Where Does Plan Validity Live in Grammar-Constrained LLM Planning? An Applicability Mask Induced from Environment Rollouts

Nijesh Upreti
School of Informatics, University of Edinburgh
n.upreti@ed.ac.uk

Abstract

LLM-augmented symbolic planning combines natural-language goal grounding with symbolic search and constrained decoding. We present **Abstraction-based Grammar-Constrained Planning** (AGCP), a framework that induces lifted action schemas from environment rollouts, compiles a bounded language of grounded action sequences that are applicable and goal-reaching under the induced model, and enforces that language during decoding. The guarantee is relative to the induced model. Environment execution additionally requires a sound induced model and a correct parsed goal. With the model, prompts, instances, and decoding settings fixed, replacing a syntax-only grammar with the applicability mask, compiled here from the domain's own transition model, moves goal-reaching samples from $0/160$ to $160/160$. On human-annotated ALFWorld goals, verifier-guided correction raises parse acceptance from 61.4% to 75.4% and execution success from 67.5% to 80.7% on 114 paired tasks. Under paraphrasing, GPT-4o and Sonnet grounding remain stable while a regular-expression baseline falls from 31.9% to 0.8%. Together, these results locate validity relative to the induced model in an applicability grammar built from a symbolically observable, bounded environment. In this setting, natural-language goal grounding is the main remaining role of the LLM.

1 Introduction

Planning from a natural-language instruction can fail in two ways. The plan can be inapplicable, or it can be applicable and solve the wrong goal. We ask which failure remains with the language model once a grammar enforces applicability. Grammar-constrained decoding methods such as XGrammar (Dong et al., 2025; Li et al., 2026), Outlines (Willard and Louf, 2023) and the grammar-constrained generation of Geng et al. (2023) mask invalid tokens at each step so that large language model (LLM) outputs conform to a fixed grammar or schema. In planning, however, action selection is often still left to the language model itself (Valmeekam et al., 2023; Yao et al., 2023; Shinn et al., 2023).

A syntax-only grammar of this kind enforces a *syntactic* constraint. It admits exactly the well-formed strings of a fixed schema, here a JSON list of action names with correct arity (Dong et al., 2025). *Applicability*, whether each action's preconditions (Fikes and Nilsson, 1971) hold in the state its predecessors reach, is a semantic property of the sequence and remains with the LLM. Suppose block A rests on the table. The grammar admits both *pick up A, then stack A on B* and *stack A on B, then pick up A*. Only the first is applicable, because stacking A requires that A is already held.

We study what changes when applicability is enforced by the grammar rather than predicted by the LLM. We call such a grammar an *applicability mask*. If the framework provides *plan validity*, an applicable plan that reaches the goal, without the LLM, the remaining LLM task is natural-language *goal grounding*: mapping an instruction to the predicates and objects of the environment (Valmeekam et al., 2023; Kambhampati et al., 2024; Valmeekam et al., 2024).

Research questions. We isolate four components that may contribute to planning success, namely the LLM, a hand-crafted parser, the applicability mask, and the parse-correction loop, so that each can be evaluated separately. **RQ1:** Can the LLM be removed from classical planning once an applicability mask is available? **RQ2:** How reliable is natural-language goal grounding on human-annotated goals, and how does it vary across models and task types? **RQ3:** Can verifier-guided correction improve grounding without consulting the target labels? **RQ4:** How much of end-to-end success is owed to the parse and how much to the

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 2965–2990
November 6-10, 2026 ©2026 Association for Computational Linguistics

planner?

To answer them we introduce Abstraction-based Grammar-Constrained Planning (AGCP). It induces lifted action schemas, the preconditions and effects of each action type, from environment rollouts, which are random action sequences executed in the environment with the state recorded before and after each step. From the schemas it compiles a bounded language of action sequences that are applicable in the initial state and reach the parsed goal, and it enforces that language during decoding as the applicability mask (§3).

AGCP assigns applicability to the induced model and the decoder constraint, and the LLM operates at the natural-language (NL) boundary, translating an instruction into the predicates and objects the symbolic system requires. The guarantee is conditional. Where compilation succeeds, every admitted sequence is applicable and goal-reaching under the induced model. Applicability in the environment further requires a sound model, and satisfying the instruction requires a correct parse. Holding the model, prompts, instances and decoding fixed and varying only the mask, we find that no sampled plan reaches the goal under a syntax-only grammar, while every sampled plan reaches it under the applicability mask.

Contributions.

- We introduce AGCP, a framework that induces an applicability grammar from environment rollouts and enforces it during decoding, and we state the conditions under which its guarantee holds.
- We isolate the applicability mask experimentally, holding everything but the grammar fixed, and we measure per domain how often the compiled grammar stays within its state budget rather than assuming the favourable case.
- We separate language grounding from symbolic planning through mask-isolation, planner and paraphrase controls across classical and human-annotated tasks.
- We evaluate verifier-guided parse correction, replicated on a second model family, and state the information boundary between environment feedback and label leakage.

2 Background and Related Work

Constrained decoding. XGrammar (Dong et al., 2025), Outlines (Willard and Louf, 2023), DOMINO (Beurer-Kellner et al., 2024) and Geng et al. (2023) enforce grammars during decoding. PICARD (Scholak et al., 2021) and Synchromesh (Poesia et al., 2022) add static semantic checks such as schema, scope and type consistency. ChopChop (Nagy et al., 2026) supports programmable semantic constraints over partially generated programs. Our G_{env} intersects the action grammar with the reachability automaton of a transition model induced from rollouts. The resulting constraint covers applicability and goal reachability wherever the intersection compiles (§3.3). In concurrent work, Meta-Ctrl (Yidou-Weng et al., 2026) enforces preconditions and goals through grounded-action tokens, and CLAMP (Ma and Kordjamshidi, 2026) combines scene-grounded decoding constraints with world-state lookahead. Both assume a supplied symbolic action model.

LLM-as-planner. Valmeekam et al. (2023) establish PlanBench and report GPT-4 one-shot plan generation at 34.3% on Blocksworld and 4.3% on deceptive Mystery Blocksworld, with Valmeekam et al. (2024) extending this to reasoning models. In LLM+P (Liu et al., 2023) the LLM emits a Planning Domain Definition Language (PDDL) (McDermott et al., 1998) problem for a classical solver, so the planner is sound while the problem-specification step is not. ReAct (Yao et al., 2023) interleaves reasoning and action, Reflexion (Shinn et al., 2023) carries verbal feedback across trials, and Self-Refine (Madaan et al., 2023) revises outputs with feedback from the same LLM. None enforces applicability during decoding. Wang et al. (2024) maintain a Minecraft skill cache, the closest prior work to our schema reuse, without soundness guarantees, and AutoPlanBench (Stein et al., 2025) benchmarks LLM-planning strategies on IPC (International Planning Competition) domains without grammar-constrained variants.

Verifiers and LLM-authored world models. Kambhampati et al. (2024) argue that autoregressive LLMs cannot plan or self-verify unaided and must be vetted by external model-based critics. Our results support that position with a stronger constraint, a decode-time applicability mask. That work also cautions that confining the LLM to syntax reformulation is a severely limited

role. Our measurements separate the two regimes in §5.2. Guan et al. (2023) also externalise planning to a sound solver, but have the LLM *write* the PDDL domain and repair it from validator and human feedback. We induce the schemas from rollouts and gate them by replay, so no LLM authors the world model. SayCan (Ahn et al., 2023) scores skills by combining language-model likelihood with learned affordances, and Grounded Decoding (Huang et al., 2023) folds environment-dependent grounding functions into token generation. AGCP instead compiles a transition model into a grammar whose accepted plans are applicable and, under the horizon-bounded construction, goal-reaching relative to that model. Xie et al. (2023) translate natural-language goals into planning representations for a classical planner, as our parser does, and Planetarium (Zuo et al., 2025) benchmarks that step. We evaluate on ALFWorld because it supports the execution and feedback experiments.

Action-schema learning. LOCM (Cresswell et al., 2013) and the action-relation modelling system (ARMS) (Yang et al., 2007) learn STRIPS action schemas from plan traces without a complete action model supplied in advance. ARMS additionally takes action signatures and typed relations as input. Safe action-model learning (Juba et al., 2021) learns conservative schemas whose plans are guaranteed applicable under the true model, assuming injective action bindings, which its E-SAM extension relaxes. Our precondition rule is the same intersection, but positional templating makes no binding assumption. Pre-minimality refinement and the causal effect rule trade the guarantee for coverage, which the replay gate then tests empirically. None of these methods integrate with LLMs or ingest natural-language goals. Our E_1 follows this line and composes schema induction with G_{env}, masked decoding, and natural-language goal parsing.

Notation. A planning problem is a tuple $(\mathcal{P}, \mathcal{O}, \mathcal{A}, s_0, g)$ where $\mathcal{P}$ is the predicate vocabulary, $\mathcal{O}$ the object set, $\mathcal{A}$ the action set, s_0 the initial state, and g the goal. A state s is a set of ground atoms over $\mathcal{P} \times \mathcal{O}^{\leq k}$, where k is the maximum predicate arity, so an atom is a predicate applied to a tuple of objects. We reserve *env* for the transition system itself, given by $\mathcal{A}$ and its transition function, and *state* for a single such atom set within it. A transition is written $(s, a(\vec{o}), s')$,

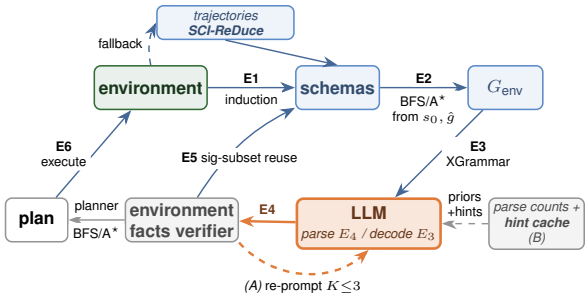


Figure 1: The six-edge closed loop. The LLM box represents two constrained calls: goal parsing under a JSON schema (E_4) and plan decoding under G_{env} (E_3). Only goal parsing requires language understanding. The parsed goal enters compilation at E_2. Two mechanisms correct the parse at E_4. *Layer A* re-prompts within a task, for up to K iterations, with the affordance violations reported by the environment-facts verifier (dashed backward arrow). *Layer B* injects per-type slot-value statistics and filtered slot-correction hints from a cache updated across tasks.

where applying the ground action $a(\vec{o})$ in state s yields successor state s'. An action $a \in \mathcal{A}$ has lifted preconditions and add/delete effects over $\mathcal{P}$. A plan $\pi = a_1, \ldots, a_n$ is applicable in s_0 if each a_i's preconditions hold in the state reached after $a_1, \ldots, a_{i-1}$, and goal-reaching if applying π to s_0 entails g. A natural-language goal g_{NL} must be parsed into a structured atom set before the planning problem is fully specified. That parse is the only LLM-dependent edge.

3 The AGCP framework

AGCP maps a natural-language goal to an executed plan through schema induction, constrained decoding and environment feedback (Fig. 1).

3.1 Six-edge closed loop

$\mathbf{E_1}$ induces lifted schemas from random rollouts (≈ 500 attempts per domain). $\mathbf{E_2}$ compiles the applicable, goal-reaching language into a context-free grammar (CFG) by breadth-first search (BFS), falling back to A*+hAdd (Bonet and Geffner, 2001) when the state budget is exhausted. $\mathbf{E_3}$ enforces the mask via XGrammar. $\mathbf{E_4}$ parses the NL goal into an atom set under a JSON schema whose allowed values are the induced predicate symbols. This is the only LLM-dependent edge. $\mathbf{E_5}$ stores replay-passing schemas in a schema library indexed by signature (App. O), and $\mathbf{E_6}$ executes the plan and reads the environment's success signal.

