REALOGIC-FT: PROGRESSIVE NEURO-SYMBOLIC POST-TRAINING FOR REASONER-VERIFIABLE LOGICAL REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models often answer logical reasoning questions correctly, yet the reasoning they display is free-form text: no machine can confirm that each step follows from the premises or that the conclusion is supported. Making such reasoning verifiable faces two obstacles. Facts, rules, and inferences must be expressed in a formal language the model does not natively speak, and training on reference proofs ties it to a single derivation per problem, penalizing equally valid alternatives while giving no signal about logical soundness. We present ReaLogic-FT, a progressive post-training framework that removes both. The model first learns to translate natural-language statements into the description logic $\mathcal{ALC}$, decidable yet expressive enough for the theories we retain, so that a reasoner can conclusively check every claimed inference. It then learns to compose such assertions into structured proofs ending in a verdict of TRUE, FALSE, or UNKNOWN. A final reinforcement learning stage drops the reference proofs entirely: an off-the-shelf reasoner checks each candidate derivation step by step, and any proof grounded in the given theory and logically valid is rewarded. We build verifier-checkable versions of ProofWriter and ProntoQA, projecting each problem onto the supported logic and adding queries that can be neither proved nor refuted. Proof supervision raises joint answer-and-proof correctness by over 40 percentage points relative to few-shot prompting. HermiT-guided reinforcement learning then matches or improves the strong SFT initialization on every reported metric and outperforms its semantic-reward ablation in mean performance; these differences are small and not statistically significant, with the clearest gains on deeper ProofWriter instances. Every accepted proof step can be independently audited by a symbolic reasoner against the given formal theory.

1 INTRODUCTION

Large language models answer rule-based reasoning questions well and produce fluent intermediate rationales Clark et al. (2020); Wei et al. (2022), yet a correct answer need not rest on correct reasoning: a model may exploit shortcuts or reach the right conclusion through flawed steps Turpin et al. (2023); Paul et al. (2024); Lanham et al. (2023), and popular remedies such as self-consistency raise accuracy without checking whether the reasoning holds Wang et al. (2023). The deeper issue is that free-form rationales, however plausible they read, leave premises, rule applications, and dependencies too implicit for reliable mechanical verification of whether each step follows and the conclusion is supported Lyu et al. (2023); Pei et al. (2025). The goal we pursue is therefore not a more convincing explanation, but reasoning that a symbolic reasoner can independently audit.

Auditable reasoning must satisfy conditions that a plausible narrative need not: every cited fact and rule comes from the given theory, each conclusion follows from its premises, dependencies respect the proof order, the whole derivation stays consistent, and the final assertions support the answer. Open-world semantics adds a subtler demand: failing to prove a query does not prove its opposite, so distinguishing FALSE from UNKNOWN requires checking whether the query or its opposite is entailed. Prior work already renders reasoning inspectable through explicit proof structures, from verifier-guided natural-language proof search Yang et al. (2022) to proofs written directly in OWL

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.

Yang et al. (2026a); what remains essential is a target with explicit syntax and unambiguous semantics that a reasoner can check Motik et al. (2012).

Auditability calls for two abilities that are easy to conflate yet fail differently: rendering natural language into formal statements, and composing those statements into a valid derivation. Learning them at once is fragile, since a failed proof leaves it unclear whether the model misread the source language, broke the target syntax, or drew an invalid inference. This motivates a progressive strategy: a model must first learn to speak a formal language before it can reason reliably in one.

ReaLogic-FT realizes this progression in two supervised stages. The first aligns individual facts, rules, and queries with canonical assertions of the description logic $\mathcal{ALC}$, a decidable fragment of first-order logic expressive enough for the theories we retain, serialized in OWL Functional-Style Syntax that a reasoner executes directly Baader et al. (2003); casting natural language into formal logic is itself an established fine-tuning target Yang et al. (2024), which we use as preparation rather than an end. The second stage composes these assertions into structured proofs whose every non-direct step records its premises, cited rule, and conclusion, ending in a TRUE, FALSE, or UNKNOWN decision.

Proof supervision improves composition but remains tied to a single reference, although a problem may admit multiple valid derivations Saha et al. (2021). Token-level imitation can penalize alternative grounded proofs and guarantees neither step entailment, chain consistency, nor answer support. Because proof correctness is a property of logical form rather than surface overlap, it must be verified.

The final stage supplements benchmark answer supervision with verifier-guided reinforcement learning that does not require reference-proof imitation. The policy generates complete derivations, and deterministic checks enforce interface validity and theory grounding, while HermiT verifies step entailment and semantic answer support Glimm et al. (2014). These judgments test exactly the properties we want to optimize but are discrete and non-differentiable, so we convert them into multidimensional rewards: proof quality is rewarded through logical validity rather than overlap with a single reference, and any grounded, valid derivation earns credit. Rewarding verified correctness follows the process-supervision line Lightman et al. (2023); Wang et al. (2024), which recent work extends to logical reasoning with formal checkers over first-order or propositional steps Chen et al. (2026); Xu et al. (2026); Feng et al. (2025). ORACLE Yang et al. (2026b) also verifies intermediate steps symbolically, but uses symbolic and LLM scores to construct offline SFT and DPO data, whereas we use strict reasoner verification as an online RL signal. Related proof-assistant methods use Lean feedback for formal mathematical proving, or to verify synthetic reasoning data and supply reinforcement-learning feedback Xin et al. (2025); Leang et al. (2025). ReaLogic-FT instead targets natural-language rule reasoning under $\mathcal{ALC}$ and open-world semantics, with grounded, sequentially verified proofs and an explicit UNKNOWN verdict treated as a first-class outcome rather than a failure to prove.

