LEARNING STATE TRANSITIONS IMPROVES DOWNSTREAM POLICY LEARNING IN GEOMETRY THEOREM PROVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Geometry theorem proving can be formulated as sequential action selection over an evolving proof state. Standard policy supervised fine-tuning (Policy-SFT) teaches a model to imitate reference actions, but does not directly supervise how those actions change the state. We study whether explicitly learning these state transitions before policy training improves downstream theorem proving. We build a stateful agent environment around the open-source GenesisGeo symbolic engine. Within this environment, we first apply world-model supervised fine-tuning (WM-SFT) to predict auxiliary line construction feedback, whose newly derived geometric relations define the action-induced state delta. We then perform Policy-SFT on actions from engine-generated trajectories using the same model. Across three runs, WM-SFT followed by Policy-SFT achieves 72.9% mean proof success, compared with 65.9% for Policy-SFT alone. Additionally, after subsequent reinforcement learning, WM-SFT followed by Policy-SFT and RL reaches up to 76.8%, compared with 71.2% for Policy-SFT followed by RL. WM-SFT is used only during training, all policies follow the same action-selection procedure at inference, with no additional prediction stage, model call, or planning. These results show that supervised learning of action-induced state transitions can provide an effective initialization for training mathematical-agent policies.[1]

1 INTRODUCTION

Neuro-symbolic geometry provers combine learned auxiliary constructions with symbolic deduction (Trinh et al., 2024; Zhu et al., 2025). When deduction from the current geometry is insufficient, a learned model can introduce a new construction, after which the symbolic engine derives additional geometric relations and continues the proof search. AlphaGeometry (Trinh et al., 2024) follows this basic loop, while InternGeometry (Zhao et al., 2026) makes the interaction explicitly multi-turn, allowing later decisions to depend on verified feedback from earlier actions. Auxiliary construction is therefore a stateful decision problem: each accepted action changes the proof state available for subsequent decisions.

Given engine-generated proof trajectories, a direct way to train such an agent is supervised fine-tuning on their construction and proposition actions. We refer to this setup as policy supervised fine-tuning (Policy-SFT). Policy-SFT matches the task performed at inference: given the current proof context, predict the next construction or proposition. However, this objective directly supervises which action to take, not how that action changes the proof state. After an auxiliary construction, the symbolic engine may derive new equal-length, parallel, perpendicular, or equal-angle relations that become part of the context for later decisions.

AlphaGeometry pretrains on large-scale synthetic proofs, including deduction traces, before fine-tuning on proofs requiring auxiliary constructions; its ablations show that pretraining improves construction performance (Trinh et al., 2024). We isolate action-conditioned transition supervision: given a proof state and a supplied construction, predict the resulting feedback, including the newly derived relational state delta. We ask whether this supervision improves subsequent policy learning without using the learned transition model at inference.

[1] Code and the full implementation will be released as open source upon acceptance.

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.

This use of transition learning also differs from the usual role of world models in model-based reinforcement learning. Methods such as PlaNet (Hafner et al., 2019) and MuZero (Schrittwieser et al., 2020) learn environment dynamics that are subsequently used to evaluate or plan future actions. When learned dynamics are used in the decision loop, prediction errors can propagate into model-generated trajectories and bias policy optimization (Janner et al., 2019). Recent language-agent methods similarly use predicted environment responses during reasoning or on-policy training (Yu et al., 2025b; Shrivastava et al., 2026).

We instead use transition prediction only as a training objective. GenesisGeo (Zhu et al., 2025) can execute construction actions and compute symbolic feedback directly. Our hypothesis is that predicting how actions change the environment provides useful supervision for the representations subsequently used to learn the policy. This follows the view that learning environment dynamics can shape representations for downstream decisions without using the learned dynamics model to execute the policy (Gelada et al., 2019). Recent language-agent work also uses state and transition supervision before policy optimization (Zhang et al., 2026; Chen et al., 2026). We study whether pretraining on verified geometric state transitions improves subsequent action imitation and reinforcement learning.

We first apply world-model supervised fine-tuning (WM-SFT) to predict construction feedback, then matched Policy-SFT on actions from the same engine-generated trajectories. The stages supervise how a supplied action changes the observable proof state and which action to select, respectively. At inference, all policies use the same agent loop and receive actual GenesisGeo feedback.

Our contributions are:

- We isolate action-conditioned state-delta prediction as a training-only world-model objective for theorem-proving agents, and study whether transition supervision improves subsequent policy learning without using a learned simulator at inference time.
- We build a stateful theorem-proving environment around the open-source GenesisGeo engine that exposes verified construction updates and supports separate world-model and policy supervision.
- We show that WM-SFT provides a better initialization for policy learning: WM-SFT followed by Policy-SFT improves mean proof success from 65.9% to 72.9% across three runs. In the RL comparison, the strongest WM-SFT-initialized policy reaches 76.8%, compared with 71.2% for Policy-SFT followed by GRPO.

2 Stateful GenesisGeo Environment

We build a stateful agent environment around the open-source symbolic engine from GenesisGeo (Zhu et al., 2025). The original engine takes a geometric configuration and a set of constraints, derives relations that follow from them, and verifies geometric propositions. Our wrapper turns these capabilities into a multi-step interaction loop in which an agent modifies the geometry, receives verified feedback, and chooses its next action from the accumulated proof context (Figure 1).

State and observations. We distinguish the engine's internal state from the public proof state exposed to the policy. The internal state contains the constructed geometry, the engine's complete set of derived relations, and proof ancestry used by the symbolic solver. The policy does not observe this full state.

Instead, each observation summarizes a public proof state with two parts. The static task context includes the formal problem, target relation, and an annotated diagram supplied in the initial observation and retained in the conversation context. The dynamic proof state includes named points, accepted auxiliary constructions, and a selected set of verified geometric relations. The policy conditions on this state together with the conversation history of prior actions and observations.

The public proof state accumulates across turns. After every accepted construction, the environment reports updated points and constructions together with newly visible geometric relations, capped and prioritized as described in Appendix C. Construction feedback also updates bookkeeping fields such as the next required point name. The engine's full deductive closure and proof traces remain private.

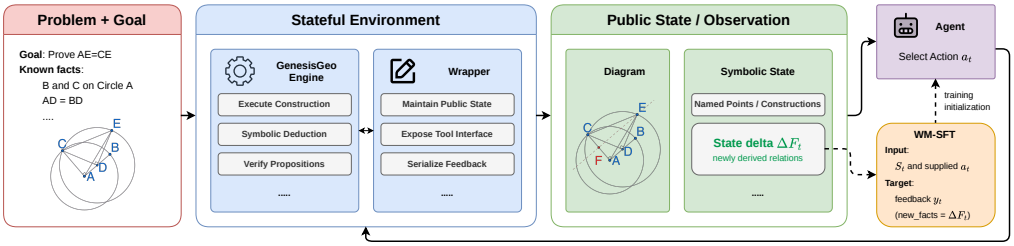


Figure 1: **Overview of the stateful geometry proving loop.** A problem and target relation initialize a GenesisGeo-based environment whose wrapper maintains public state and serializes construction feedback. The initial observation supplies the diagram and task, while later turns retain that context and append textual feedback containing named constructions and the relational state delta ΔF_t. The agent selects action a_t from this context. Dashed arrows indicate WM-SFT supervision on predicted step deltas; at inference, the engine supplies feedback directly.

