Scholars Journal of Engineering and Technology

OPEN ACCESS

Abbreviated Key Title: Sch J Eng Tech
ISSN 2347-9523 (Print) | ISSN 2321-435X (Online)
Journal homepage: https://saspublishers.com

PLaND Path to Least Non Determinism

Maddipatla Naga Venkata Sai Krishna[1]*, Asit Kumar Sahoo[1]

[1]Independent Researcher - 1 Saint Francis Pl, San Francisco, CA 94107

DOI: https://doi.org/10.36347/sjet.2026.v14i09.001 | **Received:** 28.07.2026 | **Accepted:** 06.09.2026 | **Published:** 09.09.2026

***Corresponding author:** Maddipatla Naga Venkata Sai Krishna
Independent Researcher - 1 Saint Francis Pl, San Francisco, CA 94107

Abstract | **Original Research Article**

Language-model workflows can spend tokens repeatedly applying decisions that explicit code can perform. We present Path to Least Non Determinism (PLaND), a methodology for revising an English standard operating procedure (SOP) by replacing selected model decisions with code while retaining model fallback. Two reusable skills specify baseline construction and development-guided revision; a fixed comparison contract preserves the model, evaluation conditions and complete English fallback. We evaluate the resulting classification packages on LEDGAR legal clauses, CFPB consumer complaints and SpamAssassin email using Gemini 3.5 Flash Lite. Each dataset contains 500 development, 1,000 selection and 500 reserved final-test cases. Three paired executions reuse the same cases at each reached stage. Acceptance requires an 80% accuracy floor for both the baseline and hybrid, a paired 95% accuracy-difference interval with lower endpoint at least -2 percentage points, at least 5% token reduction, and a strictly positive lower token-reduction endpoint. LEDGAR passed all final-test repeats: mean accuracy changed from 94.93% to 95.80%, with 74.56% mean token reduction. SpamAssassin also passed: mean accuracy changed from 97.67% to 97.20%, with 28.13% mean token reduction. CFPB was rejected at selection because every hybrid repeat missed the accuracy floor; mean accuracy changed from 79.73% to 79.27%. Its final test was not evaluated. These results support selective code execution under the stated tolerances. They do not establish comparative superiority, reliable autonomous rule discovery, or production-workflow performance.

Keywords: Agentic Workflows, Agent Skills, SOP Evolution, Deterministic Execution, Language-Model Agents, Token Efficiency, Hybrid Systems.

1 INTRODUCTION

Business processes often contain both stable and ambiguous decisions. A language model can interpret unfamiliar inputs and handle exceptions, but it may also spend tokens repeatedly applying the same recognizable rule. For example, a contract clause that explicitly specifies its governing law may be easier to classify than a clause that combines several legal functions. The practical question is: Which decisions can be handled by code while keeping the workflow's quality within an acceptable range?

Existing infrastructure provides ways to package and execute these workflows. The Agent Skills specification organizes reusable instructions in a SKILL.md file with optional supporting files [1]. DeepAgents supports loading skills [2], while LangGraph provides graph-based workflow orchestration [3]. These are existing systems on which a methodology can build, not components introduced or independently benchmarked in this paper.

Related research optimizes model programs, workflows and reusable skills. DSPy compiles modular language-model programs against task metrics [4]. AutoFlow optimizes natural-language workflows [5], while AFlow searches code-represented workflows using execution feedback [6]. Automated Design of Agentic Systems uses a meta agent to program and evaluate new agent designs [7]. Workflow revision and code-based agent design are therefore established research directions.

Recent skill work brings the comparison closer to PLaND. SkillOpt proposes bounded textual edits from scored execution records and accepts improvements through a held-out gate [8]. SkillRevise repairs generated skills using execution evidence and repeated verification [9]. SkillReducer reduces context consumed by skill descriptions and bodies [10], whereas ACES measures skill value through paired live evaluations under fixed conditions [11]. PLaND studies the specific intervention of adding executable decisions while preserving the

original English fallback and checking complete-workflow accuracy and token use.

The execution pattern also relates to selective classification, which studies coverage and errors among answered cases [12], and to LLM cascades such as FrugalGPT, which choose when to invoke further models [13]. In the evaluated PLaND hybrids, explicit rules form the first stage and unmatched cases proceed to one fixed model. Acceptance uses the combined outputs across all cases. The experiments compare these hybrids with their English baselines; they do not establish an advantage over learned selective classifiers, cascades or other workflow optimizers.

PLaND begins with an English standard operating procedure (SOP). We call this initial workflow the baseline: the language model follows the English instructions for every case. A hybrid SOP adds an executable step, meaning code that performs a defined operation when its conditions are met. In our experiments, this step is a Python classification script; cases it cannot classify go to the same language model. We call that path fallback. Figure 1 illustrates these two workflows. Further changes may organize stable steps as nodes in a workflow graph. That is a proposed direction. In this paper, reducing non-determinism means reducing dependence on model-mediated execution. It does not mean finding a globally optimal workflow or proving that repeated model outputs become less variable.

The contribution is an SOP-level evolution procedure with a fixed comparison contract, illustrated by three classification studies. The contract preserves the baseline instructions on fallback, records which path actually executes, and requires quality and token gates before accepting a substitution. We evaluate the resulting LEDGAR, CFPB and SpamAssassin packages. These studies test selective code execution and repeated acceptance decisions; they do not benchmark complete business processes or the reliability of independent candidate-discovery runs.

2 MATERIAL AND METHODS

2.1 Skill structure and Executable Steps

A skill is a directory whose main instructions are written in SKILL.md. Optional references/ files provide supporting instructions or domain material, scripts/ contains executable code, and assets/ holds reusable resources such as document templates, images, or lookup tables [1]. A reference file in this directory is supporting material for the agent; it is not a bibliographic citation and does not make a decision deterministic by itself.

An executable step may use Python or Bash and must have defined inputs and outputs. The workflow must actually execute the code and use its result. For example, a Python classification script can return a label directly. Merely mentioning that script in the English instructions does not establish that it ran.

