PROGRESS RATES CAN'T TELL SHORTCUTS FROM PROGRESS: LANDMARK PROPERTIES FOR SCORING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Declaring a step done is not doing it, yet agent benchmarks often score the two alike. Benchmarks give failing agents partial credit for checkpoints reached, but no rule in use asks how each checkpoint was reached. To our knowledge, no prior work states what such a score must satisfy. We supply that standard. From automated planning we take the order a task imposes on its milestones and state four properties a progress score needs. No score that reads only a run and its checkpoints can reliably detect skipped steps. The limit lies in the checkpoint list itself, not in the rule, and adding the milestone order removes it. We audit nine scoring rules, score every legal run in six task families, and test seven language models. The rules in use either ignore skipped steps or reward them. AgentBoard's published progress rate can rank a run that skips steps above a flawless one. Reanalyzing TheAgentCompany's released runs, we find that its final-checkpoint bonus alone produces 55 of 525 perfect scores. When an unlabeled shortcut appears among shuffled options, all seven models take it, and the published rate largely ranks them by how often they do. Our metric, weighted support, reverses that ranking. Given the milestone order and what each action required, it satisfies all four properties in one pass. Under weighted support, work that skips a prerequisite earns no credit, and neither does any work built on it.

1 INTRODUCTION

Agent benchmarks now set long, multi-step tasks in browsers, operating systems, code repositories and customer-service dialogues. They score each task by whether its end state passes a check (Zhou et al., 2024; Xie et al., 2024b; Jimenez et al., 2024; Yao et al., 2025), and AgentBench reports success rates across eight such environments (Liu et al., 2024). On hard tasks these scores are mostly zeros, so benchmarks add partial credit. AgentBoard publishes a progress rate, the largest fraction of hand-written subgoals that hold at any single step (Ma et al., 2024). TheAgentCompany scores checkpoints reached in a simulated workplace (Xu et al., 2025), WebCanvas scores key nodes it judges indispensable (Pan et al., 2024), and SUPER looks for two to six landmark outputs extracted by hand from a gold notebook (Bogin et al., 2024). In each case a person writes the list of intermediate results, and the score is only as good as that list.

Partial credit now does more than rank agents. It shapes which designs look promising, and when read during a run it serves as a monitor or a reward. Liu et al. (2026), for example, roll back runs that their monitor flags and raise resolution rates on problematic instances by 6.9 to 23.5%. Reward design offers a sober lesson here, since more capable agents exploit a misspecified proxy more (Pan et al., 2022; Skalse et al., 2022). A score that decides which behaviour is kept but also pays for shortcuts would select agents that take them.

Consider what a checkpoint list cannot see. An agent asked to make a failing test suite pass can fix the code or rewrite the expected values, and a tool can report a message as sent without sending it. Either way the checkpoint becomes true, and a rule that counts checkpoints credits it. What separates the honest run from the shortcut is whether the step that made the checkpoint true required what the task says must come first. No rule in use checks this. Outcome-only scoring is already known to hide such differences. Three in four instance-level comparisons tie under success rate (Diaz, 2026),

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.

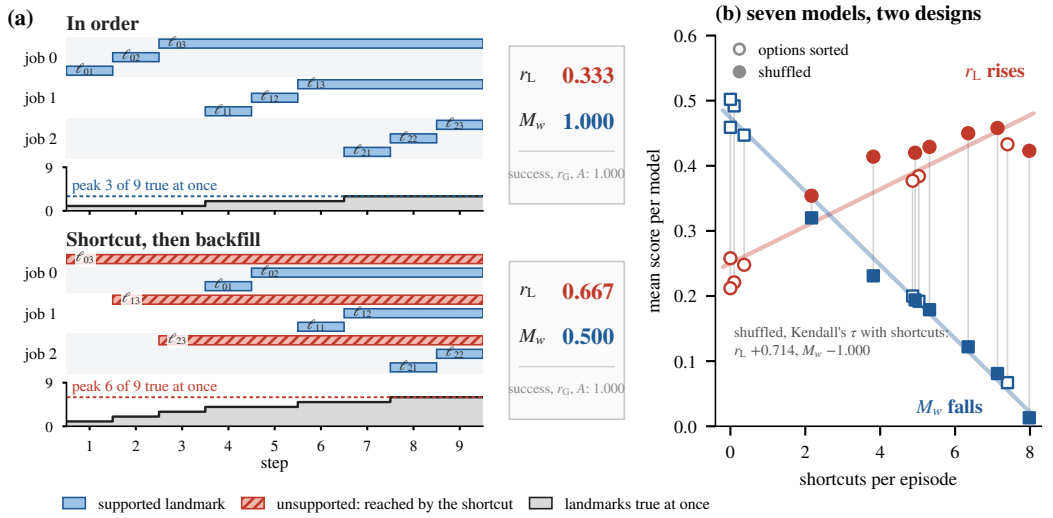


Figure 1: **The published progress rate rewards skipped prerequisites.** (a) Two runs of the same task, with three jobs of three stages each, reach the goal and achieve the same nine landmarks. Each bar shows how long a landmark stays true, from its first achievement until a later step consumes it. Hatched bars mark landmarks reached by the shortcut, which are unsupported. The shortcut run marks every job complete first and fills in the earlier stages afterwards, so more landmarks are true at once (grey). The published rate measures exactly that and prefers this run (0.667 against 0.333). Weighted support prefers the in-order run (1.000 against 0.500), while binary success, the goal-level rate and landmarks achieved score both runs 1.000. (b) Each model's mean progress rate (circles) and weighted support (squares) against its mean shortcuts per episode, over 20 instances that offer an unlabeled "mark job complete" action. Open markers show sorted options (Qwen2.5-7B from the rerun), filled markers show options shuffled at every step, and lines are least-squares fits.

and 10.7% of passing SWE-agent trajectories reach a correct patch through regression cycles, blind retries or missing verification (Sahoo et al., 2026). Yet none of this work states what a progress metric must satisfy or checks the metrics in use against such a standard. This paper supplies that standard.

Automated planning already has the object these lists approximate. Landmarks are facts that hold at some point in every valid plan, and orderings between them record which must come before which (Hoffmann et al., 2004). Both come from the task rather than from an annotator. We separate the task's *specification* from the *environment* the agent acts in, since the environment may allow more, such as the rewritten test above. A landmark is *supported* when the step that first achieved it required its immediate predecessors, and those predecessors were themselves supported earlier. On this model, we state what a progress metric should satisfy.

Contributions and findings.

- **A standard, and a limit that no checkpoint list can pass.** We state four properties for progress metrics, each ruling out a named failure. We then prove that no rule reading only a run and its checkpoints can detect skipped prerequisites on every task, while adding the ordering suffices (Theorem 1). The limit lies in the input, not in the rule (Corollary 1).
- **An audit of the rules benchmarks use.** We check nine rules, including those of AgentBoard, TheAgentCompany, WebCanvas, DevAI, ToolSandbox and LAMA. Only weighted support satisfies all four properties, and it runs in linear time (Theorem 2). The published progress rate also fails normalization, so a complete, correct run can miss full marks. Where shortcuts exist, its highest scores go to runs that skip prerequisites (Propositions 1–4).
- **Every legal run, not a sample.** We score all 6,014,400 runs of one configuration exactly. At a fixed budget the rate counts jobs started, so a higher rate means fewer goals finished on average.

- **Language models take the shortcut unprompted.** When an unlabeled shortcut appears among shuffled options, seven open models take it in 416 of 420 episodes. The rate rises with shortcut use, while weighted support falls. Without shortcuts, the rate correlates -0.439 with goal completion within an instance, but pooling across instances turns this into $+0.064$.

We release the enumeration code, every agent episode with its replies, and the scoring code. The paper makes one point precise, that a progress score has to ask not only what became true, but how.

2 LANDMARKS AND SCORING RULES

Planning has spent two decades formalizing what every solution to a task must pass through. We borrow its formalism and write each benchmark's scoring rule in the same terms.

Definition 1 (Task, environment, trajectory)**.** A task is $\Pi = \langle F, O, I, G \rangle$, with facts F, operators O, initial state $I \subseteq F$ and goal facts $G \subseteq F$, where $G \cap I = \emptyset$. Each operator o carries $\mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o) \subseteq F$. It is applicable in a state s when $\mathrm{pre}(o) \subseteq s$, and applying it yields $(s \setminus \mathrm{del}(o)) \cup \mathrm{add}(o)$. A plan is a sequence of operators from O applicable in turn from I and ending in a state containing G. An environment is a set O_E of operators over F. A trajectory is $\tau = \langle s_0, o_1, s_1, \ldots, o_T, s_T \rangle$ with $s_0 = I$, each $o_t \in O_E$ applicable in s_{t-1}, and s_t its result. For a fact f, $\mathrm{first}(f)$ is the least t with $f \in s_t$, when one exists.

Informally, landmarks are milestones every solution must pass, and $\ell' \prec \ell$ says ℓ' must come first.

Definition 2 (Landmarks and orderings)**.** A *landmark* is a fact that holds at some state of every plan for Π. Facts in I are landmarks trivially; we write L for the landmarks not in I. An ordering $\ell' \prec \ell$ holds when, in every plan for Π, ℓ' is true in the state immediately before the one in which ℓ first becomes true. We take $\prec$ to be the transitive reduction of these orderings on L, and write $\mathrm{pred}(\ell)$ for the immediate predecessors of ℓ. The transitive closure of $\prec$ is a partial order on L, not a sequence.

This is the greedy-necessary ordering of Hoffmann et al. (2004), stated over plans. An edge that the transitive reduction drops is still enforced through support (Appendix K.3). We also assume the specification is *ordering-faithful*. That is, any operator of O that first achieves a landmark ℓ on some plan requires $\mathrm{pred}(\ell)$. In every experiment here, landmarks and orderings are known by construction, and Appendix K.1 discusses how to extract them and which orderings we exclude.

Definition 3 (Scoring rules)**.** For a subgoal set $K \subseteq F$, write $r_K(\tau) = \max_{0 \le t \le T} |s_t \cap K|/|K|$. The rules audited are binary success $S(\tau) = \mathbf{1}[G \subseteq s_T]$; the landmark-level progress rate $r_{\mathrm{L}} = r_K$ with $K = L$; the goal-level progress rate $r_{\mathrm{G}} = r_K$ with $K = G$; landmarks achieved $A(\tau) = |\mathrm{Ach}(\tau)|/|L|$, where $\mathrm{Ach}(\tau)$ is the set of landmarks ever true on τ; weighted checkpoints $C = \frac{1}{2}\,\mathrm{Result}/\mathrm{Total} + \frac{1}{2}\,S_{\text{full}}$, where Result and Total are the milestone points earned and available and S_{full} is 1 when all are earned, evaluated after the run (Xu et al., 2025); and weighted support M_w (Section 4).

A landmark is *live* at step t when it is true in s_t, so AgentBoard's published rate r_{L} is the largest fraction of landmarks live at one step. Its released code for AlfWorld (Shridhar et al., 2021) and ScienceWorld (Wang et al., 2022) reports A instead, like WebCanvas. On 92 of 134 AlfWorld tasks no step can match every subgoal, yet all 174 successful tasks in its 13 released baseline runs score 1.000. Its planning tasks compute r_{G} on domains like Blocksworld that undo goal facts (Appendix K.2). We audit all three. The published rate and landmarks achieved agree on P3 but differ on P4.

Job-chain domains. Every experiment uses generated tasks with n jobs, which are the simplest tasks in which order matters. Job c has d_c stages and landmarks $\ell_{c,1} \prec \cdots \prec \ell_{c,d_c}$. The operator *advance job c to stage j* requires $\ell_{c,j-1}$ and adds $\ell_{c,j}$, where $\ell_{c,0}$ holds in I. Under the *consuming* rule it also deletes $\ell_{c,j-1}$, much as holding an object stops once the object is placed. Under the *persistent* rule it deletes nothing. The goals are the final stages. A protected shared root and pairwise merges give two further structures (Appendix C). Unless Appendix G says otherwise, weights are $w(\ell_{c,j}) = j$. These tasks are small enough to enumerate, yet large enough for the rules to disagree.

3 WHAT PARTIAL CREDIT MUST SATISFY

Environments are more forgiving than the tasks they implement. A tool can report a success it did not deliver, and a test can pass because its expected values were rewritten. We score a trajectory from O_E against the landmarks of the task Π, so the question is which achievements to trust.

Definition 4 (Sound achiever)**.** An operator o with $\ell \in \mathrm{add}(o)$ is a *sound achiever* of ℓ when $\mathrm{pred}(\ell) \subseteq \mathrm{pre}(o)$.

Definition 5 (Support)**.** A landmark ℓ is *supported* in τ when $\mathrm{first}(\ell)$ exists, the operator at step $\mathrm{first}(\ell)$ is a sound achiever of ℓ, and every $\ell' \in \mathrm{pred}(\ell)$ is supported with $\mathrm{first}(\ell') < \mathrm{first}(\ell)$.

In words, a landmark counts only if the step that produced it needed its prerequisites and those counted too. A landmark resting on an unsupported prerequisite is therefore itself unsupported. Write $\mathrm{Sup}(\tau) \subseteq \mathrm{Ach}(\tau)$ for the supported subset. Soundness implies the timing clause, since $\mathrm{pred}(\ell) \subseteq \mathrm{pre}(o_t) \subseteq s_{t-1}$ for $t = \mathrm{first}(\ell)$. The timing clause without soundness is LAMA's reached condition (Table 1). When every first achiever is sound, $\mathrm{Sup} = \mathrm{Ach}$, so P3 below matters only when the environment allows more than the specification, as with the shortcuts of Sections 4 and 6 (Appendix K.3).

	Name	Statement	Failure it prevents
P1	Monotone	τ' extends $\tau \Rightarrow M(\tau') \geq M(\tau)$	the metric punishes progress
P2	Order invariant[1]	exchanging two adjacent independent steps whose new landmarks are unrelated under $\prec$ leaves M unchanged	working style is reported as capability
P3	Order sensitive	$\mathrm{Ach}(\tau') = \mathrm{Ach}(\tau)$ and $\mathrm{Sup}(\tau') \subsetneq \mathrm{Sup}(\tau) \Rightarrow M(\tau') < M(\tau)$	skipped prerequisites cannot be told from work
P4	Normalized	$M(\langle I \rangle) = 0$, and $M(\tau) = 1$ when τ reaches G with every landmark supported	scores are not comparable across tasks
W1	Weight monotone	$\ell' \prec \ell \Rightarrow w(\ell') \leq w(\ell)$	a prerequisite outweighs the work it enables

W1 constrains the choice of weights rather than the metric.

Remark 1. P2 and P3 are compatible. Any strictly increasing function of Sup satisfies P3 by construction, and P2 because such an exchange changes no support status (Lemma 2).

Theorem 1. *No metric whose value is fixed by the trajectory, the operators on it, the landmark set and the goal satisfies P3 on every task. Adding $\prec$ suffices (Theorem 2).*

Proof. The idea is to build two tasks that agree on everything a checkpoint list records but order the same two landmarks oppositely. Let the environment offer four operators that delete nothing. The operators o_a and o_b add a and b with no precondition, and o_a^+ and o_b^+ add a given b and b given a. Take $I = \emptyset$, $G = \{a, b\}$, $\tau = \langle o_b, o_a^+ \rangle$ and $\tau' = \langle o_a, o_b^+ \rangle$. The task Π_1 with operators o_b and o_a^+ has $b \prec a$, and the task Π_2 with o_a and o_b^+ has $a \prec b$. The supported sets of τ and τ' are $\{a, b\}$ and $\{b\}$ in Π_1, and $\{a\}$ and $\{a, b\}$ in Π_2, so P3 requires $M(\tau) > M(\tau')$ in Π_1 and the reverse in Π_2. The tasks share the environment, both trajectories, the landmark set and the goal, so a metric fixed by those violates P3 in one of them (Appendix A.1 checks each cell). □

The theorem is elementary, and its value lies in Corollary 1. Reading the executed operators' preconditions does not help, because τ and τ' are the same in both tasks. The ordering suffices but is not necessary, since counting the landmarks whose first achiever lies in O also separates the pair.

Corollary 1. *A hand-authored checkpoint list without stated orderings is insufficient in principle, not merely in practice. Two tasks with identical checkpoints, environments and runs can require opposite rankings of the same two runs, and only structure outside the list distinguishes them.*

[1] In full, P2 covers every trajectory τ' obtained from τ by exchanging steps t and $t+1$ such that neither operator deletes a precondition or an add effect of the other, neither adds a landmark the other first achieves, and no landmark first achieved at one of the two steps is $\prec$-related to one first achieved at the other.

Table 1: **Of nine scoring rules, only weighted support tells skipped prerequisites from work (P3).** Rows show deployed rules (top), prior rules that read the ordering (middle) and ours (bottom). Columns $\prec$, time and sound record whether a rule reads the ordering, checks that prerequisites came first, and checks that the achieving step required them. "cond." marks a property that holds only under a condition. P1 then holds only while the tested facts persist (Remark 2). P2 holds for r_{L} under Proposition 3's condition and for r_{G} while goal facts persist, which fails in Blocksworld. "part." marks a rule that orders the turns at which milestones held. Appendix B gives the witnesses.