Action interface. At each turn, the policy emits one typed tool action. The environment supports three action types:

- A construction action adds one auxiliary point subject to geometric constraints. Supported constraints include collinearity, equal length, perpendicularity, parallelism, midpoint, cyclicity, and equal angles.
- A proposition action submits a geometric relation for symbolic verification. The proposal may be an intermediate statement or the target relation.
- A give-up action terminates the current proof attempt.

Point names persist across turns, and the environment supplies the name to use for the next auxiliary point. Lines and segments are identified by their endpoint names, so the agent does not need separate drawing actions for them. Tool calls are represented as typed JSON and are translated deterministically into engine operations. Appendix C gives naming rules, observation fields, and rollout budgets.

Deduction and state updates. Deduction derives relations logically or algebraically implied by the current geometry without modifying it. For example, from existing points and constraints, the engine may establish that two segments have equal length, two lines are parallel or perpendicular, or two angles are equal.

A construction action, in contrast, changes the geometric configuration. The action specifies a new point together with one or two supported constraints, such as placing the point on an existing line, defining a midpoint, or imposing perpendicularity. The wrapper translates the typed tool call into the GenesisGeo construction language, rebuilds the geometry with the new construction, and reruns deduction. The construction is committed only if this process succeeds.

After an accepted construction, the environment compares public relations before and after the update and reports the newly visible geometric relations. We denote these newly reported relations by ΔF_t and refer to them as the **action-induced state delta**. The delta follows the public reporting rules in Appendix C; it excludes bookkeeping changes and does not represent the engine's complete internal transition. For example, a construction may explicitly place a new point at a midpoint, while the engine subsequently derives additional perpendicular or equal-length relations. In Figure 1, the action specifies midpoint F, and $AC \perp EF$ is derived by the engine rather than specified by the construction action. The broader construction feedback, including ΔF_t and bookkeeping fields, is serialized as y_t and serves as the WM-SFT target. A rejected construction leaves the geometry unchanged and returns failure feedback instead.

Proof completion. An episode succeeds when the agent submits the target relation through a proposition action, after at least one accepted construction, and the engine certifies it in the current constructed geometry. Successful verification of an intermediate relation does not terminate the

episode. Each rollout is bounded by accepted constructions, failed proposition checks, and total interaction turns. Appendix C specifies the exact budgets and remaining interface details.

3 World-Model and Policy Supervised Fine-Tuning

We train the same model in two supervised stages. World-model supervised fine-tuning (WM-SFT) first trains the model to predict construction feedback after a supplied action. Policy supervised fine-tuning (Policy-SFT) then trains the model to select the next reference action from the current proof context.

World-model supervised fine-tuning. We derive WM-SFT examples from accepted construction transitions in the environment described in Section 2. Let h_t denote the interaction history before construction a_t, and let $\tilde{h}_t$ denote the corresponding context prepared for WM-SFT. Given $\tilde{h}_t$ and a supplied construction a_t, the model predicts the serialized construction feedback y_t returned by the environment. The new-facts field of y_t contains the action-induced state delta ΔF_t: the newly reported geometric relations selected under the public reporting rules in Appendix C. The remaining fields contain interface information such as the constructed point and the next required point name.

For one training example, WM-SFT minimizes the autoregressive cross-entropy loss

$$\mathcal{L}_{\mathrm{WM}}(\theta) = -\sum_i \log p_\theta\Big(y_{t,i} \mid \tilde{h}_t, a_t, y_{t,<i}\Big), \tag{1}$$

where i indexes target tokens. Loss is applied only to the serialized feedback tokens. The proof goal is removed from the WM-SFT context, so the prediction is conditioned on the current geometry and supplied action rather than on which state changes would be useful for the target proof. Prompt construction, feedback serialization, and training details are given in Appendix D.

Policy supervised fine-tuning. Starting from the model after WM-SFT, we then train it to imitate tool actions from engine-generated trajectories. Each Policy-SFT example contains the interaction history h_t, including the formal problem, target relation, previous actions, and engine feedback, and uses the next reference action a_t as the target. Targets include both construction actions and proposition checks. The policy loss is

$$\mathcal{L}_{\mathrm{policy}}(\theta) = -\sum_i \log p_\theta(a_{t,i} \mid h_t, a_{t,<i}), \tag{2}$$

where i indexes action tokens. Engine responses remain in the context but receive no Policy-SFT loss.

We compare this sequential training procedure with a matched baseline that applies Policy-SFT directly to the base model.

Inference and reinforcement learning. At inference, all policies select tool actions from the accumulated context and receive actual GenesisGeo feedback. WM-SFT adds no feedback prediction, simulated rollout, model call, or planning stage.

We further apply reinforcement learning to both Policy-SFT and WM-SFT→Policy-SFT checkpoints to test whether the benefit of state-transition supervision persists under subsequent policy optimization.

4 Experiments and Results

We test whether WM-SFT improves downstream policies trained by imitation or reinforcement learning on separate problems. We then examine whether explicit prediction accuracy explains these gains and how they emerge during GRPO.

4.1 EXPERIMENTAL SETUP

Our main experiments use Qwen3.5-27B and PrimeRL (Prime Intellect, 2025), with at most 10 interaction turns per proof attempt. Policy rollouts include only the initial diagram to limit context length.

Data and splits. We select 1,380 problems from 75,000 GenesisGeo source records and replay their engine-verified construction paths. The SFT split contains 1,240 training and 140 validation problems, balanced across equal-angle, equal-length, parallel, and perpendicular goals. These trajectories yield 2,752 training and 310 validation examples for Policy-SFT, and 1,512 training and 170 validation examples for WM-SFT. Both tasks inherit the same problem-level split: WM-SFT uses accepted construction transitions, while Policy-SFT also includes the final proposition action.

An expanded source pool supplies 2,076 RL training problems and the 250 held-out validation problems used to evaluate proof success. These partitions exclude all SFT source identities and are stratified by goal type and reference construction count. Appendix B details the RL composition, source hashing, filtering, trajectory selection, and split validation.

Supervised training. The 27B SFT configurations use a batch size of 32. We compare full-parameter WM-SFT, which freezes the vision encoder, with low-rank adaptation (LoRA) (Hu et al., 2021) at ranks 16 and 32. Before Policy-SFT, any WM-SFT adapter is merged into the base weights. All Policy-SFT stages then use LoRA rank 16. The Policy-SFT-only baseline receives the same total training budget as WM-SFT followed by Policy-SFT, with checkpoints selected by validation loss. We repeat the primary SFT comparison across three seeds; Appendix D gives individual scores and optimization settings.

Evaluation metrics. Pass@1 measures the fraction of problems solved in one attempt under the proof-completion rule in Section 2. New-fact F1 is evaluated on a separate held-out construction-feedback prediction set. The probe supplies geometry contexts and candidate constructions, removes images and executable tools, and generates feedback autoregressively at temperature zero. We compare predicted facts with the engine-reported relational state delta ΔF_t, pooling matched facts (TP), extra predictions (FP), and missed target facts (FN) across examples:

$$\text{micro-F1} = \frac{2TP}{2TP + FP + FN}. \tag{3}$$

The score is zero when the denominator is zero. Duplicate facts count once within each example, and responses with invalid JSON or an invalid response schema are scored as empty predicted fact sets. We normalize point arguments under the supported relation symmetries before matching, as detailed in Appendix D. The targets contain the bounded public fact reports, not the engine's complete deductive closure.

