

LLM-Modulo Planning Agents with External Plan Verification and Iterative Self-Repair

Tyler Ward¹, Yan Zhou², Stephanie Gray³

¹Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA, USA

²Department of Computer Science, Purdue University, West Lafayette, IN, USA

³Department of Computer Science, New York University, New York, NY, USA

tward88@outlook.com

Abstract

Large language models can translate natural-language goals into action sequences, but an action sequence is not a plan unless every precondition holds and the final state satisfies the goal. This study evaluates a modulo-style planning agent that assigns proposal generation to an LLM and delegates correctness to an external symbolic verifier. The evaluation used the PlanBench repository artifacts for Blocksworld: every stored GPT-4 candidate in four domain-size conditions and five prompt forms (6,000 direct plans), together with all 50 verifier-feedback trajectories containing 286 sequential candidates. Each candidate was parsed into PDDL actions and independently checked with VAL against the corresponding domain and problem file. Across the direct corpus, 934 of 6,000 candidates were valid (15.57%). Validity reached 27.53% in standard Blocksworld but only 3.60% after semantic obfuscation. State-tracking prompts were strongest in both conditions, while PDDL-form prompts did not compensate for weak state-transition reasoning. On the matched 50-instance repair set, direct execution completed 4 tasks (8%) and would have executed 46 invalid plans. A verifier gate preserved the same 8% completion while reducing invalid execution to zero through abstention. Iterative verifier-feedback repair completed 41 tasks (82%), repaired 37 of 46 initially invalid plans (80.43%), accepted no invalid plan, and required 5.72 candidate generations on average. Completion saturated after the ninth candidate; the paired gain was 74 percentage points with a 62-86 point bootstrap interval and exact McNemar $p = 1.46 \times 10^{-11}$. The results establish a clear division of labor: formal verification supplies sound acceptance, while bounded feedback-driven regeneration converts safe rejection into high task completion.

Keywords: Automated Planning, Large Language Models, Planbench, External Verification, VAL, Iterative Self-Repair, Agent Safety, Test-Time Scaling.

1. Introduction

1.1 Planning Correctness and Executable Semantics

Automated planning asks an agent to transform an initial state into a goal state through a sequence of actions whose preconditions and effects are defined by a transition model. In classical planning, the output is judged by execution semantics rather than linguistic plausibility. A plan is correct only when every action is applicable in the state reached by its predecessors and the terminal state entails the goal. STRIPS introduced the action-centric representation that underlies this view [1], PDDL standardized domain and problem descriptions [2], and subsequent planning research formalized the relationship among search, reachability, soundness, and plan quality [3]. Plan validation therefore serves a distinct role from plan generation. VAL checks a proposed sequence against the formal transition system and produces a deterministic judgment with diagnostic information [4], while systems such as Fast Downward demonstrate how explicit models support complete search and reliable plan construction [5].

1.2 LLM Planning and Agent Scaffolds

Large language models change the interface to planning. An LLM accepts instructions, demonstrations, and contextual descriptions without a task-specific parser written for every user utterance. This flexibility has supported zero-shot action planning [6], affordance-grounded robotic control [7], feedback-aware embodied reasoning [8], programmatic task plans [9], policy-code generation [10], and hybrid architectures that translate language into formal planning problems [11]. Reasoning prompts and agent scaffolds further expand the proposal process. Chain-of-thought prompting exposes intermediate reasoning [12], self-consistency samples multiple derivations [13], ReAct interleaves reasoning with actions [14], Self-Refine uses feedback to produce successive answers [15], Reflexion stores task feedback in a verbal memory [16], Tree of Thoughts searches over intermediate states [17], program-aided language models call an interpreter [18], reasoning-via-planning treats language inference as search [19], and Toolformer trains models to invoke specialized tools [20]. Together, these methods show that generation, search, execution, memory, and tool use can be composed around an LLM. Related modular systems outside planning reinforce

the same design pattern: evidence-grounded financial question answering couples retrieval with source-backed responses [21], code intelligence combines retrieval with summarization [22], and autonomous-driving perception pipelines fuse complementary signals before downstream decisions [23]. These systems do not establish plan validity, but they illustrate why specialized components can be more dependable than a monolithic generator.

1.3 The Verification Gap

The same flexibility also creates a correctness gap. A fluent action sequence can contain an inapplicable move, omit a required subgoal, use a malformed symbol, or terminate in a non-goal state. PlanBench was designed to measure these capabilities under controlled planning semantics and showed that strong language models remain unreliable on plan generation and related reasoning tasks [24]. A critical investigation of LLM planning reached the same conclusion across direct and obfuscated settings: surface competence does not imply reliable state tracking [25]. In safety-relevant agents, this distinction is operational. Executing a plausible but invalid sequence is a failure even when most steps are locally reasonable. The system therefore needs a component whose acceptance decision depends on the world model rather than on verbal confidence. Comparable reliability patterns appear in operational machine learning. Ambiguity-aware log analysis uses selective refusal rather than forcing a diagnosis [26], calibrated vision models explicitly represent human uncertainty [27], and drift monitoring links early warnings to retraining or rollback decisions [28]. Across these settings, uncertainty or anomaly evidence is useful only when it changes the system's action policy.

Predictive accuracy does not by itself define a safe decision rule. DenseNet-based cancer image classification demonstrates the value of task-specific model evaluation [29], while AI-based MRD assessment places prediction in a clinically consequential workflow [30]. Self-correction alone does not guarantee that component. Experiments on self-critiquing plans found that an LLM verifier produces false positives and that an external sound verifier is substantially more reliable [31]. Iterative prompting analyses likewise showed that a model often preserves an incorrect answer because it cannot reliably identify the underlying error [32]. By contrast, verification-guided code generation demonstrates that execution evidence can improve selection when the verifier is coupled to the semantics of the generated artifact [33]. These findings motivate a modulo architecture: the LLM proposes candidates, a deterministic parser creates an executable representation, a formal verifier certifies or rejects the candidate, and rejection feedback drives another proposal. The verifier remains authoritative throughout the loop.

1.4 Scope and Contributions

This study isolates the value of that architecture through three matched policies. The direct policy executes the first LLM

candidate. The verifier-gate policy checks the same candidate and abstains when it is invalid. The verifier-feedback policy repeatedly checks, diagnoses, and repairs until a valid plan is certified or the candidate budget is exhausted. This separation answers questions that a single accuracy number cannot answer. A verifier gate can improve safety without improving completion. A repair loop can improve completion while preserving acceptance precision. Candidate budgets expose the compute required for those gains. Error categories reveal whether feedback addresses state-transition failures or only formatting failures.