3.2 Lifted-schema induction (E_1)

The environment is a black box, so PDDL action schemas are unavailable. We recover lifted preconditions and effects for each action symbol. For an action symbol a, we collect a multiset T_a of successful transitions $(s, a(\vec{o}), s')$ and induce its lifted schema over lifted predicate sets:

$$\begin{aligned}
\mathrm{pre}(a) &= \bigcap_{T_a} \mathrm{lift}_{\vec{o}}(s) \\
\mathrm{add}(a) &= \bigcup_{T_a} \mathrm{lift}_{\vec{o}}(s' \setminus s) \cap \bigcap_{T_a} \mathrm{lift}_{\vec{o}}(s \cup s') \\
\mathrm{del}(a) &= \bigcup_{T_a} \mathrm{lift}_{\vec{o}}(s \setminus s') \setminus \bigcup_{T_a} \mathrm{lift}_{\vec{o}}(s \cap s')
\end{aligned}$$

where $\mathrm{lift}_{\vec{o}}$ replaces object ids with templated argument slots at their position (App. C). Predicates holding in s for *every* successful $a(\vec{o})$ are *candidate* preconditions, since an incidental correlation can survive the intersection. An add effect is a lifted atom that some transition adds and that no transition leaves absent both before and after, and a delete effect is one that some transition deletes and no transition keeps throughout. For Blocksworld's `pickup`(x), the intersection of $\approx$50 rollouts recovers the standard PDDL definition without schema text. Pre-minimality refinement then drops a candidate precondition when the domain's ground operator still applies with that atom removed from an observed state, a counterfactual query beyond passive rollouts (App. C) that four IPC-2023 domains require. As an acceptance test, induced schemas must replay $\geq 90\%$ of held-out transitions, and those that fail are excluded from G_{env}. Passing raises confidence without certifying the model, since a schema can clear the threshold and still mispredict transitions the mask admits. Intersection recovers only preconditions holding in every observed transition, so a type-dependent one absent from all of them is missed, which §5.4 identifies as the dominant residual error of the induced planner.

3.3 Induced applicability mask G_{env} (E_2)

From lifted schemas $\hat{\Sigma}$ and an initial state s_0 the compiler builds a finite-state machine $\mathcal{M}_\Sigma(s_0)$ over (*state*, *steps-remaining*) pairs by BFS bounded at a plan horizon H. It includes $a(\vec{o})$ as a transition from s to s' only when $\mathrm{pre}(a)[\vec{o}]$ holds in s. The machine encodes the intersection of the syntactic action grammar with the reachability relation from s_0. Bounding plan length over a finite grounded action alphabet makes the intersected language finite, and therefore regular, so it is representable as a deterministic finite automaton (DFA). The DFA is emitted in extended Backus–Naur form (EBNF), giving a CFG G_{env} whose language is exactly the grounded action sequences reaching a goal state within H actions, which XGrammar enforces as a hard token mask.

Two properties follow, and both are relative to the induced model $\hat{\Sigma}$ and the parsed goal $\hat{g}$. Under *goal-validity*, if compilation returns a non-empty language then every $\pi \in L(G_{\text{env}})$ is applicable in s_0 under $\hat{\Sigma}$, reaches $\hat{g}$, and respects the horizon H, because BFS emits only paths satisfying all three. Under *LLM-independence*, any completed output of an LLM whose token distribution has full support on the masked vocabulary reaches a goal state within H actions, since every legal continuation leads to one. Its only role is tie-breaking among already-valid continuations. The property concerns accepted outputs and does not guarantee that decoding completes within the token budget.

Two further conditions are needed before either property says anything about the environment or the user. Applicability in the environment requires $\hat{\Sigma}$ to be sound on the transitions the mask admits, which the replay gate of §3.2 tests empirically but does not certify. Satisfying the *instruction* requires $\hat{g}$ to be a correct parse, which is the LLM-dependent edge and fails at the rates of §5.2. A plan can therefore be valid under the induced model and still be the wrong plan.

Boundary condition: when the mask can be built. Both properties are conditional on G_{env} compiling, and that is a property of the environment. The construction assumes Boolean predicate state, deterministic transitions, full observability, finite-arity grounded schemas and a reachable state space enumerable within a 5000-state budget. Once that budget is exhausted the compiler falls back to A*+hAdd over the domain's PDDL model, which that path takes as input in place of the induced schemas, and emits a single sequence as a degenerate $G_{\text{env}}^{A^\star}$. Goal-validity still holds, but the grammar contains a single classical plan, so decoding no longer makes a planning choice.

On PlanBench T1, the full multi-path G_{env} compiles within the state budget on Blocksworld, Mystery Blocksworld and Logistics (30/30 pooled, N=10 per domain, cap never reached). Depots exhausts the state budget on all 30 instances and falls back to A*, which solves 30/30. At the larger

AutoPlanBench scale (Stein et al., 2025), 70 of 230 instances compile under BFS, 16 of them from a capped reachability set. A further 104 reach the cap without reaching a goal state and 17 time out, and A* recovers 93 of those 121. The other 39 never reach compilation.

Compile provenance. On PlanBench T1 the generic BFS compiler produces no grammar, and A*+hAdd supplies the grammar for all 60 instances. The 100% reported there therefore comes from classical search encoded as a grammar. Multi-path masked decoding contributes nothing on that path. The per-domain statistics above come from a corrected state encoding under which BFS does produce grammars, with applicability during BFS answered by the domain's exact ground operators rather than by the induced preconditions. The mask isolation of §5.1 uses a Blocksworld-native compiler whose transition rules are hand-coded, so it isolates the mask and not the induction. The induced schemas determine the component-F_1 result of §5.1, the induced planner of §5.4 and the reuse probe of App. O (Table 13).

3.4 NL-goal parsing with parse-layer correction (E_4)

One-shot parsing can yield structurally valid but semantically wrong atom sets. A common failure is class confusion. For example, when asked to examine the floor lamp in the bedroom, the LLM names a floor lamp as the object to toggle, even though the only lamp present is a desk lamp whose canonical class name the parse must reproduce exactly. The mask accepts the wrong name because it is well formed, so the error surfaces only when the name is checked against the scene's affordances (Layer A below) or at execution.

One-shot parser. The LLM parses the natural-language goal description under a JSON schema whose allowed values are the predicate symbols from the induced lifted schemas, or ALFWorld's slot and object-type vocabulary. XGrammar enforces the schema for the local model. Cloud models receive it in the prompt and their output is checked after generation (App. S). On PlanBench an environment-facts verifier checks that each predicate is an induced one and each argument a valid object identifier, a rejected parse counts as a failure, and the parse column of Table 7 is selected among four samples by a label-scored verifier (App. G). The ALFWorld model comparison of §5.2 scores each model's parsed output directly, with unparseable output scored as empty, and uses no ground-truth-aware reranking. For reporting we score a parse against the ground-truth goal (PDDL atoms on PlanBench, goal slots on ALFWorld) at $F_1^{\text{norm}} \geq 0.9$ (App. S).

Two correction layers. *Layer A* acts within a task. After each parse every slot value is checked against environment affordances read from ALFWorld's symbolic state observation, and if a slot is unrecognised the LLM is re-prompted with that slot and the scene's legal alternatives, up to $K{=}3$ iterations. *Layer B* acts across tasks. Two per-task-type stores are updated online: frequency tables over slot values from past *successful* parses, whose top-3 entries are added to the prompt, and short slot-correction rules extracted from past *failed* parses. A phrase blocklist removes hints that describe actions to execute, and a hint must name a slot value (App. U). The filter is a keyword rule rather than a guarantee. All 19 retained hints are slot-level corrections. Neither layer consults a ground-truth solution or an expert demonstration (App. H.1).

Layer B's only supervision is the environment's success signal and its affordances. The ground-truth goal conditions are recorded for scoring and never read by the correction loop, and the stream is online (App. U).

3.5 SCI-ReDuce: schema-free induction fallback

The pipeline above assumes symbolic transitions $(s, a(\vec{o}), s')$ exposing predicate-level state deltas. SCI-ReDuce relaxes that requirement at *induction* time. It compresses successful action sequences alone, treating actions as opaque symbols, and then indexes each macro by the ground atoms shared by the recorded states at which it was observed. Induction therefore needs recorded states but no per-action deltas and no schema, and decode-time admission still consults state. We adapt the compression procedure of ReDuce (Muggleton, 2025), add the state-based indexing, call the result **SCI-ReDuce**, and give the complete adapted algorithm in App. P. Decode-time admission requires the current state to entail the macro's entry class. If a macro's executability and exit class depend only on its entry class, this gives $L(G) \subseteq L_{\text{exec}}$, with L_{exec} the sequences executable from the current state (Prop. 3). Without that assumption, entry-class admission is a heuristic.

SCI-ReDuce provides a weaker guarantee. The horizon-bounded construction of §3.3 carries soundness *and* termination at a goal state, whereas SCI-ReDuce carries soundness only under the assumption above and admits unbounded macro loops. No sampled plan completed within the tested token budget (§5.4). We therefore scope the rollout-induction claim to environments whose state can be observed symbolically at decode time.

4 Experiments

We evaluate nine systems, eight instruction-tuned LLMs spanning four model families together with a non-neural alias-table baseline. Qwen-2.5-1.5B runs locally under XGrammar JSON-schema masking. Qwen-2.5-7B and Llama-3.3-70B run via Together.ai (Together AI, 2024; Meta AI, 2024; Grattafiori et al., 2024), GPT-3.5-turbo, GPT-4o-mini and GPT-4o (OpenAI, 2024) via OpenAI, and Claude-Haiku-4.5 and Claude-Sonnet-4.6 via Anthropic. We generate one output per ALFWorld task, no ground-truth-aware reranking there, and $N{=}251$ per cell.

We use three benchmarks. PlanBench T1 (Valmeekam et al., 2023) covers four IPC domains, scored by the VAL plan validator (Howey et al., 2004). ALFWorld (Shridhar et al., 2021) supplies human-written task descriptions, $N{=}251$ per split, covering 242 distinct tasks in distribution (seen) and 85 out of distribution (unseen), since ALFRED provides about three descriptions per task, across seven task families. Execution loads six of them and the primary execution subset is five (§5.4). Tests pair rows and do not model within-task clustering. The IPC-2023 learning track (Taitler et al., 2024) and AutoPlanBench (Stein et al., 2025) add breadth. We compare against four baselines: a hand-crafted regular-expression (regex) parser with an alias table of about 200 entries, an LLM-removed pipeline running pure A*+hAdd over the benchmark's PDDL model, ALFWorld's HandCoded TextWorld (TW) expert as the E_6 planner, and our Qwen-1.5B implementation of LLM+P (Liu et al., 2023) on Depots. We report Wilson 95% confidence intervals (Wilson, 1927) for binary outcomes and exact McNemar binomial tests (McNemar, 1947) with Cohen's h (Cohen, 1988) for paired comparisons. Full model, decoding, metric and seed settings are in App. S. Our evaluation guards against five inflation patterns, such as counting a 0-action empty plan as a solution. Three are inherited from evaluation components of earlier pipelines and two are specific to our own pipeline. Excluding them mostly lowers the figures reported here (App. H).

Domain	LLM+P[a]	No-LLM[b]	AGCP[a]
Blocksworld	—	**10/10**	**10/10**
Mystery	—	**10/10**	**10/10**
Logistics	—	**10/10**	**10/10**
Depots	0/30	**30/30**	**30/30**
Pooled	—	**60/60**	**60/60**

Table 1: PlanBench T1 plan-generation accept counts on matched instance sets, $N{=}10$ for Blocksworld, Mystery and Logistics and $N{=}30$ for Depots. [a]Qwen-1.5B; [b]pure classical (A*+hAdd over the benchmark's PDDL model, no LLM call). LLM+P is our own Qwen-1.5B implementation of the approach of Liu et al. (2023), run on Depots only and scored under the corrected non-empty-plan criterion. Without it the same run reads 12/30 (App. H). For these instances the compiled grammar is the degenerate $G_{\text{env}}^{A^\star}$ throughout, so the No-LLM and AGCP columns represent classical search over the same model.

5 Results

We examine plan validity (RQ1), language grounding (RQ2), parse correction (RQ3) and end-to-end execution (RQ4).

5.1 RQ1: what carries plan validity on classical planning

On the classical PlanBench tasks we evaluate, removing the LLM does not change plan validity.

Removing the LLM. Qwen-1.5B under G_{env} parses all 120 natural-language goals correctly under verifier-selected sampling (App. G), matching the hand-crafted regex baseline. Classical search alone, A*+hAdd over the benchmark's PDDL model, solves every instance, 60 of 60, scored on the same instances as the full pipeline (Table 1). Across 10 IPC-2023 domains, all 46 action schemas, 44 induced and 2 reused from the schema library, reach component-$F_1 \geq 0.9$ against the PDDL ground truth with no domain-specific hyperparameters. Three domains were first reduced to STRIPS by dropping negative preconditions (App. S).