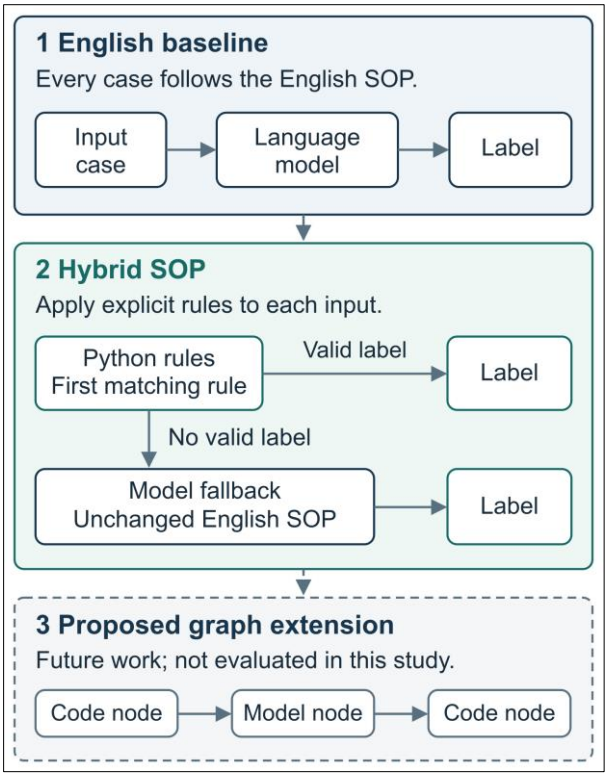

Figure 1: Baseline and hybrid execution. The baseline sends every case to the model. The hybrid first executes a rule and accepts a valid label; otherwise it uses the complete baseline instructions with the same model. A valid match is an output check, not a calibrated confidence estimate. The separately marked graph extension is proposed future work [3]

For LEDGAR, the English SOP asks the model to read a clause, identify its legal function, compare the allowed labels, and choose one. The hybrid SOP adds a rule-based classifier. Inputs that the rules cannot resolve return to the model. Section 3.2 explains how the benchmark executes this choice.

The allowed labels are predefined benchmark categories supplied to both workflows; PLaND does not discover them. The expected label for a particular case is available to the evaluator but is not passed to the model or Python classifier. Appendix B distinguishes this runtime boundary from evaluator-side dataset access.

2.2 How an SOP Evolves

PLaND has two reusable methodology skills [14]. The generate-initial-version skill instructs a host reasoning agent to create an initial agent and English SOP from requirements, approved data sources and an evaluation specification. The pland-evolver skill instructs that host agent to inspect development records and propose revisions. The host agent performs construction; the evaluation runner executes the resulting package. Gemini is the evaluation model in this study, and should not be assumed to be the model that authored the rules.

A candidate is one proposed revision of the whole SOP package: instructions and any supporting code. A bounded revision can add a classifier or restrict its rules; it need not consist of a single regular expression. Development examples guide these changes. Separate selection examples determine acceptance, and a reserved final test evaluates the selected package. The study protocol is:

1. Qualify the English baseline on development cases and save predictions, errors, model calls, tokens and execution records.
2. Inspect development evidence and propose one bounded package change. Use executable code for stable, locally testable operations whose measured benefit justifies substitution; retain English/model fallback for unresolved work. This is a development decision, not an automatic confidence estimate.
3. Evaluate the candidate on development cases. Revise a failed candidate only within the attempt budget. If no suitable change qualifies, stop and retain the baseline.
4. Freeze the first qualifying candidate and compare it with the baseline on selection cases. Every requirement in Section 3.3 must pass in every repeat. Rejection ends the dataset's study; selection outcomes cannot guide another revision.
5. After selection acceptance, evaluate the frozen package on the reserved final test and report the outcome without further tuning.

Development is a readiness check. The baseline had to reach 80% accuracy. A hybrid had to reach the same floor, reduce tokens by at least 5%, produce no execution errors, and preserve the frozen inputs and fallback SOP. Its first assessment and all three development repeats had to pass. The reviewed plan allowed up to ten baseline attempts and ten hybrid attempts to bound search effort; it did not establish ten as an optimal budget. Selection and final testing additionally used the uncertainty requirements in Section 3.3.

For a simple illustration, consider “This Agreement shall be governed by the laws of the State of California.” Governing Laws is an allowed LEDGAR category. A rule matching the phrase could return that label; other clauses would fall back to the model. This example explains routing, not how the study's rule was discovered. Correctly handling one clause does not qualify a candidate: the whole package must meet the development and held-out requirements.

The saved CFPB construction provides a concrete revision example. Its initial hybrid used an ordered set of product-phrase rules. The refinement script retained only the prepaid card, Student loan, Mortgage, and payday/title/personal/advance-loan routes, sending other complaints to the unchanged baseline. The recorded hypothesis was that routes with stronger development precision would preserve quality while avoiding some model calls. That package qualified on development but later failed selection (Section 4.2). Appendix D identifies the construction files. These files preserve explicit code changes and hypotheses, but not the proposing model, a complete interaction history, or the extent of human editing. They therefore support the recorded package progression without establishing autonomous authorship of each rule.

Only the candidate SOP and its directly related files may change within a comparison. The task, evaluation model, system prompt, cases, scorer, runtime settings, permissions and acceptance requirements stay fixed. File-content hashes detect changes, and example identifiers align paired results (Figure 2). Runtime inputs exclude each case's expected answer; the scorer gives one point for literal decoded-label equality and zero otherwise. Accuracy is the number of correct cases divided by the number evaluated.