4.2 TRANSITION PRETRAINING IMPROVES SUPERVISED POLICIES

We first test whether WM-SFT followed by Policy-SFT produces a stronger proof policy than Policy-SFT alone, under the same total training budget.

Table 1 shows higher proof success with each tested WM-SFT initialization. Across three runs, rank-32 WM-SFT followed by Policy-SFT achieves $(72.9 \pm 1.8)\%$ Pass@1, compared with $(65.9 \pm 4.3)\%$ for Policy-SFT alone. The mean improvement is 7.1 percentage points without changing inference. Reversing the order, Policy-SFT→WM-SFT reaches 55.6% new-fact F1 but only 58.8% Pass@1. This supports transition supervision as preparation for policy training: reversing the stages yields a weaker policy despite higher final accuracy in predicting the new geometric facts induced by a construction. We next test whether this advantage survives further learning from interaction.

4.3 THE BENEFIT PERSISTS UNDER REINFORCEMENT LEARNING

We apply GRPO (Shao et al., 2024) to test whether WM-SFT benefits policy optimization both with and without an action-imitation stage. All RL policies interact with the same live environment and use the same reward definition. GRPO uses a batch size of 16 and eight rollouts per example, applying

Table 1: Supervised and reinforcement-learning results. r denotes the LoRA (Hu et al., 2021) rank used for WM-SFT; all Policy-SFT stages use LoRA with $r = 16$. New-fact F1 measures prediction of the relational state delta ΔF_t, and Pass@1 measures proof success after the final training stage. All values are percentages. Pass@1 entries with $\pm$ report the mean and sample standard deviation over three runs; all other entries are single-run results.

Supervised fine-tuning	New-fact F1		
Training sequence	After WM-SFT	After Policy-SFT	Pass@1
Policy-SFT	–	1.2	65.9 ± 4.3
WM-SFT ($r = 32$) → Policy-SFT	**60.1**	13.4	$\mathbf{72.9 \pm 1.8}$
WM-SFT ablations			
WM-SFT (full-parameter) → Policy-SFT	38.5	11.7	68.4
WM-SFT ($r = 16$) → Policy-SFT	55.7	**13.5**	70.4
Policy-SFT → WM-SFT ($r = 32$)	55.6	–	58.8
Reinforcement learning			
Training sequence		New-fact F1	Pass@1
Base		6.7	22.0
Base → GRPO		7.7	59.2
Base → ECHO		4.9	36.0
Policy-SFT → GRPO		1.7	71.2
WM-SFT ($r = 32$) → ECHO		**60.6**	51.6
WM-SFT ($r = 32$) → GRPO		57.7	70.4
WM-SFT (full-parameter) → Policy-SFT → GRPO		14.3	**76.8**
WM-SFT ($r = 32$) → Policy-SFT → GRPO		13.9	76.4

policy-gradient loss only to action tokens. The episode reward gives $+1.00$ for proof completion or $+0.20$ for reaching a goal-provable state without completing the proof or giving up. A completed proof receives a further $+0.05$ when it uses no more constructions than the reference; any protocol violation incurs a single -0.05 penalty. Appendix C.1 specifies the full reward rules.

After Policy-SFT and GRPO, the two WM-SFT-initialized policies reach 76.4–76.8% Pass@1, compared with 71.2% for Policy-SFT→GRPO (Table 1). The benefit therefore persists through both action imitation and policy optimization.

Direct WM-SFT→GRPO also improves Pass@1 from 59.2% for Base→GRPO to 70.4%, showing that the benefit extends to learning from interaction without action imitation. Both comparisons test transfer beyond the WM-SFT corpus, since the RL problems exclude its source identities. Figure 2 traces how these policies improve during RL.

ECHO (Shrivastava et al., 2026) jointly trains action selection and construction-feedback prediction. It tests whether offline pretraining remains useful when transition supervision continues during RL. In our geometry adaptation of ECHO, WM-SFT raises Pass@1 from 36.0% to 51.6%, extending the benefit to every downstream training route evaluated here. Appendices D and A give the configurations and diagnostics.

4.4 Transition prediction and downstream performance

Do these policy gains depend on retaining accurate construction-feedback prediction? We compare new-fact F1, which measures prediction of the geometric facts induced by a construction, with Pass@1, which measures proof success (Table 1). In the run evaluated by the prediction probe, rank-32 WM-SFT achieves 60.1% new-fact F1, but this falls to 13.4% after Policy-SFT. The Policy-SFT-only baseline scores 1.2%. Thus, policy training substantially reduces explicit prediction accuracy, although some advantage over the baseline remains. Across three policy training runs, WM-SFT initialization also improves mean proof success from 65.9% to 72.9%.

Higher new-fact F1 does not consistently correspond to higher proof success after RL either. The WM-SFT→ECHO and WM-SFT→GRPO policies retain F1 scores near 60, but achieve 51.6% and

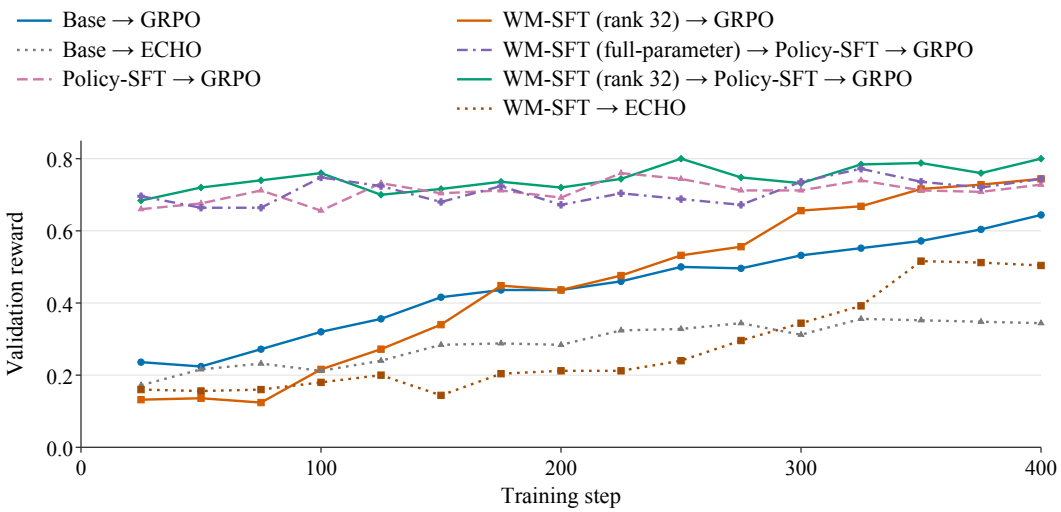


Figure 2: Validation reward during RL for seven selected training runs, with unsmoothed evaluations every 25 steps through step 400. Ranks specify LoRA for WM-SFT; all Policy-SFT stages use LoRA rank 16. The ECHO runs start from the base model or a WM-SFT checkpoint and jointly train action selection and construction-feedback prediction with full-parameter updates.

70.4% Pass@1, respectively. By comparison, the policies trained with WM-SFT, then Policy-SFT, then GRPO achieve 76.4–76.8% Pass@1 despite lower F1 scores of 13.9–14.3.