Isolating the mask. Matched columns show that the LLM is unnecessary for plan validity on these instances, but they do not isolate the mask. We

therefore hold the model (Qwen-1.5B), prompts, instances, and decoding configuration fixed, sample $K{=}8$ plans per instance, and vary only the grammar. Under a syntax-only action grammar, which admits any well-formed sequence of grounded action strings and is the guarantee a conventional constrained decoder provides, no sampled plan reaches the goal on either domain, 0 of 80 on Blocksworld and 0 of 80 on Mystery Blocksworld. All 160 samples parse, so every failure is semantic. Of these, 158 violate an action precondition, most often that a block is not on the table (39 per domain), not clear (14), or that the gripper is already full (12). The remaining two run to the horizon without reaching the goal. Under G_{env}, compiled here from a hand-coded Blocksworld transition model (§3.3), every sample reaches the goal, 80 of 80 on both domains, and the first sample is already valid on all 10 instances. The result also holds on Mystery Blocksworld, whose action names are obfuscated. The syntax-only control uses the same coarse grammar on both domains, so its Mystery result is not an independent obfuscation control. Across five jailbreak prompts at Qwen-2.5-7B, the coarse-syntax baseline admits no sound plan in 100 samples and G_{env} admits all 100.

Template sensitivity. This parse rate does not measure robust language understanding at 1.5B. PlanBench's instructions are templated. On Logistics, Qwen-1.5B parses 56.7% of goals from the base prompt and 100% with a canonicaliser and one in-context alias example (App. G). On 10 Blocksworld instances the model handles the original phrasing and a lexical paraphrase perfectly, while syntactic re-bracketing and article elision each drop it to zero. Evidence for an LLM contribution beyond templated prompts comes from the human-goal model comparison.

5.2 RQ2: what the LLM contributes to grounding

On human-annotated goals, goal grounding becomes the main source of variation. Its accuracy differs across models in distinct levels rather than continuously. By accept rate the nine systems (Table 2) fall into three performance bands, a descriptive grouping that pairwise exact McNemar tests with Cohen's h on the 251 paired rows then characterise. The **lower band**, at roughly 30%, holds the regex baseline and Qwen-1.5B ($p{=}0.81$). At this scale we detect no difference in aggregate acceptance from a hand-written parser. The **middle band**, at 40 to 44%, holds GPT-3.5, Qwen-2.5-7B, Llama-3.3-70B and GPT-4o-mini, and none of its six within-band pairs separates ($p{>}0.13$). The **higher band**, at 49 to 60%, holds Haiku, GPT-4o and Sonnet. Haiku separates from the other two ($p{<}10^{-3}$), which do not separate from each other ($p{=}0.17$). Adjacent bands separate at $p{<}0.05$ in every pairwise test, unadjusted for the 36 comparisons, and the effect at the middle-to-higher boundary is small ($h{=}0.10$ for GPT-4o-mini against Haiku). The widest separation is GPT-4o against regex at $p{=}1.6{\times}10^{-17}$ and $h{=}{-}0.52$ (App. I).

System	**IID** % [CI] $N{=}251$ per split	**OOD** % [CI] $N{=}251$ per split
Regex (alias table)*	31 [25, 37]	32 [26, 38]
Qwen-1.5B (greedy)	30 [24, 35]	42 [36, 48]§
Qwen-2.5-7B-Turbo†	42 [36, 48]	52 [46, 58]§
Llama-3.3-70B-Turbo†	41 [35, 47]	55 [49, 61]§
GPT-3.5-turbo	40 [34, 46]	50 [44, 56]§
GPT-4o-mini	**44** [38, 50]	**53** [47, 59]
Haiku 4.5‡	**49** [42, 55]	**57** [51, 63]
GPT-4o	**56** [50, 62]	**62** [56, 68]
Sonnet 4.6‡	**60** [54, 66]	**67** [61, 73]

Table 2: ALFWorld natural-language goal parse across models, accept rate with Wilson 95% CIs. Accept iff $F_1^{\text{norm}} \geq 0.9$ against the ground-truth goal conditions after the normalisation of App. S. Decoding is greedy for the local model and temperature 0.3 for cloud models, with no ground-truth-aware reranking. Horizontal rules separate the three performance bands discussed below. The OOD-minus-IID gap grows for every LLM (+6 to +14 percentage points, pp) while regex stays at 31 to 32%. *∼ 200-entry alias table, App. D. †Together.ai. ‡Anthropic. Haiku is single-seed because of rate limits. Sonnet and GPT-4o report the first of three seeds. §$N{=}255$ from a later version of the unseen split. Per-seed counts and mean F_1 in App. J. Frontier seed variance is $\leq$1.3pp.

Within one family, size matters, since Qwen-1.5B and Qwen-2.5-7B fall in different bands, but the larger open models do not reach the higher band of closed-source models of undisclosed size. Our systems vary in family, training data and size at once, so we do not attribute the ordering to any one of these. The bands track which parts of an instruction a system recovers. The lower band recovers only slot values stated explicitly in the instruction, the middle band recovers moderate paraphrase but misses implicit modifiers, and the higher band resolves those and most class confusions. The model gap is also larger out of distribution than in it, by

5 to 13pp, while the regex baseline remains at 31 to 32%. Because similar gaps appear across three model vendors, the result is consistent with frontier models recovering semantic relations that a fixed alias table cannot encode. The gap concentrates in the two families carrying implicit object-temperature semantics, Cool & Place and Heat & Place, and shrinks to +8.7pp where object and receptacle are both named explicitly (App. F).

Paraphrase control for memorisation. ALFWorld's goal strings have been public since 2020, so evaluated models may have been exposed to them, and the unseen split varies scenes while reusing goal templates. Recall of familiar strings is therefore an alternative explanation for the observed model advantage. We rewrote all 251 out-of-distribution goals as freshly generated paraphrases using Llama-3.3-70B, a third model family, holding object and location nouns verbatim and leaving prompt, scorer and threshold unchanged. If the advantage depended on the original surface strings, accuracy should fall.

It does not. GPT-4o scores 62.9% on the paraphrases against 62.2% on the originals, and Sonnet 68.5% against 67.3%, with no detected difference in either case (paired McNemar p=0.75 and p=0.55). Qwen-2.5-7B holds its position at 51.0%. The alias-table baseline, whose knowledge is the literal wording, falls from 31.9% to 0.8%, losing 78 instances and gaining none (p=6.6×10^{-24}). Sonnet outperforms regex on 171 of the 172 discordant paraphrased instances (p=5.8×10^{-50}), so the gap is wider on the rewritten text than on the original. Under a more aggressive synonym condition the LLMs decline moderately (Sonnet 57.8%) while regex remains low (17.1%). The largest Sonnet-over-regex gaps on the paraphrased text fall on exactly the implicit-modifier families, above +80pp on each of heating, cooling and cleaning. The baseline depends on the original surface form, while the frontier models do not show the same dependence. This does not establish that memorisation makes no contribution, since familiarity with the ALFRED ontology could survive a rewrite that holds object nouns fixed.

5.3 RQ3: Parse-layer correction mechanisms

Both correction layers improve grounding. All numbers below come from one uniform paired set of 114 unique ALFWorld instances, parsed by Claude Sonnet at T=0.3, so every increment shares a denominator. The stream holds 119 rows, five task ids recur, and we keep the last occurrence of each. Layer B is updated on the repeated descriptions during the run. Because correction operates on the parse, we report both parse acceptance and downstream execution. *Parse acceptance*, the share of parses at $F_1^{\text{norm}} \geq 0.9$ against the ground-truth goal slots, rises from 61.4% one-shot to 69.3% under Layer A and 75.4% with Layer B added, a gain of 14.0pp. *Task completion*, scored by execution in the environment, rises from 67.5% to 75.4% and then **80.7**%, a gain of **13.2**pp. On task completion Layer A corrects nine instances and degrades none (exact McNemar p=0.0039), and Layer B corrects a further six and degrades none (p=0.031). The parse-level increments carry the same significance (p=0.0039 and p=0.039). Layer B's separate increment is sensitive to the occurrence rule. Keeping the first occurrence instead gives $+5/-0$ (p=0.063) on task completion and $+7/-1$ (p=0.070) at parse level, while the composed gain over one-shot stays at $+15/-0$ or $+14/-0$ ($p \leq 1.2\times10^{-4}$) under either rule (App. H.1). We therefore treat the composed correction as the primary result and the Layer B increment on its own as suggestive. Without a B-only arm or an equal-call retry control we do not claim that the layers act independently or that the gain is specific to feedback rather than to added samples. Replicating both layers on GPT-4o lifts task completion from 67.2% to 75.6% over the 119-row stream, and the composed effect remains significant on the paired set (p=0.006).

The gain is concentrated rather than uniform. It falls on Examine in Light (+15pp) and Pick & Place (+15pp), the two families in which the model confuses near-synonymous object classes such as the two lamps above, and is neutral where failures lie in the planner rather than in the parse. That pattern is what targeted recovery of class confusion predicts, though it does not exclude a sampling effect that happens to be family-dependent (App. H.1).

5.4 RQ4: end-to-end success and planner credit

For end-to-end evaluation, we pass the LLM-parsed goal conditions to ALFWorld's TextWorld engine and execute action by action, reading success from the environment's own success signal (`won`). Across the 119 core-5 instances, with no filter on parse quality, the loop succeeds on **68.1**% (Wilson

E_6 planner	Success	vs expert
HandCoded TW expert	67.5%	—
Llama-3.3-70B, step-wise	36.8%	$+5/-40$
AGCP-induced planner	36.0%	$+0/-36$

Table 3: Three planners on *identical* parses: the same 114 unique Sonnet parses, with no filter on parse quality. The last column gives paired exact McNemar against the ALFWorld expert policy (p=7.9×10^{-8} and p=2.9×10^{-11}). Induced versus LLM-as-planner is $+16/-17$, p=1.0. Coverage variants over the filtered and full-6-type streams are in App. T.

[59, 76], App. T). The rate over all six task types is 56.6% of 143 instances, and the gap to core-5 reflects limits of the TextWorld library rather than of the pipeline.

Parse and policy contributions. That figure uses the expert policy provided with ALFWorld, so it measures a correct parse combined with a hand-written, schema-driven policy. To separate the two contributions we hold the parses fixed, using the same 114 unique Sonnet parses, and replace only the planner (Table 3). A planner induced entirely by our own pipeline reaches 36.0%. It uses schemas from goal-free stratified exploration, greedy search over the induced model, and generic recovery from admissible-command feedback, with no LLM in the loop and no ground truth. The expert policy strictly dominates it, winning 36 instances and losing none (p=2.9×10^{-11}). With parses held fixed, the expert policy outperforms the induced planner by roughly 31 points. The induced planner still succeeds in all five task types, and reaches 50.7% when the parse is correct. A frontier LLM in the same role does no better. Llama-3.3-70B, choosing actions step by step on the identical parses, reaches 36.8% against the induced planner's 36.0%, a difference we do not detect ($+16/-17$, p=1.0). Its success over the 119-row stream is 36.1% under both a 30- and an 80-step cap, so the cap does not bind in aggregate. The induced planner's failures locate the induction boundary. Of its 73 failures, 37 return no plan within the search budget. The inspected failures are consistent with schema incompleteness: one conjunction per action cannot express a placement precondition of the form *the receptacle is open or is not openable* (App. R).

Grammar induction without predicate deltas. SCI-ReDuce (§3.5) compresses action sequences and indexes macros by the recorded states, without per-action deltas or a schema. State is still read at decoding time from the symbolic simulator, so the comparison bounds the induction algorithm rather than the fully opaque regime. The raw grammar admits unbounded macro loops, and under it no sampled plan completed within the tested token budget on any of the 20 held-out instances. Applying the same horizon-unrolling G_{env} uses, with the horizon set to the reference plan length plus four, restores termination and yields 37.5% instance success with any of 8 samples, against 32.5% gold-plan coverage on the same splits. All 15 instances whose compiled language is non-empty are solved, and 118 of the 120 plans sampled on them reach the goal. The instance-level loss is therefore incompleteness of $L(G)$, with 62.5% of held-out instances failing to compile at all. This is a partial-coverage fallback, and its measured coverage is what bounds that claim (App. L).

6 Conclusion

Where the applicability mask compiles, it provides plan validity under the model it is compiled from. Varying only the mask, no sampled plan reaches the goal under a syntax-only grammar while every one reaches it under the applicability mask. Where the mask does not compile, classical search over the given planning model provides validity instead, and on the classical tasks we evaluate neither route needs an LLM to generate the plan. The LLM performs goal grounding, which varies across model families, improves under verifier feedback, and survives a paraphrase control under which a surface-matching baseline fails. The mask compiles on three of four PlanBench domains and schema-free induction fails on 62.5% of held-out instances, so grammar-constrained decoding provides semantic validity where the environment can be enumerated. This is also the regime in which classical planning is strongest.