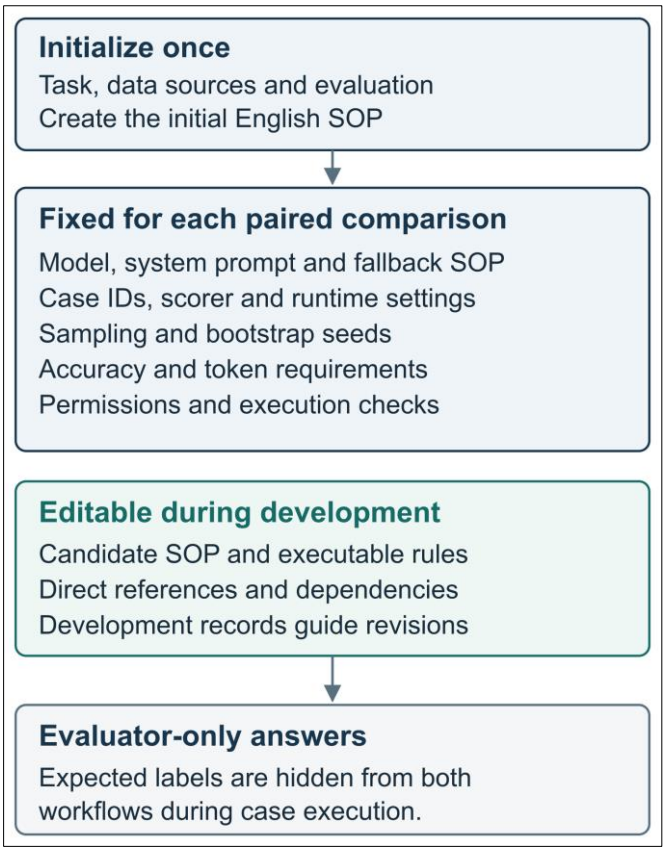

Figure 2: Fixed evaluation and editable candidate package. Development evidence may inform construction. The evaluator retains expected answers and compares predictions; neither runtime arm receives its case's expected label. Sampling and bootstrap seeds are fixed, whereas the hosted model does not support an inference seed. Integrity checking is separate from model evaluation (Appendix B)

2.3 Quality and Expense

The quality requirements specify how accurate a workflow must be before lower model use counts as an improvement. For these experiments, both the baseline and candidate must classify at least 80% of cases correctly, and the candidate may lose at most two percentage points of accuracy after accounting for sampling uncertainty. For example, a change from 90% to 88% is a two-point decrease; it is not a 2% relative decrease. Section 3.3 states the four acceptance requirements, including a minimum 5% token reduction.

Let W denote the baseline workflow and W′ a proposed replacement. Q(W) is classification accuracy, E(W) is total model input and output tokens, Qmin is the minimum acceptable accuracy, and ε is the largest allowed accuracy decrease. Here Qmin = 0.80 and ε = 0.02.

The conceptual objective is lower model use subject to an accuracy floor and an allowed loss:

$$E(W') < E(W)$$
$$Q(W) \geq Q_{min}, Q(W') \geq Q_{min}$$
$$Q(W') - Q(W) \geq -\varepsilon$$

Section 3.3 operationalizes this objective with at least 5% observed token reduction and paired confidence bounds; the inequalities alone are not the acceptance test. These values are study-specific engineering choices recorded in the comparison configuration [14]. They are not thresholds established by the dataset creators or universal standards for these tasks. The records do not provide an application-specific error-cost analysis that would justify 80% or two points for deployment. We therefore interpret acceptance only under these stated tolerances. A real application should choose its accuracy requirements before evaluation, based on the consequences of its errors. Non-inferiority testing provides a framework for assessing an allowed loss [15], it does not supply the numerical margin used here.

3 Experimental Setup

3.1 Datasets and Data Splits

We use three public sources: LEDGAR contract clauses through the LexGLUE task [16, 17], consumer complaint narratives from the Consumer Financial Protection Bureau (CFPB) [18], and the SpamAssassin public email corpus [19]. The tasks are to identify a legal clause type, a complaint's product category, or whether an email is spam. LEDGAR and CFPB each use ten labels; SpamAssassin uses two. Appendix C lists the labels and preparation details.

Table 1: Prepared dataset splits (numbers of examples)

Dataset	Dev.	Selection	Final test
LEDGAR	500	1,000	500
CFPB	500	1,000	500
SpamAssassin	500	1,000	500

In Table 1, Dev. means development. Each dataset has three non-overlapping groups, with the roles defined in Section 2.2: development guides revisions, selection selects the candidate, and the reserved test measures the selected version. The table shows examples prepared, not the number of completed model evaluations. These subsets contain equal numbers per label and do not represent natural label frequencies in production. The dataset manifests and audit records identify the selected examples and check for duplicate identifiers, duplicate content, and overlap with earlier development material [14].

These sizes were fixed before evaluation. The 500 development cases supported candidate creation, the larger 1,000-case selection set was used for the acceptance decision, and the last 500 were reserved for final confirmation. These were study design choices; no prospective power analysis or statistically optimal allocation is claimed.

In the completed evaluations reported below, LEDGAR and SpamAssassin passed selection and proceeded to their 500-case final tests. CFPB failed selection, so its 500-case test was not evaluated. Its reported result therefore covers 1,000 selection examples. The different reported sample sizes follow from this stopping rule; all three datasets had the same split sizes prepared.

3.2 How the Baseline and Hybrid Classify Each Case

The baseline and hybrid process the same examples. As defined in Section 1, the baseline sends every case to the language model with the English SOP and the list of possible labels. It returns one of the supplied categories as a label-only JSON object.

The runner calls the Python classification function directly before any model invocation; it does not ask the model to launch a shell. The hybrid tries this classifier first. It uses the first matching rule to return a category without calling the model. When no rule matches, an output guard fails, or the script raises an exception, the workflow sends the same case to the model. This is fallback. A script exception is also recorded as an execution error and prevents acceptance. For example, a clear governing-law phrase can be handled by code. The email script uses the same principle, with rules for both spam and legitimate email. Rule order matters: the script does not check every possible category conflict before choosing. No numerical confidence score is computed, and a valid label can still be incorrect.

The scorer checks every final label against the expected answer, regardless of which path produced it. A matching label scores one and any other answer scores zero. Model-token use is the sum of the input and output tokens reported by the model across all cases. A case answered entirely by the Python script uses no model tokens. Appendix B describes the code calls, output checks, and recorded fields.