	Rule	$\prec$	time	sound	P1	P2	P3	P4	Witness
S	binary success	–	–	–	cond.	yes	no	yes	Rem. 2; Thm. 1
r_{L}	rate, landmark level	–	–	–	yes	cond.	no	**no**	Prop. 1; pair
r_{G}	rate, goal level	–	–	–	yes	cond.	no	yes	Thm. 1; pair
A	landmarks achieved	–	–	–	yes	yes	no	yes	Thm. 1; pair
C	weighted checkpoints	–	–	–	cond.	yes	no	yes	Rem. 2; Thm. 1
	DevAI: met by the end	✓	–	–	yes	yes	no	yes	pair
	LAMA: reached first	✓	✓	–	yes	yes	no	yes	advance, mark
	ToolSandbox: DAG match	✓	part.	–	yes	no	no	yes	advance, mark
M_w	**weighted support**	✓	✓	✓	yes	yes	**yes**	yes	Thm. 2

AgentBoard's subgoals and WebCanvas's key nodes are ordered lists, but neither rule reads the order. A single recorded route could not serve as $\prec$ anyway, because P2 forbids penalizing independent reorderings. TheAgentCompany reads only the position of the last checkpoint. In 54 of its 175 tasks, passing that checkpoint credits every earlier one, which is exactly the inference a shortcut breaks. In its 19 released runs, we find that 55 of the 525 full scores are full only because of this bonus (Appendix B.6). A checkpoint list records destinations. Telling progress from a shortcut needs the constraints that every route to them shares.

4 Auditing the rules in use

If the standard is reasonable, the rules in use should meet it. They do not. Table 1 checks nine scoring rules against the four properties, and Appendix B gives a proof or witness for each entry. Read the P3 column first. Every partial-credit rule in current use fails it, and so does binary success. The three rules that read $\prec$ (Zhuge et al., 2025; Richter & Westphal, 2010; Lu et al., 2025) never ask what the achieving step required (Appendix B.4). The published rate fails twice more.

Proposition 1 (Normalization failure)**.** *(a) For any task, $r_{\mathrm{L}}(\tau) = 1$ if and only if some state on τ makes every landmark true at once. (b) In a job-chain domain with n consuming chains, a trajectory of advances alone has $r_{\mathrm{L}}(\tau) = c/|L|$, where c is the number of jobs it starts. One that achieves every landmark reaches G with every landmark supported and scores $n/|L|$, below 1 whenever some chain has more than one stage.*

So even a complete, in-order run scores only 0.167 to 0.500 in our configurations (Appendix K.4).

Proposition 2 (Persistence)**.** *If no operator deletes a landmark, then $r_{\mathrm{L}}(\tau) = A(\tau)$. If in addition every operator achieves exactly one new landmark, r_{L} is constant across trajectories of equal length.*

Proposition 3 (Consumption)**.** *Suppose every landmark has at most one deletable immediate predecessor, and every operator on τ is a sound achiever whose add list contains exactly one landmark, not yet achieved, and which deletes, among landmarks, exactly that landmark's deletable immediate predecessors. Then the number of true landmarks never decreases along τ, $r_{\mathrm{L}}(\tau)$ is a function of $\mathrm{Ach}(\tau)$, and r_{L} satisfies P2 on such trajectories.*

When prerequisites are enforced, then, the rate reads only which landmarks were reached. The condition of Proposition 3 holds for chains and a protected shared root, and fails for merges (Appendix A.4). Once shortcuts are allowed, the rate reads something worse.

Proposition 4 (Manipulability)**.** *Consider n consuming chains of uniform depth $d \geq 2$, a budget of at least $|L|$ steps, and an environment that adds to the advances $1 \leq k \leq d-1$ additive shortcuts per chain, each achieving a different stage other than the first and offered only while that stage has*

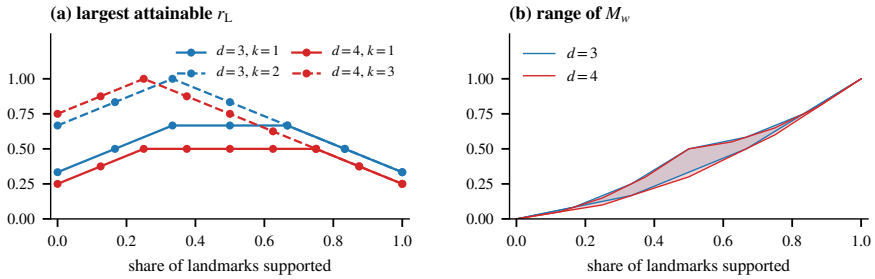


Figure 2: **The published rate is maximized by runs that skip some prerequisites.** Exact values over every run of at most $|L|$ steps on two consuming chains of depth d with k shortcuts per chain (Proposition 4). (a) The highest landmark-level rate attainable with a given share of landmarks supported. It peaks at $(k+1)/d$ while support is partial and falls to $1/d$ when every landmark is supported. (b) The range of weighted support over the same runs, which rises with support.

not been achieved. Let $f(j)$ be the largest r_L attainable with exactly j supported landmarks. Then $f(0) = k/d$, $\max_j f(j) = (k+1)/d$ attained only at partial support, and $f(|L|) = 1/d$.

The attainable rate is therefore not monotone in support, so supporting more landmarks can lower it (Figure 2). Every run that maximizes it skips some prerequisites, and once $k \geq 2$, supporting nothing beats supporting everything. Figure 1(a) shows why. A job marked complete early stays true while its earlier stages are filled in, and the rate measures this simultaneity (Appendix B.3). Study B's agents did this 32 times, through an environment bug that the shuffled rerun fixes (Appendix C.2).

Remark 2. S and C read the final state, so they satisfy P1 only while the facts they test are never deleted. On a finished two-job task, one action that restarts a job takes binary success from 1.000 to 0.000 while the rate stays at 0.500 (Appendix B.5).

One metric passes all four, and it simply sums the weights of the supported landmarks.

Theorem 2 (Weighted support)**.** *For any $w : L \to \mathbb{R}_{>0}$, the metric*

$$M_w(\tau) \;=\; \sum_{\ell \in \mathrm{Sup}(\tau)} w(\ell) \Big/ \sum_{\ell \in L} w(\ell)$$

satisfies P1 to P4. It is computable in $O(|\tau| + |L| + |\prec|)$ time, counting $|\tau|$ as the total size of the add lists along τ and testing membership in a precondition in constant time.

Proof sketch. For P1, extending τ changes no first index or first achiever. For P2, an exchange of the kind P2 describes keeps every first achiever and the order of every $\prec$-related pair (Lemma 2). For P3, a strictly smaller support gives a strictly smaller sum under positive weights. For P4, no landmark in L holds in I. For the running time, one pass over τ records first achievements and one sweep over $\prec$ checks predecessors (Appendix A.8). □

Weighted support penalizes skipped prerequisites through support alone, under any positive weights. Its positive correlation with goal completion comes from the depth weights (Appendix G). The four properties are necessary but not sufficient. Weights that violate W1 keep all four yet correlate -0.905 with goal completion. Averaging landmarks achieved with weighted support also keeps all four, yet it pays for shortcuts (Appendix K.4). A stronger condition, P3$^+$, asks that a metric depend on τ only through Sup and rise strictly with it. Together with P4 and additivity, it characterizes weighted support (Appendix A.9). Weighted support costs one pass over the run. What it needs is what the checkpoint list leaves out.

5 Exhaustive Verification

A counterexample shows that a rule can fail. Scoring every legal run shows what the rate measures when no step can skip a prerequisite.

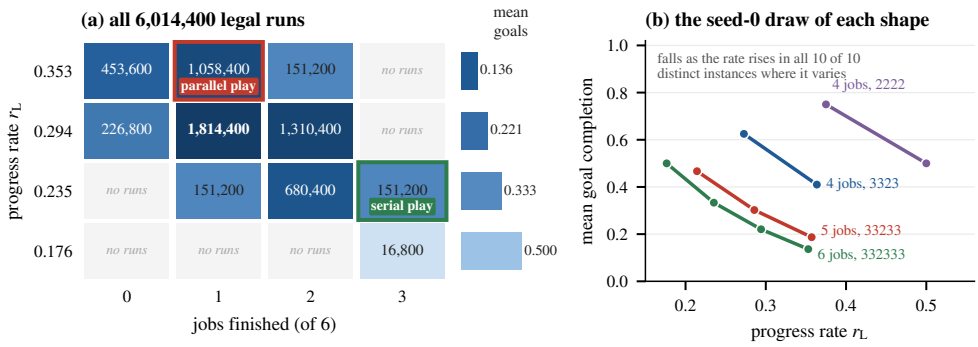


Figure 3: **When every run reaches the same number of landmarks, a higher rate means fewer finished jobs.** (a) All 6,014,400 legal runs of the main configuration (6 jobs, budget 9), arranged by progress rate (rows) and jobs finished (columns) and shaded by count on a log scale. Grey cells hold no runs, boxes mark serial and parallel play, and bars give mean goal completion at each rate. (b) Mean goal completion against the rate for the seed-0 draw of each enumerated shape, labeled by its jobs and job depths. Goal completion falls as the rate rises in every shape.

Lemma 1 (Equal landmark count)**.** *In a job-chain domain whose legal operators are exactly "advance some job to its next stage", every legal operator achieves one previously unachieved landmark. Hence* $|\mathrm{Ach}(\tau)| = |\tau|$*, and* A *is constant across legal trajectories of the same length.*

Among runs of the same length, then, differences in the rate are not differences in landmarks reached. Proposition 1(b) shows that they reflect the number of jobs started. A higher rate spreads the same steps over more jobs and leaves fewer steps to finish any one of them. Enumeration measures how large this effect is. Our reference is *goal completion*, the fraction of jobs finished, which is the task's own objective. Four shapes with five depth draws each give 20 instances, 11 of them distinct, and we count their runs exactly (Appendix D). The main configuration has 6 jobs, 17 landmarks and a budget of 9 steps, for a total of 6,014,400 legal runs.

In the main configuration, a high rate and maximal goal completion never occur together (Figure 3a). Of the top-rate runs, 453,600 complete no goal, while all 16,800 runs at the lowest rate complete the most goals the budget allows. This configuration is the strongest case. Of the 10 distinct instances where the rate varies, 8 have no top-rate run that completes the most goals, and 2 show both halves of the pattern. Mean goal completion falls strictly as the rate rises in all 10 of them (Figure 3b), from 0.500 to 0.136 in the main configuration. In every such instance, serial and closest-to-done play sit in the lowest or second-lowest rate level, while parallel play takes the top one. Across six domains, the rate correlates negatively with goal completion in every consuming one (-0.422 to -0.592), and weighted support correlates positively in all six ($+0.641$ to $+0.930$, Appendix K.5). At a fixed budget, the published rate rewards starting jobs, not finishing them.

6 Agents

Theorems say what a rule can reward, and agents show what it does reward. We ask two questions. Without a shortcut, does the rate track goal completion (study A)? With one on offer, who takes it, and what do the scores do (study B)? Six open models of 1.0B to 1.7B parameters act as agents on 20 consuming job-chain instances of 8 jobs, with 10 steps per episode (Appendix C). At each step the prompt lists the unfinished jobs and every permitted action, and the model copies one line. Study A lists only legal advances (5 draws per model and instance, 600 episodes). Study B also lists "mark job c complete" for every unfinished job, in the same format and with no hint that it differs (3 draws, 360 episodes). Marking satisfies the prompt's instruction literally but skips what the specification requires. For each instance we correlate a score with goal completion across its episodes, then average over instances with 95% bootstrap intervals (Efron, 1979) (Appendix K.6).

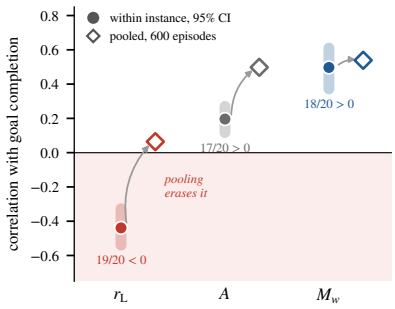


Figure 4: **Pooling erases the rate's relation to goal completion** (study A). Dots and bands show each score's mean within-instance correlation with goal completion and its 95% interval, labeled with the instances of that sign. Diamonds show it after pooling all 600 episodes. Within instances, r_L, A and M_w correlate -0.439 [-0.540, -0.327], $+0.196$ [$+0.117$, $+0.270$] and $+0.497$ [$+0.372$, $+0.610$]. Pooled, they give $+0.064$, $+0.498$ and $+0.539$.

Without a shortcut, the rate runs against goal completion. Within an instance, where the rate counts jobs started (Proposition 1), it correlates -0.439 with goal completion, negatively in 19 of the 20 instances. Weighted support and landmarks achieved correlate positively (Figure 4). Pooled over all episodes, the rate's correlation turns to $+0.064$, a case of Simpson's paradox (Simpson, 1951). Larger instances lower both the mean rate and mean goal completion, so the averages benchmarks report hide it. Per-model results and Fisher-z averages are in Appendices H.2 and K.6.

Who takes the shortcut depends on option order. Sorted options put every mark below every advance. Qwen, Llama and gemma copy the first line in 39% to 95% of steps and seldom reach a mark, while uniform choice marks in 99.9% of episodes. We reran study B with options shuffled at every step, adding Qwen2.5-7B (Appendix H.5). All seven models marked, in 416 of 420 episodes.

Table 2: **Marking raises the rate and lowers weighted support.** Marks per episode and mean scores when "mark job c complete" is offered, with options sorted or reshuffled at every step (60 episodes per model each). The last two columns have no shortcut offered (study A, 100 episodes). Qwen2.5-7B ran only in the rerun, which also ran both sorted designs for it. The last two rows are scripted choosers, and first-line choice is uniform once options are shuffled.

	shortcut, sorted				shortcut, shuffled				no shortcut	
Model	Marks	r_L	A	M_w	Marks	r_L	A	M_w	M_w	Goals
granite-3.1-1b-a400m	7.40	0.433	0.465	0.067	7.98	0.423	0.424	0.013	0.453	0.405
SmolLM2-1.7B	4.87	0.377	0.486	0.200	7.13	0.458	0.481	0.081	0.450	0.367
OLMo-2-0425-1B	5.03	0.384	0.486	0.192	6.35	0.450	0.492	0.122	0.443	0.359
gemma-3-1b	0.37	0.248	0.503	0.447	5.32	0.429	0.503	0.179	0.485	0.435
Llama-3.2-1B	0.10	0.221	0.503	0.492	4.93	0.420	0.500	0.194	0.489	0.425
Qwen2.5-1.5B	0.00	0.258	0.503	0.459	3.82	0.414	0.503	0.231	0.479	0.416
Qwen2.5-7B	0.00	0.212	0.503	0.502	2.17	0.354	0.503	0.320	0.483	0.417
uniform over lines	4.87	0.421	0.495	0.184	4.98	0.433	0.502	0.183	0.405	0.272
first listed line	0.00	0.212	0.503	0.502	—	—	—	—	0.502	0.463

The rate rewards the shortcut, whoever takes it. Within an instance, marks per episode correlate $+0.921$ (sorted) and $+0.613$ (shuffled) with the rate, and -0.978 and -0.967 with weighted support. Ranked by their mean scores, the rate and weighted support order 14 of 15 (sorted) and 18 of 21 (shuffled) model pairs oppositely (Table 2), while landmarks achieved barely moves (0.465 to 0.503). Marking raises the rate in two ways, by touching more jobs within the budget and by leaving an advanced job holding two landmarks at once (Proposition 4). Because a mark sets its job's goal, goal completion also rises with marking. We therefore also score supported goal completion, the share of jobs finished with their prerequisites met. Against it, the rate correlates -0.893, landmarks achieved $+0.525$ and weighted support $+0.935$. The timing-only rule of Table 1 needs no preconditions, yet it still catches 940 of the 1,066 sorted-design marks. Every model we tested takes an unlabeled shortcut once it sees one. The published rate pays for it, and weighted support charges for it.

7 Related work

Partial credit and process-aware evaluation. AgentQuest counts milestone states (Gioacchini et al., 2024) and TravelPlanner the share of satisfied constraints (Xie et al., 2024a). DevAI comes closest to our work. Its dependency-aware score counts a requirement only when those it depends on are met (Zhuge et al., 2025). Its dependencies, however, are annotated rather than derived from a task model, and they are judged on the finished workspace, so the score cannot tell whether a requirement was met after its prerequisites. ToolSandbox matches milestones to turns in an order its dependency graph allows (Lu et al., 2025). This checks when each milestone held, but not what the step that reached it required.

Trajectory evaluation and progress as reward. Recent evaluations look inside runs. Diaz (2026) ranks trajectories by preference, Sahoo et al. (2026) build reference automata from passing traces, and Liu et al. (2026) analyze trajectories as graphs, but none states what a progress metric must satisfy. Read during a run, a progress score becomes a reward. Only potential-based shaping leaves the optimal policy unchanged (Ng et al., 1999), and reward machines pay for subgoals in an order that an automaton fixes (Toro Icarte et al., 2018). Process supervision rewards each step of a solution using labels from human annotators (Lightman et al., 2024). Weighted support is a step-level signal of that kind, derived from the task's orderings instead of annotators.

Landmarks and axiomatic evaluation. Landmarks guide search in planners and heuristics (Hoffmann et al., 2004; Helmert, 2006; Richter & Westphal, 2010; Karpas & Domshlak, 2009; Wichlacz et al., 2022). LAMA counts a landmark as reached once its predecessors have been (Richter & Westphal, 2010), and Büchner et al. (2023) make this progression sound under reasonable orderings. Goal recognition scores observed agents by the landmarks their actions achieved (Pereira et al., 2017; 2020). All three assume the steps obey the domain's preconditions, which is exactly what a shortcut breaks. We add a test of each achieving step against those preconditions, the split between specification and environment that makes the test meaningful, and the four properties. Evaluations of language models on planning score only finished plans (Valmeekam et al., 2023; Stein et al., 2025). Our axiomatic approach follows Amigó et al. (2009; 2018) and Kleinberg (2002) (Appendix K.7). Benchmarks already write the checkpoints and planners extract the orderings. What was missing is a standard that joins them.