7 Limitations

Scope of the guarantee. The framework carries validity only when the environment is symbolically observable and its reachable state space is enumerable within the state budget. That regime largely coincides with the one classical planning already solves, so the result is a credit assignment within grammar-constrained decoding rather than a claim about planning at large. Moving beyond it requires either a compilation method that scales beyond ex-

plicit state enumeration, or an induction method that recovers the type-dependent preconditions that §5.4 identifies as the dominant residual error.

ALFWorld scope. The ALFWorld model comparison (Table 2) tests edge E_4 (NL goal to structured atoms) in isolation, scored against ALFWorld's ground-truth goal conditions after alias normalisation. It measures parse quality alone. The closed-loop E_6 result (§5.4) does provide end-to-end task completion, but only on 5 of 7 task types. ALFWorld's HandCoded TW policy does not support Pick & Place with Movable Receptacle, and Pick Two & Place fails in the library itself, running to the step limit even on ground-truth parameters. Both are excluded from the filtered rates of App. T and from the 68.1% unconditional core-5 rate. That appendix also reports the unfiltered six-type rate of 56.6%. Broader coverage would require fixing ALFWorld's TextWorld policies, which is out of scope here.

Residual contamination risk. ALFWorld (Shridhar et al., 2021) and its ALFRED trajectory data have been public since 2020. Its version-2.1.1 task descriptions and its canonical seen and unseen validation splits are indexed by Common Crawl and appear verbatim in dozens of downstream repositories. The unseen split varies room layouts while reusing goal templates. The frontier models we evaluate may therefore have been exposed to these goal strings during pretraining, a concern documented broadly for popular benchmarks (Sainz et al., 2023; Golchin and Surdeanu, 2024; Balloccu et al., 2024). The paraphrase control of §5.2 shows that the model advantage does not depend on the original surface strings, while a surface-matching baseline does. It does not establish the absence of contamination. Because our paraphrases hold object and location nouns fixed by design, exposure to the ALFRED ontology, such as which objects are coolable, could still contribute. The stronger test is a contamination-controlled NL-to-PDDL benchmark with newly authored goals held out until evaluation. Planetarium (Zuo et al., 2025) is public and does not provide that control for the models studied here.

API non-determinism of cloud LLMs. Frontier-model cloud APIs offer no setting that makes their output exactly reproducible. OpenAI documents its `seed` parameter as best-effort and Anthropic has none. Our reported cloud-LLM numbers are therefore single-seed at `temperature` $= 0.3$. We bound the expected non-determinism noise empirically. The GPT-4o-mini 4-seed variance check at $N{=}251$ ALFWorld yields $\{110, 111, 111, 109\}$ correct, stdev $= 0.4$ pp. That is well below the $+25$ to 35 pp LLM-vs-regex effect we report, and below the Wilson 95% CI half-width at $N{=}251$.

Ethical Considerations

This work studies grammar-constrained decoding for symbolic planning tasks (Blocksworld, Mystery Blocksworld, Logistics, Depots, IPC-2023 STRIPS domains). It does not involve human subjects, demographically identifying data, or generated content about people. The benchmark environments (PlanBench (Valmeekam et al., 2023), IPC-2023 learning track, AutoPlanBench (Stein et al., 2025) and ALFWorld (Shridhar et al., 2021)) are public.

A possible misuse vector for any grammar-constrained decoding method is bypassing alignment-time safety constraints by re-projecting LLM logits onto an attacker-supplied grammar. Our jailbreak ablation shows the same content-independence. Under five jailbreak prompts the coarse-syntax baseline admits no sound plan in 100 samples and the mask admits all 100 (§5.1), so the mask carries soundness regardless of the prompt. It does not substitute for safety reinforcement learning from human feedback (RLHF) or for input filtering.

All code, induced schemas, generated plans, baselines and per-experiment logs are released at `https://github.com/nuuoe/AGCP` (App. M).

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, and 26 others. 2023. Do as I can, not as I say: Grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning (CoRL), PMLR*, volume 205, pages 287–318.

Simone Balloccu, Patrícia Schmidtová, Mateusz Lango, and Ondřej Dušek. 2024. Leak, cheat, repeat: Data contamination and evaluation malpractices in closed-source LLMs. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 67–93.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2024. Guiding LLMs the right way: Fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 3658–3673.

Blai Bonet and Héctor Geffner. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33.

Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences*, 2nd edition. Lawrence Erlbaum Associates.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2019. TextWorld: A learning environment for text-based games. In *Computer Games, Communications in Computer and Information Science*, pages 41–75. Springer.

Stephen N. Cresswell, Thomas L. McCluskey, and Margaret M. West. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2):195–213.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. 2025. XGrammar: Flexible and efficient structured generation engine for large language models. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 7.

Richard E. Fikes and Nils J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 10932–10952.

Shahriar Golchin and Mihai Surdeanu. 2024. Time travel in LLMs: Tracing data contamination in large language models. In *International Conference on Learning Representations (ICLR)*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 540 others. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 79081–79094.

Richard Howey, Derek Long, and Maria Fox. 2004. VAL: Automatic plan validation, continuous effects, and mixed initiative planning using PDDL. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 294–301.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, and Brian Ichter. 2023. Grounded decoding: Guiding text generation with grounded models for embodied agents. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 59636–59661.

Brendan Juba, Hai S. Le, and Roni Stern. 2021. Safe learning of lifted action models. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 379–389.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 22895–22907.

Linzhang Li, Yixin Dong, Guanjie Wang, Ziyi Xu, Alexander Jiang, and Tianqi Chen. 2026. XGrammar-2: Dynamic and efficient structured generation engine for agentic LLMs. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pages 1009–1022. Association for Computing Machinery.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Tianyi Ma and Parisa Kordjamshidi. 2026. CLAMP: Constrained decoding for vision-language embodied planning. *Preprint*, arXiv:2609.08602. Accepted to Findings of the Association for Computational Linguistics: EMNLP 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 46534–46594.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. PDDL: The planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

Meta AI. 2024. Llama 3.3 70B instruct.

Stephen H. Muggleton. 2025. ReDuce: Linear-time inductive compression using greedy folding. In *Proceedings of the 5th International Joint Conference on Learning and Reasoning (IJCLR 2025)*. In press.

Shaan Nagy, Timothy Zhou, Nadia Polikarpova, and Loris D'Antoni. 2026. ChopChop: A programmable framework for semantically constraining the output of language models. *Proceedings of the ACM on Programming Languages*, 10(POPL):1905–1932.

OpenAI. 2024. GPT-4o system card. *Preprint*, arXiv:2410.21276.

Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations (ICLR)*.

Qwen Team. 2024. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. 2023. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10776–10787.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9895–9901.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 8634–8652.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10737–10746.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*.

Katharina Stein, Daniel Fišer, Jörg Hoffmann, and Alexander Koller. 2025. Automating the generation of prompts for LLM-based action choice in PDDL planning. In *Proceedings of the 35th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 250–259.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. 2024. The 2023 International Planning Competition. *AI Magazine*, 45(2):280–296.

Together AI. 2024. Together AI inference API. Accessed July 2026.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Advances in Neural Information Processing Systems 36, Datasets and Benchmarks Track (NeurIPS)*, pages 38975–38987.

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. 2024. LLMs still can't plan; can LRMs? A preliminary evaluation of OpenAI's o1 on PlanBench. *Preprint*, arXiv:2409.13373.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*.

Brandon T. Willard and Rémi Louf. 2023. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*.

Edwin B. Wilson. 1927. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. 2023. Translating natural language to planning goals with large-language models. *Preprint*, arXiv:2302.05128.

Qiang Yang, Kangheng Wu, and Yunfei Jiang. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2–3):107–143.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Gwen Yidou-Weng, Edward Sun, Tianyi Ma, Metin Alp Dogan, Benjie Wang, Allen Peng, Guy Van den Broeck, and Yuchen Cui. 2026. Meta-Ctrl: Guaranteed plan generation by decoupling syntactic and semantic constraints. *Preprint*, arXiv:2608.22149.

Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael L. Littman, and Stephen H. Bach. 2025. Planetarium: A rigorous benchmark for translating text to structured planning languages. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 11223–11240.

A Notation summary

The notation used below is summarised here.

Symbol	Meaning
$\mathcal{P}, \mathcal{O}, k$	predicates, objects, max arity
$\mathcal{A}$	action set of the environment
s, s_0	state (set of ground atoms), initial state
$g, \hat{g}$	goal, and the goal parsed from language
$\Sigma, \hat{\Sigma}$	lifted action schemas, and those induced from rollouts
Λ	ground action strings, the decode alphabet
H	horizon, the maximum plan length
G_{env}	compiled applicability mask
$G_{\text{env}}^{A^\star}$	degenerate single-path A* mask
$L(G)$	language of grammar G
L_{exec}	sequences executable from s
E_1–E_6	pipeline edges (Fig. 1)
$\text{sig}(s)$	ground atoms holding in s
$\text{en}(M), \text{ex}(M)$	entry and exit class of macro M
F_1^{norm}	F_1 after alias and role normalisation
h	Cohen's effect size for proportions
K	samples per instance, or iterations

B By-construction properties

The following propositions formalise the two properties stated in §3.3. Both are relative to the induced model $\hat{\Sigma}$ and the parsed goal $\hat{g}$.

Proposition 1 (Applicability relative to the induced model)**.** *If compilation returns a non-empty language then every $\pi \in L(G_{env})$ is applicable in s_0 under $\hat{\Sigma}$.*

Proposition 2 (Goal-validity relative to the induced model)**.** *Every $\pi \in L(G_{env})$ reaches $\hat{g}$ under $\hat{\Sigma}$ within the horizon H.*

Both propositions follow directly from the bounded transition graph, and the LLM-independence property of §3.3 follows from Proposition 2 for any decoder with full support on the masked vocabulary. BFS over the transition graph of $\hat{\Sigma}$ enumerates only paths that are applicable in s_0 under $\hat{\Sigma}$, reach $\hat{g}$, and respect the horizon H, so the same holds of every string in $L(G_{\text{env}})$. The mask restricts the LLM to exactly that set, so any LLM with non-empty support on the masked tokens emits one of them.

These propositions concern only the induced model and parsed goal. They do not by themselves establish validity in the environment or satisfaction of the user's instruction. A string applicable under $\hat{\Sigma}$ is applicable in the environment only if $\hat{\Sigma}$ is sound on the transitions the mask admits. The replay gate of §3.2 is an empirical acceptance test for that condition, at a threshold we chose, and a schema can pass it while still mispredicting an admitted transition. Likewise, a string that reaches $\hat{g}$ satisfies the user's instruction only if the parse is correct, which is the LLM-dependent edge measured in §5.2. Where a compiler instead consults the exact environment transition function, the first condition holds by assumption for that path, and we mark which results use it.

SCI-ReDuce terminates because each refinement split partitions a finite set of macro occurrences and the compiler's BFS stops after a fixed number of visited states (App. P).

C Induction procedure

The induction procedure of §3.2, as implemented, has four steps. Each transition $(s, a(\vec{o}), s')$ is a pair of ground-atom sets recorded before and after a successful action during random rollouts.

- **Positional templating.** Every atom in s and s' is lifted by replacing each object that occurs in $\vec{o}$ with its argument position, at every position where it occurs, so a repeated argument such as $\texttt{stack}(b, b)$ yields templates for both positions. A lifted atom is kept if at least one argument was replaced by a placeholder or if it is nullary. Objects outside $\vec{o}$ that remain in a kept atom stay as constants. Lifted atoms keep their predicate symbol.
- **Intersection and causal effects.** The precondition is the intersection of the lifted pre-state atom sets over all transitions of a. An add effect is a lifted atom added in at least one transition and absent both before and after in none. A delete effect is a lifted atom deleted in at least one transition and present both before and after in none. An effect that some rollouts observe as already true is therefore retained.
- **Pre-minimality refinement.** For each candidate precondition atom, the procedure takes an observed pre-state in which the atom holds, removes that atom, and asks whether the action is still applicable. If it is, the atom is dropped. The question is answered by the domain's ground operator, obtained from the PDDL in the IPC and PlanBench domains, so refinement uses a counterfactual applicability query rather than passive rollouts. In a black-box environment the same query needs a simulator that can be set to an arbitrary state. The operator is consulted here and, for the corrected compile statistics of §3.3, as the

applicability oracle during BFS, as marked there.