The evaluated hybrids preserve the complete English baseline on the fallback path. The results therefore measure the addition of code without shortening the model instructions. The token records show that almost all savings came from avoiding model calls; Appendix B separates the savings by path.

3.3 Acceptance Criteria and Uncertainty

Figure 3 shows where the acceptance decision belongs in the evolution process described in Section 2.2. The same requirements assess the final test, and a final-test failure would be reported without further tuning.

We fixed four acceptance requirements in the study protocol before the reported evaluations. The same requirements apply to all three datasets [14]:

1. Minimum accuracy: both workflows must classify at least 80% of examples correctly.
2. Accuracy safety check: the lower end of the paired 95% confidence interval for the candidate's accuracy change must be at least -2 percentage points.
3. Token saving: the candidate must use at least 5% fewer total model tokens.
4. Token safety check: the lower end of the paired 95% confidence interval for token reduction must be above zero.

All four must pass using unrounded values, and every evaluated output must be complete and free of execution errors. The 80% floor applies to observed accuracy, not its confidence bound. The token interval must establish a positive saving; its lower endpoint need not exceed 5%.

A confidence interval accounts for the fact that a different sample of examples could give a different result. Because both workflows process the same cases, the calculation resamples their results as matched pairs. The accuracy check asks whether the less favorable end still lies within the two-point allowance. Appendix A reports each run and its confidence intervals. Appendix B explains the 5,000 resamples used in the calculation.

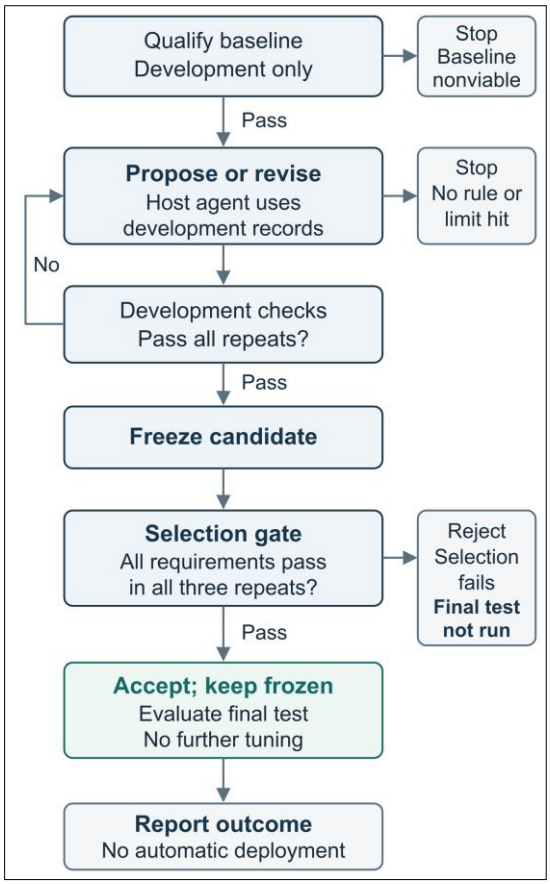

Figure 3: How a candidate is proposed and checked. A candidate is one revised SOP package. Only development examples guide revisions. Separate selection examples decide acceptance; rejection ends the study. After acceptance, the selected candidate is frozen for its reserved test.

3.4 Execution Environment and Repeated Runs

The experiments used Gemini 3.5 Flash Lite through a hosted service. Appendix B records the model settings and implementation details.

Three paired executions ran on every reached split: development, selection, and, where permitted, final test. One paired run means that both workflows process the same evaluation examples under matching model and runtime settings. Repeating an evaluation does not create new cases or new candidate SOPs. “Run 1”, “Run 2” and “Run 3” denote repeated executions, not separately sampled datasets. Both arms are evaluated afresh; fallback outcomes can differ even with identical prompts.

Rate limits were retried, and interrupted runs resumed from saved receipts rather than counting these events as case failures. Two blocked SpamAssassin selection emails were replaced with unique same-label cases under recorded amendments, without reducing the dataset size. Appendix C gives the replacement details.

4 RESULTS

Table 2 presents the main results. All accuracy and token comparisons show the natural-language baseline first and the hybrid second. Headline accuracy is the arithmetic mean of three per-run accuracies, and headline token reduction is the mean of three per-run proportional reductions. Every repeat still had to pass its gate. Final-test rows and the CFPB selection row answer different stage-specific questions and should not be treated as a common test leaderboard. A pass means that the requirements in Section 3.3 were met; it does not mean that accuracy was identical.

Table 2: Main evaluation results

Dataset and split	Baseline to hybrid result
LEDGAR, final-test, 500 cases	Mean accuracy: 94.93% → 95.80%; mean token reduction: 74.56%; Pass
CFPB, selection, 1,000 cases	Mean accuracy: 79.73% → 79.27%; mean token reduction: 23.45%; Reject
SpamAssassin, final-test, 500 cases	Mean accuracy: 97.67% → 97.20%; mean token reduction: 28.13%; Pass

4.1 LEDGAR

On the 500 LEDGAR final-test clauses, mean accuracy changed from 94.93% for the baseline to 95.80% for the hybrid. The quality and token requirements passed in every repeat.

The hybrid handled 370 clauses through the Python classification script and sent the remaining 130 to the model. Model calls therefore fell from 500 to 130 per run, and the mean token reduction was 74.56%.

4.2 CFPB

CFPB's hybrid reduced model calls from 1,000 to 788 per selection run, with a mean token reduction of 23.45%, but mean accuracy changed from 79.73% to 79.27%. Every hybrid repeat missed the 80% minimum. The candidate was rejected and the reserved test was not evaluated. Lower token use did not compensate for failing the accuracy floor.

On the 212 selection cases handled by rules in each repeat, the hybrid returned 198 correct labels; the baseline returned 204, 202, 203, respectively. On the remaining 788 fallback cases, the baseline correct counts were 596, 594, 593, and the hybrid counts were 596, 591, 597. Thus rule errors and differences between fresh fallback executions both affected the total. The baseline also missed the 80% floor in repeats 2 and 3. The failure reflects the absolute quality requirement, not the token or relative-accuracy checks.