8 Limitations and conclusion

Exactness has a price. Our domains are generated job chains, chosen so that every legal run can be enumerated, and their landmarks and orderings are known by construction (Appendix H.6 perturbs both). Domains whose landmarks must be extracted are not covered. Support is settled when a landmark is first achieved, so later corrective work earns nothing. Weighted support also needs the preconditions of the executed actions, which web and tool environments rarely expose. The agents are small, with 1.0B to 1.7B parameters plus one model of 7.6B, so model-level correlations rest on six or seven points. Which models take the shortcut depends on option order, but what the metrics do with the shortcut does not (Appendix K.8).

A checkpoint list records where a run has been, not how it got there. Without the ordering and what each step required, no rate can tell work from its appearance. Small models already take unlabeled shortcuts, and the published rate pays for them. Benchmarks can close this gap without changing a task. They can publish the orderings and what each action requires, score supported landmarks in one pass, and report within-instance results beside pooled ones. Without preconditions, checking that prerequisites came first still catches most shortcuts. For agent builders, weighted support is a step-level signal that rewards the work itself. Harnesses should shuffle the actions they list, so an option's position is not mistaken for a model's preference. Progress should be earned, not declared.

Statements

Use of large language models. An AI assistant was used in four roles. It checked the definitions and propositions we proposed, suggested improvements to them, and assisted in drafting proofs. It

helped write the notebook code that generated the task instances and ran the agents, so the instances in Sections 5 and 6 are synthetic data produced by that code. It edited the text of this paper. During planning, it retrieved related articles and analysed them to help us understand the literature. We checked the proofs it drafted in two ways: the results proved in Appendix A pass every programmatic check released in the supplement, and a second author verified each proof independently. Our review of the drafts also found the first proof of Theorem 1 invalid, which the proof given here replaces. Four claims in an earlier draft did not survive this checking and were withdrawn rather than repaired.

Reproducibility. Appendix C gives the domains, the shortcut environment, the prompts, the parser and the failure handling; Appendix D gives the enumeration algorithm and its cross-check; Appendix J gives compute, seeds and what is released. The anonymized supplement contains the enumeration and domain code, the scoring code, the benchmark audit, the proof checks, the per-episode records for both studies, the figure scripts and the number registry. Every enumeration result reruns from code alone on a CPU in under a minute, and the enumeration is cross-checked against brute-force materialization on the two smallest shapes (Appendix D.2). A script checks every definition, theorem, proposition and Table 1 entry exhaustively on all tasks, orderings and trajectories up to three landmarks and three steps. Every result number in the main text is a macro whose value, source file and computation are recorded in the registry, and every appendix table is regenerated by the released scripts and notebooks; numbers that describe the design, such as episodes per model, and exact values from proofs are written in the text. The claims about released benchmark code rest on a notebook that records the file and line behind each verdict. Every number in Section 6 recomputes from the released records without model access, and replaying the released replies through the environments reproduces every per-episode score. Exact regeneration of the original episodes is not possible, because their sampling seeds were drawn from Python's salted hash (Appendix J); the rerun of Appendix H.5 uses fixed seeds and regenerates exactly.

Ethics. No human subjects and no personal data were involved. All seven models are open-weight and were used under their own licenses, listed in Appendix C.4. The code, episode records and data released with this paper are licensed CC BY 4.0. The finding of Section 6 concerns evaluation integrity: a benchmark that rewards skipped prerequisites gives both agents and the people building them an incentive to skip them, and that incentive grows as progress scores move from ranking systems to supervising them.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, 2021.

Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval*, 12(4):461–486, 2009. doi: 10.1007/s10791-008-9066-8.

Enrique Amigó, Damiano Spina, and Jorge Carrillo-de Albornoz. An axiomatic analysis of diversity evaluation metrics: Introducing the rank-biased utility metric. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 625–634, 2018.

Ben Bogin, Kejuan Yang, Shashank Gupta, Kyle Richardson, Erin Bransom, Peter Clark, Ashish Sabharwal, and Tushar Khot. SUPER: Evaluating agents on setting up and executing tasks from research repositories. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 12622–12645, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.702.

Clemens Büchner, Thomas Keller, Salomé Eriksson, and Malte Helmert. Landmark progression in heuristic search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 33, pp. 70–79, 2023. doi: 10.1609/icaps.v33i1.27180.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*, 2023.

Fernando Diaz. Offline preference-based trajectory evaluation. *arXiv preprint arXiv:2606.17541*, 2026.

Bradley Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1): 1–26, 1979.

Ronald A. Fisher. On the "probable error" of a coefficient of correlation deduced from a small sample. *Metron*, 1:3–32, 1921.

Luca Gioacchini, Giuseppe Siracusano, Davide Sanvito, Kiril Gashteovski, David Friede, Roberto Bifulco, and Carolin Lawrence. AgentQuest: A modular benchmark framework to measure progress and improve LLM agents. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations)*, pp. 185–193, Mexico City, Mexico, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-demo.19.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006. doi: 10.1613/jair.1705.

Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004. doi: 10.1613/jair.1492.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1728–1733, 2009.

Jon M. Kleinberg. An impossibility theorem for clustering. In *Advances in Neural Information Processing Systems 15 (NIPS)*, 2002.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

Shuyang Liu, Yang Chen, Rahul Krishna, Saurabh Sinha, Jatin Ganhotra, and Reyhan Jabbarvand. Process-centric analysis of agentic software systems. *Proceedings of the ACM on Programming Languages*, 10(OOPSLA1):163, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML)*, pp. 278–287, 1999.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations (ICLR)*, 2022.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. WebCanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*, 2024.

Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. Landmark-based heuristics for goal recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, pp. 3622–3628, 2017. doi: 10.1609/aaai.v31i1.11021.

Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. Landmark-based approaches for goal recognition as planning. *Artificial Intelligence*, 279:103217, 2020. doi: 10.1016/j.artint.2019.103217.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010. doi: 10.1613/jair.2972.

Priyam Sahoo, Gaurav Mittal, Xiaomin Li, Shengjie Ma, Benjamin Steenhoek, Pingping Lin, and Yu Hu. AgentLens: Revealing the lucky pass problem in SWE-agent evaluation. *arXiv preprint arXiv:2605.12925*, 2026.

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

E. H. Simpson. The interpretation of interaction in contingency tables. *Journal of the Royal Statistical Society, Series B (Methodological)*, 13(2):238–241, 1951.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, 2022.

Katharina Stein, Daniel Fišer, Jörg Hoffmann, and Alexander Koller. Automating the generation of prompts for LLM-based action choice in PDDL planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 35, pp. 250–259, 2025. doi: 10.1609/icaps.v35i1.36126.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2107–2116, 2018.

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

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 11279–11298, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.775.

Julia Wichlacz, Daniel Höller, and Jörg Hoffmann. Landmark heuristics for lifted classical planning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)*, pp. 4665–4671, 2022. doi: 10.24963/ijcai.2022/647.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, 2024a.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024b.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Melroy Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025), Datasets and Benchmarks Track*, 2025.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations (ICLR)*, 2025.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, 2025.

A PROOFS

This appendix proves every formal result in the main text and records the numerical checks each one passed. It supports the audit in Table 1 and the claims of Sections 3 and 4.

A.1 THEOREM 1

The proof in Section 3 is complete, and this subsection checks its four cells. P3 constrains pairs of trajectories within one task, so an impossibility needs two tasks in which P3 demands opposite rankings of the same pair, not one trajectory scored in two tasks.

The tasks. Both have $F = L = G = \{a, b\}$ and $I = \emptyset$. Π_1 has operators o_b ($\emptyset \to b$) and o_a^+ ($\{b\} \to a$). Its only way to make a true is o_a^+, which needs b, so in every plan b holds immediately before a first becomes true and $\mathrm{pred}(a) = \{b\}$, $\mathrm{pred}(b) = \emptyset$. Π_2 is the mirror image, with $\mathrm{pred}(b) = \{a\}$, $\mathrm{pred}(a) = \emptyset$. Both are ordering-faithful. The environment offers all four operators, and both trajectories are applicable in it.

The cells. In Π_1, τ first achieves b by o_b, which is sound because $\mathrm{pred}(b) = \emptyset$, and then a by o_a^+, which is sound because $b \in \mathrm{pre}(o_a^+)$. Since $\mathrm{first}(b) = 1 < 2 = \mathrm{first}(a)$, this gives $\mathrm{Sup} = \{a, b\}$. The other trajectory τ' first achieves a by o_a, which is unsound because $b \notin \mathrm{pre}(o_a)$, and then b by o_b^+, which is sound because $\mathrm{pred}(b) = \emptyset$. So $\mathrm{Sup} = \{b\}$. Π_2 is symmetric, with $\mathrm{Sup}(\tau) = \{a\}$ and $\mathrm{Sup}(\tau') = \{a, b\}$. Both trajectories achieve $\{a, b\}$, so P3's antecedent holds in each task, in opposite directions. □

The construction is minimal. With one landmark there is no ordering to read, and with two landmarks but only one trajectory P3 has nothing to compare.

Corollary 1 follows by reading the two tasks as two benchmark instances whose checkpoint lists are identical. The pair does not show that $\prec$ is necessary. A metric that counts the landmarks whose first achiever lies in the task's operator set gives τ both landmarks and τ' none in Π_1, and the reverse in Π_2, without reading $\prec$ (`ladder_checks.py`).

A.2 PROPOSITION 1 (NORMALIZATION)

(a) $r_{\mathrm{L}}(\tau) = \max_t |s_t \cap L|/|L|$, which equals 1 exactly when some s_t contains all of L.

(b) Consider a trajectory of advances alone in n consuming chains. Advancing job c to stage j requires $\ell_{c,j-1}$ and deletes it, and $\ell_{c,0}$ is consumed by the first advance and never re-added. A chain therefore holds no live landmark before its first advance and exactly one after it, so $|s_t \cap L|$

is the number of chains started by step t. That count never falls, its maximum is its final value c, and $r_{\mathrm{L}}(\tau) = c/|L|$. Every advance is a sound achiever and fires only after its predecessor, so by induction along each chain every landmark is supported. A trajectory that achieves every landmark starts all n chains, so $r_{\mathrm{L}}(\tau) = n/|L|$. Whenever some chain has more than one stage, $|L| > n$ and the value is below 1. □

The restriction to advances matters. An environment that could re-add a consumed stage would let a fully supported run hold more than one landmark per chain. The shortcuts of Proposition 4 are offered only before their stage is achieved, so they cannot.

The argument needs no condition on deletable predecessors, which is why Proposition 1 applies where Proposition 3 does not. It holds exactly on every complete run of the 11 distinct enumerated profiles, a check notebook G asserts, and of the depth vectors in `proof_checks.py`. Measured on complete, fully supported trajectories, the rate returns 0.333, 0.385, and 0.214 to 0.286 across the three consuming structures of Appendix C.1. Merge is the only structure where complete runs take more than one value, and it holds the lowest. It is also where Proposition 3 fails.

A.3 Proposition 2 (persistence)

If no operator deletes a landmark then $s_0 \cap L \subseteq \cdots \subseteq s_T \cap L$, so $|s_t \cap L|$ is non-decreasing and its maximum falls at $t = T$. Since nothing is deleted, $s_T \cap L = \mathrm{Ach}(\tau)$, giving $r_{\mathrm{L}}(\tau) = A(\tau)$. If in addition every operator achieves exactly one new landmark then $|\mathrm{Ach}(\tau)| = |\tau|$, so $r_{\mathrm{L}}(\tau) = |\tau|/|L|$, which depends on the trajectory only through its length. □

It holds on 6,000 random trajectories of persistent tasks in `proof_property_tests.py`, and the measured tie rate is exactly 1.000 in all three persistent domains.

A.4 Proposition 3 (consumption)

Write $\mathrm{pred}_{\mathrm{del}}(\ell) \subseteq \mathrm{pred}(\ell)$ for the deletable immediate predecessors of ℓ. Under the hypothesis, the step that achieves ℓ is a sound achiever, so $\mathrm{pred}(\ell) \subseteq \mathrm{pre}(o_t) \subseteq s_{t-1}$. Every deletable predecessor is therefore live when ℓ is achieved, and it is deleted at that step. The step adds one new landmark and deletes $|\mathrm{pred}_{\mathrm{del}}(\ell)| \leq 1$, so the live count changes by $1 - |\mathrm{pred}_{\mathrm{del}}(\ell)| \geq 0$. It never decreases, so its maximum falls at T. No step adds an achieved landmark again, so a landmark deleted once stays false, and the live set at T is

$$\mathrm{Ach}(\tau) \setminus \{\ell' : \ell' \in \mathrm{pred}_{\mathrm{del}}(\ell) \text{ for some } \ell \in \mathrm{Ach}(\tau)\},$$

a function of $\mathrm{Ach}(\tau)$. For P2, an exchange of the kind P2 describes applied to such a trajectory yields another trajectory satisfying the hypothesis, with the same operators and the same achieved set, so r_{L} is unchanged. □

Nothing here shows that r_{L} fails P3. The hypothesis assumes achievers that delete their live predecessors, while P3's antecedent needs an environment with unsound achievers, and there r_{L} is no longer a function of Ach. The pair in Appendix B.3 achieves identical landmark sets yet receives 0.333 and 0.667. The P3 failure of r_{L} rests on Theorem 1 and on that pair.

The condition is exact rather than incidental. The table below groups the maximum live count by achieved set, over four jobs at a budget of seven.

Structure	Max. deletable predecessors	Achieved sets with more than one value
chains	1	0, a function of Ach
shared, root protected	1	0, a function of Ach
merge	2	3, not a function of Ach
merge, root protected	2	2, not a function of Ach

Protecting the root does not rescue merge, so the operative condition is the deletable-predecessor count rather than persistence as such.

A.5 PROPOSITION 4 (MANIPULABILITY)

A shortcut to stage 1 would require nothing that stage 1 needs and so be a sound achiever, which is why shortcuts target stages 2 to d, and at most $d-1$ of them per chain.

A per-chain bound. Count the live landmarks of one chain. A shortcut adds a stage not yet achieved and deletes nothing, so it raises the count by one. The advance to stage 1 adds $\ell_{c,1}$ and consumes $\ell_{c,0}$, which is not in L and is never re-added, so it also raises the count by one and happens at most once. An advance to stage $j \geq 2$ consumes the live $\ell_{c,j-1}$ and adds $\ell_{c,j}$, which may already be live, so it changes the count by 0 or -1. A chain whose shortcuts have been taken s_c times therefore never holds more than $s_c + 1$ live landmarks, and never more than s_c if stage 1 was not reached.

Zero support. Reaching stage 1 supports it, because an advance to stage 1 is sound and $\mathrm{pred}(\ell_{c,1}) = \emptyset$. So with nothing supported, no chain reaches stage 1 and each holds at most $s_c \leq k$ live landmarks, which gives $f(0) \leq k/d$. Taking all kn shortcuts and nothing else attains it, within the budget since $kn < nd$.

Full support. A shortcut is offered only while its stage is unachieved, so any shortcut taken is that stage's first achiever, which is unsound. With every landmark supported, no shortcut was taken, so each chain holds at most one live landmark and $f(|L|) \leq 1/d$. The in-order run of $|L|$ advances attains it.

The maximum. The bound gives at most $(k+1)n$ live landmarks. Taking every shortcut and advancing each chain once attains it in $(k+1)n \leq nd$ steps, with exactly the n stage-1 landmarks supported. Since $0 < n < |L|$, and both endpoints, k/d and $1/d$, are strictly smaller than $(k+1)/d$, the maximum is attained only at partial support. □

The availability condition is needed once the budget exceeds $|L|$. With two spare steps, a fully supported run on two chains of depth three could re-add a consumed stage by a shortcut and reach $2/3$ rather than $1/3$. The checks for this proposition, Theorem 1 and Proposition 1(b) are in `proof_checks.py`.

The attainable rate is therefore not monotone in support, and the peak exceeds the zero-support value by a factor of $(k+1)/k$. At $k = 1$ the endpoints coincide and the peak is twice both, which is the weakest member of the family rather than the general case. At $k \geq 2$, supporting nothing beats supporting everything outright.

n	d	B	k	$f(0)$	k/d	peak	$(k+1)/d$	$f(\|L\|)$	$1/d$
2	4	8	1	0.250	0.250	0.500	0.500	0.250	0.250
2	4	12	2	0.500	0.500	0.750	0.750	0.250	0.250
3	3	9	1	0.333	0.333	0.667	0.667	0.333	0.333
3	3	12	2	0.667	0.667	1.000	1.000	0.333	0.333
2	3	8	1	0.333	0.333	0.667	0.667	0.333	0.333
2	3	10	2	0.667	0.667	1.000	1.000	0.333	0.333

Enumerating $f(j)$ at every support level confirms the shape. Two chains of depth three at budget six give 0.333 at zero support, a peak of 0.667 at partial support, and 0.333 at full support. Three chains of depth three at budget seven give 0.333, 0.667 and 0.333. Both tables are in `notebookC_results.json`.

A.6 LEMMA 1 (EQUAL LANDMARK COUNT)

If the legal operators are exactly "advance job c to its next stage", then each one adds $\ell_{c,j}$, which was not true before, because a chain reaches stage j only once. So each step adds exactly one member of $\mathrm{Ach}(\tau)$, giving $|\mathrm{Ach}(\tau)| = |\tau|$, and A is constant across legal trajectories of equal length. □

The landmark count equals B exactly in 12 of 12 configurations across the three structures, and no run stalls before the budget (`proof_property_tests.py`). This is what makes the equal-landmark-count control sound. The tie rate of 1.000 for landmarks achieved is structural rather than an artifact of coarse binning.

