PITON: ANCHORED DEPENDENCY REPAIR FOR MULTI-TOOL PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-planning models expand a natural-language request into an ordered sequence of structured tool calls. Correct plans often require tools the request never mentions. Prior work supplies this knowledge with a dependency graph built from gold plans, joining any two tools called one immediately after the other. Treating every such adjacent pair as the same dependency conflates two situations: *(1) within-goal dependency:* two adjacent tools may serve one requested goal from the query, one tool being an unnamed prerequisite of the other; *(2) cross-goal adjacency:* two adjacent tools simply serve two different goals, requested back to back. The graph alone therefore cannot tell whether an observed adjacency represents a dependency needed by the current query or merely connects two consecutive goals. This distinction matters because frozen planners fail disproportionately on the former: on UltraTool, they recall 64.6% of cross-goal adjacencies but only 23.7% of within-goal dependencies, where the unnamed prerequisite is often missing. Motivated by this gap, we propose PITON, a model-agnostic repair framework for dependency-aware plan refinement. PITON anchors on a planner's draft, using the tools already selected as clues to missing prerequisites. It alternates an insertion reranker that recovers missing dependencies from the anchors' one-hop graph neighbourhood with a symmetric deletion reranker, until the plan stabilises. PITON is especially natural for masked diffusion models. They progressively commit confident tools and refine the remaining plan. We evaluate ten diffusion, autoregressive and closed-source planners on UltraTool, ShortcutsBench and MCP-Atlas, and show PITON raises edge F1 in all eighteen settings. The gain over the unrepaired draft reaches $+26.0$, $+10.1$ and $+24.3$ points on the three benchmarks. PITON requires only lightweight rerankers while keeping the base planner frozen, introducing minimal additional training overhead. Code is available at `https://anonymous.4open.science/r/anchor-anon-6006`.

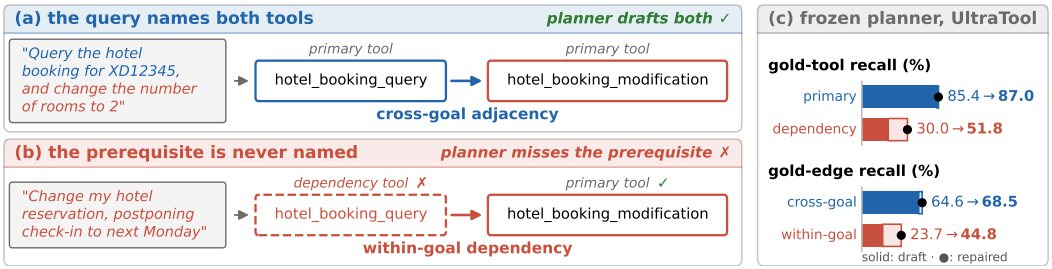


Figure 1: The same adjacent tool pair poses two different problems. In (a), the query names two goals, one for each tool. In (b), it names only one goal (change the reservation). The booking-query step is therefore an unnamed prerequisite that the planner must infer. The right panel shows how often a frozen planner (Dream) recovers each type on UltraTool, and after PITON repair (filled dots).

1 INTRODUCTION

Large language models excel across a wide range of natural-language tasks, yet requests that need exact computation or live information still require external tools such as APIs, algorithms, and

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

expert models [37, 41, 42, 44, 46, 48, 57]. *Tool planning* has therefore become a core capability of tool-augmented systems. Given a high-level user request, the model must understand the intent and produce a complete, ordered sequence of tool calls [6, 11, 49]. The main challenge is that tools depend on one another, yet the user rarely names the prerequisite tool explicitly.

To satisfy such prerequisites, prior work organises tool dependencies into a *tool dependency graph* whose nodes are tools and whose directed edges denote precedence [6, 11, 49, 54]. When tools exchange typed resources such as images or audio, an edge can be read off matching input and output types [14, 28, 49]. In a general catalogue almost every tool consumes and produces text, so the graph is instead built from adjacency in recorded plans: two tools called one after the other are joined by an edge [25, 27, 54]. Whether that graph then constrains the search space, is injected into a frozen LLM, or is trained into model parameters [6, 10, 11, 27, 31, 54], this construction implicitly treats **every adjacent pair in a gold plan as the same kind of dependency**.

We argue that this assumption conflates two situations of very different difficulty. To make the distinction precise, we first identify the **goals** named by a query. Each goal is realised by one or more tools. The **primary** tool directly performs the named action, while any **dependency** tools are required by the primary but never mentioned in the query. Adjacent pairs in a gold plan then fall into two kinds. A **WITHIN-goal dependency** joins two tools serving one goal, whereas a **CROSS-goal adjacency** joins tools serving consecutive goals. Figure 1 contrasts the two using the same UltraTool gold tool pair under two queries. A dependency graph stores the same edge for both kinds, yet the two situations differ in difficulty. We dissect the drafts of four planners on three benchmarks (§3). In ten of twelve cells, WITHIN edges are recalled less often than CROSS edges, by up to 40.9 points. Dependency tools are recalled less often than primary tools in all twelve cells, by 16 to 55 points.

From this dissection, we identify an *information-timing mismatch*. When a goal requires an unnamed prerequisite, the dependency tool is usually called *before* the primary tool in 83–93% of such edges. All prior graph-based methods we compare with (Table 4) consult the graph in generation order, so when the prerequisite must be chosen, the tool that would identify it is still unwritten. In contrast, PITON reverses the direction of inference. It treats the planner's completed draft as an **anchor**: the tools directly supported by the query are mostly already present, so their one-hop neighbourhood in the graph supplies candidates for missing prerequisites. A dependency reranker asks whether each candidate is required by the current query, while a symmetric deletion reranker removes unsupported selections. The two alternate until the plan stops changing.

Both rerankers are gradient-boosted trees trained on a corpus synthesised from gold plans. They fit on a CPU in under two minutes per benchmark while the planner stays frozen (§4). PITON can take its draft from any planner, but masked diffusion language models [35, 58] are its most natural drafters. Entropy-ordered decoding [5, 62] commits the most certain tools first and thus leaves clean anchors. Any-position generation then lets the model insert a missing prerequisite in place (§4.2). We evaluate ten diffusion, autoregressive and closed-source planners on UltraTool, ShortcutsBench and MCP-Atlas. Every published baseline is rescored against the same gold with the same scorer (§5). Under one fixed configuration, PITON improves edge F1 in all eighteen planner–benchmark settings. The gains over the unrepaired drafts reach $+26.0$, $+10.1$, and $+24.3$ points on the three benchmarks. A stratified audit confirms that the gain lands on the diagnosed gap, dependency tools and WITHIN edges, as the dissection predicts.

In sum, the main contributions of this paper are:

- We distinguish WITHIN-goal dependencies from CROSS-goal adjacencies, which existing dependency graphs conflate. Frozen planners miss the former far more often: WITHIN-edge recall trails CROSS-edge recall in ten of twelve planner–benchmark cells, by up to 40.9 points.

- We propose PITON, a model-agnostic repair layer that anchors on the completed draft, turning already selected tools into evidence for missing prerequisites.

- We evaluate PITON across ten planners and three benchmarks. It improves edge F1 in all eighteen settings ($+4.6$ to $+26.0$) and outperforms published graph-based planners. It also surpasses the strongest baseline at one to two orders of magnitude less training cost.

- We show that without gold trajectories, PITON trained on the drafts models produce still improves in seven of eight cells. Handing the repaired plans to a live agent on MCP-Atlas raises answer coverage for all seven drafters, by $+5.8$ points on average on the queries where the repair fires.

2 Related Work

Static tool planning and tool dependency graphs. LLM-generated tool plans are either produced in one shot before execution [6, 11, 23, 49, 54] or decided step by step during execution [19, 27, 40, 42, 57, 60, 64]. This paper and its main baselines belong to the former family, which usually models tool dependency explicitly as a graph [27, 49, 54]. Benchmarks divide the same way: ToolSandbox and PlanBench-XL probe implicit dependencies interactively, each call uncovering evidence for the next [26, 29]; ours carry gold plans that need no execution-time evidence (§5.1).

How the graph is used, and the gap it leaves. Early work reads the graph as a constraint on search [27, 28, 54]: GNN4Plan and ToolNet, for example, choose each tool from the previous tool's neighbours [27, 54]. Others use it to complete the retrieved tool set with the prerequisites it implies [10, 31]. More recent methods instead internalise dependency structure through additional training, in the planner [6, 11] or retriever [43, 63]. We analyse in §3 why the current search interface leaves hidden prerequisites unresolved, and show in §5 that a post-draft repair interface can recover many of them with lightweight training. Appendix A gives the per-method comparison.

3 Dissecting Dependencies in Graph-Based Tool Planning

Preliminaries. Given a query q and a candidate tool list $\mathcal{C}_q$ (retriever top-K), a planner emits a plan $\hat{P} = (t_1, \ldots, t_k)$. We score it against the gold plan by node F1, edge F1 (set semantics over adjacent pairs), exact match and normalised edit distance. Drafters are frozen throughout. The dissection below characterises the failures of such drafts.

Taxonomy of Tool Occurrences and Adjacencies. Figure 1 separated a WITHIN-goal dependency from a CROSS-goal adjacency on one pair of examples. To show that both kinds really occur, we assign every tool occurrence and every adjacent pair of a gold plan to one side of that distinction. For each query, an LLM first extracts the **goals** named in the query alone, each realised by one or a few mutually dependent tools. A second LLM pass, shown those frozen goals and the gold plan, then aligns every tool occurrence to the goal it serves: a **primary** tool directly performs the named action, and a **dependency** tool must be called for the primary to run but is never directly mentioned in the query. We label each adjacent pair from these alignments. Two endpoints in one goal form a within-goal dependency (a **WITHIN** edge), while a pair at the boundary of consecutive goals forms a cross-goal adjacency (a **CROSS** edge). These labels describe how a query **uses** a tool or pair, not its identity. The same tool can be primary in one query and a dependency in another. Likewise, the same pair can be WITHIN in one query and CROSS in another. Appendix B.1 details the protocol, quality control and dependency sub-roles. Table 1 stratifies the *unrepaired drafts* of all twelve planner–benchmark cells along both axes.

On the node axis, primary recall exceeds dependency recall in all 12 cells, by $+16.3$ to $+55.4$ points. On the edge axis, WITHIN-edge recall likewise falls below CROSS-edge recall in most cells ($+13.7$ to $+40.9$). The pattern reverses in two MCP-Atlas cells. There, 63% of CROSS edges themselves carry a dependency endpoint, so they were never an easy control group (Appendix B.2). We further find that dependency tools are hard because the query leaves them no lexical evidence; the per-benchmark retrieval tables are in Appendix B.4.

Insight 1: Dependency tools, and with them WITHIN-goal edges, are recalled far less often than primary tools and CROSS-goal edges, in almost every planner and benchmark.

But does the gap follow usage, or merely the identities of the tools in each stratum? Table 1 compares two tool populations. Its gaps could therefore reflect dependency-position tools being rarer or harder, with usage playing no part. To separate the two, we hold identity fixed and compare each tool against itself. We retain only tools that appear in both roles and edges that appear in both usages. For each tool, we compare its recall in the dependency role with its recall in the primary role. Because every comparison is within one tool, properties such as rarity and intrinsic difficulty cancel out. A large part of the raw gap survives. On UltraTool, the same tool is recalled 5 to 36 points less often as a dependency than as a primary, and the same edge 4 to 34 points less often within one goal than across a goal boundary; three of the four drafters are significant on each axis. Appendix B.3 gives the design and per-cell results.

Table 1: Draft-side gaps along the two axes (12 cells; edge gap = CROSS − WITHIN, node gap = primary − dependency). Stratum sizes are as follows. UltraTool has 936 primary / 220 dependency occurrences and 241 WITHIN / 426 CROSS edges; ShortcutsBench 363 / 393 and 476 / 45; MCP-Atlas 262 / 383 and 449 / 103.

Benchmark	Drafter	WITHIN	CROSS	edge gap	primary	dependency	node gap
UltraTool	Dream	23.65	64.55	+40.90	85.36	30.00	**+55.36**
	Qwen2.5	31.54	55.63	+24.09	84.51	43.64	**+40.87**
	LLaDA	30.57	62.99	+32.42	85.44	45.00	**+40.44**
	Llama-3.1	24.48	40.61	+16.13	85.04	61.82	**+23.22**
ShortcutsBench	Dream	19.61	33.33	+13.72	69.42	34.52	**+34.90**
	Qwen2.5	19.40	37.78	+18.38	68.87	44.53	**+24.34**
	LLaDA	17.30	37.04	+19.74	70.80	38.08	**+32.72**
	Llama-3.1	16.95	34.07	+17.12	70.16	48.77	**+21.39**
MCP-Atlas	Dream	7.92	9.15	+1.23	60.18	24.63	**+35.55**
	Qwen2.5	16.07	10.78	−5.29	60.18	40.90	**+19.27**
	LLaDA	8.15	15.36	+7.21	61.32	26.63	**+34.69**
	Llama-3.1	23.52	7.84	−15.68	79.77	63.45	**+16.32**

Insight 2: The same tool is recalled less often when used as a dependency than as a primary, and the same edge less often within a goal than across goals.

Can a capable planner reason out a tool dependency? Or is the dependency specific to a task and tool set, so only a module that has seen the relationship can supply it? We take 30 high-frequency adjacent tool pairs from UltraTool and remove every training sample containing any of these pairs. We then fine-tune a Qwen3-8B backbone and compare it with a control arm that retains the exposure under the same training budget. This intervention erases the co-occurrence, not the tools themselves. A construction gate requires every tool to remain in at least one erased-arm training sample. Both endpoints therefore retain individual exposure, while only their pairing is unseen. When an erased edge has a CROSS usage at test time, the model still recalls it at 64.7%. This is 72% of the with-exposure level (89.9%). For a WITHIN usage, recall falls to 20.8%, only 30% of the with-exposure level (70.5%) and barely above the 18.3% an untrained model reaches. The model knows how to use both tools and still does not know that one depends on the other. The protocol, the per-group numbers and the limitations are in Appendix B.5.