These results show that the policy benefit of transition pretraining can persist after much of its explicit prediction accuracy is lost. During proof search, the policy receives actual engine feedback, so it does not need to generate that feedback itself. New-fact F1 measures this generation ability. Determining which transition-related information is retained requires further analysis of the model's representations.

Source of transition supervision. Why is learning a world model online difficult for geometry theorem-proving agents? Online world-model learning relies on transitions collected by the current policy. Our pretrained base policy solves only 22.0% of evaluation problems (Table 1), suggesting that successful trajectories may be scarce early in training. We also observe that a large proportion of online traces contain fewer than two new facts. These traces provide little relational content for the world model to predict and may offer limited supervision for learning richer geometric transitions. We examine the quality of supervision available from policy rollouts through an offline comparison of three datasets for 4B WM-SFT (Table 2).

The engine-generated dataset uses trajectories produced by the symbolic engine. The raw policy-generated dataset uses trajectories collected from the 27B policy during GRPO. The filtered policy-generated dataset is a subset of those GRPO trajectories, retaining successful trajectories with nonempty new-fact feedback.

All three models are evaluated on the same held-out engine-generated transitions. The model trained on engine-generated data achieves 51.0% new-fact F1. Training on raw policy-generated data yields 6.1%, while training on filtered policy-generated data yields 10.0%. Filtering the policy trajectories therefore improves prediction accuracy, but the model trained on engine-generated data remains more accurate on this evaluation.

To compare dataset size and coverage, we focus on the engine-generated and filtered policy-generated datasets. The engine-generated dataset contains 1,682 entries in total; the filtered policy-generated dataset contains 1,719 entries but covers only 562 distinct problems, with repeated samples from eight-rollout GRPO groups. More than 90% of examples in each of these two datasets contain eight new facts. Thus, the filtered policy-generated dataset has a similar entry count and feedback length to the engine-generated dataset, but includes many examples from the same problems.

Together, these observations suggest that online world-model learning is limited by both the content and diversity of policy-generated experience: many traces contain few new facts, while the filtered informative traces cover relatively few distinct problems. This may partly explain ECHO's weaker performance in our geometry setting. ECHO jointly learns action selection and construction-feedback prediction from its current policy's rollouts, so a weak policy can also limit the supervision available to its world-model objective. Offline WM-SFT supplies informative transition examples before the policy can reliably collect such experience itself.

Table 2: Transition prediction after 4B training on different data sources, evaluated on held-out engine-generated transitions.

Training data for 4B WM-SFT	New-fact F1 (%)
Engine-generated trajectories	51.0
Policy-generated trajectories, raw	6.1
Policy-generated trajectories, successful + nonempty	10.0

We next examine how transition supervision changes downstream policy behavior during GRPO.

4.5 How does WM-SFT change learning under GRPO?

Does WM-SFT help GRPO choose constructions that make the target goal provable, or complete proofs once those states are reached? We compare Base→GRPO and WM-SFT→GRPO, which learn without an intervening Policy-SFT stage. To distinguish these behaviors, write proof success as $S = R - U$, where R is the fraction of episodes ending in a goal-provable state and U the fraction ending in such a state without a successful target proposition.

Learning constructions that make the goal provable. The WM-SFT-initialized policy first overtakes Base at the step-175 evaluation (Figure 2). Between steps 150 and 175, its success rises from 34.0% to 44.8%, compared with 41.6% to 43.6% for Base (Table 3). Of this 10.8-point improvement, 10.4 points come from increased reachability and 0.4 from fewer unfinished reachable cases. At step 175, reachability exceeds Base's (53.2% versus 48.0%), with fewer protocol violations (11.2% versus 22.4%). Accepted constructions remain approximately constant across the crossover, while rejected constructions decrease, though they remain more frequent than for Base. The emerging advantage thus reflects constructions that more often make the target goal derivable.

Turning reachable goals into completed proofs. Successful construction and successful proof submission develop at different rates. At step 125, 9.6% of episodes under the WM-SFT-initialized policy already end with a reachable goal but no successful target proposition. From steps 200 to 275, reachability rises from 56.0% to 71.2%, while unfinished reachable episodes also increase, from 12.4% to 15.6%. During this interval, the policy increasingly constructs states that support a proof but leaves many without a successful submission. From steps 275 to 400, reachability improves further to 79.6%, while unfinished reachable episodes fall to 5.2%.

This progression suggests that WM-SFT helps subsequent GRPO use geometric relationships to select goal-directed constructions. Improved proof submission later converts more of those reachable goals into completed proofs, explaining how the construction advantage translates into higher Pass@1.

5 Related Work

Neuro-symbolic geometry proving. AlphaGeometry learns auxiliary constructions from synthetic proofs and combines them with symbolic deduction (Trinh et al., 2024); Newclid provides an interpretable, extensible reasoning engine (Sicca et al., 2024). InternGeometry studies multi-turn geometry agents with symbolic construction and proposition feedback (Zhao et al., 2026). Its extended engine was not publicly available during our experiments, motivating our stateful environment built on open-source GenesisGeo (Zhu et al., 2025). Verified action-conditioned state deltas let us separately supervise transition prediction and action selection, then measure their effects on policy learning.

Table 3: Validation diagnostics around the first proof-success crossover under GRPO, with one training seed per initialization. Reachable but unsuccessful is $R - S$. Construction counts are episode means; protocol violations count episodes with at least one violation.

	Base→GRPO		WM-SFT→GRPO	
Measurement	Step 150	Step 175	Step 150	Step 175
Strict proof success (%)	41.6	43.6	34.0	44.8
Final goal reachability (%)	46.0	48.0	42.8	53.2
Reachable but unsuccessful (%)	4.4	4.4	8.8	8.4
Any protocol violation (%)	22.0	22.4	16.0	11.2
Accepted constructions/episode	2.708	2.712	2.388	2.344
Rejected constructions/episode	0.664	0.668	1.552	1.112

World-model learning for agents. World models support control through learned dynamics and imagined experience (Ha & Schmidhuber, 2018; Hafner et al., 2025), and transition objectives shape representations for model-free learning (Gelada et al., 2019). For language agents, RAP simulates transitions for planning (Hao et al., 2023), while Dyna-Think and Dyna-Mind incorporate environment simulation into reasoning (Yu et al., 2025b;a).

For downstream learning, Agent Learning via Early Experience studies future-state prediction and self-reflection using alternative-action outcomes from expert-trajectory states (Zhang et al., 2026). SPA pretrains state estimation and transition prediction on self-experience before RL (Chen et al., 2026); Reinforcement World Model Learning trains next-state prediction with semantic-similarity rewards before policy RL (Yu et al., 2026). ECHO jointly trains observation prediction and action selection on RL rollouts (Shrivastava et al., 2026).

We study transition pretraining for geometry policies acting on verified engine feedback: prediction on fixed engine-generated trajectories precedes Policy-SFT and optional GRPO. Our ECHO adaptation tests joint training with different optimization settings, comparing complete training recipes.

Assessing learned transition knowledge. Othello probes reveal internal board-state representations (Li et al., 2023), while consistency evaluations show that strong task performance need not imply a coherent implicit world model (Vafa et al., 2024). We track explicit public state-delta F1 and proof success across training stages, separating transition-prediction accuracy from downstream policy quality.