- **Replay gate.** Induced schemas must replay at least 90% of held-out transitions. Schemas below the threshold are excluded from G_{env}. Passing raises confidence without certifying the model.

D Regex baseline construction

The regex baseline used throughout Table 2 and Fig. 3 (App. N) is a fixed pattern-based parser. It extracts the eight ALFWorld goal slots from raw natural-language task descriptions using regular-expression patterns and a fixed alias table of roughly 200 entries.

Goal conditions extracted. The eight conditions encode ALFWorld's goal conditions. Four name objects and receptacles, and four are boolean modifiers on the target object.

Goal condition	Meaning
Object target	the object to manipulate
Movable receptacle	the receptacle used to carry it
Destination receptacle	where it should be placed
Target appliance	the appliance to switch on
Slicing	whether the object must be sliced
Cooling	whether it must be chilled
Heating	whether it must be heated
Cleaning	whether it must be washed

Example alias entries.

- `mug` → `Mug`
- `spray bottle` → `SprayBottle`
- `chilled` → the cooling modifier
- `warm` → the heating modifier

Symmetric alias-vocabulary protocol. The same alias table is used by the regex parser at extraction time and by the F_1^{norm} scorer (App. S) at evaluation time. Asymmetric alias vocabularies between baselines are one of the five evaluation-inflation patterns we guard against (Table 8). No iterative tuning was performed against the test split, and the parser was frozen once it matched its training-split development numbers.

E SCI-ReDuce held-out coverage

Setup. SCI-ReDuce takes n Qwen-1.5B successful plans on Mystery Blocksworld, induces a state-class-indexed grammar G_R^S, and admits a held-out plan iff G_R^S recognises it verbatim (§3.5). The setting is harder than per-instance recognition because it tests whether the induced grammar generalises to unseen state classes. Fig. 2 plots the learning curve.

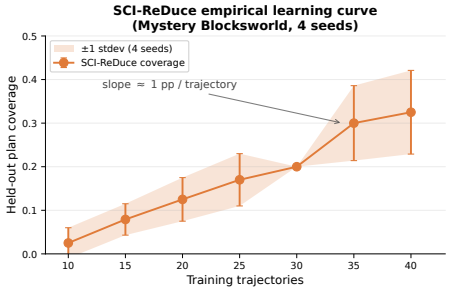


Figure 2: SCI-ReDuce held-out coverage on Mystery Blocksworld, 4 seeds. Coverage rises monotonically by about 1pp per trajectory, and the grammar reaches ∼85 macros at n=40. The zero variance at n=30 reflects identical counts across the four seeds. The pool is 50 trajectories, so the held-out set at n=30 is 20 plans, coverage moves in 5pp increments of one plan, and all four seeds admitted exactly 4 of 20 (per-seed values $[20, 20, 20, 20]\%$). Held-out size shrinks as training size grows, which is why the later points are noisier.

Coverage values:

Train	Held-out cov. (mean±sd, 4 seeds)	Macros
$n = 10$	$2.5\% \pm 3.5$pp	~ 20
$n = 15$	$7.9\% \pm 3.6$pp	~ 30
$n = 20$	$12.5\% \pm 5.0$pp	~ 45
$n = 25$	$17.0\% \pm 6.0$pp	~ 55
$n = 30$	$20.0\% \pm 0.0$pp	~ 65
$n = 35$	$30.0\% \pm 8.6$pp	~ 75
$n = 40$	$\mathbf{32.5}\% \pm 9.6$pp	~ 85

Table 4: SCI-ReDuce held-out coverage against the number of training trajectories, mean and standard deviation over 4 seeds, with the induced macro count.

The slope is about 1pp of coverage per training trajectory in this regime. The n=30 → n=35 increase from 20% → 30% reflects the point at which the induced rule set covers the canonical Mystery 2-block-and-3-block patterns.

F ALFWorld per-task-type results

We use ALFWorld's task-family names throughout. Column heads below abbreviate them as Examine in Light, Pick & Place, Clean & Place, Cool & Place, Heat & Place, Pick Two & Place, and Pick & Place with Movable Receptacle. The tables use the ALFWorld family definitions.

Pooled ALFWorld accept rates (Table 2) average across seven task families of differing difficulty. Tables 5 and 6 stratify the in-distribution and out-of-distribution splits by family, showing which families account for the pooled +30pp LLM-over-regex gap.

Two families account for most of the gap. Both require object-temperature modifiers that the regex alias table does not represent. Cool & Place rises from 29% under regex to 74% under Sonnet, a gain of 44.7pp, and Heat & Place from 41% to 79%, a gain of 38.2pp that reaches +64pp out of distribution. On simpler families the Sonnet-over-regex gap narrows, to +8.7pp on Pick & Place. Pick & Place with Movable Receptacle fails almost everywhere, with 10 of 14 system-by-split cells at $0/34$ in distribution and $0/33$ out of it. Only GPT-4o, at 6% on both splits, and Sonnet, at $11/34 = 32.4\%$ and $11/33 = 33.3\%$, achieve nonzero accuracy. The higher-performing models therefore gain mainly by recovering object-temperature modifiers that the regex was not designed to parse. This result is not evidence of broad structural superiority across all task families.

G Robustness and failure modes

NL paraphrase variation. For $N{=}10$ Blocksworld instances we generate three paraphrases of each natural-language goal. Preposition substitution gives *"X is stacked above Y"*, syntactic re-bracketing gives *"Y has X on top"*, and article elision gives *"block X"*. Qwen-1.5B parses the original and the substitution perfectly, $10/10$ each, and drops to $0/10$ on both re-bracketing and elision. Parsing survives lexical substitution that preserves the syntactic skeleton, while re-bracketing or article elision reduce the parse rate to zero. This sensitivity is specific to the templates used here and is consistent with the model comparison (§5.2). On human-annotated ALFWorld goals, which are paraphrase-rich by construction, Qwen-1.5B is indistinguishable from regex (30% vs 31% at $N{=}251$), every model from GPT-3.5 upward separates from regex (App. I), and the widest margin is GPT-4o's (paired McNemar $p = 1.6 \times 10^{-17}$). In Table 7 a failed parse counts as a failure.

Distractor sentences. We place distractor sentences around the natural-language goal on the same ten Blocksworld instances. Qwen-1.5B parses $10/10$ when the distractor follows the goal and $8/10$ when the goal is surrounded by distractors. A leading distractor is worse. The model treats the distractor as the directive in $5/10$ cases for the hard distractor and $0/10$ cases for the easy distractor.

Logistics failure modes. Logistics is the only domain whose parse rate is below 100%, and two prompt components account for the difference. Both failures are surface-form mismatches in how the model names objects. First, Qwen-1.5B outputs long object identifiers such as `package_0` and `location_1_0`, whereas the planner expects the short aliases `p0` and `l1-0`. This mismatch occurs in 10 of the 13 failing instances. A deterministic canonicaliser maps each verbose name to the domain's short alias for the same entity, so the error is a naming mismatch rather than a failure to identify the object or location, and the canonicaliser removes it in all ten. Five of them then pass, moving the domain from 56.7% to 73.3% (**+canon**). The other five also show the second fault, in which the model drops the second coordinate of a two-digit location name. That fault occurs in 8 of the 30 instances and is exactly the set still failing after canonicalisation. A Logistics-specific in-context example and an instruction to preserve both location digits resolve all eight and reach 100% (**+alias**). Both components act on the prompt, leaving the model and its weights untouched. Blocksworld, Mystery Blocksworld and Depots use single-character argument conventions under which neither mismatch arises.

PlanBench T1 NL-parse per-domain.

Domain ($N{=}30$)	**Regex**	**LLM+v.**	**+canon**	**+alias**
Blocksworld	100%	100%	100%	**100%**
Mystery Blocksworld	100%	100%	100%	**100%**
Logistics	100%	56.7%	73.3%	**100%**
Depots	100%	100%	100%	**100%**
Pooled	**100%**	89.2%	93.3%	**100%**

Table 7: NL-goal → structured-atom parse on PlanBench's templated NL (e.g. "the red block is on top of the orange block"), Qwen-1.5B-Instruct under JSON-schema mask, $N{=}30$ per domain, accept iff $F_1 \geq 0.9$ against the PDDL ground-truth atom set. The LLM columns sample four candidates at temperature 0.4, select among them by a verifier scored against the ground-truth atoms, and re-prompt up to twice with structural feedback when that score falls below 0.5. The selection uses the labels, so these columns are an upper bound on a label-free parse. An instance whose candidates all fail counts as a failure. The ALFWorld experiments use none of this. The regex column also reaches $120/120 = 100\%$ on this benchmark, so on templated NL the LLM only matches it. **LLM+v.**, the LLM parse with the verifier, is the NL parse under the corrected goal extraction of Table 8. **+canon** adds deterministic mapping from verbose object names to the domain's short aliases. **+alias** also adds a Logistics-specific in-context example and an instruction to preserve both location digits. Under the uncorrected extraction the same column reads 46.7/40.0/30.0/3.3%, pooled 30.0%.

System (IID)	Examine (N=29)	Place (N=46)	Clean (N=37)	Cool (N=38)	Heat (N=34)	Two (N=33)	Movable (N=34)
regex	31	41	24	29	41	45	**0**
Qwen-1.5B (greedy)	34	30	24	47	26	42	**0**
GPT-3.5	45	39	32	58	65	42	**0**
Qwen-2.5-7B-Turbo	41	43	32	58	71	48	**0**
Llama-3.3-70B-Turbo	41	33	41	61	68	45	**0**
GPT-4o-mini	31	46	38	63	74	52	**0**
Haiku 4.5	45	52	43	66	76	55	**0**
GPT-4o	52	59	59	66	74	76	6
Sonnet 4.6	**62**	50	**51**	**74**	**79**	76	**32**
Δ(best LLM − regex) IID	+31	+18	+35	**+45**	**+38**	+30	+32
Δ(best LLM − regex) OOD	+30	+13	+36	**+39**	**+64**	+38	+33

Table 5: ALFWorld IID per-task-type accept rate (%, $F_1 \geq 0.9$), stratified by task family. Row grouping matches Table 2.

System (OOD)	Examine (N=50)	Place (N=30)	Clean (N=36)	Cool (N=36)	Heat (N=42)	Two (N=24)	Movable (N=33)
regex	32	57	42	39	21	38	**0**
GPT-4o-mini	44	60	56	**78**	**86**	42	**0**
Claude Haiku 4.5	52	**70**	56	75	83	58	**0**
GPT-4o	54	**70**	69	**78**	83	**75**	6
Claude Sonnet 4.6	**62**	57	**78**	**78**	**86**	**75**	**33**

Table 6: ALFWorld OOD per-task-type accept rate (%, $F_1 \geq 0.9$), stratified by task family. Same column keys as Table 5. Per-task-type breakdown not collected for Qwen-1.5B, Qwen-2.5-7B, Llama-3.3-70B, and GPT-3.5 (overall OOD accept rates for these systems are reported in Table 2). On Pick & Place with Movable Receptacle (0% regex, 0% for all non-frontier LLMs), only GPT-4o (6%) and Claude Sonnet ($11/33 = 33\%$) achieve nonzero accuracy among the systems with per-task results. Accuracy on this family stays low in both splits.

H Evaluation-inflation patterns

The five patterns of §4, with where each appears and its measured impact. The three inherited patterns are of wider interest because the components that carry them recur in LLM-planning evaluations.

H.1 Parse-layer correction details

Layer A per-task-type breakdown. The gain from K=1 to K=3 on the 114-unique paired core-5 pool concentrates in Examine in Light (+15pp) and Pick & Place (+15pp), where the model confuses canonical class names (`FloorLamp` for `DeskLamp`, `Chair` for `ArmChair`) most often. Clean & Place, Cool & Place and Heat & Place are neutral to slightly positive, because failures there lie in the planner rather than in the parse (temperature modifiers correctly parsed one-shot in ≥95% of cases). The 9−0 paired dominance therefore reflects a targeted recovery of parse-class confusion.