Insight 3: A tool dependency is structural knowledge specific to the task and its tool set. A planner that has never seen the pair recovers it barely above the untrained rate, so an external module has to supply it at planning time.

Why the existing graph interface has not closed this gap. At first glance, prior graph-based planners use the dependency graph much as we do [27, 28, 54]: once a tool is chosen, the next one is selected from its graph neighbours. Their motivation is precisely what Insight 3 supports: the graph records structural dependencies that the planner may not infer reliably on its own. Yet these methods still score poorly (Table 2). Two properties of the structure explain why:

• First, the graph is read in generation order, while the dependency tool usually precedes the primary tool that identifies it. Among the WITHIN edges that join a dependency tool to a primary one, the dependency tool comes first in 82.7% on UltraTool, 83.2% on ShortcutsBench and 92.7% on MCP-Atlas. At the step where the prerequisite must appear, the tool that identifies it is still unwritten.

• Second, the graph is ambiguous by construction, yet an edge is scored by a query-independent quantity, either its frequency in past trajectories or a prediction from another model [10, 25, 27]. Built from adjacency in gold traces, it records a CROSS-goal adjacency as the same edge as a WITHIN-goal dependency. Also, a primary tool often has many neighbours, 6.6 on average on UltraTool, 26.0 on ShortcutsBench and 11.6 on MCP-Atlas. Which of them a plan lacks, or whether it lacks any, depends on the request. By Insight 2, this is exactly what such a score cannot recover.

Insight 4: The graph contains the missing dependency; what is missing is the decision context. Existing search-style interfaces read it in generation order, before the tool that reveals the prerequisite is written, and score edges without the query. A repair should instead read the graph from the finished draft and score candidate neighbours against the query.

4 PITON: REPAIRING FROZEN DRAFTS WITH GRAPH-ANCHORED RERANKING

Based on the dissection of §3, we propose PITON, a repair layer that reverses the dependency inference direction. It has two modules: a dependency reranker that inserts and a deletion reranker that removes. They alternate until the plan stops changing, and Figure 2 traces one round on a real query.

4.1 TOOL CANDIDATE GENERATION AND RERANKING

The dependency reranker. The tools in a draft $\hat{P} = (t_1, \ldots, t_k)$ serve as **anchors**. We build G by counting adjacent tool pairs in the benchmark's training gold plans, $G[a \to b]$ being the number of times b directly follows a, and take as candidates the anchors' one-hop neighbourhood: every triple (a, t, d) of an anchor $a \in \hat{P}$, a tool $t \notin \hat{P}$ joined to it by a G-edge, and a side $d \in \{\text{before}, \text{after}\}$, which fixes the slot $\text{pos}(a) + \mathbf{1}[d = \text{after}]$. A gradient-boosted tree f_{ins} [20, 38] then scores each candidate, $p_{\text{ins}}(a, t, d) = f_{\text{ins}}(\mathbf{x}_{a,t,d})$, from three groups of features $\mathbf{x}$.

- **Graph evidence** is the statistical strength of the candidate edge in the training trajectories: its count $G[a \to t]$, its lift $N\, G[a \to t] / \big(m(a)\, m(t)\big)$ over the endpoint marginals $m(\cdot)$, with N the number of adjacent pairs, and its count rank within the current candidate set.
- **Placement evidence**, for a tool proposed by several anchors, compares the support $G[a \to t]$ each anchor lends it with that anchor's position $\text{pos}(a)$ in the draft.
- **Semantic evidence** is built on $\cos\big(\text{enc}(\text{desc}_t), \text{enc}(q)\big)$ under a frozen sentence encoder enc, where desc_t is the tool's description.

Once scored, a round considers only $\arg\max_{(a,t,d)} p_{\text{ins}}$ over the whole plan and inserts it only if $p_{\text{ins}} \geq \tau_{\text{ins}}$. The gate τ_{ins} rises by a fixed step each round; it is set once on synthetic development drafts and shared by every benchmark and drafter (Appendix D). When the best score fails to clear it, the round leaves the plan untouched.

The training corpus is synthesised from the measured failure profile. Supervision for both modules comes from the training split's gold plans. We corrupt each plan by deleting a tool or replacing one with a similar tool, then learn to undo that corruption. No human annotation is involved. Appendix D gives the synthesis procedure, calibration grid and full feature list.

The deletion reranker. Some planners not only omit tools but also select many wrong ones, which calls for a deletion operator. Its features ask whether the rest of the plan still supports a drafted tool t_i by contrasting its links to its neighbours, $G[t_{i-1} \to t_i]$ and $G[t_i \to t_{i+1}]$, with the bridge $G[t_{i-1} \to t_{i+1}]$ those neighbours would form once it is gone: weak links and a strong bridge mark a tool to remove. A second tree turns these counts, the tool's marginal and its semantic fit into $p_{\text{del}}(t_i)$. It differs from the insertion side in two ways. Its candidates are the plan's own tools rather than neighbours drawn from G. A round may also drop several positions, any t_i with $p_{\text{del}}(t_i) \geq \tau_{\text{del}}$ up to a fixed cap, because removals do not compete for one slot as insertions do.

The repair loop. A single round can restore only a tool one hop away. If a plan lacks two prerequisites in a chain, the second need not be adjacent to any draft tool. We therefore iterate $\hat{P} \leftarrow \text{Insert}(\text{Delete}(\hat{P}))$ and stop as soon as a round changes nothing. Each round first lets the deletion module clear untrustworthy anchors, then rebuilds candidates on the cleaned plan and inserts one tool. Appendix C shows per-round curves and Appendix D.4 ablates each operator.

4.2 DIFFUSION LANGUAGE MODELS AS TOOL-PLAN DRAFTERS

PITON makes no demand on the planner, but the dissection also says what an *ideal* drafter would look like, and masked diffusion language models [35, 58] answer on both counts.

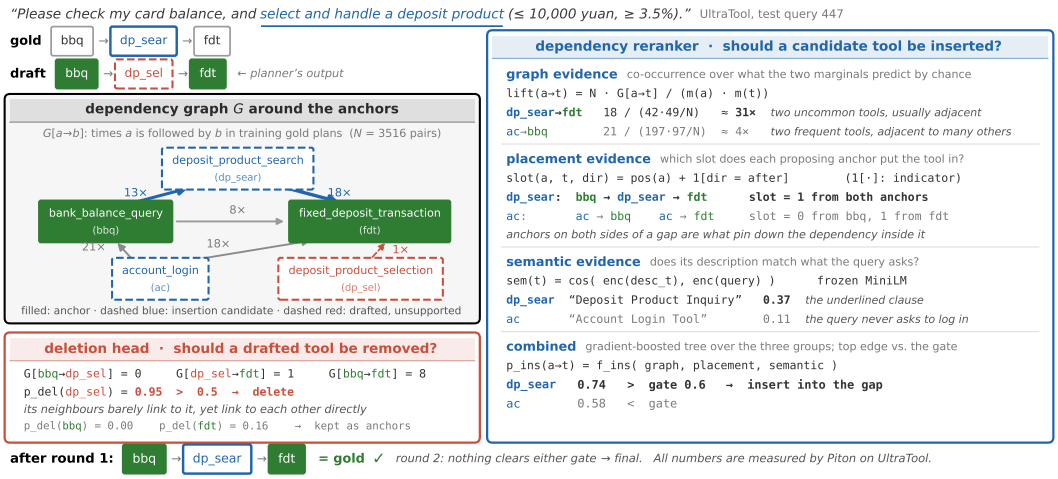


Figure 2: One repair round of PITON on UltraTool test query 447 (Qwen2.5 draft); every count and score is measured. Deletion first drops the intruder `dp_sel`; the dependency reranker then scores each candidate edge on the three evidence groups of §4.1, `dp_sear` clears the gate and enters the gap, `account_login` does not, and the plan equals gold.

Entropy-ordered decoding gives the anchor signal for free. At each step, a DLM commits the slot with the lowest entropy. Tools with strong evidence therefore land first, wherever they sit in the plan. The draft omits what the model was unsure of rather than what came late. Its anchors are therefore the tools it was sure of, leaving hidden prerequisites to the repair stage; how much purer they are is measured in Appendix E.5.

Any-position generation lets the model execute its own repair. A DLM is not bound to left-to-right order, so an edit does not invalidate the rest of the plan. Already generated tools stay in place, and the model itself can decide whether and where to apply the reranker's proposal. Appendix E.4 compares this with applying the proposal directly.

Both properties reach the repair loop through **ScaffoldSlots**. This drafting interface lets a DLM edit a plan it was never fine-tuned to produce. The prompt lists every tool under its own single-token pointer symbol with its name and description. Inserting or deleting a tool thus becomes a slot-level operation instead of a sequence rewrite. For insertion, the interface can open fresh mask slots and let the model choose which to fill under bidirectional context. Appendix E gives the configuration and compares the same DLM with and without ScaffoldSlots.

5 EXPERIMENTS

5.1 SETUP

Models and data. We use Dream-7B-Instruct and LLaDA-8B-Instruct as DLM drafters, with Qwen2.5-7B-Instruct and Llama-3.1-8B-Instruct as matched AR drafters. Dream is trained from the Qwen2.5 family and LLaDA shares a close architecture with Llama-3.1, so both pairs control for scale and minimise backbone differences. We additionally evaluate six closed-source drafters from the Claude and GPT families on MCP-Atlas. Appendix G reports them. UltraTool [17] evaluates planning before tool usage directly. ShortcutsBench [47], user-authored Apple Shortcut programs, and MCP-Atlas [1], human-verified successful trajectories, add more realistic workflows. All three write the query before the plan. Benchmarks that sample the tools first [18, 42, 49] name almost every gold tool in the query, which removes the hidden-prerequisite setting.

Configuration, baselines and metrics. All twelve planner–dataset cells use one configuration, with no per-cell search. The graph is rebuilt per benchmark, and the two rerankers are refitted per cell on a synthetic corpus (Appendix D; the shared configuration is listed in Appendix D.2). For the two diffusion drafters the model itself decides whether to apply the insertions the reranker proposes (§4.2); for the autoregressive drafters the reranker applies them. External baselines are GTool [6],

GNN4Plan [54], TaskBench (the method) [49], HuggingGPT [48], DiG-Plan [23] and ControlLLM-ToG [28]. We report edge F1, node F1 and NED (lower is better), rescoring every method against the same gold and scorer. The rescoring protocol is in Appendix F.

5.2 MAIN RESULTS

Overall reading. PITON raises edge F1 in all twelve settings (mean $+13.80$, min $+5.96$, max $+26.01$). All metrics move together, with no regression in any of the 36 metric cells. The repaired DLM beats the repaired AR model in five of six matched comparisons, consistent with §4.2. PITON also leads GTool, the strongest graph-based method, in five of six edge-F1 comparisons with the DLM drafters, and splits three to three on GTool's own AR backbones, which GTool trains (up to 185 A100-minutes) and we leave frozen: its wins are both UltraTool cells and one ShortcutsBench cell, ours both MCP-Atlas cells, by 16 to 21 points. The same rerankers attach to all six closed-source drafters without refitting and add $+8.62$ edge F1 on average (Appendix G).

MCP-Atlas gains most from repair because over half of its gold tools are dependency tools, exactly what the dependency reranker acts on. It also needs more rounds: a single round captures only 47 to 80% of the net gain, and all four drafters keep improving into a third. Appendix C traces the rounds. We also ablate the frozen rerankers under insertion only and deletion only. The loop wins in all twelve cells against either operator alone, and by more than their two gains added together. The ablation is in Appendix D.4.

5.3 DOES THE GAIN LAND WHERE THE DISSECTION SAID?

The dissection predicted that PITON would improve dependency endpoints and within-goal edges. We score the outputs before and after repair against the goal annotation of §3. The results use the same strata, scorer and preprocessing as Table 2. A star in Figure 3 marks a change whose 95% interval over 2,000 query resamples excludes zero. Figure 3 shows the UltraTool cells, while Appendix H shows all twelve. Repair lifts WITHIN-edge recall in all 12 cells ($+2.38$ to $+23.36$), significantly in 11. It also lifts dependency-tool recall in 10 of the 12 cells, by up to $+22.0$. The gain therefore lands where the dissection predicted. One case needs a closer look. On Llama-3.1 / UltraTool, primary- and dependency-tool recall both fall, even as node F1 and edge F1 rise sharply. We therefore display deletion and insertion separately: draft $\rightarrow$ deleted shows what the deletion reranker removes, and deleted $\rightarrow$ repaired shows insertion.

Scoring insertion against the post-deletion plan removes the cost of deletion from the comparison. Its contribution then becomes clear. Averaged over the four drafters, insertion adds $+11.8$, $+5.9$ and $+18.2$ on WITHIN edges for UltraTool, ShortcutsBench and MCP-Atlas. The corresponding gains on dependency tools are $+14.1$, $+6.8$ and $+19.5$. Both gains are positive in all 12 cells, and 78.4% of the 678 restored gold tools are dependency tools. The role gap from §3 narrows in 10 of 12 cells, with the largest falling from 55.4 to 35.1. The edit audit tells the same story. Across the three benchmarks, 66%, 72% and 76% of adopted insertions hit a gold tool. Of those hits, 54%, 76% and 86% are dependency tools, exactly the region targeted by the anchored candidate design.

The two operators help different drafters to different degrees, because their drafts fail in different ways (Appendix D.4). On UltraTool, for example, Llama-3.1 gains CROSS-edge recall from deletion alone. This is because its drafts contain too many tools, averaging $1.74\times$ the gold-plan length. Deleting irrelevant tools makes tools from consecutive goals adjacent again and lifts CROSS-edge recall by $+8.9$. Although deletion removes some correct tools, which lowers recall, the precision gain is much larger: node precision rises from 46.0 to 76.8.