The empirical analysis has four contributions. First, it independently revalidates every executable candidate in the selected PlanBench artifacts rather than using stored success flags. The direct analysis covers 6,000 GPT-4 plans across standard and semantically obfuscated Blocksworld with three- and four-object instances and five prompt forms. Second, it provides a matched comparison of direct execution, verifier gating, and verifier-feedback repair on all 50 available repair trajectories. Third, it quantifies safety, abstention, repair rate, candidate cost, convergence, plan length, edit distance, and difficulty-conditioned performance in addition to valid-plan rate. Fourth, it packages the complete validation pipeline, numerical tables, candidate-level records, and vector figures so that every reported result follows from one command over the archived benchmark.

The research questions are direct. RQ1 asks how prompt form, instance size, and semantic obfuscation affect externally certified direct planning. RQ2 asks what a verifier changes when it is used only as an execution gate. RQ3 asks how much iterative verifier feedback recovers from initially invalid plans. RQ4 asks how quickly repair gains saturate and how those gains vary with failure type and optimal plan length. RQ5 asks whether the matched initial-to-final improvement remains statistically decisive under paired analysis.

2. Method

2.1 Experimental Design and Corpus

The study used the PlanBench repository snapshot and its original PDDL task files, stored GPT-4 responses, backprompting trajectories, and compiled VAL executable. PlanBench contains a broader collection of prompt-based planning and reasoning tasks [24]. The present experiment selected every record for which the repository contained both an executable GPT-4 Blocksworld candidate and the exact PDDL instance required for independent validation. This criterion produced 6,000 direct candidates. The repair experiment used every trajectory in the standard four-object Blocksworld backprompting file: 50 tasks and 286 candidate plans across as many as 15 rounds. No task or round was sampled from these two executable corpora. Table I summarizes the units of analysis.

Table I
Evaluated Planbench Artifacts And Units Of Analysis

Analysis	Inputs	Observations	Coverage	Unit
Direct plan validation	4 prompt conditions x 5 prompt forms	6,000	Four domain-size groups	Every stored GPT-4 candidate
Iterative verifier feedback	One-shot GPT-4 VAL feedback	50 traces / 286 candidates	Standard four-block subset	Up to 15 candidates per trace
Formal task files	PDDL domains and problem instances	1,200 problems	Standard and mystery	Original benchmark instances

The direct corpus crossed four domain-size conditions with five prompt forms. Standard Blocksworld used the original predicate and action vocabulary; Mystery Blocksworld replaced meaningful names with opaque symbols while preserving transition structure. Three-object conditions contained 100 problems per prompt form, and four-object conditions contained 500 problems per prompt form. The prompt forms were one-shot natural language, zero-shot natural language, natural language with explicit state tracking, one-shot PDDL, and zero-shot PDDL. Each form therefore contributed 600 standard and 600 mystery candidates, and every domain-size-prompt cell was complete. The design separates combinatorial scale from semantic familiarity: increasing from three to four objects raises the state-tracking burden, while the mystery transformation removes lexical cues without changing the formal validity criterion.

The matched repair set contains 50 standard four-object problems. Each trajectory begins with an LLM plan, records VAL feedback after an invalid candidate, and continues with another LLM response until validation succeeds or the 15-candidate limit is reached. The trajectories contain 286 candidates in total. Because these records were created for backprompting analysis, they form a failure-enriched subset rather than a random sample of all Blocksworld instances. All three agent policies were evaluated on the same 50 initial candidates. Consequently, the direct-versus-gate comparison isolates the decision rule, and the gate-versus-repair comparison isolates the feedback loop. Table II states the three policies.

Table Ii Matched Agent Policies

Policy	Verifier	Feedback	Decision rule	Candidates
Direct LLM	No	No	First candidate executed	1
Verifier gate	Yes	No	Execute iff valid	1
Verifier-feedback repair	Yes	Yes	Execute first valid candidate	1-15

2.2 Parsing, Verification, and Repair Control

Figure 1 shows the implemented control flow. The LLM output first passes through a deterministic parser. For direct-plan records, the parser removes list markers, extracts parenthesized actions, lowercases symbols, normalizes

whitespace, and maps color words to the object identifiers used by the PDDL instance. For repair records, the parser follows the repository's Blocksworld extraction rules: it isolates the text between PLAN markers when present, canonicalizes pick-up and put-down, identifies stack and unstack actions, requires the correct number of full color-object arguments, orders arguments by their textual position, and emits one PDDL action per recognized line. Empty extractions remain empty candidates and are scored as invalid rather than silently removed.

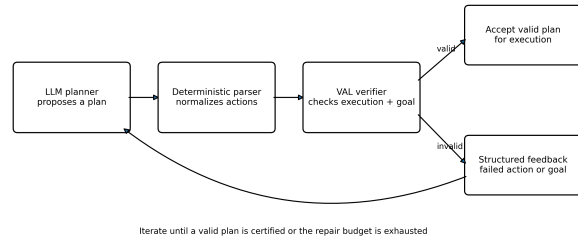


Fig. 1. Modulo planning architecture. The verifier is the sole acceptance authority, and invalid candidates return structured diagnostics to the generator.

Every parsed candidate was written to a temporary plan file and evaluated with VAL using the corresponding domain and problem file. A candidate was valid only when VAL returned a line containing exactly 'Plan valid.' The evaluator did not infer correctness from a stored Boolean field. Invalid candidates were classified as empty, syntax, unsatisfied precondition, unsatisfied goal, or other validator failure. Precondition failures indicate that at least one action was not applicable in the state created by the previous prefix. Goal failures indicate that all parsed actions executed but the terminal state did not satisfy the goal. These categories directly connect generated text to formal execution semantics.

The direct policy models unverified agent execution. It consumes one LLM call and executes the first candidate regardless of validity. The verifier-gate policy consumes the same LLM candidate, invokes VAL once, executes only a certified plan, and otherwise abstains. The verifier-feedback policy invokes VAL after each candidate, terminates immediately on a valid result, and otherwise passes structured failure information back to the next stored repair round. The maximum budget was 15 candidates because that is the complete trajectory depth present in the artifact. The accepted-plan set for both verifier policies therefore contains only candidates certified by the same sound decision rule. The fixed cap is a control constraint rather than an accuracy heuristic; analogous budget-constrained decision systems optimize utility while enforcing resource and downside-risk limits [34].