Denominator and occurrence rule. The parsed stream has 119 rows but 114 unique task ids, five being duplicates. All reported increments use the 114 unique ids, keeping the *last* occurrence of each duplicate. Composing increments across the two denominators would overstate the gain by 1.1pp. The runner builds its candidate list from every source row, so Layer B's cache is also updated on the duplicated descriptions, and deduplicating the scores does not remove that repeated exposure. Keeping the *first* occurrence instead changes one task, a Pick & Place instance that puts a watch on a side table, which fails on its first pass under Layer A+B and succeeds on its second:

Outcome, rule	1-shot	A	A+B	B vs A (p)
Completion, last	77	86	92	+6/−0 (0.031)
Completion, first	77	86	91	+5/−0 (0.063)
Parse ≥ 0.9, last	70	79	86	+8/−1 (0.039)
Parse ≥ 0.9, first	70	79	85	+7/−1 (0.070)

The composed A+B versus one-shot contrast on task completion is +15/−0 under the last-occurrence rule (p=6.1×10^{-5}) and +14/−0 under the first-occurrence rule (p=1.2×10^{-4}). Whether the second pass succeeds because of the cache or because of sampling cannot be told from this run. A unique-task online result would require the candidate stream to be deduplicated before the run

Pattern	Where it appears	Measured impact
Regex extraction confounded by the in-context-learning (ICL) example	A non-greedy regular-expression search over PlanBench prompts matched the goal sentence of the in-context *example* ("My goal is to have that…") rather than that of the test instance	NL parse increases from 30% to 89% pooled across 4 domains, and per domain by 3–30×. Affects any benchmark using ICL prompts where the metric-extracted span can match the ICL example.
LLM+P empty-plan counting error	Qwen-1.5B often emits a PDDL problem reusing init atoms as goal atoms, so the planner returns the 0-action empty plan and the metric counts it as a solution	LLM+P Depots solved instances: $12/30$ is reported as 40%. With the corrected non-empty-plan criterion it is $\mathbf{0/30}$ at Qwen-1.5B. The criterion is exact here because no PlanBench T1 instance is solved by the empty plan. In general the plan must be validated against the original problem. The check matters whenever an LLM writes the problem file.
Unreported ground-truth fallback	A closed-loop evaluation that substitutes the ground-truth goal atoms whenever the environment-facts verifier rejects the parse makes *every* downstream metric agree with the ground truth, even when the LLM produced nothing parseable	On Mystery 2–11, the verifier rejected the LLM parse on $4/10$ instances and the evaluation substituted the ground truth without exposing the parse failure, reporting $10/10$ "end-to-end". With an empty parse substituted instead, and the ICL-confound extraction fixed, the corrected LLM-driven success rate is $10/10$.
Best-of-K with ground-truth-aware reranking	A verifier-gated sampler that scores each of K samples by F_1 against the ground truth and keeps the highest is test-label leakage, since the LLM receives K attempts and an oracle selector	ALFWorld N=35, preliminary run: Qwen-1.5B at $15/35 = 43\%$ vs regex $9/35 = 26\%$. With greedy decoding (no reranking) at the same alias table, $13/35 = 37\%$ vs regex 46%.
Asymmetric alias table	The LLM prompt includes in-context aliases (e.g. card to CC) absent from the original 111-entry regex alias table	ALFWorld N=35: the asymmetric table gives regex $9/35 = 26\%$ and the merged $\sim$ 200-alias table gives $16/35 = 46\%$. LLM and baseline must share vocabulary at extraction time as well as at scoring.

Table 8: Five evaluation-inflation patterns and their measured impact. Three are inherited from evaluation components of earlier LLM-planning pipelines and two are specific to our own pipeline. The reported numbers and the released code exclude all five. The exclusion of the fourth pattern applies to the ALFWorld experiments and the closed-loop pipeline. The PlanBench parse column of Table 7 retains verifier selection and is labelled there as an upper bound. They bias in different directions. Four inflate the LLM-versus-baseline gap, while the ICL-confound extraction deflated the LLM's parse rate.

and every configuration to be rerun on the same ordered tasks. This experiment does not provide that control.

Sampling noise. Two observations weigh against sampling noise as the sole source of the Layer-A gain, without excluding it. Layer A corrects nine instances and reverses none in the paired comparison, and two independent Sonnet runs agree on 113 of the 114 acceptance decisions while giving identical task-completion totals. An equal-call retry control without feedback was not run.

Verifier inputs. At each iteration the environment-facts check consults only the affordances the TextWorld observer reports, which are the objects, receptacles and toggleable appliances in the current scene.

Sample Layer B mechanism traces. Of the six tasks that Layer A+B completes and Layer A alone does not, the reported run shows a prior at work in five and a hint in one:

- *Box and desk lamp* (Examine in Light). Layer A alone parsed the appliance as a floor lamp and failed. Every past success of this type had named the desk lamp, so the prior did, and the one-shot parse under Layer B named it too. No hint was involved.
- *Cup and microwave* (Cool & Place). Layer A alone filled the movable-receptacle slot with the microwave and the placement slot with the fridge. The prior recorded the movable-receptacle slot as empty in every past success, and the corrected parse left it empty and placed the cup in the microwave.
- *Watch and side table* (Pick & Place). Layer A alone tried a dining table and then a coffee table. The prompt held the hint that *on a table* maps to the side table rather than the coffee table, next to a conflicting hint for a CD, and the parse named the side table at the first attempt. This is the second occurrence of a duplicated task (App. H.1).

The hint filter keeps the first sentence of a proposed rule, capped at 240 characters, rejects it if the model marks it as not a parse lesson or if it contains any of twelve execution phrases such as *navigate to*, *before placing* or *while holding*, and requires the phrase *should be* or *set*. Keyword filtering does not guarantee semantic separation, so we inspected the survivors. All 19 retained hints in the reported run are slot-level corrections. One is redundant, naming the same value on both sides, and none names an action. The GPT-4o replication retained 29.

Iteration layers and cost. Action-layer iteration such as Reflexion (Shinn et al., 2023) retries whole trajectories after execution. The two parse-layer mechanisms here act before execution. Their measured call budget on the 119-row stream is 119 LLM calls one-shot, 135 under Layer A, and 160 under Layer A+B, of which 135 are parsing calls and 25 are rule-generation calls. Layer B also lengthens each prompt with priors and hints, a token cost we do not report. Both increments are computed on the same 114-unique paired set, so they compose to $67.5\% \rightarrow 75.4\% \rightarrow 80.7\%$. We do not compare the costs or gains with action-layer methods, whose evaluation sets, budgets and execution policies differ.

I Pairwise McNemar matrices

The main text groups the nine systems into three performance bands (Table 2). The full 9×9 (IID) and 7×7 (OOD) symmetric matrices below tabulate every pair, with two-sided exact binomial p-value and Cohen's h effect size. All tests pair rows, each a task-and-description trial (App. S), on the rows common to both systems, $N{=}251$ unless stated. Rows from the same task are treated as independent, so within-task clustering is not modelled, and the p-values are unadjusted for the number of comparisons. A task-clustered bootstrap (5,000 resamples of the 85 out-of-distribution task ids, rows kept together) gives 95% intervals for the row-rate difference of $[+0.27, +0.45]$ for Sonnet minus regex on the original goals, $[+0.23, +0.38]$ for GPT-4o minus regex, $[+0.60, +0.75]$ for Sonnet minus regex on the paraphrases, $[-0.02, +0.04]$ for Sonnet on paraphrases minus originals, and $[-0.11, +0.01]$ for GPT-4o minus Sonnet. The row-level conclusions of §5.2 hold under clustering. Point estimates remain row-level.

J Seed variance

The caption of Table 2 summarises seed-stability bounds for the three cloud LLMs on which multi-seed runs were feasible under provider rate limits (the Anthropic organisation-tier limit of 50 requests per minute precluded multi-seed runs on Haiku-4.5). Per-seed values are in Table 11.
All seed stdevs are ≤ 1.3pp, well below the $\sim$ 8–15pp band gaps in Table 2, so single-seed versus multi-seed reporting does not alter the band structure. The two out-of-distribution denominators differ because a later version of the unseen split contains four more instances than the one used for the first seed. Both are reported, and the four-instance difference leaves the confidence intervals overlapping.

K Additional probes

These probes examine cross-task validity, predicate discovery and sensitivity to the evaluation protocol and model choice.

PlanBench cross-task validity (local, Qwen-1.5B, $N{=}5$ per cell).

Task	BW	Myst.	Log.	Dep.
T1 plan-generation	100%	100%	100%	100%
T4 plan-reuse	100%	100%	—	—
T5 plan-generalization	100%	—[a]	—	—
T6 replanning	100%	100%	—	—
T7 plan-execution	100%	100%	—	—
T8_2 full-to-partial	100%	—	—	—
T8_3 partial-to-full	100%	—	—	—

[a] PlanBench does not provide T5 prompts for Mystery. Depots prompts are only available for T1. "—" marks cells we do not probe. Every cell we test scores 5/5. These small cells are consistent with the compiled grammar, here the single-plan A* grammar over the benchmark model (Table 13), producing goal-reaching plans regardless of the task-specific prompt directive (plan-gen, plan-reuse, plan-generalization, replanning, plan-execution, goal-shuffling). The framework's correctness guarantee is task-agnostic because G_{env} is built from the environment's transition function rather than the task prompt.

Predicate discovery, leakage-controlled evaluation. Predicate discovery from raw environment states is evaluated with a leakage-controlled verifier protocol. We require at least 3 distinct predicate names, at least 60% of them to be action-relevant, and a non-Blocksworld example (an abstract warehouse domain) so that the canonical Blocksworld

	regex	Q1.5B	GPT3.5	Q7B	L70B	4o-mini	Haiku	GPT4o	Sonnet
regex	–	0.03	0.20	0.24	0.22	0.27	0.37	0.52	0.60
Q1.5B	0.81	–	0.23	0.27	0.24	0.30	0.39	0.55	0.63
GPT3.5	$<10^{-3***}$	$<10^{-3***}$	–	0.04	0.02	0.07	0.17	0.32	0.40
Q7B	$<10^{-3***}$	$<10^{-3***}$	0.47	–	0.02	0.03	0.13	0.28	0.36
L70B	0.001^{**}	0.001^{**}	0.87	0.72	–	0.06	0.15	0.30	0.38
4o-mini	$<10^{-3***}$	$<10^{-3***}$	0.14	0.52	0.28	–	0.10	0.25	0.33
Haiku	$<10^{-6***}$	$<10^{-6***}$	$<10^{-3***}$	0.002^{**}	$<10^{-3***}$	0.017^{*}	–	0.15	0.23
GPT4o	$<10^{-9***}$	$<10^{-9***}$	$<10^{-6***}$	$<10^{-9***}$	$<10^{-6***}$	$<10^{-6***}$	$<10^{-3***}$	–	0.08
Sonnet	$<10^{-9***}$	$<10^{-9***}$	$<10^{-9***}$	$<10^{-6***}$	$<10^{-6***}$	$<10^{-6***}$	$<10^{-3***}$	0.17	–

Table 9: Pairwise exact McNemar binomial tests on ALFWorld IID (N=251 matched, threshold $F_1 \geq 0.9$). **Lower triangle**: two-sided p-value with significance marker ($^{***}p < 10^{-3}$, $^{**}p < 0.01$, $^{*}p < 0.05$). **Upper triangle**: |Cohen's h| (conventional thresholds: 0.2 small, 0.5 medium, 0.8 large). Row/column order matches the accept-rate order in Table 2. Band boundaries are visible as blocks of significance. Regex and Q1.5B form the lower band, with no within-band difference ($p = 0.81$). GPT3.5 through 4o-mini form the middle band ($p \in [0.14, 0.87]$ within). Haiku, GPT4o and Sonnet form the higher band, which is *not* homogeneous, since Haiku separates from both others, while GPT4o and Sonnet do not separate.

	regex	GPT3.5	Q7B	4o-mini	Haiku	GPT4o	Sonnet
regex	–	0.37	0.42	0.44	0.51	0.62	0.72
GPT3.5	$<10^{-6***}$	–	0.04	0.06	0.14	0.24	0.35
Q7B	$<10^{-9***}$	0.56	–	0.02	0.09	0.19	0.30
4o-mini	$<10^{-6***}$	0.29	0.88	–	0.07	0.18	0.29
Haiku	$<10^{-9***}$	0.008^{**}	0.10	0.08	–	0.11	0.21
GPT4o	$<10^{-9***}$	$<10^{-3***}$	$<10^{-6***}$	$<10^{-3***}$	0.019^{*}	–	0.11
Sonnet	$<10^{-9***}$	$<10^{-6***}$	$<10^{-6***}$	$<10^{-3***}$	$<10^{-3***}$	0.07	–

Table 10: Pairwise exact McNemar binomial tests on ALFWorld OOD (threshold $F_1 \geq 0.9$, N=251 matched rows, except that GPT-3.5 and Qwen-7B were run on the 255-row version of the unseen split, so their mutual test uses 255 rows and their tests against other systems the 251 shared rows). Same key as Table 9. Pairwise OOD McNemar values for Qwen-1.5B and Llama-3.3-70B not collected (overall OOD rates appear in Table 2). The OOD band structure largely mirrors IID. Regex stays in its own band ($p < 10^{-6}$ vs. every LLM), and the GPT3.5–4o-mini band remains statistically homogeneous.

predicates `handempty/on/ontable/clear` do not appear in-context. Under these conditions, Qwen-1.5B at N=15 seeds achieves $\mathbf{11/15 = 73.3\%}$ (Wilson 95% CI $[48\%, 89\%]$). At this scale, the task measures structured output following from an example rather than independent predicate discovery. We did not run the same leakage-controlled evaluation at frontier scale. The protocol above prevents a verifier failure mode in which 1-predicate proposals score F_1=1.0 whenever the in-context example lists the canonical predicate vocabulary. This correction is separate from the five patterns of Table 8.