5.4 GENERALISING TO A NEW TOOL SET: CAN PITON REPAIR WITHOUT GOLD TRAJECTORIES?

PITON normally builds its dependency graph from the target benchmark's training gold plans and trains both rerankers on the same source. We now remove those plans. The training split retains only queries and candidate tools, so even the graph must be rebuilt from drafts of the unlabelled queries. We devise three substitutes for that supervision. **Cross-benchmark transfer (T1)** transfers rerankers fitted on ShortcutsBench trajectories to other benchmarks. **Drafter consensus (T2)** uses a

Table 2: PITON against external tool-planning methods (UltraTool 489, ShortcutsBench 237, MCP-Atlas 166 test queries). Every result is the mean of three independent runs, for the external baselines and for PITON alike; the run-to-run spread of every row is in Appendix F. For the two diffusion drafters the model executes each insertion (§4.2); Appendix E.4 gives the reranker-executed counterpart. Training times are in minutes for UltraTool / ShortcutsBench / MCP-Atlas. PITON fits on one 8-core Apple M2 CPU. For GTool (one A100) and GNN4Plan (one RTX 6000 Ada), the value covers the full fit-and-evaluate run because the released code does not time each phase. A per-epoch regression attributes most of this time to training (Appendix F); 0 denotes a training-free method.

		UltraTool			ShortcutsBench			MCP-Atlas			
Group	Method	eF1	nF1	NED	eF1	nF1	NED	eF1	nF1	NED	train
Dream / Qwen2.5	DLM draft (Dream)	54.28	78.91	.2973	27.84	62.23	.4884	12.74	48.85	.6790	—
	PITON (Dream)	**62.74**	**82.59**	**.2499**	**35.96**	**65.14**	**.4293**	35.78	60.99	.5603	**1.5/1.2/0.39**
	AR draft (Qwen2.5)	42.35	72.65	.3617	21.06	57.37	.5314	18.13	53.09	.6784	—
	PITON (Qwen2.5)	52.67	77.27	.3247	31.12	61.81	.4654	**42.44**	**65.61**	**.5371**	**1.5/1.2/0.39**
	GTool	56.76	79.89	.2775	24.15	56.91	.4778	21.48	55.79	.6404	185/102/8.3
	GNN4Plan	43.59	68.53	.4274	17.19	40.66	.6567	28.18	48.89	.6864	3.4/1.4/0.39
	TaskBench	34.03	64.02	.4596	15.23	45.98	.6119	17.74	47.55	.6946	0
	HuggingGPT	32.15	62.12	.4709	14.51	46.04	.6228	18.16	46.65	.6974	0
	DiG-Plan	32.43	55.79	.5662	1.51	13.24	.9278	8.31	34.80	.7690	0
	ControlLLM-ToG	31.73	61.79	.4769	18.68	52.33	.6048	14.57	53.01	.6316	0
LLaDA / Llama-3.1	DLM draft (LLaDA)	51.11	77.68	.3167	23.21	62.81	.5237	12.96	48.23	.6849	—
	PITON (LLaDA)	57.07	81.16	.2807	**30.02**	**65.31**	.4755	**33.83**	**62.28**	**.5708**	**1.5/1.2/0.39**
	AR draft (Llama-3.1)	21.21	58.51	.5191	16.60	55.29	.6149	13.55	47.01	.7480	—
	PITON (Llama-3.1)	47.22	73.43	.3910	23.84	60.62	.5595	27.99	61.76	.6280	**1.5/1.2/0.39**
	GTool	**63.26**	**83.50**	**.2426**	26.28	58.34	**.4645**	21.57	52.95	.6719	153/68/6.9
	GNN4Plan	26.14	51.14	.6038	15.02	41.40	.6909	21.95	41.21	.7586	3.4/1.4/0.38
	TaskBench	17.66	43.42	.6507	15.78	47.98	.6279	15.09	46.74	.7201	0
	HuggingGPT	38.17	66.15	.4580	16.31	47.29	.6398	14.44	42.64	.7419	0
	DiG-Plan	32.40	55.87	.5697	1.61	13.00	.9305	7.28	35.07	.7713	0
	ControlLLM-ToG	22.44	50.07	.6238	14.69	40.50	.7035	12.81	46.96	.7041	0

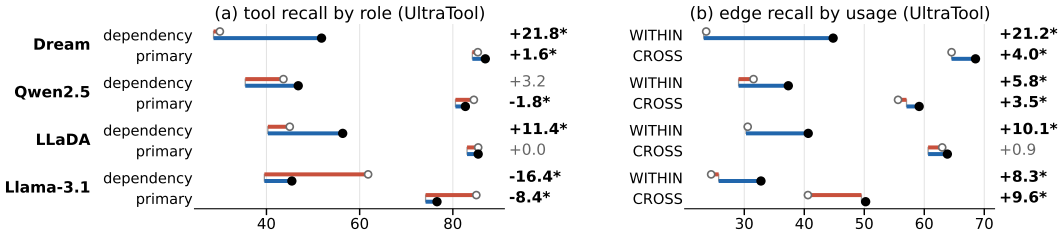


Figure 3: (a) Gold-tool recall by annotated role and (b) gold-edge recall by usage, before and after repair on UltraTool. Open circle the draft, red the deletion step, blue the insertion step, filled circle the repaired plan. Drafters are named in (a); (b) keeps the same row order. The right column of each panel is the net change, starred when the 95% interval over 2,000 resamples excludes zero.

reliability-weighted vote of four drafters as pseudo-gold. **Teacher distillation (T3)** asks a stronger model (Claude Opus) to draft once per query. Appendix I details all three.

T3 beats the draft in seven of eight cells and is flat in the eighth (Table 22). The weakest drafter gains the most by a wide margin. Against a ceiling refitted under the same zero-trajectory configuration, T3 recovers 81% of the obtainable gain on UltraTool and 55% on MCP-Atlas for the Qwen2.5 drafter, and on MCP-Atlas all four drafters gain $+8.5$ to $+13.7$ (Table 3). The flat cell is Dream on UltraTool, at -0.03; its draft is already the strongest of the eight. T1's deletion reranker still helps where the draft over-selects, in three of the eight cells by $+3.7$ to $+10.9$ (the delete-only column of Table 22), but once its dependency reranker joins, the loop ends below the unrepaired draft in six of the eight cells. This confirms the conclusion of §3: tool dependencies are task-specific and barely

Table 3: Zero-trajectory repair on MCP-Atlas, edge F1 of the full loop under each supervision source. *ceil.* is the supervised PITON row of Table 2; the UltraTool cells and the delete-only arms are in Table 22.

	draft	T1	T2	T3	ceil.
Qwen2.5	17.8	17.0	21.1	**30.1**	42.4
Llama-3.1	13.6	15.5	13.4	**22.1**	28.0
Dream	12.7	10.6	18.2	**26.4**	35.8
LLaDA	12.5	10.7	15.1	**25.6**	33.8

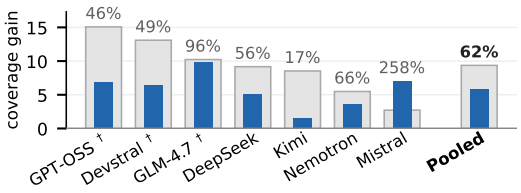


Figure 4: Downstream execution on MCP-Atlas: what a reference plan buys over the draft ($D - B$, grey), what PITON recovers ($C - B$, blue), and their ratio. A † marks drafters used to select the query set.

transferable across benchmarks or tool sets. T2 and T3 avoid this problem because they derive supervision from the target benchmark's own queries and tools. They therefore recover the structure that the benchmark actually needs.

5.5 DOWNSTREAM EXECUTION: DOES A REPAIRED PLAN HELP THE AGENT?

Every result so far scores a plan against a reference. This section asks the downstream question instead: we hand each plan to a live tool-calling agent on MCP-Atlas and grade the answer it returns against the benchmark's factual claims.

Setup. Four arms share one soft-prompt wording and differ only in the injected plan. **A** injects no plan, **B** the model's own draft, **C** the draft repaired by PITON, and **D** a reference plan derived from the official trajectory. The agent is one of seven open-weight tool-calling models (Figure 4), each writing and then executing its own plan. We follow MCP-Atlas's evaluation: coverage is the fraction of a query's gold factual claims stated in the final answer. The query set is built in two steps. We first filter for queries where a reference plan can beat injecting no plan at all, keeping the 64 queries with $D - A > 0$. Each drafter then writes its own plan for those 64 queries, and we keep the queries where its dependency reranker actually edits that plan. Each drafter therefore runs on its own subset: 173 query-by-drafter pairs, 2,076 agent runs, three runs per cell, median reported (Appendix J).

Result. Figure 4 pairs each drafter's ceiling ($D - B$, grey) with the part of it PITON takes back ($C - B$, blue). Repaired plans improve downstream execution for all seven drafters, by $+5.8$ coverage on average over the drafter's own plan, with a 95% interval over query resamples of $[+1.7, +9.8]$; the four drafters that played no part in screening the queries gain $+4.4$. The reference plan sets the ceiling, worth $+9.4$ when injected in place of the draft, and PITON's $+5.8$ is 62% of that, obtained without ever seeing a reference. Per-drafter gains range from $+1.5$ to $+9.9$; Appendix J reports the per-drafter table and the null control that fixes this measurement's noise floor.

6 CONCLUSION

A tool dependency graph stores two kinds of adjacent pair as one edge, and frozen planners disproportionately miss one of them: the WITHIN-goal dependency, whose prerequisite tool the query never names. PITON repairs exactly that gap by anchoring on the draft and reranking one-hop graph candidates against the query, raising edge F1 in all eighteen planner–benchmark settings under one configuration, with training cost one to two orders of magnitude below the strongest trained baseline. The same information-timing pattern may arise wherever a frozen model drafts a structured plan and a graph records what usually precedes what: dynamic workflow construction [36, 61], skill selection for agents [9, 15] and the routing of hand-offs in multi-agent systems [2, 39] all hinge on prerequisites the request never names, and are natural next tests for anchor-and-repair.

Limitations. The gain that survives to execution is real but bounded. Repaired plans lift claim coverage for all seven executing agents, by $+5.8$ on average, yet even the reference plan buys only $+9.4$ over the draft. The bound is set by the paradigm rather than by PITON. A plan committed before execution can do little for an agent that runs many turns and revises as it goes. The more promising setting is therefore an execution-interleaved harness, where PITON supplies a missing prerequisite at the moment the agent needs it rather than committing to a whole plan in advance.

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks with required disclosure. Large language models serve as annotators in the dissection: Mistral-Small, GPT-5.6 and Claude Sonnet 5 extract goals and assign occurrence roles under frozen prompts (Appendix B.1). Two zero-trajectory arms train on model-generated plans: Claude Opus drafts the teacher pseudo-gold and four open-weight drafters vote the consensus pseudo-gold (Appendix I). LLM coding agents implemented parts of the experiment code, launched experiment jobs under our instructions, and wrote the analysis scripts whose tables we read to interpret results. We also asked LLM reviewers to critique drafts of this paper and followed some of their suggestions on which analyses to add and how to present the diagnosis. We have not used generative AI tools to formulate or prove mathematical claims, and translation does not apply to this work. The repair corpus itself is synthesised by a fixed rule from gold plans, not by a model (Appendix D). Additionally, we used generative AI tools to edit the text for readability, to draft parts of the appendix from our notes and artefacts, to check citations against the cited papers, and to produce figures from measured data.

We have reviewed all AI-assisted work. LLM-written code was read and tested by the authors, and each cell of the main table is the mean of three independent runs. The LLM annotation is checked by automated assertions and by a blind comparison between two annotator models (Appendix B.1). Every reference was checked by the authors against the cited paper. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves no human subjects and collects no personal data. All experiments use three public benchmarks, UltraTool, ShortcutsBench and MCP-Atlas, under their released terms. The only labels we add are the goal and role annotations of Appendix B.1, produced by language models and released with the code; the one manual step, a blind re-annotation of a sample by an author, is an internal quality check rather than a study of participants. PITON proposes tools from a fixed catalogue and executes nothing itself, and the execution experiment runs inside the MCP-Atlas evaluation harness. A plan that includes the prerequisites a request leaves unsaid raises no risk beyond those of tool-augmented agents in general. We have no conflict of interest or sponsorship to disclose.

REPRODUCIBILITY STATEMENT

Every result uses one fixed configuration, set once on synthetic development drafts and shared by all benchmarks and drafters (§5.1); the thresholds, the corpus synthesis and the full feature list are in Appendix D. Each cell of Table 2 is the mean of three independent runs. Appendix F gives the rescoring protocol under which every external baseline was rerun three times against the same gold with the same scorer, and Appendix F.1 the training cost. The annotation protocol with its frozen prompts and quality checks is in Appendix B.1, the training intervention in Appendix B.5, the ScaffoldSlots drafting interface in Appendix E, the closed-source model identifiers and access dates in Appendix G, the zero-trajectory recipes in Appendix I, and the execution harness, its arms and the query selection in Appendix J. The anonymous repository linked in the abstract contains the code, the trained rerankers, the goal and role annotations, the drafts of every planner and the per-query predictions behind every table, together with the scripts that recompute the tables from them.

REFERENCES

[1] Chaithanya Bandi et al. MCP-Atlas: A large-scale benchmark for tool-use competency with real MCP servers. *arXiv:2602.00933*, 2026.

[2] Pratyay Banerjee and Ankit Chadha. Routed graph handoff: Adaptive format selection for multi-agent LLM delegation. *arXiv:2608.25277*, 2026.

[3] Sahil Bansal et al. Planning agents on an ego-trip: Leveraging hybrid ego-graph ensembles for improved tool retrieval in enterprise task planning. *arXiv:2508.05888*, 2025.

[4] Kinjal Basu et al. NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls. *arXiv:2409.03797*, 2024.

[5] Changxiao Cai and Gen Li. Confidence-based decoding is provably efficient for diffusion language models. *arXiv:2603.22248*, 2026.

[6] Wenjie Chen, Di Yao, Wenbin Li, Xuying Meng, Chang Gong, and Jingping Bi. GTool: Graph enhanced tool planning with large language model. In *ICLR*, 2026. arXiv:2508.12725.

[7] Yaxin Du et al. HyperTool: Beyond step-wise tool calls for tool-augmented agents. *arXiv:2606.13663*, 2026.

[8] Wei Fang and James R. Glass. Beyond single-shot: Multi-step tool retrieval via query planning. In *Findings of the Association for Computational Linguistics: ACL*, 2026.