2.3 Outcomes, Cost, and Statistical Analysis

The primary outcome was valid completion, defined as the fraction of instances for which the policy returned a VAL-certified plan. Accepted-plan precision was the fraction of

executed plans that were valid. Unsafe execution counted instances on which a policy executed an invalid plan. Abstention counted instances with no executed plan after the applicable budget. Repair success was computed among initially invalid tasks as the number that became valid by the final available round divided by the number initially invalid. The analysis also measured cumulative completion at each budget, average candidates consumed under that budget, one-step correction after each feedback category, and the repair saturation round, defined as the first budget after which cumulative completion never increased.

Plan structure was measured with parsed action counts. Initial, final, and optimal lengths were summarized with mean, median, minimum, and maximum. Sequence-level change was measured by Levenshtein distance over action tokens, so a substitution, insertion, or deletion contributed one unit. For valid final plans, excess length was final length minus the repository's optimal plan length. Optimal lengths also defined four difficulty bins: 2-4, 5-6, 7-8, and 9 or more actions. These bins preserve enough tasks for descriptive comparison while distinguishing short from long-horizon instances.

Uncertainty and hypothesis tests followed the paired design. Wilson score intervals summarize binomial rates because the matched sample is small and several rates are near zero or one [35]. The initial-to-final risk difference was bootstrapped by resampling the 50 paired task outcomes 20,000 times with seed 2023. Exact McNemar testing used the discordant pairs: 37 tasks changed from invalid to valid and none changed from valid to invalid [36]. The standard-versus-mystery direct comparison used the 2 x 2 validity table, from which the odds ratio and chi-square statistic were computed. Statistical claims use $p < 0.001$ when the numerical value is smaller than the displayed precision.

2.4 Reproducible Execution

The full experiment was executed locally with Python, pandas, NumPy, SciPy, statsmodels, matplotlib, and the repository's VAL binary. Validation jobs ran in 12 worker threads, each with a separate plan file, and the complete analysis finished in seconds on the available CPU environment. The archive SHA-256 digest was 7d05c4a1220bced052c8e8fac7708763dd0672bf72145448456feda4c6e5b6af. The run produced candidate-level CSV files, 11 publication tables, a machine-readable JSON summary, and seven figures in PNG and SVG formats. The fixed seed, exact file selection, parser rules, validator command, and output manifest are encoded in the accompanying script.

3. Results and Discussion

3.1 Direct Plan Validity Across Prompt and Domain Conditions

Across all 6,000 direct candidates, VAL certified 934 plans, giving an overall valid-plan rate of 15.57%. The aggregate

conceals a large semantic gap. Standard Blocksworld produced 826 valid plans from 3,000 candidates (27.53%), whereas Mystery Blocksworld produced 108 from 3,000 (3.60%). The odds of validity were 10.174 times higher in the standard condition, and the 2 x 2 chi-square statistic was 651.9 ($p < 0.001$). Because the two conditions preserve planning structure while changing names, this difference shows that the stored GPT-4 planner relied heavily on familiar lexical semantics. It did not maintain equivalent transition reasoning after the vocabulary ceased to advertise action meaning.

Table III reports every domain-size-prompt cell. The best single cell was zero-shot natural language on standard three-object tasks at 59%, followed by state-tracking natural language at 55% and one-shot natural language at 49%. Performance declined in the four-object standard condition: the corresponding rates were 30.2%, 34.8%, and 31.4%. The additional object therefore reduced accuracy even though average plans grew by only about two actions. In Mystery Blocksworld, state tracking remained strongest, reaching 21% for three objects and 8.2% for four objects. Zero-shot natural language failed on all 600 mystery tasks.

Table Iii Direct Gpt-4 Validity By Domain, Size, And Prompt

Domain	World	Obj.	Prompt	N	Valid	Valid %	Mean actions
BW-3	Standard	3	One-shot NL	100	49	49.0	5.09
BW-3	Standard	3	One-shot PDDL	100	28	28.0	4.30
BW-3	Standard	3	State-tracking NL	100	55	55.0	5.15
BW-3	Standard	3	Zero-shot NL	100	59	59.0	5.17
BW-3	Standard	3	Zero-shot PDDL	100	41	41.0	5.23
BW-4	Standard	4	One-shot NL	500	157	31.4	7.86
BW-4	Standard	4	One-shot PDDL	500	47	9.4	6.64
BW-4	Standard	4	State-tracking NL	500	174	34.8	6.11
BW-4	Standard	4	Zero-shot NL	500	151	30.2	7.11
BW-4	Standard	4	Zero-shot PDDL	500	65	13.0	7.42
Mystery BW-3	Mystery	3	One-shot NL	100	8	8.0	5.07
Mystery BW-3	Mystery	3	One-shot PDDL	100	8	8.0	4.95
Mystery BW-3	Mystery	3	State-tracking NL	100	21	21.0	4.54

Domain	World	Obj.	Prompt	N	Valid	Valid %	Mean actions
Mystery BW-3	Mystery	3	Zero-shot NL	100	0	0.0	0.91
Mystery BW-3	Mystery	3	Zero-shot PDDL	100	1	1.0	5.07
Mystery BW-4	Mystery	4	One-shot NL	500	18	3.6	7.51
Mystery BW-4	Mystery	4	One-shot PDDL	500	9	1.8	7.69
Mystery BW-4	Mystery	4	State-tracking NL	500	41	8.2	6.48
Mystery BW-4	Mystery	4	Zero-shot NL	500	0	0.0	2.42
Mystery BW-4	Mystery	4	Zero-shot PDDL	500	2	0.4	7.15

The prompt-level aggregation in Table IV and Fig. 2 establishes two consistent patterns. First, explicit state tracking was the strongest prompt form in both semantic conditions: 38.17% in standard tasks and 10.33% in mystery tasks. It exceeded one-shot natural language by 3.83 points in standard tasks and 6.00 points in mystery tasks. Second, presenting the problem in PDDL did not make the LLM behave like a symbolic planner. One-shot PDDL achieved 12.50% in standard tasks and 2.83% in mystery tasks; zero-shot PDDL achieved 17.67% and 0.50%. The formal surface representation supplied symbols, but it did not supply reliable state progression.

Table IV Prompt-Level Validity Aggregated Across Instance Sizes

World	Prompt	N	Valid	Valid %
Standard	One-shot NL	600	206	34.33
Standard	One-shot PDDL	600	75	12.50
Standard	State-tracking NL	600	229	38.17
Standard	Zero-shot NL	600	210	35.00
Standard	Zero-shot PDDL	600	106	17.67
Mystery	One-shot NL	600	26	4.33
Mystery	One-shot PDDL	600	17	2.83
Mystery	State-tracking NL	600	62	10.33
Mystery	Zero-shot NL	600	0	0.00
Mystery	Zero-shot PDDL	600	3	0.50

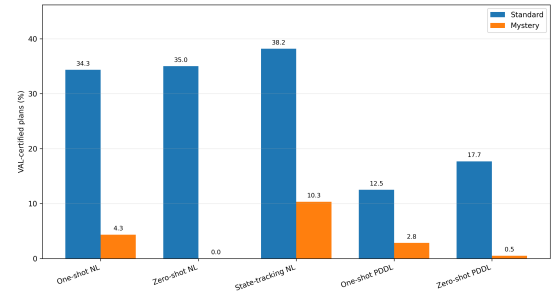


Fig. 2. VAL-certified direct plans by prompt form and semantic condition. Each bar aggregates three- and four-object instances.