6 Conclusion

Our experiments show that learning action-conditioned state transitions provides an effective initialization for geometry theorem-proving policies. WM-SFT improves proof success after Policy-SFT, and its benefits extend to GRPO both with and without an intervening action-imitation stage. These gains require no transition prediction or additional model calls at inference time.

New-fact F1 declines after Policy-SFT even as proof success improves, separating explicit transition prediction from policy quality. Future work should probe which transition knowledge policies retain and use, and study objectives that preserve useful transition knowledge during subsequent policy optimization, particularly for longer-horizon mathematical agents.

AI Use Statement

Generative AI tools were used only to polish the presentation of this manuscript, including language editing, consistency checks, and figure formatting. They were not used to generate experimental data or conduct experiments. The authors reviewed all AI-assisted changes and take responsibility for the final content of this work, including its text, claims, and figures.

REPRODUCIBILITY STATEMENT

Upon acceptance, we will publicly release our code and datasets, together with clear instructions for environment setup, data preparation, training, and evaluation. The release will include experiment configurations, random seeds, and dataset splits to facilitate reproduction of the reported results.

REFERENCES

Shiqi Chen, Tongyao Zhu, Zian Wang, Jinghan Zhang, Kangrui Wang, Ruochen Zhou, Siyang Gao, Teng Xiao, Yee Whye Teh, Junxian He, and Manling Li. Why do LLM agents fail in exploring new environments? a world-modeling perspective. arXiv preprint arXiv:2510.15047, 2026. URL https://arxiv.org/abs/2510.15047v2. Accepted to Findings of EMNLP 2026.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In International conference on machine learning, pp. 2170–2179. PMLR, 2019.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In Advances in Neural Information Processing Systems, volume 31, 2018. URL https://arxiv.org/abs/1809.01999.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In International conference on machine learning, pp. 2555–2565. PMLR, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. Nature, 640(8059):647–653, 2025.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Proceedings of the 2023 conference on empirical methods in natural language processing, pp. 8154–8173, 2023.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. Advances in neural information processing systems, 32, 2019.

Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In International Conference on Learning Representations, 2023. URL https://arxiv.org/abs/2210.13382.

Prime Intellect. prime-rl: Async rl training at scale, 2025. URL https://github.com/PrimeIntellect-ai/prime-rl. Code release, https://github.com/PrimeIntellect-ai/prime-rl.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. Nature, 588(7839):604–609, 2020.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300, 2024.

Vaishnavi Shrivastava, Piero Kauffmann, Ahmed Awadallah, and Dimitris Papailiopoulos. Echo: Terminal agents learn world models for free. arXiv preprint arXiv:2605.24517, 2026.

Vladmir Sicca, Tianxiang Xia, Mathïs Fédérico, Philip John Gorinski, Simon Frieder, and Shangling Jui. Newclid: A user-friendly replacement for alphageometry. arXiv preprint arXiv:2411.11938, 2024.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. Nature, 625(7995):476–482, 2024.

Keyon Vafa, Justin Y. Chen, Ashesh Rambachan, Jon Kleinberg, and Sendhil Mullainathan. Evaluating the world model implicit in a generative model. In Advances in Neural Information Processing Systems, volume 37, pp. 26941–26975, 2024. URL https://arxiv.org/abs/2406.03689.

Xiao Yu, Baolin Peng, Michel Galley, Hao Cheng, Qianhui Wu, Janardhan Kulkarni, Suman Nath, Zhou Yu, and Jianfeng Gao. Dyna-Mind: Learning to simulate from experience for better AI agents. arXiv preprint arXiv:2510.09577, 2025a. URL https://arxiv.org/abs/2510.09577.

Xiao Yu, Baolin Peng, Ruize Xu, Michel Galley, Hao Cheng, Suman Nath, Jianfeng Gao, and Zhou Yu. Dyna-think: Synergizing reasoning, acting, and world model simulation in ai agents. arXiv preprint arXiv:2506.00320, 2025b.

Xiao Yu, Baolin Peng, Ruize Xu, et al. Reinforcement world model learning for LLM-based agents. arXiv preprint arXiv:2602.05842, 2026. URL https://arxiv.org/abs/2602.05842.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, Jian Xie, Yuxuan Sun, Boyu Gou, Qi Qi, Zihang Meng, Jianwei Yang, Ning Zhang, Xian Li, Ashish Shah, Dat Huynh, Hengduo Li, Zi Yang, Sara Cao, Lawrence Jang, Shuyan Zhou, Jiacheng Zhu, Huan Sun, Jason Weston, Yu Su, and Yifan Wu. Agent learning via early experience. In International Conference on Machine Learning, 2026. URL https://openreview.net/forum?id=N3dXUHY5dD.

Haiteng Zhao, Junhao Shen, Yiming Zhang, Songyang Gao, Kuikun Liu, Tianyou Ma, Fan Zheng, Dahua Lin, Wenwei Zhang, and Kai Chen. Achieving olympia-level geometry large language model agent via complexity boosting reinforcement learning. In International Conference on Learning Representations, 2026.

Minfeng Zhu, Zi Wang, Sizhe Ji, Zhengtong Du, Shengqiang Tai, Junming Ke, Xiao Deng, Zanlang Yin, Xiuqi Huang, Heyu Wang, and Wei Chen. Genesisgeo: Technical report. arXiv preprint arXiv:2509.21896, 2025.

APPENDIX

A PRELIMINARY ANALYSIS OF ECHO

We examine training diagnostics and possible explanations for the ECHO results in Table 1, using base-model and rank-32 WM-SFT initializations.

Teacher-forced loss and generated predictions. Base-initialized ECHO reaches a low new-fact negative log-likelihood (NLL) of approximately 0.18 nats per token, yet its logged mismatch metrics show intermittent spikes and evaluation shows reduced JSON/schema validity. Teacher-forced NLL measures prediction conditioned on the supplied history and preceding target tokens. The world model probe instead requires autoregressive generation from a state-action prefix, and agent evaluation also requires action selection across turns. The diagnostics thus reveal a gap between fitting the training targets and generating valid responses at evaluation.

Working hypotheses. ECHO learns from the current policy's construction attempts, so the distribution of states, actions, and prediction targets changes during training. Our offline WM-SFT stage instead learns from a fixed corpus of engine-generated trajectories and their verified construction feedback. We hypothesize that small online batches and limited reuse of individual transitions make construction-feedback prediction harder to learn alongside policy updates. Dreamer demonstrates improved data efficiency with increased experience reuse (Hafner et al., 2025), motivating an ablation of transition reuse in this setting.

Optimization interference between the policy-gradient and prediction losses, and differences in parameter updates and learning rates, are additional possible contributors. Because both ECHO and WM-SFT use teacher-forced prediction losses, explaining their different outcomes requires examining how data reuse and joint optimization affect generated predictions. Controlled comparisons varying transition reuse, batch size, and the joint-loss weight would help distinguish these hypotheses and assess the role of exposure bias.

B DATA MINING AND SOURCE-BASED SPLITS

Source pools and engine-generated trajectories. GenesisGeo generates synthetic records by sampling geometric configurations, deriving facts with its symbolic engine, and tracing selected conclusions back to their premises and auxiliary constructions (Zhu et al., 2025). The SFT corpus is mined from 75,000 GenesisGeo source records containing auxiliary constructions. An expansion of 400,000 records supplies candidates for RL training and held-out evaluation. We derive learning examples from these source pools through engine qualification and trajectory selection, then exclude SFT sources from the final RL release.

