowledge-intensive processes around business artifacts (Vaculín et al., 2011). DCR models test business-process compliance with law (López et al., 2020). Proactive enforcement derives required actions from a policy and causes them in the target system (Basin et al., 2024).

These systems take structured facts or monitored events, rather than interpreting free-text case facts. Commercial claim software selects documents from a service definition at case opening; that documented step has no holds or clarification questions (ServiceNow, 2026).

PAGED benchmarks procedural-graph extraction (Du et al., 2024); WorFBench evaluates workflow generation (Qiao et al., 2025). BREX benchmarks extraction of business-rule flows, and ExIde grounds them in generated pseudo-code (Yang et al., 2026). Guideline2Graph turns clinical guidelines into executable decision graphs with provenance (Kilic et al., 2026). Graus (2026) tests whether intermediate representations help generate executable legal decision models. CANONIC checks whether content satisfies a corpus grammar; it does not plan for a case (Hadley, 2026). CASEPATH uses an LLM-generated process in Study A and a template-compiled process in Study B, then determines what an individual case needs.

Logic-LM translates a problem into a symbolic formulation for a solver (Pan et al., 2023). Guan et al. (2023) use a model to build a PDDL domain for a sound planner. CASEPATH applies this division to document requests: the process is built once, the model judges each case, and code applies the process. Rule-first instead asks the model to write and follow rules itself.

F.4 PROCESS-AWARE AGENTS AND GUARDRAILS

FlowBench supplies a workflow as text, code or flowchart and scores tool calls and goal completion (Xiao et al., 2024). In τ-bench, policy text is the system prompt; success requires the correct final database state and required information in replies. Its original best agent, gpt-4o, succeeded on 61% of retail and 35% of airline tasks (Yao et al., 2025). StateFlow separates a task state machine from actions within each state (Wu et al., 2024). ReAct interleaves reasoning and action without a process model (Yao et al., 2023).

Procedural Graphs locates the active node from the previous tool call, starting at Start and falling back to the whole graph when no node matches. A guidance model turns the surrounding graph into advice for the next action. A refiner edits the graph by comparing successful and failed trajectories (Lu et al., 2026).

These workflow systems use process context to guide LLM action choice, apart from some fixed StateFlow steps. StateFlow and Procedural Graphs track one active state or node; CASEPATH checks every obligation and, in Study B, inherited branches.

ShieldAgent compiles policy text into probabilistic rule circuits and checks an agent's action trajectory (Chen et al., 2025). GuardAgent generates code that checks proposed actions against safety requirements (Xiang et al., 2025). Both assess actions another agent chose. ShieldAgent also explains violations and suggests countermeasures. Neither plans missing requests or asks about unresolved conditions.

F.5 INFORMATION SEEKING AND GROUNDING

InfoGatherer uses evidential uncertainty to choose retrievals and questions (Taranukhin et al., 2026); Self-RAG learns when to retrieve (Asai et al., 2024). MediQ withholds uncertain diagnoses and asks follow-up questions (Li et al., 2024). Ask-before-Plan predicts when planning needs clarification (Zhang et al., 2024).

GopherCite supplies quoted evidence (Menick et al., 2022), and chain-of-verification checks draft answers with verification questions (Dhuliawala et al., 2024). GraphCompliance aligns policy and context graphs for regulatory assessment (Chung et al., 2025). CLER tests evidence reasoning in Chinese litigation (Bi et al., 2026). API-Bank and BFCL evaluate function calling (Li et al., 2023; Patil et al., 2025).

F.6 VALIDITY OF THE EVALUATION DESIGN

Partial-input baselines, annotation artifacts and shortcut learning motivate our input-free and wrong-condition controls (Kaushik & Lipton, 2018; Poliak et al., 2018; Gururangan et al., 2018; McCoy et al., 2019; Sugawara et al., 2018; Geirhos et al., 2020; Bowman & Dahl, 2021; Yuan et al., 2024). Study A uses hash-bound, LLM-generated references and code scoring. No LLM judges the outputs, removing judge-model preference as a scoring factor (Li et al., 2026).

Evaluation validity requires scoring the intended action (Thomas et al., 2026). Judgment-prediction benchmarks may not predict court decisions (Medvedeva & McBride, 2023) or assume expert-prepared inputs (Lou & Shin, 2026). We score checklist changes and document requests, the actions our method produces.