The natural-language results also separate demonstration effects from reasoning effects. In standard tasks, zero-shot natural language slightly exceeded one-shot natural language, 35.00% versus 34.33%, so the example did not improve aggregate validity. In mystery tasks, one-shot natural language reached 4.33% while zero-shot natural language remained at 0%, showing that a demonstration helped decode the obfuscated operator vocabulary. State tracking still more than doubled the one-shot mystery rate. The strongest prompt therefore combined semantic decoding with explicit intermediate-state bookkeeping rather than relying on a single example alone. Language choice is also an interface variable in computing education: bilingual data-structures instruction has been evaluated for its effects on learning and engagement [37]. Here, the corresponding question is whether a model can ground opaque symbols into executable actions.

Failure diagnostics explain why direct plans were unreliable. Table V and Fig. 3 classify all invalid candidates relative to the full generated set. Unsatisfied preconditions dominated every condition: 224 cases in standard BW-3, 1,604 in standard BW-4, 345 in mystery BW-3, and 1,987 in mystery BW-4. Goal failures were the second major category in standard tasks, with 42 and 278 cases. Mystery tasks added a large empty-plan component, especially under zero-shot natural-language prompting: 86 empty extractions in BW-3 and 361 in BW-4. Only one direct candidate produced a syntax-classified failure, and nine standard BW-4 candidates fell into the residual category. The main problem was therefore semantic execution, not PDDL punctuation.

Table V Direct Candidate Outcome Taxonomy

Domain	Total	Valid	Precond.	Goal	Empty	Syntax	Other
BW-3	500	232	224	42	2	0	0
BW-4	2,500	594	1,604	278	14	1	9
Mystery BW-3	500	38	345	31	86	0	0
Mystery BW-4	2,500	70	1,987	82	361	0	0

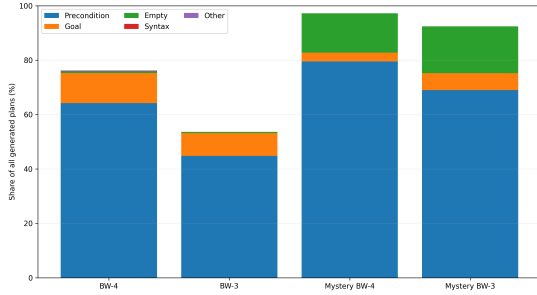


Fig. 3. Invalid direct-plan outcomes as a share of all generated candidates. Precondition violations dominate every domain condition.

Precondition failures are particularly important for agent safety. A plan with a late precondition error can appear coherent through a long prefix, yet execution stops at an impossible action. A language-only evaluator can reward the plausible prefix, while VAL rejects the complete sequence at the first violated transition. The dominance of this category supports external checking after every proposal. It also supports diagnostic feedback that identifies the failed action rather than a generic message that the answer is wrong. Goal failures require a different repair behavior: the action sequence is executable, but the planner must add or alter actions so the final state satisfies the target. Operational AIOps work similarly separates fault signals from diagnosis and remediation decisions [38], supporting the use of validator feedback as structured evidence rather than a generic confidence cue.

3.2 Matched Comparison: Direct Execution, Gating, and Repair

The matched 50-instance experiment separates safety from completion. As shown in Table VI and Fig. 4, the direct policy completed 4 tasks (8%) and executed an invalid plan on the remaining 46 tasks. Its accepted-plan precision was therefore 8%. The verifier gate completed the same 4 tasks because it received the same first candidates, but its accepted-plan precision rose to 100%. It executed no invalid plan and abstained on 46 tasks. Verification alone did not create a solution; it converted unsafe execution into safe refusal.

Table Vi Matched Policy Outcomes On 50 Repair Tasks

Policy	Valid	Comp. %	Accept prec. %	Unsafe	Unsafe %	Abstain	Abstain %	LLM calls	VAL calls
Direct LLM	4	8.0	8.0	46	92.0	0	0.0	1.00	0.00
Verifier gate	4	8.0	100.0	0	0.0	46	92.0	1.00	1.00
Verifier-feedback repair	41	82.0	100.0	0	0.0	9	18.0	5.72	5.72

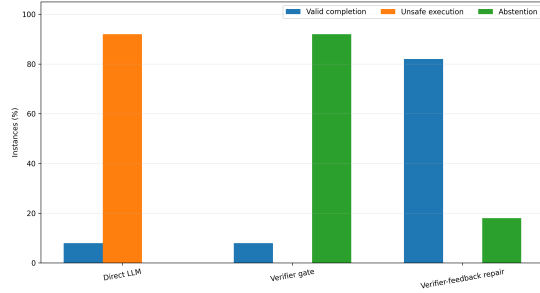


Fig. 4. Matched safety-completion tradeoff. Verification alone converts invalid execution into abstention; verifier-feedback repair converts most abstentions into certified completion.

Iterative verifier feedback changed that tradeoff. The repair policy completed 41 of 50 tasks (82%), executed no invalid plan, and abstained on 9 tasks after exhausting the available rounds. Among the 46 initially invalid tasks, 37 became valid, giving an 80.43% repair rate. All four initially valid plans remained valid because the policy stopped immediately after certification. Relative to verifier gating, repair reduced abstention from 92% to 18% while preserving 100% accepted-plan precision. Relative to direct execution, it raised valid completion by 74 percentage points and eliminated 46 unsafe executions.

The broad direct rate and the matched initial rate answer different questions. The 27.53% standard direct rate pools 3,000 candidates across two instance sizes and five prompts. The 8% matched rate comes from 50 standard four-object trajectories selected for the repository's backprompting experiment. The repair set is therefore harder and failure enriched. The valid comparison is within its paired rows: each policy sees the same initial candidate, and repair adds only verifier feedback and subsequent candidates. No claim in this study treats the matched subset as a random estimate of all PlanBench tasks.