The candidate was retained unchanged after rejection. Common thresholds make the acceptance policy consistent across tasks, but do not establish equal development effort or the best achievable accuracy. Further tuning would require a new study with appropriate held-out evaluation.

4.3 SpamAssassin

SpamAssassin's hybrid reduced model calls from 500 to 323 per final-test run. Mean accuracy changed from 97.67% to 97.20%, and the mean token reduction was 28.13%. It passed the quality and token requirements in every repeat. Its small accuracy loss stayed within the allowed tolerance; passing does not mean that accuracy improved.

Rules handled 177 final-test messages per repeat and correctly classified 173; the baseline correctly classified 175 of those same messages in every repeat. On the remaining 323 cases, the baseline correct counts were 314, 312, 314, and the hybrid counts were 313, 312, 314. The small aggregate loss therefore includes a consistent loss on routed cases and a further fallback difference in the first repeat. These path counts describe the observed mechanism; they are not additional acceptance gates.

5 DISCUSSION

5.1 What Selective Execution Demonstrates

The evaluated mechanism is a rule-first classifier with model fallback. Its useful unit of substitution is a subset of inputs: a task can remain semantic overall while recognizable cases admit executable rules. LEDGAR illustrates substantial model-call avoidance under the stated quality requirements; SpamAssassin illustrates an accepted tradeoff with a small accuracy loss. CFPB shows that reduced model use alone is insufficient.

The results support measuring the complete hybrid, including both rule errors and fallback outcomes. A label-validity guard prevents unsupported output categories; it does not certify a prediction's correctness. Development precision therefore informs a proposal but cannot replace selection testing. Preserving the full baseline prompt on fallback makes the observed token reduction attributable primarily to bypassed calls, as checked in Appendix B.3.

PLaND packages this substitution in a development and acceptance procedure. The evidence demonstrates the saved packages under that procedure, rather than superiority over other optimization methods or a guarantee that a host agent will discover the same rules. A comparative study would need to control construction effort, candidate budgets and evaluation data across methods.

5.2 What Token Savings Measure

PLaND's generation instructions ask the agent to consider resource use beyond tokens, including CPU, memory, storage, network access, caching, and setup work. The framework can represent different expense objectives. In the experiments reported here, however, the acceptance decision used model tokens as its only efficiency measure. Model-call counts explain the mechanism; they are not a second independently optimized objective.

Tokens and calls are useful measures of model use, but they do not directly establish dollar, energy, or end-to-end latency savings. Inference time can depend on the service, hardware utilization, and concurrency. Code also has execution and maintenance costs. Claims about those costs require corresponding measurements and are outside the present evaluation.

5.3 From Classification Tasks to Business Workflows

The retained tasks resemble individual decisions within business processes. They do not include long-running state, user interaction, external side effects, or recovery from partial failure. Extending the result to a complete workflow would require evaluating those behaviors as well as label accuracy. The final-test evidence is limited to LEDGAR and SpamAssassin with one hosted model. The email corpus is old, and these balanced subsets do not establish performance on current production data. Replacing blocked emails may also bias selection toward content the provider can process.

The small number of repeats limits conclusions about model variability. No controlled inference seed or stable weight fingerprint was available. The balanced subsets change the meaning of aggregate accuracy relative to natural label frequencies, and the overall gate

does not guarantee acceptable errors for each class or rule. The benchmark cases may also contain related clauses or messages; duplicate checks do not establish statistical independence. No trained small-classifier baseline or independent discovery-reliability experiment was conducted. The incomplete construction interaction record further limits reproducibility of candidate authorship.

6 Future Work

A direct test of the methodology should repeat the full construction and evolution process from fresh starts, recording the proposing model, prompts, tool interactions, human interventions and total development expense. Comparable runs should share a predefined development budget and evaluation policy. Such a study could measure discovery success, variability and effort against small trained classifiers, selective cascades and other workflow optimizers.

Broader validation should use application-specific error costs, per-class requirements and representative data distributions. New evaluation sets should account for document or message dependence, and retain provider-blocked cases in an explicit end-to-end failure analysis. These changes would constitute new experiments rather than retrospective changes to the present gates.

Finally, complete workflows require state, external tools and recovery from partial failure. Monitoring, rollback and graph-based execution could test whether local substitutions remain useful over time as inputs change. These capabilities require their own task-completion and maintenance evaluation; they are not established by the classification results.

7 CONCLUSION

PLaND is a methodology for reducing model-mediated work through evaluated changes to an English SOP. Its central mechanism is selective code execution with model fallback. On LEDGAR, the evaluated hybrid had a mean token reduction of 74.56%, with mean accuracy changing from 94.93% to 95.80%. SpamAssassin had a mean token reduction of 28.13%, with mean accuracy changing from 97.67% to 97.20%. Both passed the stated requirements in all three paired runs. CFPB did not pass selection, and its reserved test was not evaluated. These results show why both quality and token use should be tested before accepting a substitution. The study evaluates the selected packages; it does not establish how reliably independent evolver runs will discover useful candidates or how PLaND performs in complete production workflows.

Acknowledgements and Disclosures

The authors used AI-assisted tools for coding, experiment support, and manuscript preparation. The authors are responsible for the reported results and the final text. No external funding was received, and the authors declare no competing interests.

Data and Code Availability

The PLaND repository provides the two methodology skills, shared classification runner and scorer, saved SOPs, and evaluation records [14]. Appendix B identifies the current evidence and gives the verification commands. These author-produced artifacts establish provenance; they are not independent validation.

Saved run records include case identifiers, expected and predicted labels, correctness, the path used, model calls, tokens, timing, and file fingerprints. Raw datasets are subject to their source terms and are not included in the repository. Exact regeneration requires the recorded input files and model version. In particular, the CFPB inputs came from a saved local API snapshot that is not redistributed. A fresh download from the live database would constitute a different input snapshot. Repository verification checks saved files and code tests; it does not rerun model inference.