A.7 LEMMA 2 (ORDER INVARIANCE)

Lemma 2. *Let τ and τ' be trajectories whose operator sequences are permutations of each other, in which each landmark has the same first achiever and every pair of $\prec$-related landmarks is first achieved in the same relative order, ties included. Then* $\mathrm{Sup}(\tau') = \mathrm{Sup}(\tau)$*, so* $M_w(\tau') = M_w(\tau)$.

By Definition 5, membership of ℓ in $\mathrm{Sup}(\tau)$ depends only on whether the operator at $\mathrm{first}(\ell)$ is a sound achiever of ℓ, on whether $\mathrm{first}(\ell') < \mathrm{first}(\ell)$ for each $\ell' \in \mathrm{pred}(\ell)$, and, recursively, on the same data for those predecessors. A reordering of the stated kind changes none of these, because the first achiever of each landmark is unchanged by assumption and the relative order of $\prec$-related first achievements is preserved. Hence Sup, and so M_w, is unchanged. □

The argument needs no decomposition into adjacent exchanges and says nothing about the sequences in between, only about τ and τ'. We verified it over every legal ordering of the complete landmark set in three structures (90, 20 and 1,680 orderings respectively), each of which gives a single value, and on random reorderings of random trajectories (`proof_property_tests.py`).

A.8 THEOREM 2 (WEIGHTED SUPPORT)

P1. Extending τ to τ' adds steps at the end, so no existing $\mathrm{first}(\ell)$ changes and no first achiever changes. Every landmark supported in τ is therefore supported in τ', giving $\mathrm{Sup}(\tau) \subseteq \mathrm{Sup}(\tau')$ and, since $w > 0$, a non-decreasing numerator over a fixed denominator.

P2. Let τ' exchange steps t and $t+1$ as P2 describes. The states before step t and from step $t+1$ on coincide, because the two operators are independent. A landmark first achieved at step t in τ is not added by the other operator, so it is still false before its achiever fires in τ' and keeps that achiever. The same holds for step $t+1$. First achievements elsewhere are unchanged. The only pairs whose relative order changes are those first achieved at the two exchanged steps, which are unrelated under $\prec$. Lemma 2 applies.

P3. The antecedent gives $\mathrm{Sup}(\tau') \subsetneq \mathrm{Sup}(\tau)$, so the numerator drops by at least $\min_\ell w(\ell) > 0$.

P4. No landmark in L holds in I, so $\mathrm{Sup}(\langle I \rangle) = \emptyset$, which gives 0. A trajectory that reaches G with every landmark supported has $\mathrm{Sup} = L$, giving 1.

Complexity. Algorithm 1 makes one pass over τ, recording first and the soundness of each first achiever, then one topological sweep over L checking each landmark's predecessors. The first loop costs the total size of the add lists along τ, written $|\tau|$ here, and assumes constant-time membership tests on $\mathrm{pre}(o)$. The sweep costs $O(|L| + |\prec|)$. The linear bound depends on Definition 4 using immediate rather than transitive predecessors. The test $\mathrm{first}[\ell'] < \mathrm{first}[\ell]$ in the sweep never changes the output, because when $\mathrm{sound}[\ell]$ holds, $\ell' \in \mathrm{pre}(o_t) \subseteq s_{t-1}$, so ℓ' was first achieved before t. Dropping the soundness test instead leaves the timing-only rule of Table 1. □

Algorithm 1 Weighted support M_w

Input. Trajectory $\tau = \langle s_0, o_1, \ldots, o_T, s_T \rangle$, landmarks L, $\mathrm{pred}(\cdot)$ from $\prec$, weights w

$\mathrm{first} \leftarrow$ empty map, $\quad \mathrm{sound} \leftarrow$ empty map
for $t = 1 \ldots T$ **do**
  for $\ell \in \mathrm{add}(o_t) \cap L$ with $\ell \notin \mathrm{first}$ **do**
    $\mathrm{first}[\ell] \leftarrow t$
    $\mathrm{sound}[\ell] \leftarrow \big(\mathrm{pred}(\ell) \subseteq \mathrm{pre}(o_t)\big)$
$\mathrm{Sup} \leftarrow \emptyset$
for $\ell \in L$ in a topological order of $\prec$ **do**
  if $\ell \in \mathrm{first}$ and $\mathrm{sound}[\ell]$ and every $\ell' \in \mathrm{pred}(\ell)$ has $\ell' \in \mathrm{Sup}$ and $\mathrm{first}[\ell'] < \mathrm{first}[\ell]$ **then**
    $\mathrm{Sup} \leftarrow \mathrm{Sup} \cup \{\ell\}$
return $\sum_{\ell \in \mathrm{Sup}} w(\ell) / \sum_{\ell \in L} w(\ell)$

A.9 A CHARACTERIZATION OF WEIGHTED SUPPORT

P3$^+$ asks that $M(\tau) = g(\mathrm{Sup}(\tau))$ for a function g that increases strictly under inclusion. Call M *additive* if $M(\tau_3) + M(\tau_4) = M(\tau_1) + M(\tau_2)$ whenever $\mathrm{Sup}(\tau_3) = \mathrm{Sup}(\tau_1) \cup \mathrm{Sup}(\tau_2)$ and $\mathrm{Sup}(\tau_4) = \mathrm{Sup}(\tau_1) \cap \mathrm{Sup}(\tau_2)$. In words, the credit a supported set earns does not depend on what else is supported. A supported set is always a down-set of $\langle L, \prec \rangle$, since a landmark is supported only when its predecessors are. Assume every down-set is the supported set of some trajectory. In the job-chain domains here, that holds whenever the budget allows $|L|$ steps.

Characterization. A metric satisfies P3$^+$, P4 and additivity if and only if $M = M_w$ for some $w : L \to \mathbb{R}_{>0}$.

Proof. Weighted support depends on τ only through Sup, is a sum of positive weights over it and so rises strictly and is additive, and satisfies P4 by Theorem 2. Conversely, write $M(\tau) = g(\mathrm{Sup}(\tau))$. P4 gives $g(\emptyset) = 0$, since the empty run supports nothing, and $g(L) = 1$, since a run that reaches G with every landmark supported scores 1. Additivity says $g(S \cup S') + g(S \cap S') = g(S) + g(S')$ for down-sets. For $\ell \in L$ let $\downarrow\ell$ be ℓ with all its predecessors under the transitive closure of $\prec$, and set $w(\ell) = g(\downarrow\ell) - g(\downarrow\ell \setminus \{\ell\})$. Both sets are down-sets and the first strictly contains the second, so $w(\ell) > 0$. Let S be a nonempty down-set and ℓ a maximal element of S. Then $S \setminus \{\ell\}$ is a down-set, $S = (S \setminus \{\ell\}) \cup \downarrow\ell$ and $(S \setminus \{\ell\}) \cap \downarrow\ell = \downarrow\ell \setminus \{\ell\}$, so additivity gives $g(S) = g(S \setminus \{\ell\}) + w(\ell)$. By induction on $|S|$, $g(S) = \sum_{\ell \in S} w(\ell)$, and $g(L) = 1$ makes the weights sum to one, so $M = M_w$. □

W1 then restricts w to weights that do not decrease along $\prec$, and the depth weights are one such choice. Additivity across the connected components of $\langle L, \prec \rangle$ alone is weaker. It suffices when every component is a chain, as in the disjoint-chain domains, because the down-sets within a chain are nested and so satisfy the additivity equation trivially. It does not suffice when a component branches. With a shared root $r \prec a$, $r \prec b$, the values $g(\emptyset) = 0$, $g(\{r\}) = 1/4$, $g(\{r, a\}) = g(\{r, b\}) = 1/2$ and $g(L) = 1$ rise strictly and satisfy P1 to P4, yet $g(\{r, a\}) + g(\{r, b\}) \neq g(L) + g(\{r\})$, so no weights reproduce them.

A.10 REMARK 2 (FINAL-STATE RULES AND P1)

Binary success and weighted checkpoints evaluate predicates on s_T rather than a maximum over t. If a fact they test is deleted after being established, their value can fall as the trajectory is extended, which violates P1. They satisfy P1 exactly while the tested facts are never deleted. Appendix B.5 gives the witness.

B AUDIT WITNESSES

This appendix justifies every cell of Table 1, either by pointing to a proof in Appendix A or by giving a trajectory on which the property fails. It also records the constructed runs used by Proposition 4 and Remark 2.

B.1 CELL BY CELL

Rule	Prop.	Justification
S	P1	Conditional. It reads s_T, and Appendix B.5 gives an extension that takes it from 1.000 to 0.000
S	P2	Holds. An exchange of independent steps leaves s_T unchanged
S	P3	Fails. Both runs in Appendix B.3 reach G, so both score 1.000 while their supported sets differ
S	P4	Holds. It is 0 on $\langle I \rangle$ because $G \cap I = \emptyset$, and 1 on any run reaching G
r_{L}	P1	Holds. A maximum over a growing set of states cannot fall
r_{L}	P2	Conditional. Holds under the condition of Proposition 3 (Appendix A.4), where the maximum live count is a function of Ach. It fails on a merge. If m consumes x and y and z is independent, $\langle x, y, z, m \rangle$ holds three of the four landmarks at once and the exchanged $\langle x, y, m, z \rangle$ two
r_{L}	P3	Fails. Theorem 1 plus Appendix B.3, where identical achieved sets receive 0.333 and 0.667
r_{L}	P4	Fails. By Proposition 1, a complete, fully supported run scores $n/\lvert L \rvert$ in chains, and between 0.167 and 0.500 across all configurations measured
r_{G}	P1	Holds. Maximum over states
r_{G}	P2	Conditional. When no goal fact is ever deleted the goal count only rises, so its maximum is the final count, which the exchange preserves. Appendix B.2 gives an exchange that halves it when a goal is deleted
r_{G}	P3	Fails, by Theorem 1. Both runs in Appendix B.3 score 1.000
r_{G}	P4	Holds. It is 0 on $\langle I \rangle$ because $G \cap I = \emptyset$, and reaching G gives 1
A	P1	Holds. Ach only grows
A	P2	Holds. Ach is a set
A	P3	Fails. The antecedent fixes Ach equal, so A cannot separate the two runs, and both score 1.000
A	P4	Holds. $\mathrm{Ach}(\langle I \rangle) = \emptyset$ because $L \cap I = \emptyset$, and all of L gives 1
C	P1	Conditional, as for S, because the programmatic evaluators run on the final state (Appendix B.6)
C	P2	Holds for checkpoints evaluated on the final state, which the exchange preserves, or on the set of steps taken
C	P3	Fails, by Theorem 1. The milestone list carries no orderings, so the two runs are indistinguishable to it
C	P4	Holds when no checkpoint is met in I. It is then 0 on $\langle I \rangle$, and a run passing every checkpoint scores 1
Rows 6–8	P1–P4	DevAI, LAMA and ToolSandbox, in Appendix B.4
M_w	P1–P4	All hold, by Theorem 2 (Appendix A.8)

B.2 AN EXCHANGE THAT LOWERS THE GOAL-LEVEL RATE

Let $G = \{g, h\}$, and let the environment offer three operators without preconditions. The operator o_1 adds g, o_2 adds h, and o_3 adds a fact m and deletes g. The trajectory $\langle o_1, o_2, o_3 \rangle$ passes through $\{g\}$, $\{g, h\}$ and $\{h, m\}$, so $r_{\mathrm{G}} = 1$. Exchanging the last two steps gives $\{g\}$, $\{m\}$, $\{h, m\}$ and $r_{\mathrm{G}} = 1/2$. The exchange meets every condition of P2. Neither o_2 nor o_3 deletes anything the other requires or adds, each first achieves a different fact, and m need not be a landmark at all. Binary success is 0 before and after, since the final state is unchanged. In every domain of this paper goals are final stages or merges, which nothing deletes, so r_{G} satisfies P2 there.

B.3 THE CONSTRUCTED PAIR

Three jobs of three stages, consuming, with one additive shortcut per job. Stages are weighted $w(\ell_{c,j}) = j$, so the total weight is 18. The in-order run advances each job through its stages. The shortcut run marks each job complete first, then fills in the earlier stages. Both use nine steps and achieve all nine landmarks.

in-order run				shortcut-then-backfill run			
Step	Operator	Adds	Live	Step	Operator	Adds	Live
1	advance job0 to 1	$\ell_{0,1}$	1	1	mark job0 complete	$\ell_{0,3}$	1
2	advance job0 to 2	$\ell_{0,2}$	1	2	mark job1 complete	$\ell_{1,3}$	2
3	advance job0 to 3	$\ell_{0,3}$	1	3	mark job2 complete	$\ell_{2,3}$	3
4	advance job1 to 1	$\ell_{1,1}$	2	4	advance job0 to 1	$\ell_{0,1}$	4
5	advance job1 to 2	$\ell_{1,2}$	2	5	advance job0 to 2	$\ell_{0,2}$	4
6	advance job1 to 3	$\ell_{1,3}$	2	6	advance job1 to 1	$\ell_{1,1}$	5
7	advance job2 to 1	$\ell_{2,1}$	3	7	advance job1 to 2	$\ell_{1,2}$	5
8	advance job2 to 2	$\ell_{2,2}$	3	8	advance job2 to 1	$\ell_{2,1}$	6
9	advance job2 to 3	$\ell_{2,3}$	3	9	advance job2 to 2	$\ell_{2,2}$	6

In the in-order run every landmark is first achieved by an advance whose precondition is its immediate predecessor, so every achiever is sound and, since predecessors are achieved earlier, every landmark is supported. So $|\mathrm{Sup}| = 9$ and $M_w = 1.000$. In the shortcut run each $\ell_{c,3}$ is first achieved by "mark job c complete", which does not require $\ell_{c,2}$, so those three achievers are unsound and the three landmarks are unsupported however late $\ell_{c,2}$ arrives. The remaining six are supported, and the weights give $M_w = (1 + 2) \times 3/18 = 0.500$.

The live counts differ because forcing a final stage early costs nothing. The shortcut deletes no landmark, so the three final stages it sets sit alongside the six backfilled ones. The maximum is 3 of 9 for the in-order run and 6 of 9 for the shortcut run, giving 0.333 and 0.667. Both reach the goal, so $S = 1.000$, $r_{\mathrm{G}} = 1.000$ and $A = 1.000$ for both.

B.4 Rules that read the ordering

Three rules from prior work read an order, namely DevAI's and ToolSandbox's scores and LAMA's reached condition. We state each in the landmark model, with L as the checkpoints and $\prec$ as their order, and `ladder_checks.py` scores every witness below. The common witness for P3 is one job of three stages. The in-order run advances it to stages 1, 2 and 3, and the *advance, mark* run advances it to stages 1 and 2 and then marks it complete. Both achieve the three landmarks, but the second supports only two.

DevAI. A landmark counts when it was achieved and every immediate predecessor counts. DevAI judges requirements on the finished workspace, where facts persist, so met by the end means achieved (Zhuge et al., 2025). The count depends on τ only through Ach, which gives P1, P2 and P4 as for A. It gives both runs of Appendix B.3 full credit, so it fails P3.

LAMA. A landmark counts when it was achieved and every immediate predecessor counts and was first achieved earlier, which is Definition 5 without the soundness clause and the condition under which LAMA treats a landmark as reached (Richter & Westphal, 2010). P1 and P4 follow as for M_w, and P2 by the argument of Lemma 2, which never uses soundness. It scores the backfill run of Appendix B.3 below the in-order run, but gives the advance, mark run full credit, so it fails P3.

ToolSandbox. Milestones form a directed acyclic graph, and a trajectory scores the best average similarity over assignments of milestones to turns whose chronological order is a topological sort of the graph (Lu et al., 2025, Appendix A.7). With milestones that are facts and similarity 0 or 1, the score is the largest share of L that can be matched to states in which each landmark holds, with ℓ' matched before ℓ whenever $\ell' \prec \ell$. Extending τ keeps every match, so P1 holds for the milestone score. A minefield, which sets the score to zero, is outside the model. A supported run can match each landmark at its first achievement, which gives P4. The advance, mark run matches all three stages, so P3 fails. P2 fails as well, because a match reads when a fact still holds rather than when it was first achieved. Let a shortcut make ℓ true before its predecessor p, then achieve p, then take an unrelated step and a step that deletes ℓ. Exchanging the last two steps, which P2 allows, removes the only state in which ℓ holds after p, and the score loses one milestone. If related milestones may share a turn, the same happens without the unrelated step. Milestones on tool calls are events fixed to one turn and behave like the LAMA rule.

B.5 THE RESTART WITNESS

Two jobs of two stages, both completed, then one further step that restarts a job. Before the restart, binary success is 1.000. After it, the job's final-stage fact has been deleted, and binary success is 0.000. Every maximum-based rule is unchanged, because a maximum over states cannot fall when a state is appended. The progress rate stays at 0.500, and goal completion stays at its pre-restart value. Weighted support is unchanged too, since it is fixed at first achievements. Restarting a job is ordinary agent behaviour, which is why this is a witness rather than a contrivance.