3.3 Repair Convergence and Diagnostic Effectiveness

Repair gains arrived early but not immediately. Table VII and Fig. 5 show that a one-candidate budget reproduced the 8% initial rate. Allowing a second candidate raised cumulative completion to 36%, a gain of 28 points at an average cost of 1.92 candidates. The third candidate raised completion to 48%; the fourth to 58%; and the sixth to 70%. By the seventh candidate, 38 tasks were complete (76%). Candidate eight added one success and candidate nine added two, reaching the final 82%. Candidates 10 through 15 added no further success. The saturation round was therefore nine.

Table Vii Cumulative Completion Under Candidate Budgets

Maximum candidates	Solved	Completion %	Mean consumed
1	4	8.0	1.00
2	18	36.0	1.92
3	24	48.0	2.56

Maximum candidates	Solved	Completion %	Mean consumed
4	29	58.0	3.08
5	31	62.0	3.50
6	35	70.0	3.88
7	38	76.0	4.18
8	39	78.0	4.42
9	41	82.0	4.64
10	41	82.0	4.82
11	41	82.0	5.00
12	41	82.0	5.18
13	41	82.0	5.36
14	41	82.0	5.54
15	41	82.0	5.72

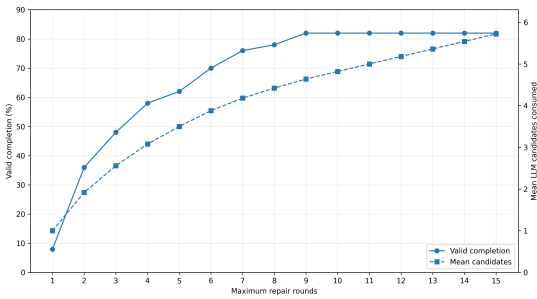


Fig. 5. Cumulative valid completion and mean candidate consumption as the repair budget increases. Completion saturates at candidate nine.

The budget curve supports a bounded deployment policy. A four-candidate cap obtained 58% completion with 3.08 average candidates consumed, while a seven-candidate cap obtained 76% with 4.18 candidates. Extending from seven to nine candidates added six completion points at an average additional consumption of 0.46 candidates per task. Extending beyond nine consumed additional candidates only on the nine unresolved traces and produced no benefit. In this corpus, the efficient frontier ends at nine rounds. A production controller would stop there unless a different model, prompt, or planner changed the observed tail.

Figure 7 displays the validator outcome at each repair round. The first round contained 4 valid plans, 41 precondition failures, 4 goal failures, and 1 empty candidate. Round two produced the largest single conversion, with 14 valid candidates among the 46 remaining traces. Later rounds contained progressively fewer active traces because successful tasks terminated. Precondition failures remained the dominant residual outcome through the final round. The total repair corpus contained 197 precondition failures, 42 goal failures, 41 valid candidates, and 6 empty candidates across 286 evaluations. No repair candidate was classified as syntax or other failure after parsing.

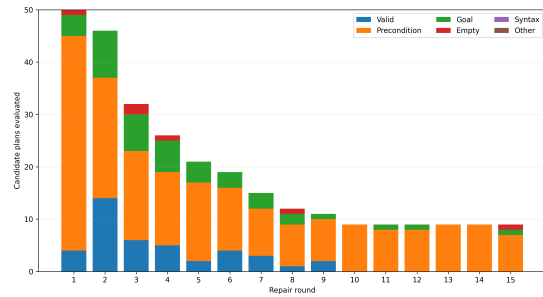


Fig. 7. Validator outcomes by repair round. Active trajectories decrease after success; precondition failures dominate the unresolved tail.

Table VIII measures whether a particular diagnostic was followed by a valid candidate on the next recorded transition. Goal feedback was followed by a valid plan in 13 of 46 transitions (28.26%). Precondition feedback was followed by a valid plan in 24 of 190 transitions (12.63%). The higher one-step rate for goal feedback is consistent with the smaller search needed after an executable but incomplete plan: the model often needed to add or redirect a suffix. Precondition feedback demanded restoration of state consistency, and a local edit could create another upstream or downstream violation. The feedback still repaired many precondition failures over multiple iterations, but the correction path was longer.

Table VIII One-Step Correction After Verifier Feedback

Feedback	Transitions	Next valid	Correction %
Goal	46	13	28.26
Precondition	190	24	12.63

3.4 Difficulty, Plan Quality, and Compute

Difficulty-conditioned results are reported in Table IX and Fig. 6. Initial validity was concentrated entirely in the shortest bin: 4 of 11 tasks with optimal length 2-4 were valid, and no longer task was initially valid. After repair, completion reached 90.91% in both the 2-4 and 5-6 bins, 69.23% in the 7-8 bin, and 80.00% in the 9+ bin. The longest bin exceeded the 7-8 bin despite containing longer optimal plans, so the 50-task sample does not support a monotone length-performance curve. It does establish that repair was not confined to short plans: 12 of 15 tasks requiring at least nine optimal actions were solved.

Table IX Completion By Optimal Plan Length

Optimal actions	N	Initial valid	Initial %	Final valid	Final %	Gain (pp)
2-4	11	4	36.36	10	90.91	54.55
5-6	11	0	0.00	10	90.91	90.91
7-8	13	0	0.00	9	69.23	69.23
9+	15	0	0.00	12	80.00	80.00

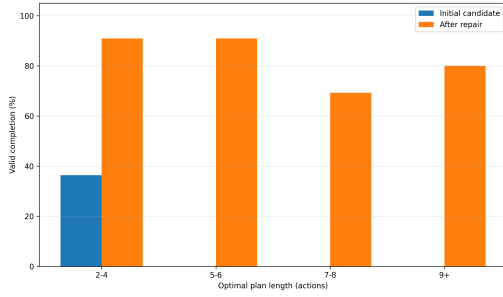


Fig. 6. Initial and repaired completion by optimal plan length. Repair succeeds across all horizon bins.

Table X quantifies candidate length and editing. Initial candidates averaged 6.98 parsed actions, final candidates averaged 8.42, and optimal plans averaged 7.12. The median was eight for all three quantities. The mean initial-to-final Levenshtein distance was 4.38 actions, with a maximum of 11. Repair therefore involved substantive sequence changes rather than repeated formatting. Among the 41 valid final plans, excess length averaged 1.41 actions, the median was zero, and the maximum was six. Many repairs reached an optimal-length solution, while others traded compactness for correctness.