Appendix A Individual Held-Out Executions

Repeats 1, 2, and 3 correspond to identifiers 20260903, 20260904, and 20260905. Each pair uses identical cases. Repeats are not additional independent samples and are not pooled to enlarge the case count.

Table A1 shows what happened in each run. The baseline and hybrid values are shown in that order, separated by a slash. In Tables A3, A4 and D1, B denotes the baseline and H the hybrid.

Table A1: Results from each run

Dataset and stage	Run	Accuracy: baseline / hybrid (%)	Model tokens: baseline / hybrid
LEDGAR selection	1	97.0 / 96.7	472,670 / 140,424
LEDGAR selection	2	97.3 / 96.9	472,717 / 140,392
LEDGAR selection	3	97.4 / 96.9	472,800 / 140,484
LEDGAR final test	1	94.8 / 95.8	242,557 / 61,653
LEDGAR final test	2	95.0 / 95.8	242,421 / 61,700
LEDGAR final test	3	95.0 / 95.8	242,372 / 61,707
CFPB selection	1	80.0 / 79.4	700,508 / 536,424
CFPB selection	2	79.6 / 78.9	700,526 / 536,141
CFPB selection	3	79.6 / 79.5	700,470 / 536,111
SpamAssassin selection	1	99.0 / 97.9	2,636,096 / 1,788,601

SpamAssassin selection	2	98.8 / 98.1	2,636,096 / 1,788,658
SpamAssassin selection	3	99.0 / 98.1	2,636,117 / 1,788,728
SpamAssassin final test	1	97.8 / 97.2	1,464,817 / 1,052,802
SpamAssassin final test	2	97.4 / 97.0	1,465,131 / 1,052,891
SpamAssassin final test	3	97.8 / 97.4	1,464,835 / 1,052,948

Table A2 shows the uncertainty check for the same runs. CI means confidence interval. It is not the spread across the three runs.

Table A2: Confidence intervals used for acceptance

Dataset and stage	Run	Accuracy difference 95% CI (points)	Token reduction 95% CI (%)
LEDGAR selection	1	-1.10 to 0.50	67.38 to 73.16
LEDGAR selection	2	-1.20 to 0.40	67.40 to 73.17
LEDGAR selection	3	-1.30 to 0.30	67.39 to 73.15
LEDGAR final test	1	-0.40 to 2.60	70.65 to 78.36
LEDGAR final test	2	-0.60 to 2.20	70.62 to 78.34
LEDGAR final test	3	-0.60 to 2.20	70.61 to 78.33
CFPB selection	1	-1.60 to 0.40	20.56 to 26.33
CFPB selection	2	-1.80 to 0.40	20.60 to 26.38
CFPB selection	3	-1.20 to 1.10	20.60 to 26.38
SpamAssassin selection	1	-1.80 to -0.50	28.01 to 36.63
SpamAssassin selection	2	-1.40 to -0.10	28.01 to 36.63
SpamAssassin selection	3	-1.60 to -0.30	28.00 to 36.63
SpamAssassin final test	1	-1.60 to 0.40	22.13 to 35.18
SpamAssassin final test	2	-1.60 to 0.60	22.14 to 35.18
SpamAssassin final test	3	-1.40 to 0.60	22.13 to 35.17

For accuracy, positive values favor the hybrid and negative values favor the baseline. The lower number had to be at least -2.00 points. For token reduction, both numbers had to be above zero. CFPB passed these interval checks but was rejected because hybrid accuracy was below 80% in all three runs. Calculations use unrounded values.

Example: LEDGAR final-test run 1 has an accuracy interval of -0.40 to 2.60 points. Because -0.40 is above the -2.00-point limit, it passed the accuracy check. Its token-reduction interval, 70.65% to 78.36%, was above zero, so it passed the token check.

Table A3: Selection means and decisions

Dataset	Mean accuracy B → H (%)	Mean token reduction (%)	Decision
LEDGAR	97.23 → 96.83	70.29	Accept
CFPB	79.73 → 79.27	23.45	Reject
SpamAssassin	98.93 → 98.03	32.15	Accept

Table A4: Final-test means and model calls

Dataset	Mean accuracy B → H (%)	Calls B → H	Mean token reduction (%)
LEDGAR	94.93 → 95.80	500 → 130	74.56
SpamAssassin	97.67 → 97.20	500 → 323	28.13

Appendix B Reproduction Details

B.1 Runtime and Scoring

The evaluated model was google/gemini-3.5-flash-lite, accessed through OpenRouter with Google AI Studio as the sole permitted provider endpoint. The setup used DeepAgents 0.7.12, low reasoning effort, a 1,024-token completion limit, non-streaming requests, and a 300-second timeout. Temperature was omitted, leaving the service default. Provider fallback was disabled. The configuration hash identifies request and routing settings, not model weights. Hosted weight bytes and a stable system fingerprint were unavailable.

The runner used eight concurrent case workers, preserving the frozen setting between arms. HTTP 429 responses were recorded and retried, honoring Retry-After or using exponential backoff with case-specific jitter and a maximum delay of 300 seconds. They were not classified as failed examples. Interrupted runs retained completed case receipts and resumed missing work after checking frozen inputs and configuration.

Three paired executions ran on every reached split with the same cases, SOPs, label format, and runtime. Their identifiers are 20260903, 20260904, and 20260905. Although the plan calls them repeat seeds, the

adapter records seed_supported: false and sends no inference seed. They are repeat identifiers, not controlled model-randomness settings. Data sampling and bootstrap sampling used seed 20260902; the token bootstrap used that value plus one. We report each held-out result without assuming deterministic service behavior.

The evaluator loads metadata for all prepared splits, derives the allowed vocabulary from that metadata, and reads input bytes across splits to verify a dataset hash. Thus reserving a final test means withholding its model evaluation and use for candidate tuning; it does not mean the files were never read by preparation or integrity-checking code. The supplied model prompt and classifier arguments contain the current text and allowed labels, not its expected answer. The repository preserves gate decisions but does not provide a complete human/host-agent access history.