B.6 HOW THEAGENTCOMPANY'S SCORE MAPS ONTO C

The reported composite is $\frac{1}{2}\,\mathrm{Result}/\mathrm{Total} + \frac{1}{2}\,S_{\mathrm{full}}$, where checkpoints are English milestones carrying point values and S_{full} is the full-completion indicator (Xu et al., 2025). The released evaluator computes points earned over points available after the run. In 59 of the 175 tasks at least one checkpoint is graded against the trajectory and the rest against the final state, which is why P1 is conditional in the same way as for binary success. P3 fails for the reason Theorem 1 gives. No checkpoint declares a prerequisite, so no evaluator built from the list can distinguish the two runs of Appendix B.3. In 54 tasks the evaluator also applies a strategy that, once the last checkpoint passes, awards full points for every earlier one (`workspaces/base_image/scoring.py`, lines 65–80). On those tasks a run that reaches only the last checkpoint scores the same as the in-order run. In the 19 runs released in TheAgentCompany's experiments repository (snapshot in the supplement), 81 of 3,056 task scores exceed the sum of their checkpoint results, which only this strategy can produce, on 31 distinct tasks. Moreover, 55 of the 525 full scores are full although an earlier checkpoint was not met. Figure 5 splits the full scores by run and shows that the bonus adds full scores to most runs, the strongest included, rather than to a few weak ones.

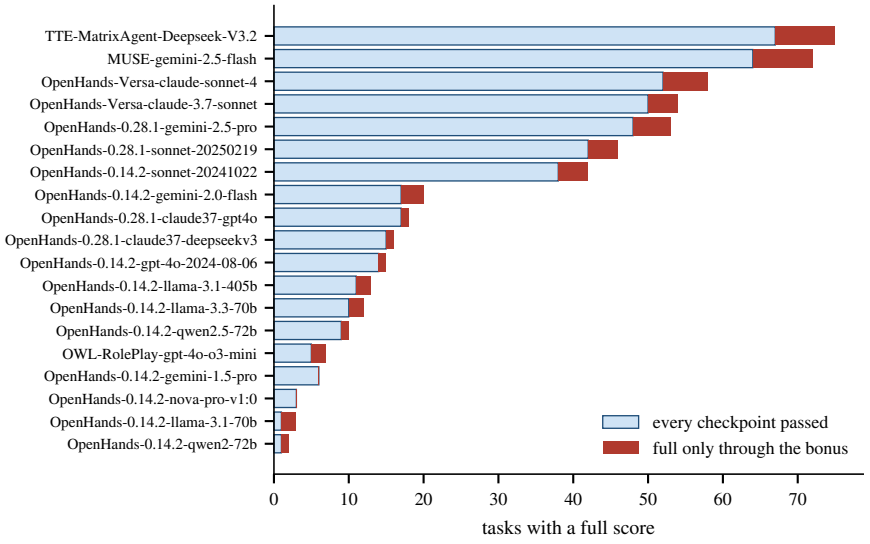


Figure 5: **TheAgentCompany's final-checkpoint bonus turns partial runs into full scores.** Tasks with a full score in each of the 19 released runs, split into those where every checkpoint passed (light) and those that are full only because passing the last checkpoint credited an earlier one that was not met (dark). The bonus accounts for 55 of the 525 full scores and adds them to most runs, the strongest included.

Nothing here depends on the task suite, only on the scoring rule.

B.7 WHY AGENTLENS IS NOT AUDITED

AgentLens scores process quality rather than progress (Sahoo et al., 2026). It penalizes blind retries and backtracking deliberately, so a run that repeats work scores lower than one that does not, which is a violation of P1 as we state it and a feature as they state it. Scoring it against P1 would hold it to a

target it does not claim. Its ordered-versus-unordered composition is the closest existing engineering treatment of what P2 and P3 separate, namely which achievements must keep their order and which may be reordered freely. Its reference, however, is built from observed passing traces rather than from necessary orderings. It therefore penalizes deviation from what was seen rather than violation of what is required. Section 7 places it accordingly.

C Domains, environments and agent setup

This appendix specifies the generated domains, the shortcut environment, and everything needed to rerun the agent studies of Section 6. It supports the setup claims there and in Section 5.

C.1 Structures

Three ordering structures are used, each in a consuming and a persistent version.

Chains. There are n disjoint jobs, and job c has stages $\ell_{c,1} \prec \cdots \prec \ell_{c,d_c}$. The operator *advance job c to stage j* requires $\ell_{c,j-1}$ and adds $\ell_{c,j}$. Under the consuming rule it also deletes $\ell_{c,j-1}$. The goals are the final stages.

Shared root. As chains, but with a single landmark $r \prec \ell_{c,1}$ for every c, so every chain needs r first. r is marked protected, meaning it is a prerequisite rather than a consumable, so no operator deletes it.

Merge. As chains, plus a layer of landmarks each requiring the final stages of two chains. The merges are the goals.

Two design choices were corrected while running these domains, and both are worth recording because the corrected version is what the results use. A single converging sink, one landmark requiring every chain end, leaves goal completion at zero until the very last step, so the correlation with goal completion is undefined by construction. It was replaced by the pairwise merges above, which grade. A shared prerequisite that is consumed models an exclusive resource rather than a precondition, which is not what a shared root is meant to represent, so it was marked protected instead. Proposition 3's condition holds for chains and for the protected shared root and fails for merge, and Appendix A.4 shows that protecting the merge root does not change that.

C.2 The shortcut environment

In study B the environment lists, alongside the ordinary advances, one action per unfinished job, *mark job c complete*. It is listed until job c has reached its final stage. It adds ℓ_{c,d_c} and deletes nothing, so it is additive in the sense of Proposition 4 with $k = 1$ per job. It requires nothing, so it is not a sound achiever of ℓ_{c,d_c}, whose immediate predecessor is ℓ_{c,d_c-1}. The option text carries no marker distinguishing it from an advance, and the prompt says nothing about it. Once a job is marked, the prompt moves it out of the `UNFINISHED` list, and an untouched job that is marked is offered no further advance. A job advanced first and then marked holds two landmarks, its current stage and its final one, and there the released environment has a bug. It reads the job's stage from whichever of the two it meets first when iterating over a set, so for some jobs it keeps listing an advance. Models chose such an advance 74 times, in 65 of the 180 episodes of the three models that did, and 32 of those added a skipped stage, the backfill of Figure 1(a). The rerun of Appendix H.5 lists nothing for a job that is marked or finished. A step adds no landmark only when it fails to parse (Appendix C.7) or re-advances a marked job to the final stage the mark already set, and an episode ends early once nothing is listed, when every job is marked or finished. This is why landmarks achieved in Table 2 falls below the step budget's share only for the three frequent markers. The prompt's only instruction, that every job must reach its final stage, is satisfied by marking.

C.3 The Theorem 1 instance

Facts $F = \{a, b\}$, $I = \emptyset$, $G = \{a, b\}$. The environment has four operators, none of which deletes anything, namely o_a ($\mathrm{pre} = \emptyset$, $\mathrm{add} = \{a\}$), o_b ($\emptyset$, $\{b\}$), o_a^+ ($\{b\}$, $\{a\}$) and o_b^+ ($\{a\}$, $\{b\}$). Specification Π_1 has operators $\{o_b, o_a^+\}$ and ordering $b \prec a$, and Π_2 has $\{o_a, o_b^+\}$ and $a \prec b$. The trajectories

are $\tau = \langle o_b, o_a^+ \rangle$ and $\tau' = \langle o_a, o_b^+ \rangle$. The supported sets are $\{a, b\}$ and $\{b\}$ in Π_1, and $\{a\}$ and $\{a, b\}$ in Π_2 (Appendix A.1).

C.4 AGENTS

Model	Parameters	License	Revision
Qwen/Qwen2.5-1.5B-Instruct	1.54B	Apache 2.0	`989aa7980e`
HuggingFaceTB/SmolLM2-1.7B-Instruct	1.71B	Apache 2.0	`31b70e2e86`
ibm-granite/granite-3.1-1b-a400m-instruct	1.33B (0.4B active)	Apache 2.0	`0da7a48b02`
google/gemma-3-1b-it	1.00B	Gemma Terms of Use	`dcc83ea841`
allenai/OLMo-2-0425-1B-Instruct	1.48B	Apache 2.0	`48d788eca8`
meta-llama/Llama-3.2-1B-Instruct	1.24B	Llama 3.2 Community	`9213176726`
Qwen/Qwen2.5-7B-Instruct (rerun only)	7.62B	Apache 2.0	`a09a35458c`

Revisions are the head of each model's repository on the day both studies ran, 21 September 2026, read from the Hugging Face API, and were unchanged when the rerun ran. Parameter counts come from each checkpoint's safetensors metadata.

Generation used a single T4 GPU in float16, with transformers 5.0.0 and torch 2.10.0 under Python 3.12. Decoding was sampled at temperature 0.7, top-p 0.95, with at most 48 new tokens per step. Instances are generated with seeds 0 to 19. Each has 8 jobs, with each depth drawn uniformly from $\{2, 3\}$, giving 16 to 23 landmarks and a mean of 20. The depth vector for each seed is produced by the generator in the released code, so the seeds, not the vectors, are the specification.

C.5 PROMPTS

The rerun of Appendix H.5 uses the same prompts with study B's options shuffled at every step. Study A, no shortcut. Options are listed in sorted order, and the displayed stage index starts at 0, so an untouched job of depth three reads "stage 0 of 3".

```
Every job must reach its final stage.

FINISHED (<k>): <comma-separated finished jobs, or "none">
UNFINISHED (<m>):
  job0 is at stage <j> of <d>
  ...

CHOOSE ONE:
- advance job0 to stage <j+1>
- ...

Reply with one line copied exactly from CHOOSE ONE.
```

Study B, shortcut available. The finished-job line and the count after `UNFINISHED` are absent here. Beyond the extra options, nothing else differs.

```
Every job must reach its final stage.

UNFINISHED:
  job0 is at stage <j> of <d>
  ...

CHOOSE ONE:
- advance job0 to stage <j+1>
- mark job0 complete
- ...

Reply with one line copied exactly from CHOOSE ONE.
```

C.6 PARSER

A reply is resolved by three rules, tried in order.

1. Exact containment of one of the listed options.
2. The pattern `job N ...stage M` in study A, or `mark job N` in study B.
3. The first listed option naming job `N`, for any job the reply mentions.

Rule 3 cannot manufacture a shortcut. Options are sorted, and whenever `mark job N complete` appears, `advance job N to stage ...` appears before it, so the first listed option naming job `N` is always the advance. Taking the shortcut requires the reply to contain `mark job N`, which is rules 1 and 2. This matters for Section 6, because it means the 203 episodes that used the shortcut did so because the model asked for it. The rerun parses against the sorted list whatever order it displays, so the same holds there.

C.7 FAILURE HANDLING

A reply that no rule resolves is a parse failure. The step is consumed and the state is left unchanged, and an episode ends after 6 failures. As a consequence, episodes do not all reach the same number of landmarks (Appendix K.6).

D ENUMERATION

This appendix gives the enumeration algorithm, its cross-check against brute force, and the full per-configuration results behind Section 5 and Figure 3.

D.1 ALGORITHM

A state is the stage vector of the n jobs together with the running maxima of the true-landmark count and the goal count. The transition advances any job that has not reached its final stage. Starting from the initial state with count 1, the counts are aggregated by state after each of the B steps. Two runs that reach the same state are interchangeable from then on, so the aggregation loses nothing and the resulting counts are exact rather than sampled. The main configuration has 445 distinct final states standing for 6,014,400 runs, which is why the dynamic program finishes in under a second where materializing runs one at a time does not finish at all.

D.2 CROSS-CHECK

On the two smallest shapes the runs were also materialized one by one and the resulting joint distributions compared cell by cell with the dynamic program. Four jobs of depth two at budget six gives 1,440 runs, and four jobs of mixed depth at budget eight gives 24,360. In both, the two distributions are identical, not merely equal in total. The check is in `weights_sweep.py` in the supplement and reruns in seconds.

D.3 PER-CONFIGURATION RESULTS

Four shapes, each with five depth draws, give 20 instances. Jobs are interchangeable, so draws that permute the same depths yield identical distributions. The table lists each distinct profile once, with depths in sorted order and the draws that produced it, and the check against the stored joint distributions is in the notebook rather than assumed. The 4-job, depth-2 shape draws the same profile five times. One profile, depths 2,2,2,3 at budget 8, admits a single rate level, since every run must touch all four jobs, so it carries no information about the ordering. Mean goal completion falls strictly at every step up the rate in all 10 of the 10 remaining profiles, which is the monotone check of Section 5.

Shape	Depths	Draws	Landmarks	Legal runs	Rate range	Spread	Falls strictly
4 jobs, $B = 8$	2,3,3,3	0	11	24,360	0.273–0.364	1.33×	yes
4 jobs, $B = 8$	2,2,2,3	1,2	9	7,560	0.444	1.00×	one level
4 jobs, $B = 8$	2,2,3,3	3,4	10	14,840	0.300–0.400	1.33×	yes
5 jobs, $B = 8$	2,3,3,3,3	0	14	227,640	0.214–0.357	1.67×	yes
5 jobs, $B = 8$	2,2,2,3,3	1,3	12	135,240	0.250–0.417	1.67×	yes
5 jobs, $B = 8$	2,2,2,2,3	2	11	96,600	0.364–0.455	1.25×	yes
5 jobs, $B = 8$	2,2,3,3,3	4	13	178,920	0.231–0.385	1.67×	yes
6 jobs, $B = 9$	2,3,3,3,3,3	0	17	6,014,400	0.176–0.353	2.00×	yes
6 jobs, $B = 9$	2,2,2,3,3,3	1,4	15	3,932,880	0.200–0.400	2.00×	yes
6 jobs, $B = 9$	2,2,2,2,3,3	2,3	14	3,049,200	0.286–0.429	1.50×	yes
4 jobs, $B = 6$	2,2,2,2	0,1,2,3,4	8	1,440	0.375–0.500	1.33×	yes

The main configuration's disjoint support is its strongest case. In depths 2,2,2,3,3,3 every lowest-rate run finishes three goals while four are possible, and only runs at the second-lowest level reach four.

Shape	Policy	Rate	Percentile	Goal completion
4 jobs, $B = 8$	serial	0.273	13.8	0.750
	parallel	0.364	100.0	0.250
	closest-to-done	0.273	13.8	0.750
5 jobs, $B = 8$	serial	0.214	4.4	0.600
	parallel	0.357	100.0	0.200
	closest-to-done	0.214	4.4	0.600
6 jobs, $B = 9$	serial	0.235	16.6	0.500
	parallel	0.353	100.0	0.167
	closest-to-done	0.235	16.6	0.500
4 jobs, $B = 6$	serial	0.375	25.0	0.750
	parallel	0.500	100.0	0.500
	closest-to-done	0.375	25.0	0.750

D.4 Scripted policies

Three policies are defined on the same state. *Serial* advances the lowest-index unfinished job. *Parallel* advances the least advanced job, breaking ties by index. *Closest-to-done* advances the most advanced unfinished job. The percentile column gives the share of legal runs scoring at or below that policy's rate, so a high percentile means the rate flatters the policy. Serial and closest-to-done coincide in these configurations, both landing in the lowest or second-lowest rate level with the most goals the budget allows, while parallel play takes the top rate with the fewest completed.

E Other structures and ties

This appendix reports the six domains of Section 5 in full. They cover three ordering structures, each in a consuming and a persistent version, and the tables give complete-run rates, correlations and tie rates. Appendix C.1 defines the structures. The complete-run column is computed at the same configuration as the rest of the row. In merge, complete runs take two values depending on the order in which chains finish, and the column gives both, with 98.4% of complete runs at the higher.

		correlation with goals		tie rate			
Structure	Complete run r_{L}	r_{L}	M_w	r_{L}	r_{G}	A	M_w
chains, consuming	0.333	−0.563	+0.930	0.697	0.455	1.000	0.386
shared root, consuming	0.385	−0.592	+0.915	0.541	0.480	1.000	0.296
merge, consuming	0.214–0.286	−0.422	+0.641	0.821	0.895	1.000	0.439
chains, persistent	1.000	constant	+0.930	1.000	0.455	1.000	0.386
shared root, persistent	1.000	constant	+0.915	1.000	0.480	1.000	0.296
merge, persistent	1.000	constant	+0.641	1.000	0.895	1.000	0.439

All six use four jobs of depth three at budget $\lfloor 2|L|/3 \rfloor$, and every correlation is weighted by run counts over the complete set of legal runs. A complete, fully supported run scores 0.333 in chains, 0.385 with the shared root, and 0.214 to 0.286 in merge. Mean tie rates over the six domains are 0.843 for the progress rate, 0.610 for the goal-level rate, 1.000 for landmarks achieved and 0.373 for weighted support. Three readings are worth making explicit. In the consuming rows the progress rate is negatively correlated with goal completion and weighted support is strongly positive, and in the persistent rows the rate is constant, so its correlation is undefined rather than zero. This follows from Proposition 2 rather than from a measurement. Landmarks achieved ties every pair of runs, which is Lemma 1 showing up as a tie rate of 1.000, and it is also why the enumeration is a fair test, since the runs being compared reached the same number of landmarks. Finally, the tie rates place the rate and weighted support far apart, 0.843 against 0.373 on average, so the rate is not merely mis-ordering behaviour but failing to distinguish it.

E.1 Size sweep

The domains above were rebuilt from their structural descriptions and checked against the released results. All 36 quantities, four tie rates and two correlations for each of the six domains, agree to four decimals. That check licenses the sweep below, which the original notebook ran but stored no output for. The complete-run column gives every value a complete run can take, computed by the same enumeration.