Table X Candidate Cost, Length, And Sequence Change

Quantity	Mean	Median	Min	Max
Initial candidate plan length	6.98	8.00	0.00	12.00
Final candidate plan length	8.42	8.00	0.00	16.00
Optimal plan length	7.12	8.00	2.00	12.00
Candidates per trace	5.72	4.00	1.00	15.00
Initial-final edit distance	4.38	4.00	0.00	11.00
Excess actions in valid final plans	1.41	0.00	0.00	6.00

The candidate-cost distribution was skewed. Mean consumption was 5.72 candidates, but the median was four; some tasks solved immediately, whereas nine exhausted all 15 rounds. This asymmetry explains why a fixed maximum budget and an average cost must be reported together. A policy that allocates one call to every task is cheap but unsafe without a gate and incomplete with one. A policy that allocates nine calls only when earlier candidates fail obtains the full measured completion rate while avoiding six fruitless rounds on the unresolved tail. External verification makes this adaptive allocation possible because it provides a reliable stopping event. Candidate ordering could reduce this cost further: personalized recommendation methods offer practical ranking machinery [39], while spectral rank-aggregation theory provides a formal basis for combining noisy preferences [40]. Any such ranker would alter search efficiency, not the acceptance boundary; selected plans would still require VAL.

3.5 Statistical Evidence and System Implications

Table XI summarizes inferential results. The initial valid rate had a Wilson 95% interval of 3.15-18.84%, and the final rate

had an interval of 69.20-90.23%. The repair rate among initially invalid tasks had an interval of 66.83-89.35%. The paired risk difference was 74 points with a 62-86 point bootstrap interval. Exact McNemar testing yielded $p = 1.46 \times 10^{-11}$ because 37 pairs improved and none deteriorated. These results reject the explanation that the observed gain arose from unpaired instance variation. The repair policy changed outcomes on the same tasks.

Table Xi Statistical Estimates And Tests

Estimate	Outcome	Point estimate	Interval / test	Method
Initial valid rate	4/50	8.00	3.15-18.84	Wilson 95% CI
Final valid rate	41/50	82.00	69.20-90.23	Wilson 95% CI
Repair among initially invalid	37/46	80.43	66.83-89.35	Wilson 95% CI
Paired risk difference	37/50	+74.00 pp	62-86 pp	20,000 paired bootstrap
Initial vs. final	37 improve / 0 worsen	-	$p = 1.46 \times 10^{-11}$	Exact McNemar
Standard vs. mystery	826/3000 vs. 108/3000	OR = 10.174	chi-square = 651.9; $p < 0.001$	2 x 2 validity table

Taken together, the experiments identify three distinct roles. Prompt design affects proposal quality, as shown by the advantage of explicit state tracking. Formal verification affects decision quality, as shown by the gate's change from 8% to 100% accepted-plan precision without any change in completion. Iterative regeneration affects coverage, as shown by the increase from 4 to 41 completed tasks under the same acceptance rule. Conflating these roles obscures system behavior. A high-quality planner needs good proposals, a sound acceptance boundary, and a budgeted mechanism for acting on rejection. Data-driven marketing decision systems offer a parallel distinction between predictive insight and the policy that acts on it [41]; the planning results make that separation explicit through proposal quality, acceptance precision, and completion.

The results also clarify why a language model should not serve as the final verifier for deterministic planning. Prior work found that LLM self-verifiers accept invalid plans and that self-critique can reduce planning performance [31], [32]. The present evaluation uses VAL as the acceptance authority and obtains zero false acceptance in the measured set. The LLM is still useful: it generates diverse repairs that recover 37 failed tasks. The architecture does not ask one model to be simultaneously creative and sound. It assigns creativity to generation and soundness to formal execution checking. An analogous separation is important in credit-risk modeling, where a decision-tree score can support assessment but does not by itself define policy or verification [42].

The mystery results reinforce that architecture beyond the matched repair subset. Familiar action names allowed substantially higher direct validity, but the formal verifier was

invariant to whether predicates were semantically meaningful or opaque. This is a core advantage of symbolic checking: once the candidate is grounded to the correct action schema, acceptance follows from state transitions, not from linguistic familiarity. A future repair agent for mystery domains therefore needs stronger symbol grounding and proposal generation, but it does not need a different correctness criterion. Long-document RAG for contractual and insurance analysis provides a parallel form of grounding: conclusions are tied to retrieved clauses rather than surface fluency [43]. In planning, the PDDL transition system supplies the authoritative grounding.

PDDL prompting deserves the same distinction. Giving formal syntax to the LLM reduced ambiguity in representation, yet direct PDDL rates remained below natural-language rates in standard tasks and near zero in mystery tasks. Formal input is not equivalent to formal reasoning. The parser and validator make the representation executable, while the generator still has to predict a globally consistent sequence. The empirical value of PDDL in this pipeline lies in machine-checkable semantics, not in an assumption that an LLM will automatically search the induced state space. Broader AI-driven financial-service systems likewise combine data-derived insight with explicit risk analysis instead of treating a prediction as a self-authorizing decision [44].

For agent engineering, the measured policy is straightforward. The system should never execute an uncertified deterministic plan when an exact domain model and validator exist. A single verifier call removes unsafe execution but creates abstention. Structured failure feedback should then be used under an adaptive cap. In the measured trajectories, four candidates provided a substantial low-cost operating point, seven captured most of the available benefit, and nine captured all observed benefit. The final unresolved tasks should be escalated to a classical planner, a human operator, or a model with a different search mechanism rather than repeated with identical feedback. Where an exact world model is unavailable, monitored evidence must supplement the controller; deep-autoencoder IoT traffic analysis can identify abnormal operating states [45]. Such monitors complement rather than replace plan certification.

4. Limitations

The evaluation measures stored GPT-4 responses in the PlanBench repository. It establishes the behavior of those candidates under fresh validation, not the performance of newer models, different decoding temperatures, or newly sampled outputs. Model comparison was intentionally excluded because the only complete verifier-feedback trajectories aligned with the original GPT-4 backprompting records. The reproducible unit is the candidate plan, its problem file, and its validator result.

The repair analysis contains 50 standard four-object Blocksworld tasks and is failure enriched. Its paired estimates are internally valid, but its 82% final completion rate is not a population estimate for every PlanBench domain. Logistics,

Sokoban, unsolvable-task recognition, and other benchmark capabilities require domain-specific parsers, validators, and outcome definitions. The broad direct analysis covers standard and mystery Blocksworld only because those conditions supplied the complete prompt grid needed for controlled comparisons.

Formal verification is sound relative to the supplied PDDL model. The experiment does not test whether that model perfectly represents a physical environment, uncertain sensing, continuous control, or hidden effects. In a deployed robot or software agent, model errors can make a formally valid plan unsafe in the real system. The modulo architecture still applies, but the verifier must incorporate execution monitors, state estimation, and domain constraints that match the deployment environment. In embodied systems, perception fusion [23] and target-detection improvements based on RFACConv and triplet attention [46] can strengthen the estimated state, but neither certifies a downstream action sequence.