Further empirical probes. Additional results in the release (App. M) include: AutoPlanBench (APB) breadth scan ($63/96 = 65.6\%$ on 32 APB domains at Qwen-1.5B), a 7-step ablation series, adversarial mask robustness (from $0/100$ to $100/100$ sound plans across 5 jailbreak prompts at Qwen-2.5-7B), the non-PDDL environment interface (ALFWorld via TextWorld fact extraction), and a cross-domain LLM-hint reuse probe in which Qwen-1.5B matches the signature-subset baseline.

Multi-LLM PlanBench extension. Local Qwen-1.5B anchors the PlanBench T1 matrix at 100% across all four domains (Table 1). The LLM-independence property predicts near-100% performance for the broader $4 \times 4 \times 8$ matrix (Qwen-1.5B / Qwen-2.5-7B / Llama-3.1-8B / Mistral-7B-v0.3 $\times$ T1–T8 $\times$ Blocksworld / Mystery / Logistics / Depots), because G_{env} carries plan validity regardless of LLM family. We have not run this matrix, so the prediction is untested. The scripts are part of the code release.

L Schema-free induction fallback

The rollout-based induction claim rests on symbolic observability. SCI-ReDuce (§3.5) compresses successful action sequences and indexes the resulting macros by the recorded states, without per-

Model	IID seeds	IID stdev	OOD seeds	OOD stdev
GPT-4o-mini	$\{110, 111, 111, 109\}$	0.4pp	$\{134\}$ (single)	–
GPT-4o	$\{141, 136, 142\}$	1.3pp	$\{156, 158, 158\}$	0.1pp
Claude Sonnet 4.6	$\{151, 150, 149\}$	0.4pp	$\{169, 173, 173\}$	0.3pp

Table 11: Per-seed accept counts. IID denominator $N{=}251$ throughout. OOD denominator $N{=}251$ for first cell (GPT-4o seed1, Sonnet seed1), $N{=}255$ for subsequent seeds (a later version of the unseen split adds 4 instances). Stdev is point-estimate sample stdev over the listed seed counts.

action deltas or a schema. We measure its end-to-end coverage on held-out Mystery Blocksworld, over the same splits as the coverage curve of App. E.

Decoding under the *raw* induced grammar admits unbounded macro loops, and no sampled plan completed within the 384-token budget, on $0/20$ held-out instances over 2 seeds and $0/28$ sampled plans. Applying the same horizon-unrolling that G_{env} uses, with the horizon set to the held-out reference plan length plus four, restores termination. Counting an instance as solved when any of its 8 samples reaches the goal, success is $15/40 = 37.5\%$ over 4 seeds, Wilson $[24, 53]$, against $13/40 = 32.5\%$ gold-plan coverage on the same splits. Every instance whose compiled language is non-empty is solved $(15/15)$, and 118 of the 120 plans sampled on them reach the goal. The two failing samples are not classified further. Instance success slightly *exceeds* gold-plan coverage because the grammar sometimes admits a different valid plan. Every instance-level failure is therefore a compile failure, with $25/40 = 62.5\%$ of held-out instances yielding an empty language. Coverage therefore bounds this fallback, and the schema-free induction regime is where the framework is weakest.

M Reproducibility

The release contains the experiment scripts, the saved outputs used for the reported analyses, and a script that checks the reported numbers against those outputs.

Statistics. We use Wilson 95% confidence intervals for binary outcomes, exact McNemar binomial tests for paired outcomes, and Cohen's h for effect sizes on proportions. Random seeds are fixed and documented per experiment in the code release.

N Parse accuracy across models

Figure 3 plots the accept rates of Table 2 against system, with both splits on one axis. The plot shows two patterns that the table does not. Within each band the accept rate is nearly constant and it rises between bands, and the seen and unseen series diverge as accuracy rises, since every LLM gains out of distribution while the regex baseline does not.

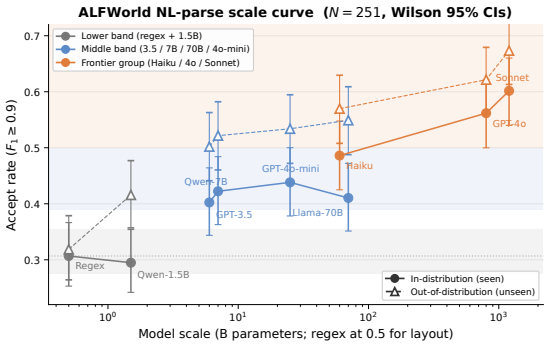


Figure 3: ALFWorld NL-parse accept rate ($F_1 \geq 0.9$) across models with Wilson 95% CIs, $N{=}251$ per split except the cells marked in Table 2 at $N{=}255$. The regex baseline is the dotted line. Filled circles are the seen split and open squares the unseen split. Every LLM gains out of distribution and the regex baseline does not.

O Cross-domain schema reuse (E_5)

Many domains share structurally identical operators. A *move-agent* action with one location-typed source and one location-typed target recurs across Logistics, Depots and several IPC suites, so repeating the ∼500-rollout induction for each domain duplicates work. A schema library stores replay-tested lifted schemas indexed by their (arity, predicate-signature) tuple. For a new action a' we enumerate library entries whose signature is a subset of the pre/add/del predicate multiset observed for a', treat the match as a candidate model, and verify it by held-out environment replay at $F_1 \geq 0.9$. On success we mark a' as reused and skip induction. On failure we fall back to fresh intersection-statistics induction, so a reused schema never displaces a domain-specific one without passing the replay test.

P SCI-ReDuce algorithm

The schema-then-CFG pipeline of §3.2 assumes symbolic transitions $(s, a(\vec{o}), s')$ that record predicate-level state deltas. SCI-ReDuce relaxes that requirement at induction time. It compresses successful action sequences alone and then indexes each macro by the recorded states at which it was observed, so induction needs recorded states but no per-action deltas and no schema. Decode-time admission still consults state.

Notation. Let Λ be the set of ground action strings and $\mathcal{P}$ the finite set of ground atoms observed in any recorded state. For a trajectory $\tau = \langle (s_0, a_1, s_1), \ldots, (s_{n-1}, a_n, s_n) \rangle$ write $w(\tau) = a_1 \cdots a_n \in \Lambda^*$ for its *action word*. Abstract a state by the set of ground atoms that hold in it, $\mathrm{sig}(s) = s \cap \mathcal{P}$, which in the implementation is s itself. A *state class* is any $c \subseteq \mathcal{P}$ and $s \models c$ iff $c \subseteq \mathrm{sig}(s)$. Since $\mathcal{P}$ is finite there are at most $2^{|\mathcal{P}|}$ classes, which is what keeps the compiled grammar finite.

A *macro* $M \in \Lambda^*$ is a right-hand side introduced by compression. Writing $\mathrm{Occ}(M) = \{(s, s') : M \text{ was observed to run from } s \text{ and yield } s'\}$, define its entry and exit classes by intersection, exactly as E_1 intersects candidate preconditions,

$$\mathrm{en}(M) = \bigcap_{(s,s') \in \mathrm{Occ}(M)} \mathrm{sig}(s),$$

$$\mathrm{ex}(M) = \bigcap_{(s,s') \in \mathrm{Occ}(M)} \mathrm{sig}(s').$$

Adapted compression. Algorithm 1 adapts the alternating repetition and folding operations of ReDuce (Muggleton, 2025) so that this appendix is self-contained. Our variant uses positive closure, so StarReplace rewrites a maximal run a^k with $k \geq 2$ as a^+, one or more repetitions, whereas ReDuce uses Kleene star and admits zero repetitions. MaxFreqPair returns an adjacent pair of symbols of maximum frequency in the grammar bodies. The procedure is an *inductive* compression in the sense of Muggleton (2025), meaning the encoding is shorter than the input while the repeats generalise it. Our variant rescans the grammar bodies on every round, so it does not retain the linear running time of the original.

State-class indexing (this work). Plain ReDuce treats sequences as symbol-only, so a macro may be inapplicable at the current state. Algorithm 2 admits a macro in a concrete state that entails its entry class and emits right-linear productions $N_s \to M \,.\, N_{s'}$ over the concrete states its BFS reaches. Proposition 3 reads these productions at the level of classes. Three modifications distinguish this from the original. Repeats use positive closure in place of Kleene star, the productions are indexed by state class, and the output is emitted as EBNF for XGrammar where the original emits H_2^2 Prolog clauses.

Algorithm 1 Compression procedure adapted from ReDuce (Muggleton, 2025)

Require: initial grammar G_0 with one body per action word

$G \leftarrow \textsc{StarReplace}(G_0)$
while some adjacent pair occurs at least twice in G **do**
  $xy \leftarrow \textsc{MaxFreqPair}(G)$
  introduce a fresh nonterminal $N \to xy$
  replace every occurrence of xy in the existing production bodies of G, excluding the new definition, by N
  $G \leftarrow \textsc{StarReplace}(G)$
end while
return G

Refinement is per macro. A class is split whenever two occurrences of the *same* macro in it lead to different exit signatures, choosing the atom whose presence best separates the exits. Each split partitions a finite occurrence set and removes the chosen atom from further consideration, so a macro with k occurrences is refined at most $k - 1$ times and to depth at most $|\mathcal{P}|$. The implementation records the concrete exit state of each occurrence rather than an exit class, and the compiler's BFS from the concrete s_0 follows an edge whenever the current state entails its entry class and moves to that recorded exit state. The BFS stops after a fixed number of visited states rather than at a depth, and the plan horizon is imposed afterwards by unrolling the emitted grammar over pairs of class and remaining budget, the construction G_{env} uses (App. L). The emitted language is therefore sound only if a macro admitted from any state entailing its class reaches a state of the recorded exit's class, which is the assumption of Proposition 3 and is not checked by the compiler.

Proposition 3 (Soundness relative to the abstraction)**.** *Admit a macro M at state s only when $\mathrm{en}(M) \subseteq \mathrm{sig}(s)$. If, for every macro, both exe-*

Algorithm 2 SCI-REDUCE, state-class-indexed grammar construction

Require: trajectories T with recorded states, initial state s_0, goal test, visited-state cap D

$G \leftarrow \textsc{ReDuce}(\{\, w(\tau) : \tau \in T \,\})$, plus every single action as a length-one macro
for all macros M in G **do**
  $\mathrm{Occ}(M) \leftarrow$ all (s, s') at which M occurs in T
  $\mathrm{en}(M) \leftarrow \bigcap_{(s,\cdot)\in \mathrm{Occ}(M)} \mathrm{sig}(s)$
  while two occurrences of M in one class differ in $\mathrm{sig}(s')$: split the class on an atom separating them
end for
$E \leftarrow \{\, (\mathrm{en}(M), M, s') : M \in G,\ (s, s') \in \mathrm{Occ}(M) \,\}$
BFS from s_0 over concrete states until D states are visited: from s follow each edge with $\mathrm{en}(M) \subseteq \mathrm{sig}(s)$ to its recorded s'
emit $N_s \to M\,.\,N_{s'}$ per traversed edge, $N_s \to \varepsilon$ per visited goal state, start symbol N_{s_0}
return EBNF encoding, empty if no goal state was visited

cutability from a state and the class of the resulting state depend only on the entry state's class, then every string of $L(G)$ is executable from s_0, that is $L(G) \subseteq L_{\text{exec}}$.