The shared runner calls the Python classifier directly; it does not ask the model to launch a shell for each case. Both arms request label-only JSON. The scorer compares the decoded label with the expected label by exact string equality, without changing capitalization, punctuation, or aliases. Expected answers are not supplied to either classifier. Invalid answers score zero; execution errors are recorded separately.

Completion tokens include any reported reasoning tokens, which are not counted twice. Cached input tokens remain part of input-token use. The reported totals cover completed evaluation runs, not construction work, screening, failed attempts, or retries. Resumed-run wall times are not complete execution times and are not headline results.

B.2 Uncertainty Calculation

The comparison code sorts records by case identifier and draws 5,000 paired bootstrap resamples [20]. Each resample selects cases with replacement and retains both workflows' results for each selected case. It then recalculates the accuracy difference and token reduction. The 2.5th and 97.5th percentiles form the reported 95% interval. For acceptance, the lower endpoint must be at least −2 percentage points for accuracy change and strictly above zero for token reduction. It is not a guaranteed worst possible outcome.

These percentile intervals describe case-sampling uncertainty conditional on the saved outputs and prepared data. They do not quantify model drift, construction variability or production performance. The bootstrap treats cases as exchangeable and resamples across the pooled balanced subset rather than preserving class counts or document groups. Related clauses or messages may violate that assumption. No multiplicity adjustment is applied across datasets or runs; the all-repeat gate is an engineering requirement, not a calibrated joint confidence statement. Repeats are not pooled as independent samples.

B.3 Token Savings by Path

Fallback input-token counts matched the baseline on every corresponding final-test case. In LEDGAR final-test repeat 1, the hybrid saved 180,904 tokens: 180,828 from avoiding model calls and 76 from shorter model outputs on fallback cases. In SpamAssassin final-test repeat 1, the corresponding amounts were 412,015, 411,938, and 77 tokens. The measured savings therefore came almost entirely from bypassing model calls. They did not come from shortening the fallback prompt. For one SpamAssassin selection email, provider-reported input counts differed by 1, 3, 2 tokens across the three pairs despite the unchanged prompt-construction contract. The saved receipts do not explain this small discrepancy; it is retained in the reported totals.

B.4 Evidence and Verification

The consolidated collection is in experiments/gemini-3.5-flash-lite/2026-09-05-study/. Original and amended runs are retained in separate subdirectories. These directories contain the frozen plan, recorded amendments, controllers, comparisons, and evidence manifests. Paper-calculations.json lists the exact files and hashes used for every reported result, including the superseded SpamAssassin records. Distinct conditions are retained separately.

The manuscript audit is in the manuscript/ subdirectory of the consolidated collection. Its paper-calculations.json identifies every reported pair, file hash, SOP hash, classifier hash, gate, and recomputed interval. Evidence-files.json inventories saved collection and code files; paper-artifacts.json identifies manuscript outputs. The provenance addendum records superseded metadata issues without changing frozen results.

Run uv sync --project reproduce --frozen, then reproduce/.venv/bin/python reproduce/verify.py. Run reproduce/.venv/bin/python paper/audit_paper.py --artifacts to recompute results and verify manuscript files. These verification commands operate on saved evidence and do not invoke a model or evaluate CFPB's reserved test. The audit derives accuracy and tokens from case outputs, independently rebuilds paired bootstrap intervals, and checks saved comparisons. Command ledgers retain original collection and retry commands.

Appendix C Labels and Sources

LEDGAR uses Governing Laws, Counterparts, Notices, Entire Agreements, Severability, Amendments, Survival, Assignments, Expenses, and Terms. These are corpus categories [16], distributed through LexGLUE [17]. All three study splits were newly sampled from the source training partition after prior-case exclusions.

CFPB uses the product field associated with published narratives in the saved snapshot [18]. The labels are Mortgage; Checking or savings account; Student loan; Money transfer, virtual currency, or money service; Vehicle loan or lease; Prepaid card; Payday loan, title loan, personal loan, or advance loan; Credit card; Debt collection; and Credit reporting or other personal consumer reports. These strings come from the frozen snapshot, not label discovery.

SpamAssassin uses ham from easy-ham and hard-ham archives and spam from spam and spam-2 archives dated 20030228 [19]. In its final selection set, blocked spam messages mail-3125dcdc7726abc1c128 and mail-449598896e25273135c3 were replaced by mail-023db858450b084e7238 and mail-0ae0fc25ca50549e73a8, respectively. The set retained 500 ham and 500 spam cases. Data audits record source hashes, case identifiers, label balance, and duplicate/overlap checks.

Each amendment restarted all three selection pairs on the amended set. Development and final-test inputs, packages, model settings, and thresholds stayed unchanged. Original blocked attempts and superseded outputs were retained. Screening checked whether the provider would process the email, not whether it classified it correctly. The responses were not used to tune rules, and provider safeguards were not weakened.

Appendix D what the two PLaND skills instruct

The generate-initial-version skill specifies the initial agent, one English SOP, approved data access, and task-specific runner and scorer files. It derives the scaffold from requirements and the evaluation structure without copying case answers into the agent. Python or Bash replacements for SOP steps are introduced only during subsequent evolution.

The pland-evolver skill instructs the host agent to measure the current SOP, inspect development records, propose one bounded change, and compare the candidate under fixed conditions. It saves the hypothesis, package changes, measurements, and accept/reject decision. A rejected candidate leaves the previous accepted version in place. Once held-out results are available, the skill instructs the agent to stop evolving and report them. The study protocol makes selection rejection terminal and permits final-test evaluation only after selection acceptance.

The approved study plan allowed up to ten English-baseline attempts and ten hybrid attempts. Exhausting the budget was not acceptance. The baseline first had to reach 80% development accuracy. A hybrid then had to reach that floor, reduce tokens by at least 5%, produce no execution errors, and satisfy the frozen-input and SOP contracts. Its primary assessment and all three development repeats had to pass before selection. These development checks used observed results; the bootstrap requirements applied to selection and final testing.