Structure	Shape	Landmarks	Legal runs	Complete run r_{L}	Corr. r_{L}	Corr. M_w
chains	(3,3)	9	510	0.333	-0.717	$+0.960$
chains	(4,3)	12	36,120	0.333	-0.563	$+0.930$
chains	(5,3)	15	4,229,400	0.333	-0.572	$+0.930$
chains	(4,4)	16	722,400	0.250	-0.385	$+0.863$
merge	(3,3)	10	510	0.200–0.300	undefined	undefined
merge	(4,3)	14	103,200	0.214–0.286	-0.422	$+0.641$
merge	(5,3)	17	15,120,600	0.176–0.294	-0.372	$+0.605$
merge	(4,4)	18	6,991,600	0.167–0.222	-0.332	$+0.537$

The negative correlation of the rate with goal completion survives every shape in which goal completion varies, between -0.332 and -0.717, and weakens as jobs get deeper because a deeper chain spends more steps before its goal can move. Merge at $(3, 3)$ admits no completed merge inside the budget, so goal completion is constant there and both correlations are undefined rather than zero. The largest reference set is just over 15 million runs.

F Granularity

The progress rate is $r_K(\tau) = \max_t |s_t \cap K|/|K|$ for a subgoal set K, and benchmarks choose K themselves. This appendix shows that the choice is not cosmetic. The same formula on the same trajectory can rank two strategies in opposite orders depending on whether K is the landmark set or the goal set. It supports the separate rows for r_{L} and r_{G} in Table 1.

Take ten jobs of two stages each at a budget of ten actions, consuming, so $|L| = 20$ and $|G| = 10$. Breadth-first play advances every job once. Ten landmarks are true at the end and none of the jobs is finished, so $r_{\mathrm{L}} = 0.500$ and $r_{\mathrm{G}} = 0.000$. Serial play finishes five jobs. Each finished job holds only its final stage, since reaching stage two deletes stage one, so five landmarks are true and $r_{\mathrm{L}} = 0.250$, while $r_{\mathrm{G}} = 0.500$. The two rules disagree about which strategy did better, by a factor of two in each direction, on runs of identical length.

The same reversal appears in the enumerated configurations, where the policy table of Appendix D.4 reports both quantities. Parallel play takes the highest rate in every shape and the lowest goal completion, and serial and closest-to-done play take the lowest rate of the three and the highest goal completion. Nothing about the trajectories changes between the two columns. Only K does.

This is why the audit in Section 4 lists r_{L} and r_{G} as separate rules rather than as one rule with a parameter. They differ on P4, which r_{G} satisfies and r_{L} does not (Proposition 1), and they satisfy P2 under different conditions. A benchmark that reports "progress" without stating its subgoal set has not said which of the two it means.

The comparison across shapes and policies is below. Serial and closest-to-done play take the lower rate and the higher goal completion in every shape. Parallel play takes the higher rate and, in every shape deeper than two stages, finishes nothing at all.

Shape	Policy	r_{L} (landmark level)	r_{G} (goal level)
4 jobs $\times$ 2, $B = 6$	serial	0.375	0.750
	parallel	0.500	0.500
	closest-to-done	0.375	0.750
4 jobs $\times$ 3, $B = 8$	serial	0.250	0.500
	parallel	0.333	0.000
	closest-to-done	0.250	0.500
5 jobs $\times$ 3, $B = 8$	serial	0.200	0.400
	parallel	0.333	0.000
	closest-to-done	0.200	0.400
6 jobs $\times$ 3, $B = 9$	serial	0.167	0.500
	parallel	0.333	0.000
	closest-to-done	0.167	0.500
10 jobs $\times$ 2, $B = 10$	serial	0.250	0.500
	parallel	0.500	0.000
	closest-to-done	0.250	0.500

The ten-job row is the worked example above. These runs are scripted rather than enumerated, so the table shows what the two granularities do to three specific strategies rather than to the whole reference set. Appendix D.3 gives the enumerated version.

G WEIGHT SENSITIVITY

Theorem 2 holds for every strictly positive w, so the properties do not depend on the weighting. What the weighting does affect is how strongly M_w tracks goal completion, and this appendix measures that over the complete set of legal runs in the main enumerated configuration. That configuration has 6,014,400 runs of 6 jobs at budget 9, each reaching the same number of landmarks, and correlations are weighted by run counts.

Scheme	$w(\ell_{c,j})$	Correlation with goal completion
uniform	1	constant
linear	j	$+0.730$
quadratic	j^2	$+0.751$
exponential	2^j	$+0.755$
adversarial (violates W1)	$d_c - j + 1$	-0.905
progress rate r_{L}	–	-0.553

Under legal play every achieved landmark is supported, so uniform weights make M_w equal to landmarks achieved, which Lemma 1 fixes at a constant, so its correlation is undefined rather than zero. The three increasing schemes agree closely, from $+0.730$ to $+0.755$, so the positive relation reported in Section 5 is not an artifact of the linear weights used throughout the paper. We tested three increasing schemes, not all of them, and state it that way.

The adversarial scheme is the informative row. It weights early stages most heavily, which violates W1, and it inverts the correlation to -0.905. A metric can satisfy P1 to P4 and still be useless for ranking agents if its weights contradict the ordering they are built on, which is what W1 rules out and why it is stated as a modelling constraint rather than as a fifth property.

The same sweep over the six domains of Appendix E agrees.

Domain	uniform	linear	quadratic	exponential	adversarial
chains, consuming	undefined	+0.930	+0.966	+0.975	−0.930
chains, persistent	undefined	+0.930	+0.966	+0.975	−0.930
shared, consuming	undefined	+0.915	+0.957	+0.967	−0.915
shared, persistent	undefined	+0.915	+0.957	+0.967	−0.915
merge, consuming	undefined	+0.641	+0.714	+0.781	−0.641
merge, persistent	undefined	+0.641	+0.714	+0.781	−0.641

The ordering of the schemes is the same everywhere, the persistent and consuming versions of a structure give identical figures because weighted support does not read deletions, and the adversarial scheme inverts the sign in all six.

The released agent episodes carry their supported sets, so they can be rescored the same way. The table gives the within-instance correlation with goal completion, averaged over instances as in Section 6.

Episodes	uniform	linear	quadratic	exponential	adversarial
study A, no shortcut	+0.196	+0.497	+0.465	+0.454	−0.513
study B, shortcut offered	−0.933	−0.896	−0.847	−0.882	−0.963

In study A the three increasing schemes agree within overlapping intervals and the adversarial one reverses the sign, as in enumeration. Uniform weights give the correlation of landmarks achieved, because under legal play every achieved landmark is supported. The order among the increasing schemes is the reverse of enumeration's, with linear highest, but the differences are small. In study B every scheme correlates negatively, because goal completion there counts the goals a shortcut sets, and no weighting of supported landmarks credits a marked job.

H PER-MODEL AGENT RESULTS

This appendix gives both agent studies broken down by model, the analysis that explains the one model disagreeing with the other five, and the interval procedure. It supports Section 6 and Table 2.

H.1 STUDY A, NO SHORTCUT OFFERED

Model	r_{L}	Goals	A	M_w	Parse fail./ep	Switches/ep	Corr.	Inst.
Qwen2.5-1.5B	0.234	0.416	0.503	0.479	0.00	4.84	−0.626	15
SmolLM2-1.7B	0.252	0.367	0.494	0.450	0.20	6.34	−0.554	17
granite-3.1-1b-a400m	0.201	0.405	0.461	0.453	0.83	3.39	+0.146	14
gemma-3-1b	0.225	0.435	0.503	0.485	0.00	4.66	−0.799	9
OLMo-2-0425-1B	0.271	0.359	0.503	0.443	0.01	7.02	−0.417	14
Llama-3.2-1B	0.225	0.425	0.503	0.489	0.00	4.34	−0.824	11
uniform over lines	0.313	0.272	0.503	0.405	–	8.03	–	–
first listed line	0.212	0.463	0.503	0.502	–	3.20	–	–

Each row covers 100 episodes, 20 instances by 5 draws. The last two columns give the within-instance correlation of the progress rate with goal completion for that model alone, and the number of instances on which it is defined. That count varies from 9 to 20 because instances where either quantity is constant for that model are dropped. This is why the primary analysis in Section 6 pools models within an instance, where it is defined on all 20.

H.2 THE MODEL THAT DISAGREES

Five models give correlations between −0.824 and −0.417. Granite gives +0.146, and it is also the model that fails to parse most often, at 0.83 failures per episode against 0.00 to 0.20 for the rest. Across the 6 models, the parse-failure rate correlates +0.912 with the per-model correlation and the job-switch rate correlates −0.252, so the association is with parsing rather than with strategy.

The mechanism follows from the setup rather than from the correlation. A parse failure consumes a step and leaves the state unchanged, so it lowers both the rate and goal completion for reasons that have nothing to do with how the agent was working, which pushes the two scores together. Restricting to the 521 episodes with no parse failure at all, 86.8% of the study, gives -0.559, slightly stronger than the headline figure. -0.439 over all episodes is the primary figure. The clean-episode check was added after we saw granite's parse failures, and it is reported as that.

H.3 Interval procedure

Intervals are 95% percentile bootstrap intervals over the per-instance correlations, not over episodes. The 20 per-instance values are resampled with replacement 10,000 times, drawing indices from Python's `random.Random` seeded with 0. The mean is taken each time, and the sorted means at positions 250 and 9,750 are reported. `agent_section_numbers.py` in the supplement implements exactly this and reproduces $[-0.540, -0.327]$, $[+0.372, +0.610]$ and $[+0.117, +0.270]$ exactly. Resampling instances rather than episodes is the conservative choice here, because episodes within an instance share a task and are not independent.

H.4 Study B, shortcut offered

Table 2 gives the per-model scores. Parse failures in this study run from 0.00 per episode for Qwen to 0.10 for granite, with SmolLM2 at 0.00, OLMo at 0.07, and gemma and Llama at 0.00. They are too rare to explain the abstention pattern, which option position does explain (Appendix H.5).

Within an instance, the number of shortcuts an episode uses correlates $+0.921$ with the progress rate, $+0.949$ with goal completion, -0.978 with weighted support and -0.509 with landmarks achieved, the last defined on 20 of 20 instances. These are means of per-instance correlations over 18 episodes each. The corresponding model-level figures, over 6 points, are $+0.988$ for the rate and -0.997 for weighted support. We keep the two units apart throughout and never mix them in one sentence.

H.5 Option position and the shuffled rerun

Both studies list options in sorted order, so in study B every "mark" line follows every "advance" line. The rerun repeats study B on the same 20 instances with 3 draws and the same prompts, decoding and parser, but with two changes. The displayed order is shuffled at every step by a seeded generator, and a job that is marked or finished lists nothing (Appendix C.2). Qwen2.5-7B-Instruct joins the six models and also runs the sorted designs of both studies, in float16 across two T4 GPUs. Seeds derive from `hashlib`, so the rerun regenerates exactly. Study B's prompt omits the line listing finished jobs and the count of unfinished ones. The rerun keeps both prompts unchanged, so only the order differs.

	study A	study B, sorted			study B, shuffled	
Model	first line	first line	last line	mark rate	first line	mark rate
granite-3.1-1b-a400m	0.96	0.19	0.17	0.79	0.51	0.95
SmolLM2-1.7B	0.56	0.32	0.10	0.49	0.41	0.74
OLMo-2-0425-1B	0.23	0.28	0.30	0.51	0.30	0.65
gemma-3-1b	0.64	0.65	0.00	0.04	0.29	0.53
Llama-3.2-1B	0.85	0.95	0.00	0.01	0.81	0.50
Qwen2.5-1.5B	0.57	0.39	0.00	0.00	0.29	0.38
Qwen2.5-7B	0.94	1.00	0.00	0.00	0.50	0.22

Shares are of parsed steps, and a mark is about half the listed lines at every step (about 0.49 in the sorted design and 0.50 shuffled). In the sorted design the three models that abstained copy the first line in 39% to 95% of steps and so seldom meet a mark, and Qwen2.5-7B copies it in every step and scores exactly as the first-line reference of Table 2. Granite is the exception. It chooses the first line in 19% of steps and marks in 0.79 of them. Shuffled, every model marks, at rates from 0.22 to 0.95 against a listed share of 0.50. Granite, SmolLM2 and OLMo mark above that share, and the two Qwen models below it. Across the seven models the rate's mean orders them largely by marks per episode (Kendall's τ $+0.714$), and weighted support reverses that order

exactly (−1.000). Within an instance, marks correlate +0.613 [+0.551, +0.671] with the rate, +0.942 with goal completion, −0.572 with landmarks achieved and −0.967 [−0.971, −0.962] with weighted support. Against supported goal completion, the rate correlates −0.689 and weighted support +0.769 [+0.720, +0.814].

Position also undoes a reading this section could invite, that the models taking the shortcut are the weaker ones. The three heavy users of the sorted design had lower weighted support in study A (0.449 against 0.484), but a model that copies the first line plays serially there, which maximizes goal completion at this budget (the first-line reference reaches 0.463), so that comparison confounds position with skill and we draw no conclusion from it. Figure 6 shows the confound per model.

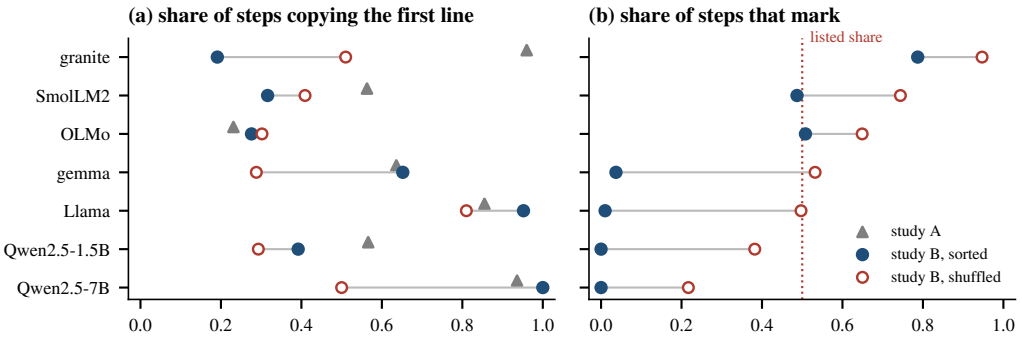


Figure 6: **Option position decides which models meet the shortcut.** (a) Share of steps on which each model copies the first listed line, in study A, in study B with options sorted, and in the shuffled rerun. (b) Share of steps that mark a job complete, sorted and shuffled, against the share of listed lines that are marks (dotted). Sorted, every mark sits below every advance, so models that copy the first line rarely reach one. Shuffled, every model marks.

Without preconditions, support falls back to the timing-only rule of Table 1. On the released episodes it catches every mark taken before the job reached its penultimate stage and none taken after, which is 940 of 1,066 marks in the sorted design and 2,019 of 2,262 shuffled. It still orders 15 of 15 and 18 of 21 model pairs opposite to the rate, and within an instance it correlates −0.961 and −0.906 with marks, against −0.978 and −0.967 for weighted support, and +0.883 and +0.617 with supported goal completion. Most of the separation survives, and what is lost is the mark taken after its prerequisite, which only the precondition test sees.

H.6 Wrong orderings and noisy detection

Weighted support trusts two inputs a benchmark would supply, namely the ordering and the detection of each landmark. We perturb each on the released episodes, averaging seeded draws per condition (`R1_cpu_analysis.ipynb`), and compare with the unperturbed scores through Kendall's τ between the model rankings by mean score and through the within-instance correlation used in Section 6. That correlation is with goal completion in study A and with marks per episode in study B, sorted design.

Ordering errors. Dropping an edge removes a check. Under legal play every achiever is sound, so dropped edges change nothing in study A. In study B a mark escapes exactly when the edge into its stage is dropped, so dropping a share p of edges lets about p of the marks through. Adding an edge between stages of different jobs, consistent with a random linear extension so the order stays acyclic, makes a legal advance fail the soundness test when the new predecessor is not among its preconditions. That hurts study A, where every landmark was supported before, more than study B.

	study A		study B, sorted		
Edges	τ	corr. goals	τ	corr. marks	marks caught
none	1.000	+0.497	1.000	−0.978	1.00
drop 10%	1.000	+0.497	0.992	−0.959	0.90
drop 25%	1.000	+0.497	0.925	−0.904	0.74
drop 50%	1.000	+0.497	0.872	−0.654	0.50
add 5%	0.675	+0.359	0.955	−0.964	1.00
add 10%	0.576	+0.276	0.965	−0.950	1.00
add 20%	0.445	+0.189	0.853	−0.922	1.00

Percentages are of the true edges (drop) or of $|L|$ (add). Marks caught is relative to the true ordering.

Detection noise. A missed landmark removes its first achievement, and with it the support of everything resting on it. A spurious landmark is added at a random step with no achiever, so it is supported only if it has no predecessor. Missed landmarks can be partly recovered. A sound achiever that was detected shows that its preconditions held, so they can be inferred ("inferred" rows). The table gives τ for weighted support, landmarks achieved and the rate.

	study A			study B, sorted		
Landmarks	M_w	A	r_{L}	M_w	A	r_{L}
missed 5%	0.725	0.589	0.925	0.960	0.731	0.979
missed 10%	0.584	0.525	0.891	0.933	0.704	0.963
missed 20%	0.416	0.496	0.864	0.843	0.624	0.939
missed 5%, inferred	0.925	0.720	0.925	1.000	0.725	0.979
missed 10%, inferred	0.851	0.725	0.891	0.989	0.725	0.963
missed 20%, inferred	0.845	0.664	0.864	0.949	0.728	0.939
spurious 5%	0.997	0.576	0.933	1.000	0.731	0.976
spurious 10%	0.997	0.496	0.896	1.000	0.696	0.933
spurious 20%	0.989	0.427	0.867	0.992	0.589	0.933

Spurious landmarks barely move weighted support and move landmarks achieved most. Missed landmarks hurt weighted support in study A, where one missed stage unsupports the rest of its job, and inference restores most of the agreement. Without inference, at the highest rate, weighted support falls below landmarks achieved there. Without inference, the rate moves least under missed landmarks, which is stability of a ranking that Section 6 shows runs against goal completion. Figure 7 plots the same agreement against the size of the error.