The deterministic parser favors precision over recall. It rejects malformed or incomplete natural-language action lines and counts an empty extraction as invalid. A human reader could occasionally infer an intended action that the parser discards, but such intent is not an executable plan. The reported metric therefore evaluates end-to-end machine executability. Alternative parsers would change the boundary between empty, syntax, and semantic failures, although every accepted plan would still require VAL certification.

Compute was measured in candidate and verifier calls because archived traces do not provide a uniform token-accounting basis across every record. The study does not report dollar cost, wall-clock generation latency, or energy use. The repair policy also uses the feedback wording recorded in the repository, so it does not isolate alternative diagnostic templates. Finally, the mystery transformation tests semantic obfuscation, not arbitrary distribution shift; its results should be interpreted as evidence about lexical dependence within a fixed transition family. Deployment also introduces nonstationary operational risks. Predictive DDoS mitigation requires live monitoring and policy-constrained response [47], while shared-infrastructure degradation can require telemetry-aware risk models [48]. More generally, uncertainty quantification for spectral estimators [49] addresses confidence in noisy estimates, whereas VAL addresses logical validity under a fixed symbolic model; a deployed agent may need both.

5. Conclusion

This study demonstrates that external verification and iterative repair solve different parts of the LLM planning problem. On 6,000 direct PlanBench candidates, validity was low overall and fell from 27.53% in standard Blocksworld to 3.60% under semantic obfuscation. Explicit state tracking improved proposals, but no prompt form produced dependable direct execution. On the matched 50-task set, a verifier gate eliminated invalid execution and raised accepted-plan precision to 100%, yet completion remained

8%. Feeding verifier diagnostics back into the generator raised completion to 82%, repaired 80.43% of initially invalid plans, preserved zero invalid acceptance, and reached its full measured benefit by candidate nine.

The central design principle is a strict division of labor. The LLM generates and edits candidate plans; the symbolic verifier alone decides validity; the controller allocates additional generation only after a concrete failure. This arrangement turns verification from a passive evaluation tool into an active control signal. For deterministic domains with an executable model, modulo planning provides a practical agent policy: certify before execution, repair under a bounded budget, and escalate unresolved cases instead of trusting fluent but unchecked plans.

References

- [1] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, nos. 3-4, pp. 189-208, 1971.
- [2] D. McDermott et al., "PDDL - The Planning Domain Definition Language," Yale Center for Computational Vision and Control, New Haven, CT, Tech. Rep. CVC TR-98-003/DCS TR-1165, 1998.
- [3] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. San Francisco, CA, USA: Morgan Kaufmann, 2004.
- [4] R. Howey, D. Long, and M. Fox, "VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL," in *Proc. 16th IEEE Int. Conf. Tools with Artificial Intelligence (ICTAI)*, 2004, pp. 849-857.
- [5] M. Helmert, "The Fast Downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191-246, 2006.
- [6] W. Huang et al., "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *Proc. 39th Int. Conf. Machine Learning (ICML)*, PMLR, vol. 162, 2022, pp. 9118-9147.
- [7] M. Ahn et al., "Do as I can, not as I say: Grounding language in robotic affordances," in *Proc. 6th Conf. Robot Learning (CoRL)*, PMLR, vol. 205, 2023, pp. 287-318.
- [8] W. Huang et al., "Inner monologue: Embodied reasoning through planning with language models," in *Proc. 6th Conf. Robot Learning (CoRL)*, PMLR, vol. 205, 2023, pp. 1769-1782.
- [9] Singh et al., "ProgPrompt: Generating situated robot task plans using large language models," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2023, pp. 11523-11530.
- [10] J. Liang et al., "Code as policies: Language model programs for embodied control," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2023, pp. 9493-9500.
- [11] B. Liu et al., "LLM+P: Empowering large language models with optimal planning proficiency," *arXiv:2304.11477*, 2023.
- [12] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 24824-24837.
- [13] X. Wang et al., "Self-consistency improves chain of thought reasoning in language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.
- [14] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.
- [15] Madaan et al., "Self-Refine: Iterative refinement with self-feedback," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.
- [16] N. Shinn et al., "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.
- [17] S. Yao et al., "Tree of thoughts: Deliberate problem solving with large language models," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 11809-11822.
- [18] L. Gao et al., "PAL: Program-aided language models," in *Proc. 40th Int. Conf. Machine Learning (ICML)*, PMLR, vol. 202, 2023, pp. 10764-10799.
- [19] S. Hao et al., "Reasoning with language model is planning with world model," *arXiv:2305.14992*, 2023.
- [20] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.
- [21] S. Zhou, Z. Li, and E. Wang, "Evidence-Grounded RAG for Tokenized Trade Receivable Disclosure QA under U.S. Capital Market Standards," *J. Adv. Comput. Syst.*, vol. 3, no. 7, pp. 41-57, Jul. 2023, doi: 10.69987/JACS.2023.30704.
- [22] Y. Li, "Findable then Explainable: Retrieval-Summary Integration for Code Intelligence on a Lightweight CodeSearchNet Subset," *J. Adv. Comput. Syst.*, vol. 4, no. 7, pp. 65-82, Jul. 2024, doi: 10.69987/JACS.2024.40706.
- [23] Y. Wang, S. Du, Q. Xin, Y. He, and W. Qian, "Autonomous Driving System Driven by Artificial Intelligence Perception Fusion," *Acad. J. Sci. Technol.*, vol. 9, no. 2, pp. 193-198, Feb. 2024, doi: 10.54097/e0b9ak47.
- [24] K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, and S. Kambhampati, "PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 38975-38987.
- [25] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati, "On the planning abilities of large language models: A critical investigation," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 75993-76005.