The CFPB example in Section 2.2 is recorded in cfpb/operations/build_candidate.py, cfpb/operations/refine_candidate_02.py, and the candidate packages' construction.json files under the consolidated collection directory identified in Appendix B.4. The scripts materialize explicit regular-expression rules; running them reproduces those package edits, not the host agent's reasoning that chose them. The saved hypotheses and development assessments document the revision and its eligibility.

Every baseline qualified on its first attempt. LEDGAR and CFPB qualified their second hybrid candidate; SpamAssassin qualified its first. Table D1 reports the development repeats.

Table D1: Development attempts and mean accuracy

Dataset	B attempts	H attempts	Mean accuracy B → H (%)
LEDGAR	1	2	96.53 → 97.20
CFPB	1	2	81.67 → 81.87
SpamAssassin	1	1	98.87 → 98.67

The evolver requires a new executable step to preserve the complete English fallback, with any later instruction shortening evaluated separately. The evaluated hybrids followed that requirement.

REFERENCES

1. Agent Skills. (n.d.). Specification. Retrieved September 6, 2026, from https://agentskills.io/specification
2. LangChain. (n.d.). Skills in Deep Agents. Retrieved September 6, 2026, from https://docs.langchain.com/oss/python/deepagents/skills
3. LangChain. (n.d.). LangGraph overview. Retrieved September 6, 2026, from https://docs.langchain.com/oss/python/langgraph/overview
4. Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., & Potts, C. (2023). DSPy: Compiling declarative language model calls into self-improving pipelines (Version 1) [Preprint]. arXiv. https://arxiv.org/abs/2310.03714v1
5. Li, Z., Xu, S., Mei, K., Hua, W., Rama, B., Raheja, O., Wang, H., Zhu, H., & Zhang, Y. (2024). AutoFlow: Automated workflow generation for

large language model agents (Version 1) [Preprint]. arXiv. https://arxiv.org/abs/2407.12821v1

6. Zhang, J., Xiang, J., Yu, Z., Teng, F., Chen, X., Chen, J., Zhuge, M., Cheng, X., Hong, S., Wang, J., Zheng, B., Liu, B., Luo, Y., & Wu, C. (2025). AFlow: Automating agentic workflow generation (Version 4) [Preprint]. arXiv. https://arxiv.org/abs/2410.10762v4
7. Hu, S., Lu, C., & Clune, J. (2025). Automated design of agentic systems (Version 2) [Preprint]. arXiv. https://arxiv.org/abs/2408.08435v2
8. Yang, Y., Gong, Z., Huang, W., Yang, Q., Zhou, Z., Huang, Z., Li, Y., Gao, X., Dai, Q., Liu, B., Qiu, K., Yang, Y., Chen, D., Yang, X., & Luo, C. (2026). SkillOpt: Executive strategy for self-evolving agent skills (Version 2) [Preprint]. arXiv. https://arxiv.org/abs/2605.23904v2
9. Liu, Y., Su, Z., Xie, L., Zhang, Y., Zong, Q., Guo, J., Xie, Z., Ji, Y., Yim, Y., Luo, H., Ren, X., Chenyu, R., Li, H., & Song, Y. (2026). SkillRevise: Improving LLM-authored agent skills via trace-conditioned skill revision (Version 3) [Preprint]. arXiv. https://arxiv.org/abs/2606.01139v3
10. Gao, Y., Li, Z., Yuan, Y., Ji, Z., Ma, P., & Wang, S. (2026). SkillReducer: Optimizing LLM agent skills for token efficiency (Version 2) [Preprint]. arXiv. https://arxiv.org/abs/2603.29919v2
11. Kevin, C., Raghavan, N., Puget, J.-F., Malani, R., Puvvadi, M., Abramovitch, M., Gupta, M., Akkiraju, R., Prabhu, S., Dangi, Y., Luo, W., & Lee, S. H. (2026). Evaluating skills, not just agents: Agentic continuous evaluation of skills (Version 1) [Preprint]. arXiv. https://arxiv.org/abs/2608.20614v1
12. El-Yaniv, R., & Wiener, Y. (2010). On the foundations of noise-free selective classification. Journal of Machine Learning Research, 11(53), 1605–1641. https://jmlr.org/papers/v11/el-yaniv10a.html
13. Chen, L., Zaharia, M., & Zou, J. (2024). FrugalGPT: How to use large language models while reducing cost and improving performance. Transactions on Machine Learning Research. https://openreview.net/forum?id=cSimKw5p6R
14. Maddipatla, N. V. S. K., & Sahoo, A. K. (2026). PLaND experiment records and implementation (Collection commit 315785fa3d87b1225d8b4a6bced49d6a23e03473) [Software and research data]. GitHub. https://github.com/mnvsk97/PLaND/tree/315785fa3d87b1225d8b4a6bced49d6a23e03473
15. Walker, E., & Nowacki, A. S. (2011). Understanding equivalence and noninferiority testing. Journal of General Internal Medicine, 26(2), 192–196. https://doi.org/10.1007/s11606-010-1513-8
16. Tuggener, D., von Däniken, P., Peetz, T., & Cieliebak, M. (2020). LEDGAR: A large-scale multi-label corpus for text classification of legal provisions in contracts. In Proceedings of the Twelfth Language Resources and Evaluation Conference (pp. 1235–1241). European Language Resources Association. https://aclanthology.org/2020.lrec-1.155/
17. Chalkidis, I., Jana, A., Hartung, D., Bommarito, M., Androutsopoulos, I., Katz, D., & Aletras, N. (2022). LexGLUE: A benchmark dataset for legal language understanding in English. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 4310–4330). Association for Computational Linguistics. https://doi.org/10.18653/v1/2022.acl-long.297
18. Consumer Financial Protection Bureau. (n.d.). Consumer Complaint Database [Data set]. Retrieved September 6, 2026, from https://www.consumerfinance.gov/data-research/consumer-complaints/
19. Apache SpamAssassin. (2003). Public mail corpus [Data set]. https://spamassassin.apache.org/old/publiccorpus/
20. Efron, B. (1979). Bootstrap methods: Another look at the jackknife. The Annals of Statistics, 7(1), 1–26. https://doi.org/10.1214/aos/1176344552