[9] Dawei Fu et al. SE-GoS: Self-evolving graph-of-skills for skill library at scale. *arXiv:2609.08228*, 2026.

[10] Linfeng Gao, Yaoxiang Wang, Minlong Peng, Jialong Tang, Yuzhe Shang, Mingming Sun, and Jinsong Su. Tool graph retriever: Exploring dependency graph-based tool retrieval for large language models. *arXiv:2508.05152*, 2025.

[11] Xinyi Gao, Xinyu Ren, Junliang Yu, Tong Chen, Quoc Viet Hung Nguyen, and Hongzhi Yin. GRAFT: Graph-tokenized LLMs for tool planning. *arXiv:2605.11706*, 2026.

[12] Zhibin Gou et al. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *ICLR*, 2024. arXiv:2305.11738.

[13] Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. In *NeurIPS*, 2019. arXiv:1905.11006.

[14] Advait Gupta, NandaKiran Velaga, Dang Nguyen, and Tianyi Zhou. CoSTA*: Cost-sensitive toolpath agent for multi-turn image editing. *arXiv:2503.10613*, 2025.

[15] Yu Hao et al. HiSkill: Empowering LLM agents with hierarchical skill graphs. *arXiv:2607.25853*, 2026.

[16] Chengrui Huang et al. HTAA: Enhancing LLM planning via hybrid toolset agentization & adaptation. *arXiv:2604.10917*, 2026.

[17] Shijue Huang, Wanjun Zhong, Jianqiao Lu, et al. Planning, creation, usage: Benchmarking LLMs for comprehensive tool utilization in real-world complex scenarios. In *Findings of ACL*, 2024. arXiv:2401.17167.

[18] Yue Huang et al. MetaTool benchmark for large language models: Deciding whether to use tools and which to use. In *ICLR*, 2024. arXiv:2310.03128.

[19] Yan Jiang, Hao Zhou, Lizhong Gu, Tianlong Li, Ruinan Jin, Wanqi Zhou, and Ai Han. NaviAgent: Graph-driven bilevel planning for scalable tool orchestration. In *ICML*, 2026. arXiv:2506.19500.

[20] Guolin Ke et al. LightGBM: A highly efficient gradient boosting decision tree. In *NeurIPS*, 2017.

[21] Sehoon Kim et al. An LLM compiler for parallel function calling. In *ICML*, 2024. arXiv:2312.04511.

[22] Zae Myung Kim et al. The amazing agent race: Strong tool users, weak navigators. *arXiv:2604.10261*, 2026.

[23] Yansi Li and Zhuosheng Zhang. DiG-Plan: Mitigating early commitment for tool-graph planning via diffusion guidance. *arXiv:2606.05728*, 2026.

[24] Dawei Liu et al. Graph-of-skills: Dependency-aware structural retrieval for massive agent skills. *arXiv:2604.05333*, 2026.

[25] Hao Liu and Dongyu Li. SkillGraph: Graph foundation priors for LLM agent tool sequence recommendation. *arXiv:2604.19793*, 2026.

[26] Jiayu Liu et al. PlanBench-XL: Evaluating long-horizon planning of LLM tool-use agents in large-scale tool ecosystems. *arXiv:2606.22388*, 2026.

[27] Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. ToolNet: Connecting large language models with massive tools via tool graph. *arXiv:2403.00839*, 2024.

[28] Zhaoyang Liu et al. ControlLLM: Augment language models with tools by searching on graphs. *arXiv:2310.17796*, 2023.

[29] Jiarui Lu et al. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. *arXiv:2408.04682*, 2024.

[30] Pan Lu et al. Chameleon: Plug-and-play compositional reasoning with large language models. In *NeurIPS*, 2023. arXiv:2304.09842.

[31] Elias Lumer et al. Graph RAG-tool fusion. *arXiv:2502.07223*, 2025.

[32] Ziyang Luo et al. MCP-Universe: Benchmarking large language models with real-world model context protocol servers. *arXiv:2508.14704*, 2025.

[33] Aman Madaan et al. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023. arXiv:2303.17651.

[34] Guozhao Mo et al. LiveMCPBench: Can agents navigate an ocean of MCP tools? *arXiv:2508.01780*, 2025.

[35] Shen Nie et al. Large language diffusion models. *arXiv:2502.09992*, 2025.

[36] Boye Niu et al. Flow: Modularized agentic workflow automation. In *ICLR*, 2025. arXiv:2501.07834.

[37] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. *arXiv:2305.15334*, 2023.

[38] Fabian Pedregosa et al. Scikit-learn: Machine learning in Python. *JMLR*, 12, 2011.

[39] Chen Qian et al. Scaling large language model-based multi-agent collaboration. In *ICLR*, 2025. arXiv:2406.07155.

[40] Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. *arXiv:2504.13958*, 2025.

[41] Yujia Qin et al. Tool learning with foundation models. *arXiv:2304.08354*, 2023.

[42] Yujia Qin et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *ICLR*, 2024. arXiv:2307.16789.

[43] Changle Qu, Sunhao Dai, Xiaochi Wei, et al. Towards completeness-oriented tool retrieval for large language models. In *CIKM*, 2024. arXiv:2405.16089.

[44] Changle Qu et al. Tool learning with large language models: A survey. *arXiv:2405.17935*, 2024.

[45] Laura Ruis, Mitchell Stern, Julia Proskurnia, and William Chan. Insertion-deletion transformer. *arXiv:2001.05540*, 2020.

[46] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023. arXiv:2302.04761.

[47] Haiyang Shen et al. ShortcutsBench: A large-scale real-world benchmark for API-based agents. *arXiv:2407.00132*, 2024.

[48] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in hugging face. In *NeurIPS*, 2023. arXiv:2303.17580.

[49] Yongliang Shen et al. TaskBench: Benchmarking large language models for task automation. In *NeurIPS Datasets and Benchmarks*, 2024. arXiv:2311.18760.

[50] Kaustubh Vyas, Damien Graux, Yijun Yang, Sébastien Montella, Chenxin Diao, Wendi Zhou, Pavlos Vougiouklis, Ruofei Lai, Yang Ren, Keshuang Li, and Jeff Z. Pan. From an LLM swarm to a PDDL-empowered HIVE: Planning self-executed instructions in a multi-modal jungle. *arXiv:2412.12839*, 2024.

[51] Ruiyang Wang et al. GAVEL: Graph world models for verified and efficient long-horizon LLM task planning. *arXiv:2609.19315*, 2026.

[52] Zhenting Wang et al. MCP-Bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers. *arXiv:2508.20453*, 2025.

[53] Xiaolong Wei, Yuehu Dong, Xingliang Wang, Xingyu Zhang, Zhejun Zhao, Dongdong Shen, Long Xia, and Dawei Yin. Beyond ReAct: A planner-centric framework for complex tool-augmented LLM reasoning. *arXiv:2511.10037*, 2025.

[54] Xixi Wu et al. Can graph learning improve planning in LLM-based agents? In *NeurIPS*, 2024. arXiv:2405.19119.

[55] Zijian Wu et al. MCPMark: A benchmark for stress-testing realistic and comprehensive MCP use. *arXiv:2509.24002*, 2025.

[56] Weijia Xu and Marine Carpuat. EDITOR: An edit-based transformer with repositioning for neural machine translation with soft lexical constraints. *TACL*, 2021.

[57] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023. arXiv:2210.03629.

[58] Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion large language models. *arXiv:2508.15487*, 2025.

[59] Junjie Ye et al. ToolHop: A query-driven benchmark for evaluating large language models in multi-hop tool use. *arXiv:2501.02506*, 2025.

[60] Zian Zhai, Xingyu Tan, et al. HyperAgent: Planning and acting over tool–schema hypergraphs for tool-use LLM agents. *arXiv:2608.02650*, 2026.

[61] Jiayi Zhang et al. AFlow: Automating agentic workflow generation. In *ICLR*, 2025. arXiv:2410.10762.

[62] Shaorong Zhang, Longxuan Yu, Rob Brekelmans, Luhan Tang, Salman Asif, and Greg Ver Steeg. Generation order and parallel decoding in masked diffusion models: An information-theoretic perspective. *arXiv:2602.00286*, 2026.

[63] Zijing Zhang et al. ToolExpNet: Optimizing multi-tool selection in LLMs with similarity and dependency-aware experience networks. In *Findings of ACL*, 2025.

[64] Yuchen Zhuang et al. ToolChain*: Efficient action space navigation in large language models with A* search. In *ICLR*, 2024. arXiv:2310.13227.

A PER-METHOD COMPARISON WITH PRIOR WORK

Section 2 groups prior work by what it does with a tool dependency graph. Table 4 makes the comparison per method. The *graph source* column shows that our prior, adjacency in gold plans, is one the field already uses (GNN4Plan builds the same graph on UltraTool), so our claims concern how a graph is used rather than where it comes from. No prior method removes a tool it has already selected, which is the operator an over-generating drafter needs.

Table 4: Static tool planners that model tool dependencies. *Direction* is whether the graph constrains generation as the plan is emitted (forward) or is applied to a plan that already exists (post hoc).

Method	Graph source	How the graph is used	Direction
TaskBench	tool graph built by the benchmark: an edge where one tool's output type matches another's input type, or a temporal link (daily-life APIs)	sampled paths and subgraphs become gold trajectories and queries; at inference the LLM sees only the tool list and emits nodes and links	forward
GNN4Plan	the benchmark's tool graph, or adjacency in gold trajectories where none ships (UltraTool)	the LLM decomposes the query into steps; a GNN picks each step's tool from the previous node's neighbourhood	forward
GTool	request-specific subgraph of the benchmark's tool dependencies, assumed incomplete	a GNN encodes it into a graph token for a frozen LLM; trained with missing-dependency prediction	forward
GRAFT	the benchmark's tool graph, internalised into the LLM	each tool is a dedicated token; directed dependencies learned in the representation space, plus on-policy distillation	forward
DiG-Plan	none as input; the invocation graph is the output	a diffusion proposer emits a candidate tool set, an AR refiner predicts the edges	forward
ControlLLM	prebuilt tool graph linking tools through parameter and resource relations	Thoughts-on-Graph searches a solution path on the graph	forward
HIVE	none; the capabilities of the available models	compiles the request to a PDDL plan	forward
PITON	adjacency counts in the training split's gold plans	anchors on the draft, scores each one-hop neighbour against what this query still lacks, alternates insertion and deletion	post hoc

Other static planners. HuggingGPT [48] and Chameleon [30] emit modular programs, LLMCompiler [21] and Beyond ReAct [53] emit parallel DAGs, ControlLLM [28] and CoSTA* [14] search a tool path on a graph, and HIVE [50] compiles the request to a PDDL plan. All plan before execution and share the forward direction of Table 4.

Adjacent families that answer a different question. Dependency-aware *retrieval* decides which tools to hand downstream, not in what order to call them: Tool Graph Retriever [10] learns a pairwise dependency discriminator, Graph RAG-Tool Fusion [31] completes a dependency closure along a predefined graph, COLT [43] enforces completeness of the retrieved set, ToolQP [8] discovers prerequisites through multi-turn query planning, and ToolExpNet [63] learns a tool-experience network. *Argument wiring* takes the tool set as given and decides the execution order and which output feeds which input [4, 21, 59]; our question comes before that. *Execution-interleaved* methods decide the next tool after observing the last one's result: ToolRL [40] learns the calling policy by reinforcement learning, NaviAgent [19] picks the next call from a tool graph after each observation, and HyperAgent [60], the closest in motivation, restores tools that are operationally required but semantically remote. HyperAgent's dependencies come from expert-annotated parameter schemas and its gap from post-execution state, neither of which a static benchmark provides.

Benchmarks we do not evaluate on, and why. Tool-first benchmarks such as TaskBench, ToolBench and MetaTool [18, 42, 49] sample the tools first and then prompt a model to write the query, so almost every gold tool is named in the request (§5.1). Interactive benchmarks such as MCP-Bench, MCP-Universe, LiveMCPBench, MCPMark, ToolSandbox and PlanBench-XL [26, 29, 32, 34, 52, 55] score the answer or state after live execution, or a set or partial order of expected calls, and none carries a single reference sequence; MCP-Atlas provides human-verified trajectories with claim-level scoring, which is why §5.5 runs its harness. Benchmarks whose gold

plans are DAGs, such as The Amazing Agent Race [22] and TaskBench's DAG split, need a scorer and candidate rule over partial orders; ours treat linear chains.

Methods we do not compare against, and why. We do not compare with methods whose planner is the LLM itself fine-tuned on the benchmark's trajectories: fine-tuning brings a large gain but can change the base model's general ability and transfers poorly. GRAFT [11] trains the planner on its own trajectories and had no public implementation at the time of writing. HTAA [16] trains a hierarchical planner over agentised tool sets and HyperTool [7] moves execution to code blocks; both act at execution time. Ego-graph ensembles [3] and Graph-of-Skills [24] belong with the retrieval family above. GAVEL [51], concurrent with this work, verifies and repairs embodied plans against a graph world model in a different domain.

Diffusion LMs and edit-based generation. Discrete diffusion language models have developed from absorbing-state formulations and masked objectives to 7B- and 8B-scale models such as LLaDA and Dream [35, 58]. Confidence- or entropy-ordered decoding [5, 62] is their standard inference strategy. Edit-based generation with insertion and deletion as primitives goes back to the Levenshtein Transformer family [13, 45, 56]. Our edits are tool-level decisions by external rerankers around a frozen planner, not token-level moves of a trained generator. Verify-then-repair loops [12, 33] share the division of labour between a critic and a rewrite, but realise the rewrite by regenerating the sequence autoregressively.

B THE DISSECTION: PROTOCOL, FULL TABLES, AND THE TRAINING INTERVENTION

Section 3 tightens its conclusion in three steps, and this appendix follows the same progression: the annotation protocol (Appendix B.1), edge endpoints and missed-edge decomposition (Appendix B.2), the paired design with every per-cell result (Appendix B.3), the evidence distribution behind the dependency-tool gap (Appendix B.4) and the training intervention (Appendix B.5). Every stratified number uses the predictions and the scorer of Table 2, with predictions de-duplicated by first occurrence and node and edge counts accumulated micro over set semantics.