- [26] J. Nie and D. Zheng, "Ambiguity-Aware HDFS Log Anomaly Detection with Retrieval-Augmented Failure Narratives and Selective Refusal," *J. Adv. Comput. Syst.*, vol. 3, no. 1, pp. 66-80, Jan. 2023, doi: 10.69987/JACS.2023.30105.
- [27] Z. S. Zhong, R. Ma, and H. Zhao, "Human-Uncertainty Distillation for Calibrated Vision Models on CIFAR-10H," *J. Adv. Comput. Syst.*, vol. 3, no. 2, pp. 77-89, Feb. 2023, doi: 10.69987/JACS.2023.30206.
- [28] H. Zhang, "DriftGuard: Multi-Signal Drift Early Warning and Safe Re-Training/Rollback for CTR/CVR Models," *J. Adv. Comput. Syst.*, vol. 3, no. 7, pp. 24-40, Jul. 2023, doi: 10.69987/JACS.2023.30703.
- [29] Z. Zhong, M. Zheng, H. Mai, J. Zhao, and X. Liu, "Cancer image classification based on DenseNet model," *J. Phys.: Conf. Ser.*, vol. 1651, no. 1, Art. no. 012143, Nov. 2020, doi: 10.1088/1742-6596/1651/1/012143.
- [30] J. Chen, J. Xiong, Y. Wang, Q. Xin, and H. Zhou, "Implementation of an AI-based MRD Evaluation and Prediction Model for Multiple Myeloma," *Front. Comput. Intell. Syst.*, vol. 6, no. 3, pp. 127-131, Jan. 2024, doi: 10.54097/zJ4MnbWW.
- [31] K. Valmeekam, M. Marquez, and S. Kambhampati, "Can large language models really improve by self-critiquing their own plans?" in *NeurIPS 2023 Workshop on Foundation Models for Decision Making*, 2023.
- [32] K. Stechly, M. Marquez, and S. Kambhampati, "GPT-4 does not know it is wrong: An analysis of iterative prompting for reasoning problems," in *NeurIPS 2023 Workshop on Foundation Models for Decision Making*, 2023.
- [33] Ni, S. Iyer, D. Radev, V. Stoyanov, W.-T. Yih, S. Wang, and X. V. Lin, "LEVER: Learning to verify language-to-code generation with execution," in *Proc. 40th Int. Conf. Machine Learning (ICML)*, PMLR, vol. 202, 2023, pp. 26106-26128.
- [34] H. Zhang, "Risk-Aware Budget-Constrained Auto-Bidding under First-Price RTB: A Distributional Constrained Deep Reinforcement Learning Framework," *J. Adv. Comput. Syst.*, vol. 4, no. 6, pp. 30-47, Jun. 2024, doi: 10.69987/JACS.2024.40603.
- [35] E. B. Wilson, "Probable inference, the law of succession, and statistical inference," *Journal of the American Statistical Association*, vol. 22, no. 158, pp. 209-212, 1927.
- [36] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153-157, 1947.
- [37] G. S. Raj, H. Zhang, V. Abhyankar, S. Mukherjee, E. Zhang, J. Williams, R. Halverson, and J. M. Patel, "Impact of bilingual CS education on student learning and engagement in a data structures course," in *Proc. 19th Koli Calling Int. Conf. Comput. Educ. Res.*, 2019, Art. no. 24, pp. 1-10, doi: 10.1145/3364510.3364518.
- [38] Y. He, Y. Pan, Y. Wang, S. Du, and Q. Xin, "Intelligent Fault Analysis With AIOps Technology," *J. Theory Pract. Eng. Sci.*, vol. 4, no. 1, pp. 94-100, Feb. 2024, doi: 10.53469/jtpes.2024.04(01).13.
- [39] K. Xu, H. Zhou, H. Zheng, M. Zhu, and Q. Xin, "Intelligent classification and personalized recommendation of e-commerce products based on machine learning," in *Proc. 6th Int. Conf. Computing and Data Science, Applied and Computational Engineering*, vol. 64, no. 1, pp. 143-149, 2024, doi: 10.54254/2755-2721/64/20241365.
- [40] Z. S. Zhong and S. Ling, "Improved theoretical guarantee for rank aggregation via spectral method," *Inf. Inference: J. IMA*, vol. 13, no. 3, Art. no. iaac020, Sep. 2024, doi: 10.1093/imaia/iaac020.
- [41] J. Wang, Q. Xin, Y. Liu, J. Wang, and T. Yang, "Predicting Enterprise Marketing Decision Making with Intelligent Data-Driven Approaches," *J. Ind. Eng. Appl. Sci.*, vol. 2, no. 3, pp. 12-19, Jun. 2024, doi: 10.5281/zenodo.11357252.
- [42] Q. Xin, R. Song, Z. Wang, Z. Xu, and F. Zhao, "Enhancing Bank Credit Risk Management Using the C5.0 Decision Tree Algorithm," *J. Comput. Technol. Appl. Math.*, vol. 1, no. 4, pp. 100-107, Nov. 2024, doi: 10.5281/zenodo.14032041.
- [43] S. Zhou, Z. Li, and E. Wang, "Long-Document RAG for Contractual and Insurance Clause Analysis in Receivables RWA Structures," *J. Adv. Comput. Syst.*, vol. 4, no. 8, pp. 88-104, Aug. 2024, doi: 10.69987/JACS.2024.40810.
- [44] T. Yang, Q. Xin, X. Zhan, S. Zhuang, and H. Li, "Enhancing Financial Services through Big Data and AI-Driven Customer Insights and Risk Analysis," *J. Knowl. Learn. Sci. Technol.*, vol. 3, no. 3, pp. 53-62, Jul. 2024, doi: 10.60087/jklst.vol3.n3.p53-62.
- [45] Q. Xin, Z. Xu, L. Guo, F. Zhao, and B. Wu, "IoT traffic classification and anomaly detection method based on deep autoencoders," in *Proc. 6th Int. Conf. Computing and Data Science, Applied and Computational Engineering*, vol. 69, no. 1, pp. 64-70, Jul. 2024, doi: 10.54254/2755-2721/69/20241511.
- [46] Z. Ling, Q. Xin, Y. Lin, G. Su, and Z. Shui, "Optimization of Autonomous Driving Image Detection Based on RFACnv and Triplet Attention," in *Proc. 2nd Int. Conf. Software Engineering and Machine Learning, Applied and Computational Engineering*, vol. 77, pp. 210-217, 2024, doi: 10.54254/2755-2721/77/2024MA0067.
- [47] Wang, Y. He, Z. Shui, Q. Xin, and H. Lei, "Predictive optimization of DDoS attack mitigation in distributed systems using machine learning," in *Proc. 6th Int. Conf. Computing and Data Science, Applied and Computational Engineering*, vol. 64, no. 1, pp. 94-99, May 2024, doi: 10.54254/2755-2721/64/20241350.

- [48] J. Nie and D. Zheng, "Noisy-Neighbor-Aware VM Degradation Risk Modeling with Unsupervised Residual Fusion," *J. Adv. Comput. Syst.*, vol. 4, no. 4, pp. 112-123, Apr. 2024, doi: 10.69987/JACS.2024.40409.
- [49] Z. S. Zhong and S. Ling, "Uncertainty Quantification of Spectral Estimator and MLE for Orthogonal Group Synchronization," *arXiv:2408.05944*, Aug. 2024.