For each source, the mining procedure considers supported goals and tests the recorded construction path together with sampled alternatives. Each retained path is executed and certified by GenesisGeo. Goals refer to the initial points, differ from the constraints directly imposed by the construction actions, and become provable after construction, with at least two logical deductions beyond the initial deductive closure. This count excludes premises, trivial steps, and numerical checks. The general SFT selector admits one to three auxiliary constructions and balances equal-angle, equal-length, parallel, and perpendicular goals. Sampled successful paths are ranked by logical deduction count, construction count, proof depth, numerical-check count, and a deterministic action-order tie-breaker. Sources are processed in a deterministic hash order. For each source, the selector prioritizes an eligible goal from the relation type with the largest remaining quota, using a hash-based tie-breaker. If a relation type has insufficient eligible cases, the builder reduces all four quotas to the largest common capacity while preserving the validation quota.

SFT problems and one-step examples. The 75,000-record pool yields 1,380 selected problems, with 310 training and 35 validation problems per goal type, totaling 1,240 training and 140 validation problems. We split problems before deriving either dataset, so both tasks share the same training and validation assignments. Each problem contributes one engine-verified trajectory. Compiling these trajectories into one-step Policy-SFT examples yields 2,752 training and 310 validation examples.

The corresponding WM-SFT dataset contains 1,512 training and 170 validation examples: Policy-SFT includes each construction action and the final proposition action, whereas WM-SFT uses only construction transitions. Each example includes its preceding context and trains on a single target action or construction response.

RL selection and held-out evaluation. The expansion yields 8,684 qualified problems. We remove 1,139 whose source hashes match the SFT corpus, leaving 7,545 eligible candidates. The selector retains all 1,526 candidates with at least two reference constructions and adds 800 one-construction cases. Within each goal type, these cases are ranked in descending order by post-construction proof depth, post-construction logical deduction count, full-proof logical deduction count, total dependency count, numerical checks, point count, and construction-clause count, with a seeded tie-breaker. Quotas account for the goal distribution of the retained multi-construction cases and the availability of one-construction candidates. The resulting one-construction quotas are 255 equal-length, 221 parallel, 202 perpendicular, and 122 equal-angle problems.

We split the resulting 2,326 problems into 2,076 training and 250 validation problems, stratifying proportionally by goal type and reference construction count. Roughly two-thirds of the release has two or three reference constructions. The four goal types are nearly balanced within each split, with 518–520 training and 62–63 validation problems per type.

Source identity and validation. A source is identified by the SHA-256 hash of its original question and solution. Derived goals retain this parent identity. The canonical SFT builder selects at most one trajectory per source record and assigns whole trajectories to train or validation; all WM-SFT and Policy-SFT views inherit that split. The expanded RL release excludes SFT sources, and its training and held-out partitions contain distinct source records. Validators check source-hash disjointness, exclusion lists, artifact checksums, and engine replay.

C Environment interaction details

Initial context and subsequent observations. The initial message contains the formal problem, target fact, named points, initially visible facts, construction history, required next-point name, and an annotated diagram. This message remains in the conversation context. Subsequent tool responses report the action outcome and a cumulative summary of named points, accepted constructions, visible facts, the next-point name, and the failed-proposal budget. The target fact is repeated in this feedback; the formal problem and initial diagram remain available through the initial message. Proposal bookkeeping also records the last failed proposal and whether a construction has subsequently changed the geometry. To reduce context length, policy rollouts receive subsequent observations as text.

Point naming and action vocabulary. The environment maintains the list of existing point names and supplies the exact name required for the next auxiliary point. A construction call specifies that point and one or two constraints. Every constraint must involve the new point and may otherwise refer only to existing points. Supported constraints are collinearity, equal length, perpendicularity, parallelism, midpoint, cyclicity, and equal angles. For example, constructing the foot D from C onto line AB specifies that A, B, D are collinear and CD is perpendicular to AB. The proposition action accepts equal-length, perpendicular, parallel, or equal-angle statements, represented by a predicate and its point arguments. An intermediate proposal can succeed without solving the target theorem.

Engine execution and feedback. The wrapper translates a construction into the engine's construction language and appends it to the current problem. The backend rebuilds the geometry, runs deductive database and algebraic reasoning (DDAR) to depth 500, extracts the newly derived facts, and renders an updated diagram. The construction is committed only after these operations succeed. Rejected constructions leave the geometry unchanged. Updated diagrams are generated internally even when the rollout configuration sends only textual feedback to the policy. Each public fact report contains at most eight canonical new facts, prioritizing facts involving the new point. Reports accumulate in the visible context, while the engine retains its complete deductive closure internally. Reference actions, source identity, full closure, proof ancestry, and the private goal-reachability flag are withheld from the policy.

Interaction budgets. Each rollout permits at most four accepted constructions, three failed proposals, and ten total turns, with one tool call per turn. The construction limit counts accepted points, and the proposal limit counts failed checks; the total-turn limit also bounds rejected constructions and successful intermediate proposals. A give-up action ends the rollout immediately.

State and conversation history. Let x_t denote the complete internal environment state, including the engine's deductive closure and proof ancestry, and let $s_t = O(x_t)$ denote the public observation. Valid actions induce deterministic transitions $x_{t+1} = T(x_t, a_t)$ in a finite-horizon, partially observed process. The policy conditions on the conversation prefix $h_t = (s_0, a_0, s_1, \ldots, a_{t-1}, s_t)$, which preserves the initial context and the preceding actions and observations. For world model pretraining, this history is prepared with a dedicated prediction prompt as described in Appendix D, and the target $y_t = g(s_t, a_t, s_{t+1})$ serializes the compact public consequence of the supplied construction.

C.1 RL ROLLOUTS AND REWARDS

For RL, each episode starts from a training problem's initial geometry and collects a complete interaction trajectory by executing the policy's actions in the live GenesisGeo backend. The policy uses the tools, cumulative feedback, and interaction budgets described above. Episodes end on proof completion, a give-up action, exhaustion of the failed-proposal or turn budget, an engine worker failure, or an assistant response without a tool call.

GRPO and ECHO use the same scalar episode reward, computed at the end of the trajectory as the sum of the components in Table 4. The completion term rewards solving the theorem, while the partial-credit term rewards reaching geometry from which the engine can prove the goal, even if the agent has not successfully submitted it. The efficiency bonus favors economical constructions, and the protocol penalty discourages malformed tool use. Goal provability and the reference construction count are private scoring information and are not included in the policy's observations.

Table 4: RL episode reward components. Each contribution is applied at most once per episode, and the total reward is their sum. The completion and partial-credit terms are mutually exclusive.

Component	Reward	Condition
Proof completion	$+1.00$	The agent has made at least one accepted construction and successfully submitted the target relation through a proposition action.
Goal provable	$+0.20$	The engine can prove the goal in the final geometry, but the agent has not completed the proof and has not chosen to give up.
Efficiency bonus	$+0.05$	The proof-completion condition holds, and the number of accepted constructions is no greater than the positive reference construction count.
Protocol violation	-0.05	At least one turn has an incorrect number of tool calls, an unknown tool, invalid JSON arguments, or a call rejected by the syntax/schema parser.