B.1 ANNOTATION PROTOCOL, QUALITY CONTROL AND DEPENDENCY SUB-ROLES

The annotator is a model. All three passes below are run by an LLM; no role label is written by hand. UltraTool and ShortcutsBench use Mistral-Small as the Pass A and Pass B teacher, with GPT-5.6 for selective re-annotation and for the Pass C adjudications; MCP-Atlas uses Claude Sonnet 5 for its 150 newly annotated queries. "Annotator" below therefore names a model and a prompt version. The roles only stratify results and never enter any score; every number is computed against the benchmark's own gold plan.

Three passes, each with restricted input. Pass A sees **only the query** and returns the goals named by the user. For each goal, it records the literal form, a normalised form and the source spans. Pass 0 sees the query and the raw call trajectory and returns a predictability label. Pass B sees the frozen Pass A goals and the de-duplicated tool nodes and produces the goal-to-tool alignment, the role of every occurrence, and the dependency edges with their evidence. The frozen goals may not be rewritten, merged, split or reordered in Pass B; where the gold plan does not match the goal segmentation the mismatch is recorded as a disposition. Four frozen boundary rules define a goal. A goal is one thing the user explicitly asked for that can stand on its own. Search, identifier resolution, format conversion and login are hidden steps unless the user names them literally. A goal may require several tools and is not split merely because it takes several steps. Parameters, constraints, formats and result attributes are not goals.

Roles. The annotation enumerates five roles that the main text collapses to two. A `DIRECT_REALIZER` is a primary tool: the query names it or directly implies its effect. The next three roles are dependency tools. A `HIDDEN_PREREQUISITE` must run before a direct tool. A `HIDDEN_BRIDGE` connects the middle of a plan by converting a format, entity or parameter, and a `HIDDEN_POSTCHECK` confirms a direct effect afterwards. `SHARED_ACROSS_GOALS`, a de-duplicated tool node reused by two goals, occurs only on MCP-Atlas and counts as a dependency here. Table 5 shows that the composition differs sharply across benchmarks, which matters because a graph prior tells more about prerequisites than about bridges.

Table 5: Dependency sub-roles by benchmark, counted over query × unique gold tool. On MCP-Atlas the shared nodes number 124 at the raw occurrence level and 61 after de-duplication.

Benchmark	primary	prerequisite	bridge	postcheck	shared	dependency total
UltraTool	936	181 (82.3%)	6	33	n/a	220
ShortcutsBench	363	166 (42.2%)	205 (52.2%)	22	n/a	393
MCP-Atlas	262	312 (81.5%)	10	0	61	383

Quality control. Two assertions are re-run every time the analysis loads the annotation. A role conflict is the same query-tool pair receiving two different roles, and an index-block mismatch is a role index block that fails to reproduce the local tool plan position by position. Both are zero on all three benchmarks, over 1,156 / 756 / 708 occurrence roles and 1,156 / 756 / 645 unique query-tool pairs, giving 2,620 role-bearing tool occurrences in total. The protocol also requires every tool node to be owned by a goal or explicitly marked as an unexplained reference. Every dependency edge must carry evidence: either the target's parameters contain an entity or identifier produced by the source, or the reasoning explicitly consumes the source's result.

Three provenance facts. First, 150 of MCP-Atlas's 166 scored queries are newly annotated and 16 come from an earlier pilot with a different annotator model; on those 16, a blind comparison gives 77.8% role agreement ($\kappa = 0.591$, or 0.713 once goal granularity is set aside). Second, where `SHARED_ACROSS_GOALS` is placed matters: counting shared nodes as dependencies moves between-group values by 2 to 5 points and is a main source of MCP-Atlas's null result in the paired reading (odds ratios 1.13 to 1.92 with these nodes, 1.76 to 3.09 without), because reused nodes are mostly infrastructural calls recalled like primary tools. Third, one author relabelled goal and role for 70 queries (30 UltraTool, 20 ShortcutsBench, 20 MCP-Atlas; 207 occurrences), blind to the LLM labels. Agreement on the primary-versus-dependency role is 89.4% (Cohen's $\kappa = 0.78$, 95% query-bootstrap CI $[0.68, 0.87]$; 0.74, 0.77 and 0.78 per benchmark), and on the goal an occurrence belongs to 78%, 97% and 95%. The 22 disagreements split evenly in direction; the recurring case is a selection or lookup step the query names that the LLM labelled hidden. Restricted to the audited queries, the primary-versus-dependency recall gap of Insight 1 keeps its sign for every drafter under either labelling. Moreover, the UltraTool bridge count ($n = 6$) and both postcheck counts (33 and 22) are too small for sub-role conclusions, which are supported only for UltraTool prerequisites, ShortcutsBench bridges and MCP-Atlas prerequisites.

B.2 Endpoint composition of edges, and where a missed edge is lost

The edge gap in Table 1 reverses on two MCP-Atlas cells because the CROSS bucket is not the same object on the three benchmarks. That a WITHIN edge carries at least one dependency endpoint holds everywhere; that a CROSS edge joins two named primary tools holds only on UltraTool. This is measured, not definitional, because a CROSS edge ends at the *first* tool of the next goal and a goal's first tool is very often its own hidden prerequisite.

The share of CROSS edges carrying a dependency endpoint, 9.15 / 35.56 / 63.11%, orders the same way as the mean draft-side CROSS minus WITHIN gap, $+28.4$ / $+16.8$ / -3.2. Three points are not statistical evidence, but this is a mechanical decomposition, since edge recall is built from endpoint recall. MCP-Atlas's CROSS bucket was never an easy control group. Endpoint composition does not explain the gap away either. The paired design of Appendix B.3 fixes the edge identity and thus the endpoint tools. It still finds odds ratios of 2.45 to 7.34 in the three UltraTool cells whose intervals exclude one. Excluding MCP-Atlas's shared nodes leaves the direction unchanged, at 89.11% (n=358) and 36.67% (n=60) for the first two columns.

B.3 The paired design and its per-cell numbers

Design. Table 1 compares one pile of WITHIN edges with another of CROSS edges, so identity composition alone could explain the difference. The paired design keeps only the edge or tool identities that occur in both usages and compares each identity against itself, with a Mantel-Haenszel odds ratio stratified by identity as the primary estimator. Units match the scorer: an edge is a query paired with an ordered gold adjacent pair, a tool a query paired with a unique gold tool, under the

same de-duplicated set semantics. Thresholds of one, two and three occurrences per usage are all reported, together with the Wilcoxon signed-rank test, an exact sign test and the Mantel-Haenszel χ^2; the bootstrap resamples identities ($B = 2000$).

Table 6: The same tool in two roles, on the unrepaired draft, threshold ≥ 1. A positive paired difference means the tool is recalled less often as a dependency than as a primary. Confidence intervals resample identities.

Benchmark	Drafter	ids	paired diff [95% CI]	up/down/tie	Wilcoxon p	MH OR [95% CI]
UltraTool	Dream	31	$+35.86$ [$+20.79$, $+50.83$]	20/4/7	0.0004	7.52 [4.26, 13.26]
	Qwen2.5	31	$+24.58$ [$+9.46$, $+39.84$]	16/5/10	0.0045	3.83 [2.26, 6.51]
	LLaDA	31	$+20.86$ [$+5.27$, $+37.25$]	16/7/8	0.018	4.41 [2.56, 7.61]
	Llama-3.1	31	$+4.68$ [-12.31, $+21.45$]	11/12/8	0.63	2.33 [1.29, 4.21]
ShortcutsBench	Dream	46	$+14.07$ [-0.41, $+28.65$]	23/11/12	0.080	2.35 [1.38, 4.01]
	Qwen2.5	46	$+14.52$ [$+1.91$, $+26.80$]	22/11/13	0.035	1.97 [1.15, 3.40]
	LLaDA	46	$+21.95$ [$+8.17$, $+36.87$]	23/8/15	0.0061	3.14 [1.84, 5.38]
	Llama-3.1	46	$+18.37$ [$+4.90$, $+31.65$]	24/9/13	0.0091	1.96 [1.16, 3.31]
MCP-Atlas	Dream	41	$+16.18$ [$+3.61$, $+28.34$]	23/12/6	0.032	1.67 [0.95, 2.92]
	Qwen2.5	41	$+2.46$ [-9.38, $+13.90$]	17/17/7	0.62	1.13 [0.65, 1.98]
	LLaDA	41	$+16.59$ [$+3.87$, $+30.54$]	17/13/11	0.056	1.92 [1.07, 3.44]
	Llama-3.1	41	$+2.04$ [-8.15, $+10.74$]	17/10/14	0.17	1.55 [0.80, 3.00]

On UltraTool and ShortcutsBench all eight odds ratios exceed one with intervals excluding one, from 1.96 to 7.52: the same tool is recalled at between one seventh and one half the odds as a hidden dependency as it is as a primary. The result is insensitive to the threshold (UltraTool with Dream 7.52 / 7.43 / 7.20, with Qwen2.5 3.83 / 3.75 / 3.57; ShortcutsBench with Dream 2.35 / 2.44 / 3.62, with Qwen2.5 1.97 / 1.87 / 2.15). MCP-Atlas is a null result, 1.13 to 1.92 with only the LLaDA cell excluding one, about half of which comes from the shared-node convention of Appendix B.1.

Table 7: The same edge in two usages, on the unrepaired draft, threshold ≥ 1, UltraTool only. A positive paired difference means the edge is recalled less often when it serves a single goal.

Drafter	ids	paired diff [95% CI]	up/down/tie	Wilcoxon p	MH OR [95% CI]
Dream	41	$+34.39$ [$+19.47$, $+48.29$]	24/7/10	0.0002	7.34 [3.79, 14.21]
Qwen2.5	41	$+20.75$ [$+4.97$, $+36.30$]	21/9/11	0.021	2.45 [1.39, 4.33]
LLaDA	41	$+33.81$ [$+18.37$, $+49.25$]	25/6/10	0.0003	6.36 [3.20, 12.64]
Llama-3.1	41	$+3.83$ [-10.65, $+17.83$]	13/13/15	0.61	1.62 [0.87, 3.01]

All four odds ratios exceed one and three of four intervals exclude it: the same edge $a \to b$ is recalled at between one seventh and three fifths the odds within a goal as across a boundary, and thresholds barely move it (Dream 7.34 / 9.81 / 8.69, Qwen2.5 2.45 / 2.16 / 2.31).

B.4 Evidence distribution: why dependency tools are hard

The paired design shows the gap follows usage; this section shows why, namely that the query leaves no lexical evidence for a dependency tool. We use a light retriever that trains no parameter and uses no dependency annotation, aggregating the training goals by their primary tool into centroids and ranking the query against them by cosine.

Table 8: Retrieval of a goal's own tools. The same top-3 that finds nine primary tools in ten finds fewer than half of the dependencies.

Benchmark	Top-3 primary	Top-3 non-primary	Top-5 primary	Top-5 non-primary	Top-5 full cover
UltraTool	93.4%	43.2%	95.1%	61.4%	59.2%
ShortcutsBench	66.4%	23.5%	72.1%	31.5%	16.7%

B.5 TRAINING INTERVENTION: ERASING AN EDGE FROM THE TRAINING SET

Design. A stronger model might simply understand these dependencies. This experiment fixes model capacity and all other training exposure and manipulates only whether the edge was ever seen. We take the 30 most frequent adjacent tool pairs of UltraTool (frequency 33 to 292) and split them at random into two disjoint groups of 15 that serve as replication checks. The *plain* arm deletes every training sample containing the pair; the *matched control* arm keeps the same budget but retains the exposure; a *base* arm with no training gives a reference. Construction gates enforce zero leakage, keep at least one plain-arm training example for every tool and require at least ten test occurrences per held-out edge; failed gates trigger deterministic substitution (three edges in one group, one in the other). The backbone is Qwen3-8B under QLoRA for five epochs at 8192 context with thinking off.

Table 9: Erasing an edge from the training set, two disjoint groups of 15 edges.

Group	Arm	held-edge recall (macro)	edge F1 (generalisation split)	trajectory EM
g1	plain	.428	.470	.387
	matched control	.789	.759	.707
g2	plain	.465	.518	.421
	matched control	.825	.793	.755
mean	plain → control	.447 → .807	.494 → .776	.404 → .731

The two groups give held-edge gaps of $+.361$ and $+.360$, so this is not one unlucky partition. A pilot on eight edges under the same protocol adds a confound control. On the split where both arms have seen the edges, control minus plain edge F1 is -2.5; on the generalisation split, it is $+25.4$. This conservative difference in differences of $+28.0$ points rules out sample volume and distribution as explanations.

Split by usage. Over 821 sample-by-edge occurrences drawn from 750 generalisation-split records, stratified by how the edge is used in the query, Table 10 separates the two. A CROSS usage largely survives erasure, at 64.7% recall (72% of the with-exposure level), while a WITHIN usage falls to 20.8%, barely above the untrained 18.3%.

Table 10: The cost of erasure, by how the held-out edge is used in the test query ($n = 312$ WITHIN and $n = 456$ CROSS occurrences).

Training setting	WITHIN usage	CROSS usage	CROSS − WITHIN
base (no training)	18.3%	51.5%	+33.3
plain (edge erased)	20.8%	64.7%	+43.9
matched control (same budget, exposed)	70.5%	89.9%	+19.4
cost of erasure	−49.7	−25.2	
retained without exposure	29.5%	72.0%	

Conclusion and limitations. The WITHIN gap is not a capacity problem: a model may know two tools separately and still not know that this task requires A before B. Five limitations bound the claim. The backbone and cohort differ from Table 2, so this is mechanistic evidence rather than part of that table. It runs only on UltraTool, the one benchmark with training gold frequent enough for a zero-exposure control. The held-out edges are high-frequency edges whose erasure also drains multi-step samples from a few domains, which the matched control is for. The usage labels come from this experiment's own goal annotation, with 53 of the 821 occurrences excluded as ambiguous. Per-edge coverage is 13 to 64, so only aggregate and paired readings are used; 26 of the 30 edges show a positive exposure benefit and four do not.