Proof. Induct on the number of macros emitted. The start symbol is N_{c_0} with $c_0 \subseteq \mathrm{sig}(s_0)$, so $s_0 \models c_0$. Assume the derivation has reached N_c in state s with $s \models c$. A production $N_c \to M\,.\,N_{c'}$ applies only if $\mathrm{en}(M) \subseteq \mathrm{sig}(s)$, and M ran from every observed state of that class, so by the assumption M runs from s and yields some s'' with $\mathrm{ex}(M) \subseteq \mathrm{sig}(s'')$, hence $s'' \models c'$. The invariant is preserved and every emitted macro executes. □

The assumption defines both the benefit and the limitation of this abstraction. An entry class is the intersection of the observed entry states, so it records only the atoms those states share. Two states that both entail c can differ in atoms outside c that matter to M, and the assumption fails exactly there. This is the same abstraction gap that bounds symbol-only induction in §5.4.

Example. Take two action words $w_1 = u\,a\,b\,c\,c$ and $w_2 = v\,a\,b\,c\,c\,c$, where each letter abbreviates a ground action. STARREPLACE yields $u\,a\,b\,c^+$ and $v\,a\,b\,c^+$. The pair $b\,c^+$ occurs twice, so folding introduces $N_1 \to b\,c^+$, leaving $u\,a\,N_1$ and $v\,a\,N_1$. The pair $a\,N_1$ then occurs twice, giving $N_2 \to a\,N_1$ and the bodies $u\,N_2$ and $v\,N_2$. Indexing attaches to N_2 the intersection of the states holding at its first symbol in the two observed occurrences, and the decoder admits N_2 only in states entailing that class.

Soundness without termination. This fallback is strictly weaker than G_{env}. Under the assumption of Proposition 3 it preserves soundness, and it does not carry termination. The (*state*, *budget*) construction of §3.3 carries both soundness and termination at a goal state. Proposition 3 bounds the language from above by L_{exec} and says nothing about reaching $\hat{g}$, and the grammar admits unbounded macro loops, so no plan sampled under the raw grammar completed within the tested token budget (§5.4). Applying the same horizon-unrolling used for G_{env} bounds derivation length at the cost of coverage. Completing a plan still requires a non-empty accepting language and a sufficient decoding budget. The rollout-based induction claim is scoped to environments whose state can be observed symbolically at decode time.

Q Broader impact

A 1.5B LLM under an applicability mask inherits the plan validity of a classical solver, so LLM-planning benchmarks run without grammar constraints may underestimate the planning performance achievable by small LLMs. This bears on deployment of small models in on-device planning, on verifier-gated orchestration of classical solvers, and on grammar-aware benchmark design. The framework gives the LLM no new planning capability. It lets a classical planner use the LLM's language understanding without inheriting its unsoundness.

R Induced-planner failure decomposition

Of the 73 failures of the AGCP-induced E_6 planner on the 114 paired Sonnet parses (§5.4): 37 return no plan within the search budget. The inspected failures are consistent with schema incompleteness: the induced model represents applicability as one conjunction per action and so cannot express a placement precondition of the form *the receptacle is open or is not openable*, which is a disjunction. A bounded search does not by itself distinguish a missing or spurious precondition from an exhausted budget, so this attribution rests

on inspection rather than on the failure signal alone. Another 16 are goal-construction failures, of which 15 follow an imperfect E_4 parse. A further 17 are divergences between the induced model and the environment, which incorrect induced effects can also produce, and 3 are rendering failures. The first class is the one we attribute to the induction procedure. The count of 37 is an observed failure category, and the cause is a diagnosis from inspected traces rather than a trace-by-trace audit. The coverage limit of the schema-free fallback (App. L) is a separate boundary.

S Experimental setup

Models. We evaluate nine systems, eight instruction-tuned LLMs across four model families and one non-neural baseline. The local model is Qwen-2.5-1.5B-Instruct (Qwen Team, 2024) under XGrammar JSON-schema masking, package version 0.2.0 (Dong et al., 2025; Li et al., 2026). The cloud models are Qwen-2.5-7B-Instruct-Turbo and Meta-Llama-3.3-70B-Instruct-Turbo through Together.ai (Together AI, 2024; Meta AI, 2024; Grattafiori et al., 2024), GPT-3.5-turbo, GPT-4o-mini and GPT-4o through OpenAI (OpenAI, 2024), and Claude-Haiku-4.5 and Claude-Sonnet-4.6 through Anthropic. No fine-tuning or reinforcement learning from human feedback was applied to any model, and the exact API identifiers are below. We generate one output per task, with greedy decoding for the local model and `temperature` $= 0.3$ for the cloud LLMs. No cloud API offers a setting that makes its output exactly reproducible (§7). XGrammar enforces the JSON schema for the local model. Cloud models receive the schema and slot conventions in the prompt and return JSON, through a JSON object for OpenAI or as plain text parsed as JSON for Anthropic and Together, which fixes the format but does not enforce the allowed values. An out-of-vocabulary value therefore fails at scoring rather than at decoding. No ground-truth-aware reranking is used on ALFWorld. The PlanBench parse column of Table 7 is verifier-selected. N=251 rows per cell, with multi-seed checks where rate limits permit (App. J).

System	API identifier	Provider
Qwen-1.5B	`Qwen2.5-1.5B-Instruct`	local
Qwen-7B	`Qwen2.5-7B-Instruct-Turbo`	Together
Llama-70B	`Llama-3.3-70B-Instruct-Turbo`	Together
GPT-3.5	`gpt-3.5-turbo`	OpenAI
GPT-4o-mini	`gpt-4o-mini`	OpenAI
GPT-4o	`gpt-4o`	OpenAI
Haiku 4.5	`claude-haiku-4-5`	Anthropic
Sonnet 4.6	`claude-sonnet-4-6`	Anthropic

Table 12: Models evaluated, with the interface identifier and provider for each.

Benchmarks. PlanBench T1 (Valmeekam et al., 2023) on four IPC domains (Blocksworld, Mystery Blocksworld, Logistics, Depots), scored by the VAL plan validator (Howey et al., 2004). ALFWorld (Shridhar et al., 2021), built on ALFRED (Shridhar et al., 2020) and TextWorld (Côté et al., 2019), contributes N=251 in-distribution (seen) and N=251 out-of-distribution (unseen) rows across the seven families of Table 5. Each row is one human-written description of a task, and ALFRED supplies about three per task, so the out-of-distribution rows cover 85 distinct tasks and the in-distribution rows 242. Execution loads six families and the primary execution subset is five. Breadth comes from the IPC-2023 learning track (Taitler et al., 2024) and AutoPlanBench (Stein et al., 2025) (32 domains). The ten IPC domains are Blocksworld, Childsnack, Ferry, Floortile, Miconic, Rovers, Satellite, Sokoban, Spanner and Transport. Childsnack, Ferry and Satellite use negative preconditions, which we drop before loading so that the STRIPS parser accepts them. Their induced schemas are scored against the reduced models and are under-constrained relative to the originals. Of the 46 schemas, 44 are induced afresh and 2 are reused from the cross-domain schema library (App. O).

Baselines. (i) A hand-crafted regex parser for NL-goal parsing, paired with an alias table of about 200 entries (implementation details in App. D). (ii) An LLM-removed pipeline that runs pure A*+hAdd over the benchmark's PDDL model with no LLM call. (iii) The HandCoded TextWorld (TW) expert provided with ALFWorld, as the schema-driven planner at E_6. (iv) Our own Qwen-1.5B implementation of the LLM+P approach (Liu et al., 2023), run on Depots and scored under the corrected non-empty-plan criterion. The published LLM+P evaluation covers other domains and is not reproduced here.

Experiment	Transition model	Planner / compiler	Outcome
No-LLM (Table 1)	benchmark PDDL	A*+hAdd	VAL-validated solution
AGCP on PlanBench (Table 1)	benchmark PDDL, via the A* fallback	single-plan grammar, Qwen-1.5B decode	VAL-validated solution
Mask isolation (§5.1)	hand-coded Blocksworld rules	BFS over (state, budget)	goal reached
Compile statistics (§3.3)	induced schemas for the action set, exact ground operators for applicability	BFS, A* fallback	compile, solution
Component F_1 (§5.1)	induced schemas	—	schema accuracy
ALFWorld induced planner (§5.4)	induced schemas from exploration	greedy search	env won
Schema reuse (App. O)	induced schemas	signature match	transfer
SCI-ReDuce (§5.4)	trajectory macros with recorded states	class-indexed BFS	goal reached

Table 13: Model provenance per experiment, the transition model and planner each result depends on.

Metrics. We report plan execution success, a binary outcome read from VAL or from TextWorld's own success signal (won), and NL-parse F_1 against the ground-truth goal slots after normalisation, which we abbreviate F_1^{norm}. Object names are canonicalised, so *mug* matches *Mug*. A sliced object named as such and the base object with the slicing modifier set are treated as one. The slot naming the appliance to toggle and the slot naming a movable receptacle are folded into one auxiliary role before matching, so a lamp placed in the receptacle slot still matches. The metric is therefore a normalised goal-slot F_1, more lenient than strict slot accuracy, and distinct from execution success. We also report component F_1 for induction against PDDL ground truth, and held-out grammar coverage for SCI-ReDuce at 4 seeds per training size.

T End-to-end coverage variants

Using the ALFWorld-provided HandCoded TextWorld expert, end-to-end success on ALFWorld varies with how the instance set is filtered. On a high-confidence subset of 30 instances whose parses reach $F_1^{\text{norm}} \geq 0.95$, the loop succeeds on 96.7%. On the larger set of 93 instances meeting the same filter, it succeeds on 79.6%. Without any parse filter it reaches 68.1% over the 119 core-5 instances (Wilson $[59, 76]$) and 56.6% over all 143 instances of all six types. The gap to core-5 reflects limits of the TextWorld library. Pick Two & Place runs to the step limit even on ground-truth parameters, and Pick & Place with Movable Receptacle has no policy. Parsed parameters are accepted by the environment policy on 133 of 143 instances. The BUTLER text agent of Shridhar et al. (2021) reports 40% on seen and 37% on unseen tasks across all task types in the text environment, under a different training and evaluation protocol, so the figures are not a matched comparison.

U Parse-correction mechanisms

Parse correction combines within-task re-prompting (Layer A) with statistics and hints accumulated across tasks (Layer B).

Layer A: within-task re-prompting. After each parse, every slot value is checked against environment affordances. These are read from the per-step fact list that ALFWorld's TextWorld engine returns as its symbolic state observation, which gives the ground atoms holding in the current scene. If a slot is unrecognised, as in the lamp case of §3.4, the LLM is re-prompted with that slot and the scene's legal alternatives, up to $K{=}3$ iterations. No ground-truth solution or expert demonstration is consulted.

Layer B: cross-task statistics and hints. Two per-task-type stores are updated online across runs of the pipeline. (i) *Per-type parse statistics*: frequency tables over slot values from past *successful* parses. On a new instance the LLM prompt includes the top-3 values per slot for the current task type. (ii) *Slot-correction hint cache*: short slot-correction rules extracted from past *failed* parses. A typical rule reads "for Pick & Place tasks where the instruction says 'on a table', the placement slot should be the side table rather than the coffee table". The filter keeps the first sentence of a proposed rule, rejects it if it contains any of twelve execution phrases such as *navigate to* or *while holding*, and requires the phrase *should be* or *set*. The filter is a keyword rule rather than a guarantee, so we also inspected what it retained, and all 19 surviving hints are slot-level corrections (App. H.1). The two layers draw on different information (Layer A on the current task's affordances, Layer B on past outcomes for the same task type), and §5.3 reports their increments, with the Layer B increment sensitive to the occurrence rule.

Layer-B information boundary. Because Layer B carries corrections across instances, its information boundary must be made explicit. Priors come from execution-successful prior tasks and hints from execution-failed ones, so

the only supervision is the environment's success signal (won) and its affordances. The ground-truth goal conditions are recorded for scoring but are never read by the correction loop, and the stream is online, so instance t sees only the instances that precede it. Learning through the prior that instructions saying "floor lamp" refer to the desk lamp class in this environment is thus environment-ontology learning from interaction, which is the mechanism's claimed contribution.