A completed proof earns 1.05 with the efficiency bonus and 1.00 otherwise, before protocol penalties. The bonus compares construction counts and permits alternative solutions. Well-formed rejected constructions and unsuccessful propositions receive no separate penalty. Repeated protocol violations incur only one 0.05 deduction. ECHO adds its prediction loss to the training objective separately from the episode reward. Figure 2 plots validation reward at logged checkpoints.

D TRAINING AND EVALUATION DETAILS

World model prompt and context. We use a dedicated system prompt that specifies one-step construction-feedback prediction, geometric syntax and conventions, and the required JSON response. It includes a syntax-only example of a candidate construction and response. The final user message supplies a candidate construction and requests its immediate consequences. No executable tool definitions are provided. The input contains the formal construction state, named points, visible facts, and earlier construction actions and feedback, with the proof goal and the candidate construction's response removed. Each example includes only the latest diagram; earlier constructions and feedback remain as text. Policy rollouts instead receive only the initial diagram.

World model targets. The target is compact JSON containing an acceptance flag, the constructed point, a new-facts list, and the next required point name. The list contains canonical equal-length, perpendicular, parallel, or equal-angle relations, selected using the observation rules in Appendix C. These offline examples use accepted constructions from engine-generated trajectories, so the acceptance flag is always true. Only the final target response tokens receive cross-entropy loss.

27B supervised training and checkpoint selection. The 27B SFT and RL experiments use eight NVIDIA H100 GPUs. SFT uses a batch size of 32. Unless otherwise noted, LoRA uses a constant learning rate of 10^{-4} and full-parameter fine-tuning uses a learning rate of 10^{-5}. The full-parameter WM-SFT variant freezes the vision encoder and updates all non-vision parameters; LoRA variants update the language attention and MLP projections at ranks 16 or 32. Before Policy-SFT, any WM-SFT adapter is merged into the base weights. All Policy-SFT configurations use LoRA rank 16. The Policy-SFT-only baseline starts from the pretrained Qwen3.5-27B base and receives the same total training budget as WM-SFT and Policy-SFT combined. Both training routes use the same policy-training settings and select checkpoints by the lowest validation loss. To assess robustness across training seeds, we run the primary SFT comparison with three seeds per training sequence. Policy-SFT alone obtains 63.2%, 63.6%, and 70.8% Pass@1; rank-32 WM-SFT followed by Policy-SFT obtains 74.0%, 74.0%, and 70.8%. The respective means and sample standard deviations are $(65.9 \pm 4.3)\%$ and $(72.9 \pm 1.8)\%$. Standard deviations use an $n - 1$ denominator, with $n = 3$; the 7.1-point mean improvement is computed before rounding. The remaining SFT configurations and all new-fact F1 measurements in Table 1 are reported for a single run.

For the reverse-order ablation, we initialize WM-SFT from the Policy-SFT checkpoint evaluated in Table 1, use the same training budget as the other WM-SFT runs, and select the final checkpoint by the lowest world-model validation loss.

Reinforcement-learning objectives. All RL configurations use the same random seed. GRPO uses a batch size of 16, eight rollouts per example, and a maximum of 12 in-flight rollouts in PrimeRL. GRPO applies policy-gradient loss only to action tokens; environment responses remain context. We evaluate ECHO from both base Qwen3.5-27B and the rank-32 WM-SFT checkpoint; for the latter, the adapter is merged before RL. Our ECHO adaptation adds cross-entropy supervision on selected public consequence values, chiefly the new-facts list, from the same on-policy trajectories. The original ECHO recipe uses a prediction-loss weight of 0.05 and 16 rollouts per prompt (Shrivastava et al., 2026). Both geometry configurations use a prediction-loss weight of 0.1 and reduce the rollout count to eight per problem to fit our compute budget. We retain full-parameter fine-tuning, a learning rate of 10^{-6}, and a batch size of 16, following ECHO. Appendix A discusses preliminary diagnostics.

World model evaluation. Section 4.1 defines the held-out transition probe, generation settings, invalid-output handling, and new-fact micro-F1 (Equation 3). Before matching facts, we normalize the ordering of point arguments under the supported relation symmetries. Reversing a segment's endpoints or exchanging the two segments in an equal-length relation preserves the fact, so $AB = CD$ and $BA = DC$ match. Parallel and perpendicular relations similarly treat both lines as undirected and interchangeable. Equal-angle relations allow endpoint reversal, exchange of the two lines defining each angle, and exchange of the two angles, following the environment's line-angle convention. Matching then compares the predicate and normalized point arguments. Reference sets use the bounded public fact reports described in Appendix C.

D.1 WORLD MODELLING EXAMPLE

This held-out example shows the current geometry in mathematical notation and reproduces the candidate action and engine-generated target as recorded JSON.

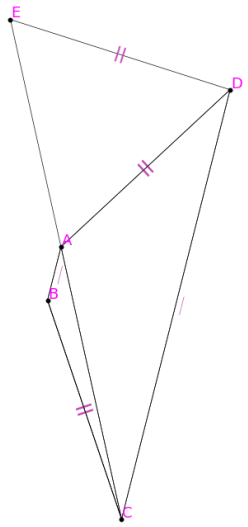
Input: current geometry. The named points are A, B, C, D, E.

The formal construction state specifies $AB \parallel CD$, $AD = BC$, collinear points A, C, E, and $BC = DE$.

The five already-visible facts are $AB \parallel CD$, $AD = BC$, $AD = DE$, $BC = DE$, and

$$\angle(AD, AE) = \angle(AE, DE),$$

using the supporting-line angle convention.
There is no prior auxiliary-construction history; the required next point name is F. The proof goal is removed from this input.



Original current-state diagram, before constructing F.

Input: supplied candidate action. The final user message requests its immediate public consequence.


```
{"arguments":{"constraints":[
  {"args":["a","b","d","f"],"predicate":"parallel"},
  {"args":["a","d","b","f"],"predicate":"parallel"}
],"point":"f"},"tool_name":"construct_auxiliary"}
```


Supervision target: engine-generated feedback.


```
{"accepted":true,"new_facts":[
  {"args":["a","b","d","f"],"predicate":"equal_length"},
  {"args":["a","d","b","f"],"predicate":"equal_length"},
  {"args":["b","c","b","f"],"predicate":"equal_length"},
  {"args":["a","b","c","f"],"predicate":"parallel"},
  {"args":["a","b","d","f"],"predicate":"parallel"},
  {"args":["a","d","b","f"],"predicate":"parallel"},
  {"args":["a","b","a","c","b","d","d","f"],"predicate":"equal_angle"},
  {"args":["a","b","a","d","b","f","d","f"],"predicate":"equal_angle"}
],"next_point_name":"g","point":"f"}
```


The supplied constraints $AB \parallel DF$ and $AD \parallel BF$ complete parallelogram $ABFD$. The target therefore includes derived equalities $AB = DF$ and $AD = BF$, among other reported facts. The model is trained to predict this feedback given the construction and current geometry.

D.2 SYSTEM PROMPTS

The following system prompts are reproduced verbatim, with line wrapping for readability. The generic agentic prompt describes updated diagrams in construction feedback; our rollout configuration retains only the initial diagram (Appendix C). The world-modelling task uses the dedicated world-model profile and receives a candidate action in the final user message. The agentic task uses the neutral agent profile and receives executable tool schemas.