C THE REPAIR LOOP, ROUND BY ROUND

One round is not enough. It captures 47 to 80% of the net gain on MCP-Atlas ($+8.58$ to $+19.55$ edge F1), yet all four drafters keep improving into the third round and three into a fourth, to final

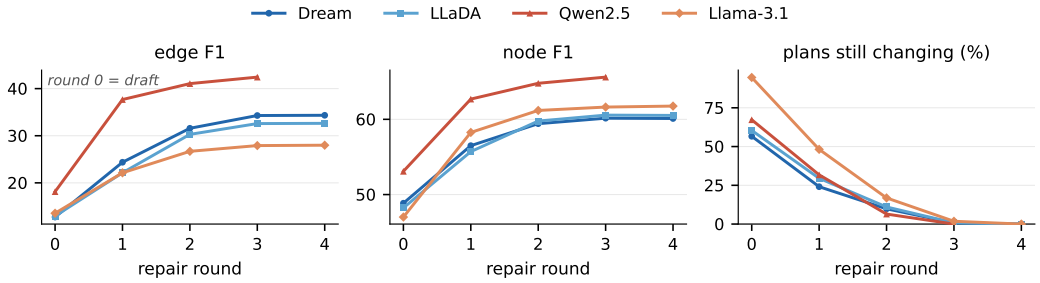


Figure 5: The number of repair rounds is decided by the plan, not by a budget. Four drafters on MCP-Atlas (166 queries) under the frozen configuration. Round 0 is the draft; every later round first lets the deletion module clear untrustworthy anchors, then rebuilds candidates on the cleaned plan and inserts one tool. The right panel is the share of plans that the next round still changes, and its reaching zero is the stopping condition. Every point is the mean of the same three runs Table 2 averages. For the two autoregressive drafters the last round of each curve is the number that table reports; for the two diffusion drafters that table reports the model-executed variant of Appendix E.4, which ends 1.4 and 1.2 points above the reranker-executed curves shown here.

gains of +21.60 (Dream), +24.31 (Qwen2.5), +19.67 (LLaDA) and +14.44 (Llama-3.1). A tool restored in one round becomes an anchor in the next, and only then does its own prerequisite enter a one-hop neighbourhood. Plans stop at different rounds, 9 of Llama-3.1's 166 after round 0 against 72 for Dream, so the number of rounds is a property of the query.

MCP-Atlas needs the loop because of its composition: primary tools are only 38.1 to 47.0% of its gold occurrences, against 81% on UltraTool, so more than half of its gold tools are unnamed dependencies, and its one-hop coverage is lowest. One-hop coverage, the share of a draft's missing gold tools that lie one hop from an anchor in G, is 85.7%, 79.5% and 55.8% on UltraTool, ShortcutsBench and MCP-Atlas, so on MCP-Atlas a single round cannot reach almost half of what is missing until an earlier insertion has become an anchor.

D THE TWO RERANKERS: CORPUS, CALIBRATION, FEATURES AND THE OPERATOR ABLATION

Both rerankers are gradient-boosted trees that fit on CPU in under two minutes per benchmark. All supervision is synthesised automatically from the target benchmark's training gold plans, without human annotation. The only quantities taken from the drafter's own drafts are the pooled corruption rates of Table 11; no per-query label of the evaluation split enters the corpus.

D.1 SYNTHESISING THE TRAINING CORPUS

Each training gold plan is corrupted in two ways to form a pseudo-draft: dropping gold tools creates insertion positives, injecting non-gold tools creates deletion positives, and the labels say which tools to restore and remove. A real drafter keeps the tools the query names and drops the ones it does not, so a uniform drop rate would get the size of the synthesised gap right and its shape wrong. We therefore bin each gold occurrence by evidence strength, the maximum cosine between the tool description and the query's clause vectors. A per-occurrence label of this kind exists only for UltraTool, whose training occurrences fall 45.08% almost unmentioned, 8.41% marginal and 46.51% explicitly named; every benchmark's own evidence distribution is cut into three bins at these proportions. The retention rates, measured on UltraTool drafts at 0.431, 0.728 and 0.874, are shared by all three benchmarks and rescaled so that expected retention still matches the per-cell keep rate; otherwise the change would alter both the size and the shape of the gap.

The two rates read the cell's real drafts. MCP-Atlas keeps as little as 0.390 (omission), while UltraTool with Llama-3.1 injects 0.542 (over-selection), the two pathologies that §5.3 and §5.4 run

Table 11: Calibration values, all measured by the script on that cell rather than chosen. *keep* is the gold retention rate of that cell's real drafts and *extra* the share of drafted tool occurrences that are not in gold, both pooled over the cell; the synthesiser injects into a pseudo-draft of length k a number of non-gold tools equal to $extra \cdot k/(1 - extra)$ rounded stochastically and capped at $\min(k, 6)$. The pair thus writes into the corpus whether that drafter's disease is omission or over-selection. *exact* is the share of that cell's drafts that already match gold, which supplies the negatives that stop the dependency reranker from learning that one more tool can always be added.

Benchmark	Drafter	plans	keep	extra	bins after rescaling	exact	ins. AUROC	del. AUROC
UltraTool	Dream	2339	0.682	0.167	0.444 / 0.750 / 0.901	0.442	.955	.975
	LLaDA	2339	0.707	0.227	0.461 / 0.778 / 0.934	0.403	.932	.966
	Qwen2.5	2339	0.700	0.315	0.455 / 0.769 / 0.924	0.331	.945	.981
	Llama-3.1	2339	0.735	0.542	0.479 / 0.808 / 0.970	0.145	.946	.981
ShortcutsBench	Dream	1114	0.509	0.198	0.331 / 0.560 / 0.672	0.165	.874	.921
	LLaDA	1114	0.534	0.237	0.348 / 0.587 / 0.705	0.152	.888	.930
	Qwen2.5	1114	0.555	0.407	0.361 / 0.610 / 0.733	0.122	.878	.945
	Llama-3.1	1114	0.582	0.477	0.379 / 0.640 / 0.769	0.093	.890	.943
MCP-Atlas	Dream	330	0.390	0.353	0.254 / 0.428 / 0.514	0.006	.892	.855
	LLaDA	330	0.406	0.416	0.264 / 0.446 / 0.536	0.018	.885	.886
	Qwen2.5	330	0.481	0.417	0.313 / 0.529 / 0.635	0.006	.871	.897
	Llama-3.1	330	0.703	0.647	0.458 / 0.774 / 0.929	0.000	.905	.917

into. Almost no MCP-Atlas draft is already complete (exact-match rate 0.000 to 0.018, against 0.145 to 0.442 on UltraTool). Distractors in the deletion corpus follow the measured composition of each cell's wrong selections. One rule is applied per cell; what branches is the benchmark's failure profile.

D.2 Thresholds and operating points

Calibration uses the training and development splits only and never touches the test split. The development drafts are themselves synthetic, because a development split has no decoder output. Where the dataset ships its own development split we use it, and otherwise we cut the last 20% of the fitting half. Development drafts carry repeated tools too.

Table 12: The operating point, shared by all three benchmarks. Across benchmarks we refit only the dependency graph and the two rerankers.

Quantity	Value	How it was set
insertion gate	0.6, $+0.15$ per round	development grid
deletion threshold	0.5	development grid
deletions per round	at most 6	fixed, not tuned per cell
rounds	at most 8, stop at a fixpoint	fixed, not tuned per cell
oscillation	undo the round and stop	fixed
orchestration	loop, deletion first	development grid over four orchestrations
over-deletion guard	25th length percentile	stop deleting below it
placement features	forced on	pinned for all three benchmarks, not grid-selected

D.3 Features

The graph G counts adjacent pairs in the training gold plans, with $G[a \to b]$ the ordered adjacent count, $m(t)$ the tool marginal and N their total; $k = |\hat{P}|$. A candidate is a triple (a, t, d), $d \in \{\textit{before}, \textit{after}\}$, with c its count in direction d and r the reverse. v_t is the frozen sentence embedding of a tool description, q the query embedding and F the matrix of query-clause vectors, clauses being cut at sentence punctuation and at the connectives *then*, *finally*, *after that*, *next*, *also* and *and*, with the whole query kept as one further row. All are L_2 normalised, so an inner product is a cosine. Eleven columns are replaced by their empirical quantile under the benchmark's own distribution,

written ecdf, and k is divided by the mean plan length; every other column is dimensionless, which is what makes the rerankers transferable across catalogues (§5.4).

Table 13: The dependency reranker's 23 features. Semantic features embed the tool description; on MCP-Atlas, which ships none, the tool identifier read as words stands in for it.

Feature	Definition
graph evidence (11)	
`log_edge_count`	$\mathrm{ecdf}(\log(1+c))$
`pmi`	$\mathrm{ecdf}\Big(\log \frac{c/N}{(m(a)/N)(m(t)/N)}\Big)$, the log lift
`cond_prob`	$c/m(a)$
`log_tool_marginal`	$\mathrm{ecdf}(\log(1+m(t)))$
`rank_in_draft`	rank of this candidate by c within the query's candidate set, divided by its size
`n_candidates`	ecdf of the candidate set size
`k_draft`	k divided by the mean training plan length
`length_cdf`	$F_{\mathrm{len}}(k)$, the empirical CDF of training plan length
`anchor_pos_frac`	$\mathrm{pos}(a)/\max(k-1,1)$
`dir_after`	$\mathbf{1}[d = \textit{after}]$
`log_reverse_edge`	$\mathrm{ecdf}(\log(1+r))$
placement evidence (4), informative only when a tool has several anchors	
`anchor_rank_for_tool`	rank of a by c among all anchors nominating t, divided by their number
`n_anchors_for_tool`	number of anchors nominating t, divided by k
`anchor_at_edge`	$\mathbf{1}[\mathrm{pos}(a) \in \{0, k-1\}]$
`dir_log_ratio`	$\mathrm{ecdf}(\log(1+c) - \log(1+r))$
semantic evidence (8)	
`sim_tool_query`	$v_t \cdot q$
`sim_tool_draft_max`	$\max_{a \in \hat{P}} v_t \cdot v_a$
`sim_margin`	$v_t \cdot q - \mathrm{mean}_{t' \in \mathrm{Cand}} v_{t'} \cdot q$
`uncovered_evidence`	$\max_f \left[F_f \cdot v_t - \max_{a \in \hat{P}} F_f \cdot v_a\right]$
`draft_coverage`	$\mathrm{mean}_f \max_{a \in \hat{P}} F_f \cdot v_a$
`tool_best_fragment`	$\max_f F_f \cdot v_t$
`sim_anchor_query`	$v_a \cdot q$
`sim_tool_anchor`	$v_t \cdot v_a$

Table 14: The deletion reranker's 17 features, for position i of the draft $\hat{P} = (t_1, \ldots, t_k)$, with $\textit{left} = G[t_{i-1} \to t_i]$, $\textit{right} = G[t_i \to t_{i+1}]$ and $\textit{bridge} = G[t_{i-1} \to t_{i+1}]$, all zero at a boundary. Write $s_j = v_{t_j} \cdot q$.

Feature	Definition
structural signals (11)	
`log_left_edge`, `log_right_edge`	$\mathrm{ecdf}(\textit{left})$, $\mathrm{ecdf}(\textit{right})$
`log_pair_support`	$\mathrm{ecdf}(\textit{left} + \textit{right})$, quantiled against the single-edge reference
`both_sides_supported`	$\mathbf{1}[\textit{left} > 0 \wedge \textit{right} > 0]$
`orphan`	$\mathbf{1}[\textit{left} = 0 \wedge \textit{right} = 0]$
`log_bridge`	$\mathrm{ecdf}(\textit{bridge})$, the bridge test specific to deletion
`log_tool_marginal`	$\mathrm{ecdf}(m(t_i))$
`k_draft`, `length_cdf`	k over the mean length, and $F_{\mathrm{len}}(k)$
`position_frac`	$i/\max(k-1,1)$
`duplicate_occurrence`	$\mathbf{1}[\,t_i$ occurs more than once in $\hat{P}\,]$
semantic signals (6)	
`sim_to_query`	s_i
`sim_minus_plan_mean`	$s_i - \mathrm{mean}_j s_j$
`sim_rank_in_plan`	$\lvert\{j : s_j > s_i\}\rvert/(k-1)$
`sim_to_plan_max`	$\max_{j \neq i} v_{t_i} \cdot v_{t_j}$
`covers_own_fragment`	$\max_f F_f \cdot v_{t_i}$
`fragment_uniquely_covered`	$\max_f \left[F_f \cdot v_{t_i} - \max_{j \neq i} F_f \cdot v_{t_j}\right]$

D.4 Ablation: insertion, deletion, and the loop

Table 15: One frozen pair of rerankers under four orchestrations, edge F1. Same configuration, drafts and three-run means as Table 2, so the draft and loop columns reproduce it cell for cell for the autoregressive drafters; for the diffusion drafters the loop column is the reranker-executed loop on which that table's model-executed rows build (Appendix E.4).

Benchmark	Drafter	draft	insert only	delete only	loop
UltraTool	Dream	54.28	60.45	55.68	**62.18**
	LLaDA	51.11	53.52	53.90	**57.00**
	Qwen2.5	42.35	44.31	49.80	**52.67**
	Llama-3.1	21.21	21.38	39.20	**47.22**
ShortcutsBench	Dream	27.84	35.14	28.57	**35.95**
	LLaDA	23.21	29.02	24.08	**30.17**
	Qwen2.5	21.06	25.33	24.33	**31.12**
	Llama-3.1	16.60	18.33	19.74	**23.84**
MCP-Atlas	Dream	12.74	22.53	13.43	**34.34**
	LLaDA	12.96	21.50	13.20	**32.63**
	Qwen2.5	18.13	30.97	22.37	**42.44**
	Llama-3.1	13.55	15.43	17.89	**27.99**

The loop beats either single operator in all 12 cells, and its advantage is not the sum of the two. On MCP-Atlas with Dream, insertion alone gains $+9.8$ and deletion alone $+0.7$, whereas the loop gains $+21.6$. Once deletion cleans the anchors, the dependency reranker sees a different plan with different candidates and reference positions. This is the mechanism §4.1 describes.