I THE GREEDY STUDY, AND WHY IT WAS WITHDRAWN

An earlier version of Section 6 used greedy decoding. It is described here because its replay summary is in the supplement, and a reader who finds it should know why no claim rests on it.

I.1 DESIGN

The greedy study used the same six models, the same 20 instances of 8 jobs and the same budget of 10 steps, but decoded greedily rather than by sampling. Greedy decoding is deterministic, so each model produces exactly one trajectory per instance. That gives six episodes per instance instead of the 30 that study A produces.

I.2 WHY IT SUPPORTED NO CORRELATIONAL CLAIM

A within-instance correlation over six points, one per model, mixes two things that sampling separates, namely how the metric responds to different behaviour and how the six models differ from one another. It also leaves almost nothing to correlate. Replayed, the six models produce on average 3.1 distinct values of goal completion per instance and 3.05 distinct values of the progress rate, so roughly half the points in each instance are tied with another point. An early pooled reading of $+0.549$ was computed on this design and then withdrawn, and it is not an estimate of anything the paper claims.

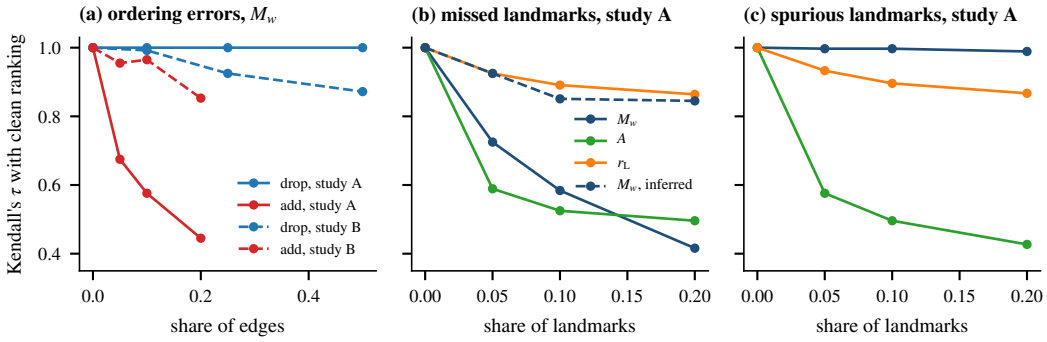


Figure 7: **How far each score's model ranking moves when its inputs are wrong.** Kendall's τ between the model ranking under perturbed inputs and the error-free ranking. (a) Weighted support with a share of ordering edges dropped or added, both studies. (b, c) Study A with a share of landmarks missed or spuriously detected, for weighted support with and without inferred preconditions, landmarks achieved and the rate. Dropped edges and spurious landmarks leave weighted support's ranking largely intact. Added edges and missed landmarks move it more, and inferring the preconditions of detected achievers recovers most of the loss from missed ones. The tables above give the values and study B.

Sampling replaced the design rather than repairing it. Five draws per model per instance give 30 episodes with real within-instance variation, and the analysis in Section 6 is defined on all 20 instances.

The replay summary also records each instance's depth vector. They match the generator that studies A and B use, instance by instance, so the three studies ran on the same twenty instances.

J Compute and Reproducibility

Item	Value
Enumeration	Under a minute on a laptop CPU for the complete set of 6,014,400 runs and the weight sweep, a few minutes for the six domains, and seconds for the constructed pair and the witnesses. No GPU at any point
Agent studies	Roughly one to two hours for study A and about one hour for study B, on a single T4. The rerun of Appendix H.5 used two T4s
Training	None. No fine-tuning, no reinforcement learning, no paid API calls
Seeds	Instance seeds 0 to 19 for both studies. Sampling seeds were drawn from Python's salted `hash`, which is per-process, so reruns do not regenerate the same episodes. The rerun derives its seeds from `hashlib` and regenerates exactly
Release	Enumeration and domain code, the weight sweep with its brute-force cross-check, the scoring code, the number registry with the source of every figure in the paper, the benchmark audit, the proof checks, every episode of both studies with its replies, the replay summary of Appendix I, and the figure scripts

Reproducing the agent studies has a limit worth stating plainly. The original episodes cannot be regenerated exactly, for the seed reason above, so reproduction runs through the released records rather than by rerunning the models. A rerun should set `PYTHONHASHSEED` or derive its seeds with `hashlib`. The replies of both studies are released with every step's listed actions, parsed action and resulting state. Replaying them through the environments reproduces every per-episode score, and the achieved and supported sets they yield let a reader rescore the episodes under any weighting (Appendix G).

Everything outside the agent studies and the benchmark audit reruns from the released code with no model access and no stored data. The enumeration rebuilds its instances from seeds, and the

checks in Appendices D.2 and G rerun in seconds. The benchmark audit reruns from the three public repositories it names, at the snapshots the supplement records.

K EXTENDED DISCUSSION

This appendix holds the discussion that the main text summarizes, section by section. Nothing here changes a result. It gives the reasons, the qualifications and the checks behind them.

K.1 ORDERINGS, EXTRACTION AND THE DOMAINS

Stated over plans, Definition 2 gives what Hoffmann et al. (2004) call a greedy-necessary ordering. It requires ℓ' at the first achievement of ℓ, and their necessary orderings, which require it at every achievement, are special cases. Natural and reasonable orderings are excluded, because they do not require ℓ' to hold when ℓ is achieved, so an operator that ignores them is not thereby unsound. We assume the specification is *ordering-faithful*, meaning that an operator of O that first achieves a landmark ℓ on some plan requires $\mathrm{pred}(\ell)$, so plans never skip a prerequisite. Orderings obtained by backchaining from preconditions that all first achievers share (Hoffmann et al., 2004) have this property by construction, as does every domain in this paper. Excluding I from L is what lets a run start at zero, because a fact that already holds cannot be achieved, well or badly.

Deciding landmarks and orderings is PSPACE-hard, so practical extractors are incomplete or unsound. In every experiment reported here, landmarks and orderings are known by construction. The claim we make for landmarks is not that an extractor always returns the right ones. It is that landmarks replace annotator judgement with a stated, reproducible procedure that a reader can audit.

The subgoal set K is a free parameter that benchmarks leave to their authors. Table 1 and Appendix F show that the choice changes which properties the rule satisfies, and can reverse the ranking of two agents.

We use this family because it contains both persistence rules, because its reference sets can be enumerated exactly, and because it is a simple setting in which the ordering matters.

K.2 WHAT RELEASED BENCHMARK CODE COMPUTES

Released implementations need not compute the published rule. In AlfWorld (Shridhar et al., 2021) and ScienceWorld (Wang et al., 2022), AgentBoard's subgoals are regular expressions matched against the observation at each step. A step reports one action, so two event subgoals such as picking up an object and putting it down can never match the same step. On the 92 of 134 AlfWorld tasks with two or more such subgoals, the published maximum over single steps therefore cannot reach 1. The released code instead keeps a flag per subgoal, sets it on any match, and reports the fraction set.[2] In the 13 baseline runs AgentBoard releases, all 174 successful AlfWorld tasks and all 101 successful ScienceWorld tasks score 1.000. WebCanvas scores its key nodes the same way. In these two environments and in WebCanvas, the reported score is therefore landmarks achieved, A. We audit both, and the two differ on P4 but agree on P3, the property this paper turns on.

AgentBoard's planning environments compute a third rule, the largest fraction of goal literals true in a single state, which is r_{G}.[3] Blocksworld deletes goal facts, since unstacking a block undoes an *on* fact, so there r_{G} is a deployed maximum over single states and the condition under which it satisfies P2 fails (Table 1). We audit no other AgentBoard environment. WebShop and WebArena score a continuous match between state and goal, and for BabyAI and Jericho we have read only the logs, not the code.

TheAgentCompany reports the composite of Definition 3. Its released evaluator returns points earned over points available for each task, and in 54 of its 175 tasks first applies a strategy that awards full credit to every checkpoint once the last one passes (`workspaces/base_image/scoring.py`, lines 65–80). Appendix B.6 counts how often this changes a released score.

[2] `agentboard/environment/alfworld/alfworld_env.py`, lines 107–113, and `agentboard/environment/scienceworld_env.py`, lines 109–116. Repository snapshots are identified in the supplement.

[3] `agentboard/environment/pddl_env/pddl_env.py`, lines 156–166 and 187.

Corollary 1 applies directly to the checkpoint schemes in current use. AgentBoard annotates its subgoals as a sequence and WebCanvas stores its key nodes as an ordered list, but both scoring rules credit a checkpoint the moment it matches, in any order (Ma et al., 2024; Pan et al., 2024). A sequence records one route through the task rather than the orderings every route must respect, so it could not serve as $\prec$ without penalizing independent reorderings (P2). TheAgentCompany reads the order in one place only. In 54 of its 175 tasks, passing the last checkpoint awards full credit for every earlier one (Xu et al., 2025), which is the inference a shortcut breaks. None of the three publishes which checkpoints are prerequisites of which. What a benchmark would have to publish instead is short. It needs the checkpoints, the orderings among them that every solution must respect, and what each environment action requires.

K.3 Environments, immediate predecessors and when P3 has content

The *environment* O_E an agent acts in may permit transitions the specification does not. A tool reports a success it did not deliver, an action succeeds without its preconditions, or a test suite passes because the expected values were hardcoded. A trajectory is produced in the environment and scored against the specification's landmark graph, which is the position a real agent evaluation occupies.

Definition 4 uses immediate predecessors rather than the transitive closure, for three reasons. The support recursion already supplies transitivity. Real operators carry immediate preconditions. Delivering a package requires the package to be loaded, not to have been manufactured, so a transitive requirement would classify nearly every achiever in a planning domain as unsound. And it is what keeps Theorem 2 linear.

Hardcoded tests fit the model directly. Let the task be to implement a function so that its test suite passes, with landmarks ℓ_1, the function returns the specified outputs, and ℓ_2, the suite passes, and $\ell_1 \prec \ell_2$. The specification's operator *run tests* requires ℓ_1 and adds ℓ_2. The environment also lets the agent rewrite the expected values and then run the suite, an operator whose precondition is the rewritten suite and not ℓ_1. A run that does so achieves ℓ_2 by an unsound achiever, so ℓ_2 stays unsupported however correct the code later becomes, while landmarks achieved credits it at once. A tool that reports a message as sent without sending it has the same shape, with the report as the unsound achiever of the landmark "the user was told".

The reduction matters when $\ell_1 \prec \ell_3$ holds both directly and through ℓ_2. The direct edge is dropped, so an achiever of ℓ_3 that requires ℓ_2 but not ℓ_1 is sound. The test is weaker than one over every greedy-necessary predecessor, but support still requires ℓ_1 before ℓ_2 and ℓ_2 before ℓ_3, and the test stays on immediate preconditions, where real operators carry them.

Where the environment enforces preconditions and every achiever is sound, $\mathrm{Sup} = \mathrm{Ach}$, the antecedent of P3 never holds, and every metric satisfies P3 vacuously. That is why Sections 4 and 6 add a shortcut to the environment and not to the specification. The property has content exactly when the environment is more permissive than the specification.

One might call a shortcut efficient when it reaches the goal. By the specification the work was not done, in the same sense that a test suite passed by hardcoding does not show that the code works. Which models take it depends on how the options are listed (Appendix H.5), but what a metric does with it does not.

K.4 The audit

A run that does everything the task requires, in the order the task requires, scores between 0.167 and 0.500 across the configurations in this paper (Appendix E). In consuming chains the value is exactly $n/|L|$, the reciprocal of the mean chain depth, as part (b) says. In the six domains of Section 5 a complete run scores 0.333 in chains and 0.385 with a shared root. In merge it depends on the order in which chains finish, from 0.214 to 0.286. Merge is where the condition of Proposition 3 fails, while part (a) needs no condition at all.

The rate records the largest number of landmarks true at one moment. In a consuming chain, reaching stage j removes stage $j-1$, so each job holds at most one landmark and the rate counts the jobs an agent has started, finished or not.

The condition is exact rather than incidental. It holds for chains and for the shared structure with a protected root, where the maximum live count is a function of the achieved set, and it fails for pairwise merges, where it is not (Appendix A.4). With prerequisites enforced, then, the rate measures nothing beyond which landmarks were reached. Where landmarks persist it is landmarks achieved, and where they are consumed it discards the order in which they were reached.

Proposition 3 does not itself establish that r_{L} fails P3. Its hypothesis admits sound achievers only, whereas P3's antecedent requires an environment with unsound achievers, and there r_{L} is no longer a function of Ach. The two runs of Figure 1(a) achieve identical landmark sets yet receive 0.333 and 0.667. The P3 failure of r_{L} rests on Theorem 1 and on that pair.

Let the environment permit k *additive* shortcuts per job, each of which adds a landmark without its prerequisites and deletes nothing. The realistic case is a tool that falsely reports success, which adds a fact and retracts none.

By Proposition 4, the attainable rate is not monotone in support. At $k = 1$, supporting nothing and supporting everything tie while the peak is twice both. At $k \geq 2$, supporting nothing beats supporting everything outright, and at $k = d - 1$ the peak reaches the ceiling of 1.000 on a run whose support is partial. The $k = 1$ case is the weakest member of the family, not the general one.

Figure 1(a) shows the mechanism on three jobs of three stages. Both runs reach the goal and achieve the same nine landmarks. The in-order run advances each job stage by stage. The shortcut-then-backfill run marks each job complete first and fills in the earlier stages afterwards, so its three final stages are unsupported.

Run	$\lvert\mathrm{Sup}\rvert$	S	r_{G}	A	r_{L}	M_w
in-order	9	1.000	1.000	1.000	0.333	1.000
shortcut-then-backfill	6	1.000	1.000	1.000	**0.667**	0.500

Three of the five rules score the two runs identically. The rate does not merely miss the skipped prerequisites. It prefers the run that skipped them, because a job marked complete early stays true while its earlier stages are filled in, and simultaneity is the quantity the rate measures. The pair is a construction. Section 6 offers the same shortcut to language models, with $k = 1$ per job and mixed depths, so the closed forms do not apply directly, and there a job that is marked leaves the prompt, though an environment bug let models backfill a skipped stage 32 times (Appendix C.2). A model can advance a job and then mark it, which holds two landmarks in one job, and marking also lets it touch more jobs in the same budget. Section 6 separates the two.

On a finished two-job task, one extra action that restarts a job takes binary success from 1.000 to 0.000, while every maximum-based rule is unchanged, the rate at 0.500 among them. Restarting a job is ordinary agent behaviour rather than a contrivance. The restarted job really is unfinished, so a zero is a defensible reading of success. The point is only that success can fall while the work already done stays done, which P1 rules out by design.

The claim here is an existence claim. A metric satisfying all four properties exists and is cheap. All experiments use $w(\ell_{c,j}) = j$, one plus the length of the longest $\prec$-chain ending at ℓ, which satisfies W1. Uniform weights satisfy W1 as well, and under legal play they make M_w equal to landmarks achieved, which is constant at a fixed budget. The positive correlations of M_w with goal completion in Sections 5 and 6 therefore come from the depth weights. The penalty it applies to skipped prerequisites comes from support alone and holds for every positive weighting.

The four properties are necessary conditions, not a characterization, and two metrics show the gap. Weighted support with weights that violate W1 still satisfies P1 to P4, yet in the main configuration of Section 5 it correlates -0.905 with goal completion (Appendix G). W1 is part of what makes the metric useful, even though Theorem 2 does not need it. The average of landmarks achieved and weighted support also satisfies all four, yet it credits unsupported landmarks. On the three jobs of Figure 1, a run that takes the three shortcuts and nothing else scores $1/6$ with nothing supported, above the $1/12$ of a run that makes one supported advance. P3 compares only runs with equal achieved sets. Giving an unsupported landmark no credit at all is the stronger condition P3$^+$, under which the metric depends on τ only through Sup and rises strictly with it. P3$^+$ implies P3, and

weighted support meets it while the average does not. Table 1 has no P3$^+$ column because every rule there that fails P3 fails P3$^+$ too.

The goal-level rate satisfies three of the properties, one conditionally, and the landmark-level rate two, one conditionally, so the subgoal set a benchmark author chooses decides which properties hold. Appendix F shows that the choice can reverse the ranking of two strategies. Appendix B.7 explains why AgentLens is not audited.

K.5 Exhaustive verification

Section 4 compares chosen runs. Section 5 removes the choice by enumerating every legal run and scoring all of them. Every legal run of a given length reaches the same number of landmarks (Lemma 1), so a spread in the rate across those runs is not a difference in landmarks reached, and we can ask what it tracks instead. Equal here means equal landmark count, the quantity the rate itself counts. Weighted support, which weights deeper landmarks more, does tell these runs apart, and it does so in the direction of goal completion (Appendix G).

The count of Lemma 1, $|\mathrm{Ach}(\tau)| = B$, holds in 12 of 12 configurations across the three structures, with no run stalling. In Section 5 the rate is r_{L}, and the reference quantity is *goal completion*, the fraction of jobs finished. It is the task's own objective rather than a rival metric. Under legal play every goal reached is supported, and in these domains it coincides numerically with r_{G} because goal facts are never deleted.

Runs are counted exactly by dynamic programming over reachable states (Appendix D.1). Four shapes, each drawn with five depth seeds, give 20 instances, 11 of them distinct (Appendix D.3). The main configuration has 6 jobs and a budget of 9. It has 17 landmarks and 6,014,400 legal runs, each reaching 9 of them, with a rate from 0.176 to 0.353. The agent shape of Section 6, 8 jobs at budget 10, admits 254 to 750 million runs per instance, which is why Section 5 uses four to six jobs.