D.2.1 WORLD MODELLING TASK

```
THIS IS A NEXT-STATE PREDICTION TASK.

You are predicting GenesisGeo's immediate public state transition. The
conversation gives the current model-visible geometry state and may
include prior agent actions and engine feedback. The final user
message supplies one candidate construct_auxiliary action that has not
yet been executed.

Predict only the immediate public symbolic consequence of executing
that candidate action. The candidate action is input, not a request to
execute it and not a target to repeat. DO NOT CALL OR EXECUTE A TOOL.
Do not choose a different action, solve the proof goal, or predict
private GenesisGeo state.

Construction constraints and argument shapes:
- collinear [P1,P2,P3]
- equal_length [P1,P2,P3,P4]
- perpendicular [P1,P2,P3,P4]
- parallel [P1,P2,P3,P4]
- midpoint [M,A,B], meaning M is the midpoint of AB
- cyclic [P1,P2,P3,P4]
- equal_angle [P1,P2,P3,P4,P5,P6,P7,P8]

Proposal predicates and argument shapes:
- equal_length [P1,P2,P3,P4]
- perpendicular [P1,P2,P3,P4]
- parallel [P1,P2,P3,P4]
- equal_angle [P1,P2,P3,P4,P5,P6,P7,P8]

CRITICAL ANGLE CONVENTION: each point pair is an undirected supporting
line. For equal_angle, the line-angle between P1P2 and P3P4 equals the
line-angle between P5P6 and P7P8 modulo 180 degrees. Reversing a line,
exchanging the lines in one angle, or exchanging the two angles does
not change the fact. Therefore supplementary-looking interior angles
may represent the same equal_angle fact.

Diagram annotations are certified context: equal-length segments use
same-color tick strokes, perpendicular lines use a right-angle box,
and equal line-angles use same-color arcs. Do not confuse annotations
already present with new feedback.

CRITICAL FEEDBACK SEMANTICS:
- An accepted construction changes the geometry but does not guarantee
progress toward the goal.
- new_facts contains only the bounded newly reported public facts and
may be empty. It is not the engine's complete internal proof state.
- A rejected construction leaves the geometry unchanged.
- Use the updated diagram, construction history, and certified
feedback when choosing the next action.
- A proposal consumes the limited proposal budget and returns only
binary verification.

Return exactly one compact JSON object with these keys:
- accepted: whether GenesisGeo accepts the construction.
```

```
- point: the candidate auxiliary point name.
- new_facts: the bounded canonical public facts newly reported after
the construction.
- next_point_name: the required name for a subsequent auxiliary point.

Syntax-only example (not facts about the current problem):
Candidate action: {"tool_name":"construct_auxiliary","arguments":{"po⌋
int":"d","constraints":[{"predicate":"collinear","args":["a","b","d"]⌋
},{"predicate":"perpendicular","args":["c","d","a","b"]}]}}
Required response: {"accepted":true,"point":"d","new_facts":[{"predic⌋
ate":"perpendicular","args":["a","b","c","d"]}],"next_point_name":"e"}

RETURN ONLY THE NEXT-STATE JSON OBJECT. DO NOT RETURN OR CALL
construct_auxiliary. Do not include Markdown, explanation, an updated
image, the complete DDAR closure, proof ancestry, goal reachability,
or unreported future facts. The environment remains authoritative;
this is a prediction of its public feedback.
```

D.2.2 AGENTIC TASK

```
You are a geometry-proof agent interacting with GenesisGeo. Solve the
stated formal goal by constructing useful auxiliary points and
proposing engine-certifiable facts.

The initial message gives the current diagram, formal problem, named
points, bounded visible facts, proof goal, and the exact name required
for the next auxiliary point. Tool schemas are authoritative for JSON
formatting. Make exactly one tool call per turn.

Available actions:
- construct_auxiliary creates the required next point using one or two
constraints. Every constraint must include that new point. Use only
named points plus the required new point. An accepted action rebuilds
GenesisGeo, reruns DDAR, and returns an updated diagram and bounded
newly certified facts. A rejected action changes nothing.
- propose asks whether one supported formal fact is certified in the
current state. It returns only binary success/failure and whether the
stated goal was solved; it gives no proof, hint, missing premise, or
distance to the goal.
- give_up ends the rollout unsuccessfully. Call it only when no useful
legal construction or proposal remains.

Construction constraints and argument shapes:
- collinear [P1,P2,P3]
- equal_length [P1,P2,P3,P4]
- perpendicular [P1,P2,P3,P4]
- parallel [P1,P2,P3,P4]
- midpoint [M,A,B], meaning M is the midpoint of AB
- cyclic [P1,P2,P3,P4]
- equal_angle [P1,P2,P3,P4,P5,P6,P7,P8]

Proposal predicates and argument shapes:
- equal_length [P1,P2,P3,P4]
- perpendicular [P1,P2,P3,P4]
- parallel [P1,P2,P3,P4]
- equal_angle [P1,P2,P3,P4,P5,P6,P7,P8]

CRITICAL ANGLE CONVENTION: each point pair is an undirected supporting
line. For equal_angle, the line-angle between P1P2 and P3P4 equals the
line-angle between P5P6 and P7P8 modulo 180 degrees. Reversing a line,
exchanging the lines in one angle, or exchanging the two angles does
not change the fact. Therefore supplementary-looking interior angles
may represent the same equal_angle fact.
```

```
Diagram annotations are certified context: equal-length segments use
same-color tick strokes, perpendicular lines use a right-angle box,
and equal line-angles use same-color arcs. Do not confuse annotations
already present with new feedback.

CRITICAL FEEDBACK SEMANTICS:
- An accepted construction changes the geometry but does not guarantee
progress toward the goal.
- new_facts contains only the bounded newly reported public facts and
may be empty. It is not the engine's complete internal proof state.
- A rejected construction leaves the geometry unchanged.
- Use the updated diagram, construction history, and certified
feedback when choosing the next action.
- A proposal consumes the limited proposal budget and returns only
binary verification.

Strategy:
1. Read the goal predicate and identify the missing relation needed to
certify it.
2. Prefer a construction that creates a standard bridge: a
perpendicular foot, line intersection, midpoint, parallel line,
equal-distance point, concyclic point, or angle-transfer point.
3. Use returned new facts and the updated diagram as the current
state. Do not assume consequences that the engine did not certify.
4. Keep the exact stated goal in view after every turn. Propose that
exact goal as soon as it is certified. Do not spend the limited
proposal budget on unsupported guesses.
5. A failed proposal does not change the geometry. NEVER repeat the
same failed proposal unless at least one construction has subsequently
been accepted and changed the state.
6. Intermediate proposals are allowed only when supported by the
current state and useful for the proof; a successful intermediate
proposal does not solve the rollout unless it is the stated goal.
7. Never invent point names, reuse a constructed point name, expose
raw GenesisGeo DSL, or call another tool after success or give_up.

Example tool-call shapes (syntax only, not facts about the current
problem):
construct_auxiliary({"point":"d","constraints":[{"predicate":"colline⌋
ar","args":["a","b","d"]},{"predicate":"perpendicular","args":["c","d⌋
","a","b"]}]})
propose({"predicate":"perpendicular","args":["c","d","a","b"]})

You may make at most four accepted constructions, three failed
proposals, and ten total turns. The rollout succeeds only when propose
accepts the stated goal.
```