What each operator contributes tracks the drafter's pathology. UltraTool with Llama-3.1 over-selects badly, with an injection rate of 0.542. Deletion alone captures most of the gain at $+18.0$, while insertion alone adds $+0.2$. The three MCP-Atlas cells that mostly omit, with retention from 0.39 to 0.48, show the reverse pattern. An additive-only method therefore has no route forward on an over-selecting drafter; deletion is the operator that Appendix A shows every existing graph method lacks. This also explains why the delete-only arm holds up under the zero-trajectory supervision of Appendix I. Judging a tool redundant needs only local evidence within the plan; insertion needs the dependency knowledge itself.

E Drafting with a diffusion LM: ScaffoldSlots and anchor purity

E.1 What ScaffoldSlots is

The interface lets a diffusion LM that was never fine-tuned on this task be edited. Two obstacles stand in the way: tool names span several tokens, so replacing or inserting one changes the sequence length and prevents a local edit, and free generation produces out-of-catalogue tools and malformed structures. ScaffoldSlots maps every candidate tool to a single-token pointer holding one slot, so inserting or deleting a tool is a slot-level operation. Before decoding, a length probe asks the drafter how many steps the request describes, reads the most probable digit c from one appended mask slot, decodes once at each of $\{c-1, c, c+1\}$ and keeps the plan with the highest mean per-slot log-probability. The probe counts named actions, so the draft covers the named goals and leaves prerequisites to the repair stage. Deletion is always applied by the deletion reranker directly, because neither diffusion LM can re-mask a committed slot.

E.2 The scaffold buys legality, not accuracy

Setting: UltraTool, 489 queries, Dream-v0-Instruct, no adapter, 48 single-token pointers in one candidate set.

Table 16: Output interface against plan quality on a frozen drafter.

Output interface	parse valid	catalogue violations	edge F1	node F1
atomic pointer, free generation, 24-slot canvas	30.88%	—	18.56	26.91
atomic pointer, ScaffoldSlots	100%	0%	47.91	72.54
tool strings, free generation, 64-token canvas	96.73%	7.98%	47.16	70.55

Within the pointer representation the scaffold is worth $+29.35$ edge F1, because atomic pointers collapse under free generation. Against free generation over tool strings it is worth only $+0.75$. What it buys is not accuracy but addressable slots, which let the plan be edited in place.

E.3 Fixing the plan length before decoding is free for a DLM and costly for an AR model

The same length probe transplanted to an autoregressive model: one forward pass answers how many steps the request takes, and the plan is then decoded at a length fixed near that answer. Both sides are measured against free decoding to end-of-sequence.

Table 17: The cost of fixing the plan length before decoding, edge F1 on UltraTool.

Architecture	free to EOS	length fixed first	Δ
Dream, ScaffoldSlots fixed canvas	47.16	47.91	$+0.75$
AR, tool names, told k	39.01	31.71	-7.30
AR, pointers on a fixed canvas	39.01	26.12	-12.89

A fixed canvas is nearly free for the DLM and costs the AR model about seven points, and its length accuracy falls from 64.7% to 29.6%. An AR model commits token by token from left to right and cannot place the tools it is sure of first; being told k only forces it to fill k slots once it has nothing left to say, and its per-slot hit rate falls by 35.3 points from the first slot to the fourth, whereas the fourth slot stays level under free decoding. A diffusion LM commits in entropy order with every slot seeing all the others, so the fixed canvas costs it nothing. A fixed-length canvas is therefore not a general recipe; it is free only on a decoder that does not commit in sequence order.

E.4 Model-executed versus reranker-executed repair

Table 2 reports the two diffusion drafters with the model executing each insertion: the reranker's gated candidates and their positions are written into the prompt, a mask slot opens at each, and the model fills those it agrees with. The reranker-executed loop instead applies the reranker's top candidate directly; it is what the autoregressive rows, the per-round curves (Appendix C) and the ablation (Appendix D.4) use. Both variants run on the same frozen rerankers, the same three drafts per cell and in the same GPU process, so their difference isolates the model's own judgement. Replaying the reranker-executed loop on the GPU reproduces the offline plans for all but 0 to 4 queries per cell, under 1%, which moves UltraTool's edge F1 by $+0.47$ and -0.34 and the other cells by at most 0.04; the reranker-executed column below is that replay.

The model's judgement adds $+0.51$ edge F1, $+0.49$ node F1 and -0.002 NED on average: clearly positive on MCP-Atlas, slightly positive on UltraTool and flat on ShortcutsBench. The gain is smaller than the run-to-run spread of the reranker-executed loop itself (mean three-run range 1.06), and by edge F1 the model's overrides win 71 queries and lose 64 (sign test $p = 0.61$). The rerankers do the work; the diffusion interface adds a small, consistent right of veto. We read the small gain as a limit of the drafter rather than of the interface. Repairing a plan is too hard for a model never trained to do it: the missing dependency is knowledge the planner never held (§3), so it has little to add when asked whether a graph-supplied candidate belongs; what it can still judge is whether the candidate fits the plan it wrote, and that is the veto. The diffusion drafters' edge therefore comes from drafting rather than from executing the repair: entropy-ordered decoding leaves purer anchors (Appendix E.5), and the same rerankers repair a purer draft to a higher score in five of six matched cells (Table 2).

Table 18: Reranker-executed against model-executed repair on the six diffusion cells, three-run means under the frozen configuration. Δ is model minus reranker.

		reranker executes			model executes			
Benchmark	Drafter	eF1	nF1	NED	eF1	nF1	NED	Δ eF1
UltraTool	Dream	62.65	82.43	.2509	62.74	82.59	.2499	$+0.09$
	LLaDA	56.66	80.96	.2836	57.07	81.16	.2807	$+0.41$
ShortcutsBench	Dream	35.92	65.17	.4292	35.96	65.14	.4293	$+0.04$
	LLaDA	30.15	65.32	.4754	30.02	65.31	.4755	-0.13
MCP-Atlas	Dream	34.34	60.11	.5649	35.78	60.99	.5603	$+1.44$
	LLaDA	32.64	60.53	.5746	33.83	62.28	.5708	$+1.19$
mean								$+0.51$

E.5 Anchor purity: diffusion drafts omit, autoregressive drafts over-select

Because PITON anchors on the draft, what matters is not the draft's score but how many of the tools it commits are right. A wrong selection becomes an anchor and pulls one-hop candidates from the wrong place, whereas an omission is simply work left for the dependency reranker. Purity here is node precision, and the last column is the share of queries whose draft contains no wrong selection at all.

Table 19: Draft-side purity on the full test splits, UltraTool 489, ShortcutsBench 237, MCP-Atlas 166, on the frozen drafts of Table 2. Counts are per-query totals and so are reported for one run rather than averaged; across the drafters whose three repeats are not byte-identical, no column here moves by more than 1.8 points between runs.

Drafter	Benchmark	purity P	recall R	wrong	missed	wrong/missed	clean drafts
Dream	UltraTool	83.8	74.6	168	297	0.57	71.0%
LLaDA	UltraTool	77.8	77.4	258	264	0.98	63.6%
Qwen2.5	UltraTool	69.1	76.6	400	273	1.47	47.2%
Llama-3.1	UltraTool	46.0	80.3	1099	230	4.78	16.4%
Dream	ShortcutsBench	80.2	50.9	96	376	0.26	68.8%
LLaDA	ShortcutsBench	76.3	53.4	127	357	0.36	63.3%
Qwen2.5	ShortcutsBench	59.3	55.5	292	341	0.86	36.7%
Llama-3.1	ShortcutsBench	52.3	58.2	407	320	1.27	24.9%
Dream	MCP-Atlas	64.7	39.0	146	420	0.35	42.8%
LLaDA	MCP-Atlas	58.4	40.6	199	409	0.49	32.5%
Qwen2.5	MCP-Atlas	58.3	48.1	237	357	0.66	19.9%
Llama-3.1	MCP-Atlas	35.3	70.3	887	204	4.35	3.6%

Both diffusion drafters are purer than both autoregressive ones on all three benchmarks, and Dream is the purest in every cell; the price is lower recall in eleven of twelve cells. This is the error profile PITON prefers: a wrong selection contaminates the anchors, whereas an omission is what the dependency reranker undoes. Llama-3.1 is the opposite extreme, with the highest UltraTool recall (80.3) at a purity of 46.0. The mechanism is committing less: Dream's drafts are shorter than gold on all three benchmarks (2.12, 2.05 and 2.49 against 2.39, 3.23 and 4.14). Interface and architecture are entangled here, since Dream uses ScaffoldSlots and Qwen2.5 free generation; with both decoding tool strings freely on UltraTool, Dream still reaches 73.0 purity at 68.3 recall against 66.2 at 71.0 for Qwen2.5, so the architecture contributes about $+6.8$ purity points.

F Rescoring protocol and baseline reproduction

Every method within a dataset is rescored against the same gold with the same scorer, since published numbers differ in tokenisation, de-duplication and edge semantics. The scorer accumulates node and edge counts micro over set semantics, de-duplicates by first occurrence on both sides, and scores UltraTool against the baseline gold. Edge F1 is computed over ordered adjacent pairs (a, b)

of the de-duplicated sequence, so a reversed pair is a miss, node F1 over the tool sets, and the normalised edit distance over the ordered sequences. External baselines run from their released code with our retrieval candidates and catalogue, so a difference between two rows of Table 2 reflects the plans rather than the scripts.

Every external baseline was run three times under independent seeds, each run decoding the full test split, and Table 2 reports the mean of the three. Two of the six methods cannot vary with the seed at all: ControlLLM-ToG decodes greedily, and TaskBench samples at a temperature and nucleus that leave a single token in the candidate set, so their three runs are bit-identical. The zero spread in those rows is a property of the decoder rather than a measured variance.

GTool's spread matters most because it is the strongest external method and the comparison in §5.2 turns on it. Both conclusions survive the spread, not only the means. On UltraTool, we lead Qwen2.5-trained GTool by 62.74 to 56.76, above its best run (58.46). On its home ground with Llama-3.1, GTool leads us by 63.26 to 57.07, and even its worst run (61.83) remains ahead.

PITON's own twelve cells repeat under the same protocol, three runs each, and Table 2 reports the mean. Our drafters decode at temperature 0, so a repeat can differ only through the node it lands on. Four of the twelve cells returned byte-identical predictions in all three runs and their half-range is exactly zero. In the remaining eight, two of the three runs coincide and the third differs, so what the half-range measures there is the distance between two GPU nodes (Table 20).

Table 20: Run-to-run repeatability of PITON's twelve cells: distinct decodes among the three runs, and draft and repaired edge F1 as mean $\pm$ half-range.

Benchmark	Drafter	distinct decodes	draft eF1	PITON eF1
UltraTool	Dream	1	54.28 ± 0.00	62.74 ± 0.00
	Qwen2.5	1	42.35 ± 0.00	52.67 ± 0.00
	LLaDA	2	51.11 ± 0.00	57.07 ± 0.29
	Llama-3.1	1	21.21 ± 0.00	47.22 ± 0.00
ShortcutsBench	Dream	2	27.84 ± 0.28	35.96 ± 0.16
	Qwen2.5	2	21.06 ± 0.16	31.12 ± 0.73
	LLaDA	2	23.21 ± 0.02	30.02 ± 0.53
	Llama-3.1	2	16.60 ± 0.01	23.84 ± 0.01
MCP-Atlas	Dream	2	12.74 ± 0.01	35.78 ± 0.42
	Qwen2.5	2	18.13 ± 0.46	42.44 ± 1.10
	LLaDA	2	12.96 ± 0.37	33.83 ± 0.62
	Llama-3.1	1	13.55 ± 0.00	27.99 ± 0.00

Every cell sits below GTool's smallest spread of ± 1.25, and five of the twelve exceed GNN4Plan's ± 0.33. The largest is on MCP-Atlas, where repair amplifies variation in the draft. Draft variation of ± 0.01 to ± 0.46 becomes ± 0.42 to ± 1.10 after tools are inserted from a changed anchor set. Node F1 moves by at most ± 0.61 and NED by at most ± 0.0053 in any cell. The two comparisons against GTool in §5.2 survive this spread from our side as well, since the UltraTool cells they turn on move by at most ± 0.29.

F.1 TRAINING COST

The last column of Table 2 is measured, not estimated. For GTool each entry is the mean of the same three runs as the scores beside it. PITON's entry is not: it is a single timing pass over the same twelve cells on one machine, averaged over the four drafters of a benchmark. Repeating that pass reproduces 1.5 and 1.2 minutes on UltraTool and ShortcutsBench and gives 0.41 rather than 0.39 on MCP-Atlas, so the second decimal there is the machine's own variation and not a measured difference.

GTool and GNN4Plan are each timed on one GPU: an A100-80GB for GTool and an RTX 6000 Ada for GNN4Plan. The value covers the full fit-and-evaluate run because the released code reports one elapsed time and does not separate fitting from testing. GTool early-stops on validation loss, so its stopping epoch varies across runs, and its cost is linear in that epoch to within about one percent. Regressing runtime on the stopping epoch puts the fixed part, which contains the test pass, at most

12 minutes of a run lasting up to 193, so fitting accounts for 91 to 98 percent of the reported time in five of six cells; only MCP-Atlas with Llama-3.1 is too short for the decomposition. Reporting the whole run therefore overstates GTool's training cost by a few percent and understates nothing. Training longer is what makes GTool expensive; it is not what makes it good, since ordering a cell's three runs by stopping epoch agrees with ordering them by edge F1 in only 11 of 16 comparable pairs.

G CLOSED-SOURCE DRAFTERS

PITON makes no demand on the drafter, and the same rerankers attach directly to models reachable only through an API. We evaluate six closed-source drafters on the hardest benchmark, the 166 MCP-Atlas test queries; Table 21 reports the per-cell numbers. All six are public API models from the Claude and GPT families, accessed in September 2026; exact identifiers are released with the code.