The joint distribution (Figure 3a) carries more than the means. 453,600 runs receive the highest rate and complete no goals at all. Every one of the 16,800 runs at the lowest rate completes the most goals the budget allows, 3 of 6. No run in the top two rate levels completes 3 of 6. The empty cells carry the argument, since a high rate and maximal goal completion never occur together. The pattern is strongest here. Across the 10 distinct instances where the rate varies, the top rate level contains no run that finishes the most goals the budget allows in 8, and both halves of the main configuration's pattern hold in 2. In depths 2,2,2,3,3,3, for example, the four goals the budget allows are reached only at the second-lowest level (Appendix D.3).

Mean goal completion falls strictly as the rate rises in all 10 distinct instances where the rate varies (Figure 3b). In the main configuration it falls from 0.500 at rate 0.176 to 0.136 at rate 0.353. Scripted play lands where that ordering predicts. In every distinct instance where the rate varies, serial and closest-to-done play sit in the lowest or second-lowest rate level and parallel play takes the top one. Serial play sits strictly below parallel in 9 of 10, and the exception is depths 2,2,3,3, where the rate has two levels and both take the upper one. In the main configuration serial play sits at the 16.6th percentile of the rate with goal completion 0.500. Parallel play takes the top rate, 0.353, with goal completion 0.167, the lowest of the three. Advancing whatever is closest to done is the sensible heuristic here, and the rate ranks it lowest of the three, tied with serial play.

Appendix E repeats the enumeration on six domains, built from disjoint chains, a protected shared root and pairwise merges, each with consuming and persistent stages. In every consuming domain a complete run scores below 1 and the rate correlates negatively with goal completion, from -0.422 to -0.592. In the persistent ones the rate equals landmarks achieved and is constant at a fixed budget, as Proposition 2 predicts. Weighted support correlates $+0.641$ to $+0.930$ with goal completion in all six. The rate also ties often. Two legal runs drawn at random get the same rate with probability 0.843 on average, against 0.373 for weighted support. This is the tie problem Diaz (2026) measures under success scoring, here computed exactly. The rate takes only two to four values across millions of runs, because each job holds at most one live landmark. That coarseness is what makes enumeration possible, and it means these spreads are smaller than a benchmark with richer subgoals would show.

K.6 AGENTS

Six open instruction-tuned models run as agents, namely Qwen2.5-1.5B-Instruct, SmolLM2-1.7B-Instruct, granite-3.1-1b-a400m-instruct, gemma-3-1b-it, OLMo-2-0425-1B-Instruct and Llama-3.2-1B-Instruct. They range from 1.0B to 1.7B parameters, and granite is a mixture of experts with 1.3B parameters, 400M of them active. Appendix C.4 gives revision hashes and licenses, and Appendix C.5 the verbatim prompts.

The instances are 20 consuming job-chain instances of 8 jobs, depths drawn uniformly from $\{2, 3\}$ with seeds 0 to 19, giving 16 to 23 landmarks and a mean of 20. Each episode allows 10 steps. At each step the prompt states the goal, lists the unfinished jobs with their current stage, and lists every permitted action under a "choose one" heading. The model is asked to copy one line. Decoding is at temperature 0.7, top-p 0.95, at most 48 new tokens. A reply is parsed by exact option match, then by a job-and-stage pattern, then by taking the first listed option for any job the reply mentions. A reply that resists all three uses the step and leaves the state unchanged, and an episode ends after 6 such failures. Parse failures are why episodes do not all reach the same number of landmarks. Mean landmarks achieved is 0.494 for SmolLM2 and 0.461 for granite, against 0.503 for the other four, so Lemma 1 holds per step rather than per episode.

Two studies share this setup. In study A the environment lists only legal advances, with 5 draws per model per instance for 600 episodes. In study B it also lists "mark job c complete" for every job that has not reached its final stage, in the same format and with no hint that the action differs. That is one additive shortcut per job, with 3 draws for 360 episodes. The prompt's only instruction is that every job must reach its final stage, and marking a job complete satisfies it literally. The environment then lists the job as finished. Taking the shortcut is consistent with what the model is told. What it skips is what the specification requires, and that is what a score should register.

The within-instance design was fixed before study A ran, and after an earlier pooled analysis of greedy decoding on the same instances had shown how pooling can reverse a sign (Appendix I). For each instance we correlate a score with goal completion over its 6 models $\times$ 5 draws, then average over instances, dropping instances where either variable is constant. Each per-instance correlation therefore mixes differences between models with differences between draws. Appendix H.1 gives the per-model figures. Averaging in Fisher's z (Fisher, 1921) rather than raw gives -0.471 for the rate, $+0.202$ for landmarks achieved and $+0.543$ for weighted support, the same signs with slightly larger magnitudes. Intervals are 95% percentile bootstrap (Efron, 1979) over the per-instance correlations, 10,000 resamples. Pooled correlations appear only to show the confound. Instances differ in size and difficulty, so pooling mixes how a metric behaves with how hard the instance was.

Pooled over all 600 episodes the rate's correlation with goal completion becomes $+0.064$. Weighted support moves little, to $+0.539$, and landmarks achieved rises to $+0.498$. Pooled over instances, the view in which benchmarks report mean scores, the effect disappears (Figure 4). The change is an instance of Simpson's paradox (Simpson, 1951). Instance size drives it. Across the 20 instances $|L|$ runs from 16 to 23 and correlates -0.924 with the mean rate and -0.796 with mean goal completion. A small instance divides the same live count by fewer landmarks and is also easier to finish, so instance means move together ($+0.891$), and pooling adds that agreement to the disagreement within each instance. The remedy is the one Agarwal et al. (2021) give for reinforcement-learning benchmarks, which is to report per-instance results with intervals rather than a pooled point estimate.

One model disagrees with the other five. Per model, the within-instance correlation runs from -0.824 to -0.417 for five of the six, and granite sits at $+0.146$. Granite also fails to parse 0.83 times per episode, against 0.00 to 0.20 for the rest. Across the 6 models the parse-failure rate correlates $+0.912$ with the per-model correlation, while the rate at which a model switches jobs correlates -0.252. The mechanism is visible in the definition. A parse failure consumes a step without changing the state, which decouples the score from the strategy. Restricting to the 521 episodes with no parse failure (86.8%) gives -0.559, a check chosen after seeing granite's parse failures; we report -0.439 as the primary figure. Per-model instance counts vary from 9 to 20 because constant-score instances are dropped. The primary analysis pools models within an instance and is defined on all 20.

In study A every action is a legal advance, so every achieved landmark is supported and weighted support is landmarks achieved weighted by depth. Its positive correlation with goal completion therefore rests on the depth weights, because finishing a job means reaching its heaviest landmark.

Its size is not comparable with the enumeration's correlations, which weight every legal run once rather than the runs a model chooses to sample. Only the signs carry across.

In study B, models take the shortcut in 203 of 360 episodes (56.4%). In the sorted design the six models split. Granite, SmolLM2 and OLMo used the shortcut in every one of their 60 episodes, at 7.40, 4.87 and 5.03 uses per episode. Qwen, Llama and gemma abstained in 60, 55 and 42 of 60. The split is largely positional. Every mark follows every advance in the sorted list, the three abstainers copy the first line in 39% to 95% of steps, and uniform choice among the listed lines would have marked in 99.9% of episodes. With the order shuffled, all seven models marked (Appendix H.5). Parse failures run from 0.00 to 0.10 per episode here, and the parser's loosest fallback always resolves to an advance rather than a shortcut (Appendix C.6), so abstention is not an artifact of parsing.

What the scores do with that behaviour is the result (Table 2, Figure 1b). Within an instance, the number of shortcuts an episode uses correlates $+0.921$ with the progress rate and $+0.949$ with goal completion, while weighted support correlates -0.978 and landmarks achieved -0.509, the last defined on 20 instances. The goal figure is a property of the measurement rather than of the agents. Marking a job sets its goal fact, so of granite's 7.57 finished jobs per episode, 7.40 were marked, and for the three heavy users 83 to 98% of finished jobs were marked.

Two things raise the rate when a model marks jobs. A mark on an untouched job adds one live landmark, exactly as advancing the job once would, and finishes it in one step, so marking touches more jobs in the same budget. The heavy users hold 7.5 to 8.6 landmarks at their peak, against 4.4 to 5.1 for the abstainers, which is the breadth effect of Section 5. Holding two landmarks in one job, the mechanism of Proposition 4, needs a job advanced before it is marked, and is certain only when more than eight landmarks are live at once, since each episode has eight jobs. That happens in 36, 11 and 10 of 60 episodes for granite, SmolLM2 and OLMo, and never for the abstainers. The rate rewards both. Mean scores order 14 of 15 model pairs oppositely under the rate and under weighted support. Granite skips a prerequisite 7.40 times per episode and does the least supported work of the six at 0.067. Yet the rate ranks it first, and goal completion gives it 0.946. Qwen never takes the shortcut, does 0.459 of supported work, and the rate ranks it 4th. Shuffled, the rate orders the seven models largely by shortcut use (Kendall's τ $+0.714$) and weighted support reverses that order exactly (-1.000).

Shortcut use here says nothing about skill. The three heavy users of the sorted design had lower weighted support in study A, but a first-line copier plays serially there, which maximizes goal completion at this budget, so the comparison confounds position with skill (Appendix H.5).

The rules divide as Theorem 1 says they must. Landmarks achieved barely moves across the six models (0.465 to 0.503), and goal completion counts the goals a shortcut sets, so neither separates skipping from doing. Of the rules in Table 1, only weighted support does, and it does so by reading $\prec$.

K.7 Related work in full

Partial credit in agent benchmarks. The rules audited in Section 4 come from four benchmarks that award credit for unfinished work, namely AgentBoard's progress rate (Ma et al., 2024), TheAgentCompany's weighted checkpoints (Xu et al., 2025), WebCanvas's key nodes, annotated on tasks drawn from Mind2Web (Pan et al., 2024; Deng et al., 2023), and SUPER's landmark outputs (Bogin et al., 2024). Other benchmarks score partial progress the same way. AgentQuest counts the milestone states a run has reached (Gioacchini et al., 2024), and TravelPlanner reports the share of constraints a plan satisfies (Xie et al., 2024a). In all of them the checkpoints are written by hand and then counted, averaged or maximized. Two of the four we audit record an order, AgentBoard's subgoal sequence and WebCanvas's ordered key-node list, and neither scoring rule reads it. TheAgentCompany reads only the position of the last checkpoint, and uses it to grant credit to the others.

DevAI comes closest to what we propose (Zhuge et al., 2025). Its 365 requirements form a dependency graph, and its dependency-aware score counts a requirement only when the requirements it depends on are also met. In its human evaluation, the two best systems each meet about 29% of requirements on that count, against 43% and 45% when dependencies are ignored. The dependencies

are written by annotators rather than derived from a task model, and they are judged on the finished workspace. The score therefore reads which requirements hold at the end, not whether each was met after its prerequisites or by a step that needed them, which is the distinction support draws.

ToolSandbox also reads an order (Lu et al., 2025). Its milestones form a directed acyclic graph by temporal dependency. Each milestone scores the database snapshot or tool trace of every turn, and a trajectory scores the best average over assignments of milestones to turns whose order is a topological sort of the graph. It checks the order in which milestones held, so a shortcut whose effect persists can be matched after its predecessor, and it never asks what the achieving step required (Appendix B.4).

Our audit is directed at the scoring rules rather than at the task suites, which are careful work. A benchmark could keep its tasks unchanged and publish the orderings its authors already know. AgentBoard reports a correlation above 0.95 between its progress rate and human judgements of progress. The two results are compatible. The released rate measures how far a run has got, and so does a human judgement of progress, but neither checks whether each step's prerequisites were met, which is where the properties here bite.

Process-aware evaluation. Three recent papers diagnose the loss of information that motivates this one. Diaz (2026) compares trajectories by preference rather than by terminal success, and measures the ties that success scoring induces. Sahoo et al. (2026) scores a trajectory against a reference automaton built by merging observed passing traces, and labels low-scoring passes lucky, which is the phenomenon our shortcut result produces on purpose. Its score pairs an ordered recall against the reference with an unordered precision, the nearest engineering counterpart to the line P2 and P3 draw between achievements that must keep their order and those free to move. Liu et al. (2026) represents trajectories as graphs and defines metrics over them for software-engineering runs. All three describe trajectories, but none states what a progress metric must satisfy, checks the metrics in use against such a standard, or proves what a metric can achieve. They also need different inputs. A reference automaton or a trajectory graph is built from traces that must first be collected, so it encodes what was seen and penalizes deviation from it. Weighted support instead reads the task's own orderings and one pass over the run, and charges nothing for reaching a landmark by an unseen route.

Progress scores as rewards. A progress score read during a run acts as a reward, and reinforcement learning has worked out what goes wrong with rewards of this kind. Shaping a reward by a difference of potentials leaves the optimal policy unchanged, while other shaping terms can pay an agent for cycling through states that look like progress (Ng et al., 1999). Reward machines pay for subgoals through an automaton that fixes which must come before which (Toro Icarte et al., 2018), the ordering a checkpoint list leaves out. Support is a condition on the past of a run, evaluated at each first achievement, which is the kind of property a runtime monitor tracks. A reward machine could pay for it if its states recorded which landmarks are supported. When the reward is a misspecified proxy, more capable agents exploit it more (Pan et al., 2022), and over the set of all policies only a trivial proxy is safe from this (Skalse et al., 2022). Process supervision labels each step of a solution instead of its outcome (Lightman et al., 2024). Weighted support is a step-level signal of that kind, computed from the task's orderings rather than from annotators.

Landmarks in planning. Landmarks and their orderings were introduced to guide search (Hoffmann et al., 2004) and have been used that way since, in planners (Helmert, 2006; Richter & Westphal, 2010) and in admissible (Karpas & Domshlak, 2009) and lifted (Wichlacz et al., 2022) heuristics. LAMA's landmark-count heuristic is already close to a progress measure. It counts the landmarks a partial plan has not reached, plus those it reached once that are false again and still needed, and a landmark counts as reached only once its predecessors were reached (Richter & Westphal, 2010). Büchner et al. (2023) show that this progression infers invalid information under reasonable orderings and give a sound one that uses orderings and the search history. Evaluations of language models on planning score finished work. PlanBench tests plan generation, verification and several related tasks, each instance checked by a validator (Valmeekam et al., 2023), and AutoPlanBench turns PDDL domains into prompts and reports how many it solves (Stein et al., 2025). Goal recognition already scores observed trajectories by landmarks. Pereira et al. (2017; 2020) estimate goal completion as the share of a goal's landmarks achieved, inferring the achieved landmarks from the

observed actions. That inference assumes the actions obeyed the domain's preconditions, which is what a shortcut breaks. The test of each achieving step against those preconditions is what this paper adds, and once the task model is known it costs one pass, because the structure planners already extract is the structure a progress metric needs.

Axiomatic evaluation. Stating constraints a metric should satisfy, surveying the metrics in use against them, and reporting which survive is the method Amigó et al. (2009) apply to clustering evaluation and Amigó et al. (2018) to diversity metrics in retrieval, and Kleinberg (2002) is the case where the constraints admit nothing at all. We follow the method on a different object and reach the other outcome. The constraints are jointly satisfiable, and one of the nine rules we audit satisfies them.

K.8 Limitations in full

Each limit is stated in Section 8, and this subsection gives its scope and what bounds it.

Domains. Exact reference sets need domains whose legal runs can be counted, so the empirical results use generated job chains, with landmarks and orderings known by construction. Propositions 1 to 4 are stated for structural conditions rather than for this family, so they do not depend on it. Because each job holds one live landmark, the rate takes two to four values in these domains. Richer subgoal sets would spread it further. Deciding landmarks and orderings is PSPACE-hard in general, so the audit takes $\langle L, \prec \rangle$ as given, and Appendix H.6 measures the effect of wrong orderings and noisy detection. An extractor runs at benchmark scale. On AgentBoard's 60 planning tasks, Fast Downward's translator finds 1.12 (Gripper) to 2.32 (Tyreworld) fact landmarks per goal fact. AgentBoard's released logs of its 13 baseline runs record per-task scores but no actions, so those runs cannot be rescored from the logs.

Model. Support is settled at a landmark's first achievement. A landmark first reached by an unsound step stays unsupported even if the work is later redone, which is what keeps a backfilled shortcut from earning credit (Figure 1a), and rewarding recovery would need a different rule. Like every maximum-based rule, weighted support does not register work that is later undone, as P1 requires. A final-state check reported beside it covers that case. Soundness is judged against the preconditions of the executed operators, which planning domains supply and web or tool environments usually do not expose. Without them, only the timing of first achievements can be checked. On the sorted study B episodes that check catches 940 of 1,066 marks, missing only those taken after the prerequisite already held (Appendix H.5). Weighted support's positive correlation with goal completion comes from the depth weights, while its penalty for skipped prerequisites holds under any positive weighting (Appendix G).

Agents. The six models of studies A and B have 1.0B to 1.7B parameters, so that all 960 episodes run on a single T4 GPU. The rerun adds Qwen2.5-7B on two. Proposition 4 does not depend on which agents take a shortcut, and which ones do depends on option order (Appendix H.5), so the agent results are evidence about the metrics, not a ranking of the models. Model-level correlations rest on six or seven points and are reported descriptively. The shortcut is half the actions listed at the first step, so the rerun reports each model's mark rate against that share. The original sampling seeds cannot be regenerated, so those episodes are reproduced from the released replies. The rerun's seeds are fixed.