All six models improve, by $+8.62$ edge F1 on average, the gain is largest for the weakest draft (Haiku, $+21.08$), and the strongest drafts still have headroom (Fable, 64.46 to 69.08; Opus, 64.18 to 69.31).

Table 21: Repair on closed-source drafters (MCP-Atlas, 166 test queries): edge F1 of the draft and of the repaired plan. The rerankers are fitted under the same configuration as the open-weight experiments, with no per-model tuning.

Drafter	Draft	+ PITON	Δ
Haiku	25.06	**46.14**	+21.08
Luna	51.70	**60.00**	+8.30
Sol	59.12	**66.08**	+6.96
Terra	46.59	**52.23**	+5.64
Fable	64.46	**69.08**	+4.62
Opus	64.18	**69.31**	+5.13

H THE STRATIFIED COMPARISON: THE FULL TWELVE CELLS

Figures 6 and 7 are the full versions of Figures 3a and 3b, covering all three benchmarks. The visual grammar is the one defined in §5.3, with the deletion step and the insertion step drawn separately.

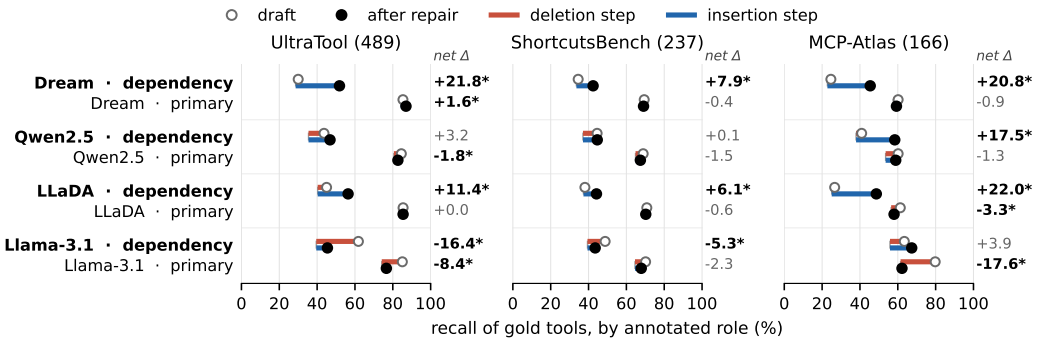


Figure 6: Gold-tool recall before and after repair, by annotated role, for all 12 cells. The open circle is the draft, the red segment is deletion, the blue segment is insertion, and the filled circle is the repaired plan. Each panel's right column gives the net change; a star means its 95% interval over 2,000 query resamples excludes zero.

Precision, which the figures do not show. Recall alone can be inflated by pouring tools into a plan. A stratum's precision is the share of emitted tools that belong to that stratum's gold set.

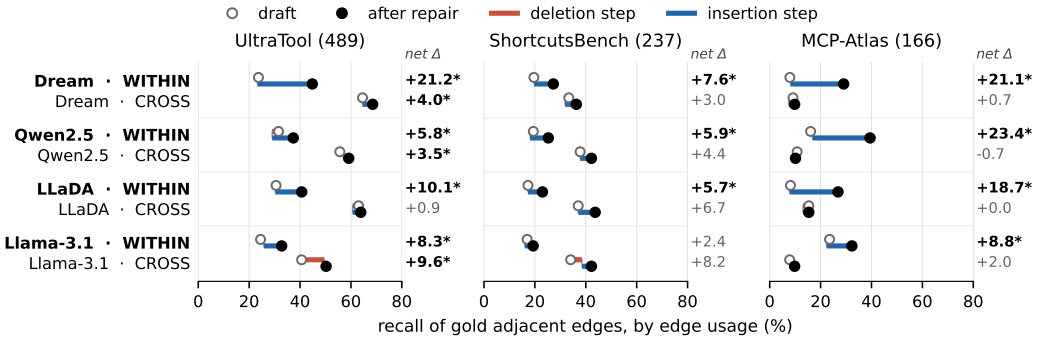


Figure 7: Recall of gold adjacent edges before and after repair, stratified by edge usage, all 12 cells. Visual grammar identical to Figure 6.

This is a decomposition of node precision by role, since the strata sum exactly to node precision. Dependency precision rises in all 12 cells, by $+7.16$ on average and $+1.77$ to $+15.96$ across cells. All 12 bootstrap intervals exclude zero, and the sign test gives $p < 0.05$. Precision also rises in both cells where dependency recall falls. Primary precision rises in 7 of 12, so repair is buying dependency precision partly with primary precision.

I ZERO-TRAJECTORY GENERALISATION

I.1 HOW THE THREE SUPERVISION SOURCES ARE BUILT

PITON depends on the target benchmark in exactly two places, the dependency graph counted from adjacency in the training trajectories and the synthetic corpus built from the training gold plans. This section removes both, leaving the training split with queries and candidate tools only. All three substitutes draft on the unlabelled training queries and count adjacent pairs in those drafts to form $\hat{G}$, which has the same structure as G but a different source; they differ only in who drafts and where the rerankers' supervision originates.

Cross-benchmark transfer (T1) fits both rerankers on another benchmark's real trajectories, ShortcutsBench, and moves them over frozen with their operating point included; the target domain supplies only $\hat{G}$, the drafts and the catalogue. What makes this possible is the feature normalisation in Appendix D. Every graph statistic is relative, so an edge's reliability relative to its neighbours retains the same meaning in another catalogue.

Drafter consensus (T2) asks a panel to plan every training query. The panel has four drafters in general and three effective members on MCP-Atlas. Their pseudo-gold is a reliability-weighted vote, where reliability is mean Jaccard agreement with the other members. The method admits tools at a threshold of 0.5 and orders them by the medoid. A panel is necessary because a single model systematically omits some tools, which would then be absent from both its pseudo-gold and its $\hat{G}$; the vote suppresses each member's idiosyncratic errors but not their shared habits, which is where T2's ceiling lies.

Teacher distillation (T3) lets one stronger model, Claude Opus, draft each training query once. This draft directly supplies the pseudo-gold, from which $\hat{G}$ and the corpus follow. The teacher sees only the query and that row's candidate tools, with no gold field. The teacher is a one-off offline cost, 2,670 calls across the two benchmarks, and is absent at inference. The three sources are monotone in how much target-domain inference they spend, T1 none, T2 four panel passes and T3 one strong-model pass.

Table 22: Zero-trajectory PITON, edge F1, each cell reported as delete-only / loop. Operating points are carried over with no target-domain calibration and all eight cells share one set of rerankers. The loop here is orchestrated insertion first, whereas Table 2 deletes first; on T3 the two differ by about one point (30.07 against 29.06 on MCP-Atlas with Qwen2.5). *Ceiling* is the PITON row of Table 2, which uses a different configuration, so the two are not subtractable; see below for the matched-configuration ceiling. The UltraTool arms average three runs that differ only in the corpus-synthesis and tree-building seed, with per-cell ranges of 0.26 to 3.35; the drafts there are frozen and decoded once, so this measures reranker-fitting variance and not the end-to-end decoding variance that Table 2 averages over. The MCP-Atlas arms are single runs on a single draft, which is the whole of the small disagreement in the *Draft* column there against Table 2.

Drafter	Draft	T1 transfer (SB rerankers)	T2 consensus	T3 teacher (Opus)	Ceiling
UltraTool (489)					
Qwen2.5-AR	42.35	46.71 / 40.68	47.42 / 44.57	48.15 / **49.47**	52.67
Llama-3.1-AR	21.21	32.07 / 32.65	33.14 / 37.68	35.79 / **42.20**	47.22
Dream	54.28	53.92 / 45.91	54.11 / 50.22	54.80 / **54.25**	62.74
LLaDA	51.11	50.95 / 43.82	51.73 / 48.52	53.17 / **52.64**	57.07
MCP-Atlas (166)					
Qwen2.5-AR	17.83	16.53 / 17.00	18.41 / 21.06	19.64 / **30.07**	42.44
Llama-3.1-AR	13.55	17.20 / 15.45	13.57 / 13.41	18.11 / **22.07**	27.99
Dream	12.73	6.48 / 10.58	12.58 / 18.23	10.75 / **26.42**	35.78
LLaDA	12.47	8.60 / 10.66	11.94 / 15.09	10.69 / **25.58**	33.83

I.2 THE FULL GRID

Two different ceilings, and only one of them is subtractable. The ceiling column above uses the full recipe including the corpus switches, so the distance to it mixes the loss from having no gold with the difference between two configurations. Refitting the same zero-trajectory configuration on the target domain's real training gold gives the matched ceiling, 40.09 on MCP-Atlas and 51.12 on UltraTool for the Qwen2.5 drafter. Against that, T3 recovers $(30.07 - 17.83)/(40.09 - 17.83) = 55\%$ of the obtainable gain on MCP-Atlas and $(49.47-42.35)/(51.12-42.35) = 81\%$ on UltraTool.

J DOWNSTREAM EXECUTION: DOES A REPAIRED PLAN HELP AN AGENT FINISH THE TASK?

J.1 SETUP

Why these seven agents. The executing cohort is not the drafters of Table 2: the 8B drafters cannot drive a 33-server MCP loop reliably, and the closed-source drafters of Appendix G would cost too much over 2,076 agent runs. The seven are open-weight tool-calling models, served through the MCP-Atlas evaluation harness at the time of writing (September 2026), and each writes and then executes its own plan.

Arms. As in §5.5, arm A injects no plan, B the drafter's own draft, C the draft repaired by PITON, and D the reference plan derived from the official trajectory.

Selecting the queries in two steps. Of MCP-Atlas's 166 test queries, 151 (91%) are runnable end to end once 33 MCP servers and 300 tools are brought up; the remaining 15 need credentials or servers we could not bring online. Neither selection step can see the quantity we report: the first runs only arms A and D, which involve no draft or repair, and the second looks only at whether the dependency reranker changed a draft, fixed before any agent runs.

The first step selects for headroom. We run arm A (no plan) and arm D (the reference plan) with three drafters and keep every query whose three-model mean of $D - A$ is positive: **64 queries**, with mean $D - A$ of $+25.2$. The other 87 fall outside the conditional estimand defined below, because on them a reference plan does not improve execution over no injected plan. The threshold is the sign of $D - A$, not a tuned cut.

The second step keeps, for each drafter, the queries where the dependency reranker edits its plan. Where the reranker does not fire, arms B and C inject byte-identical text, so the treatment effect there is zero by construction. Our estimand is therefore intervention efficacy conditional on positive plan headroom and on PITON firing; a deployment-level effect would include those zeros, and the null control below measures exactly that column. Each drafter runs on its own subset, averaging 24.7 of the 64 queries.

The first step introduces a selection bias, because the queries were screened with three of the drafters, so those three carry a dagger below and we report the clean and the biased subsets separately. A biased *selection* is not a biased *treatment*, which is what the split is there to show.

Runs that returned no answer. A minority of agent runs ended in a gateway error, which the judge would score as zero claims. We re-ran every affected run once; 49 runs still returned nothing, spread over 20 of the 193 cells, and we drop those cells whole, all four arms together. The rule was fixed before any coverage number was computed and looks only at whether a run errored, so it cannot favour an arm. This leaves the **173 cells** reported below, over 2,076 agent runs.

Judging. We use the benchmark's own metric: one to nine gold factual claims per query (3.38 on average), coverage being the share stated in the final answer. Each claim scores zero or one, an answer that recites the plan without a result scores zero, and semantically equivalent answers count. We deviate in one respect: all twelve answers for a query, four settings by three runs, are shuffled, blinded and judged in one call, so judge leniency is common-mode across arms. Re-judging with a different blinding order and judge shifts results by 0.66 points under this protocol, against 11.0 when arms are judged separately.

J.2 RESULTS

Table 23: Execution-layer claim coverage, each drafter on its own subset of the 64 headroom queries, restricted to the queries where its dependency reranker fired. Each cell is the median of three runs, averaged over queries; intervals resample the queries 20,000 times, carrying every drafter's value for a resampled query together. A dagger marks a drafter that took part in screening the query set. Recovery is $(C-B)/(D-B)$; Mistral's exceeds one because its reference plan is worth only $+2.72$ over its own draft.

Drafter	n	A	B	C	D	$\Delta = C - B$	$D - B$	recovery
Kimi-K2.5	28	79.9	84.0	85.5	92.6	$+1.49$	$+8.53$	17%
DeepSeek-V3.2	25	45.3	70.3	75.5	79.5	$+5.17$	$+9.17$	56%
Nemotron-120B	17	50.7	54.5	58.1	60.0	$+3.63$	$+5.49$	66%
GPT-OSS-120B †	32	55.2	62.4	69.4	77.5	$+6.91$	$+15.07$	46%
GLM-4.7 †	22	53.0	60.2	70.0	70.4	$+9.85$	$+10.23$	96%
Mistral-L3-675B	28	27.9	31.3	38.3	34.0	$+7.02$	$+2.72$	258%
Devstral-123B †	21	17.0	24.6	31.0	37.6	$+6.43$	$+13.10$	49%
pooled (173 pairs)		48.0	56.4	62.2	65.7	$+5.79$	$+9.36$	62%
clean 4 drafters						$+4.38$	$+6.51$	67%
selection-involved 3 drafters †						$+7.64$	$+13.10$	58%

A null control that costs nothing to run. Where the dependency reranker does not fire, arms B and C inject byte-identical prompts and differ only in the agent having run again, so the true effect in that column is zero and it measures the noise floor directly. This control comes from an earlier round of the same protocol in which the non-firing cells were also executed. Pooled over its 63 non-firing pairs the column reads -2.82 [-10.62, $+4.72$]: if anything it leans against the sign we report. The same column under a single run per cell instead of three reads $+5.26$, with one cell at $+15.80$ whose interval excludes zero, a clear false positive. Repeating three times brings it back to the noise floor. This is why the direction above is not an artefact of the judge.

Reading the table. The reference plan lifts coverage by $+17.74$ [$+12.26$, $+23.27$] over injecting no plan and by $+9.36$ [$+3.06$, $+15.65$] over the drafter's own plan; the repaired plan recovers $+5.79$ [$+1.74$, $+9.84$] of that, or 62% of the ceiling.