Inference-time approaches instead call an external solver to verify or filter outputs, such as Faithful CoT Lyu et al. (2023), Logic-LM Pan et al. (2023), LINC Olausson et al. (2023), SatLM Ye et al. (2023), and FoVer Pei et al. (2025), or emulate symbolic reasoning within the language model, as in SymbCoT Xu et al. (2024). ReaLogic-FT, like the training-time methods above, uses verifier feedback to reshape the policy itself rather than only its outputs, so the model learns to produce grounded, logically valid, reasoner-verifiable proofs.

To train and evaluate under these requirements, we build a unified, verifier-checkable dataset from two complementary benchmarks: ProofWriter Tafjord et al. (2021) for multi-step rule reasoning under open-world semantics, and ProntoQA Saparov & He (2023) for controlled ontology-style reasoning across hop counts. Both are projected into the supported $\mathcal{ALC}$ fragment and cast as structured OWL proofs with TRUE, FALSE, and UNKNOWN verdicts.

Our contributions are threefold:

- We introduce ReaLogic-FT, a progressive neuro-symbolic post-training recipe in which a single policy first learns to speak a formal logic, then to compose proofs in it, and finally is optimized against a reasoner rather than a reference, turning each ability's failure mode into a separate, addressable stage.

- We operationalize reasoner-verifiable reasoning as deterministic and reasoner-based checks covering grounding, structural validity, sequential entailment over an accepted state, answer support, and open-world UNKNOWN, turning the grounding, entailment, and answer-support checks into a reward that reinforcement learning can optimize directly, while HermiT certifies UNKNOWN verdicts at evaluation.
- On verifier-checkable versions of ProofWriter and ProntoQA spanning reasoning depths up to five, ReaLogic-FT produces proofs that HermiT can audit step by step: proof supervision raises joint answer-and-proof correctness by over 40 percentage points over few-shot prompting, significant after Holm correction, and the verifier reward yields directionally consistent but statistically non-significant mean gains over its semantic-reward ablation.

2 PRELIMINARIES

2.1 LOGICAL SETTING AND THREE-WAY ENTAILMENT

Each reasoning instance consists of a natural-language theory and a query, paired with a formal ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ and a gold query assertion q. Here, the TBox $\mathcal{T}$ contains general concept inclusions (GCIs), while the ABox $\mathcal{A}$ contains concept and role assertions about named individuals. We consider the description logic $\mathcal{ALC}$ Baader et al. (2003), whose concepts are defined by $C, D ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists r.C \mid \forall r.C$, where A is a concept name and r is a role name. We serialize the corresponding axioms in OWL Functional-Style Syntax Motik et al. (2012), retaining concept assertions, role assertions, and GCIs.

The model does not receive q directly; it must produce its own canonical OWL query assertion $\hat{q}$ from the natural-language query, with a valid interface requiring $\hat{q} = q$. Let $\bar{q}$ denote the negated assertion of q: $(\neg C)(a)$ for $q = C(a)$, and, for ProofWriter role queries $q = r(a, b)$, the negative role assertion $\neg r(a, b)$, which lies outside $\mathcal{ALC}$ and is used only by the verifier. Under the open-world assumption Hitzler et al. (2012), the three-way label is $y(\mathcal{O}, q) = \text{TRUE}$ if $\mathcal{O} \models q$, FALSE if $\mathcal{O} \models \bar{q}$, and UNKNOWN if $\mathcal{O} \not\models q$ and $\mathcal{O} \not\models \bar{q}$. We assume that $\mathcal{O}$ is consistent and treat inconsistent instances as invalid.

2.2 VERIFIER-CHECKABLE STRUCTURED PROOFS

For a predicted outcome $\hat{y}$, the model generates a structured proof $\pi = (F_0, s_1, \ldots, s_m)$ with at most $m \leq 8$ inference steps, where $F_0 \subseteq \mathcal{A}$ contains optional supporting input facts and each step $s_i = (P_i, r_i, c_i)$ consists of premises P_i, a cited rule r_i, and a conclusion c_i. We characterize a generated proof using the following properties. (1) Directness: A proof is direct when q or $\bar{q}$ is an input fact recorded in F_0; otherwise, it contains one or more inference steps. (2) Grounding: A proof is grounded if, for every step i, $r_i \in \mathcal{T}$ and $P_i \subseteq \mathcal{A} \cup \{c_1, \ldots, c_{i-1}\}$. (3) Structural validity: A step is structurally valid when its fields are well formed and well typed, its citations are resolvable, and its dependencies respect the proof order. (4) Semantic validity: An eligible step is semantically valid when HermiT verifies $P_i \cup \{r_i\} \models c_i$. Only verified conclusions enter the accepted state and may support subsequent steps. (5) Answer support: The accepted state must contain q for a TRUE prediction or $\bar{q}$ for a FALSE prediction; an UNKNOWN prediction requires an empty proof, and HermiT must confirm that $\mathcal{O}$ entails neither q nor $\bar{q}$. (6) Overall validity: An output $(\pi, \hat{q}, \hat{y})$ is verifier-valid when its model–reasoner interface is valid, $\hat{q} = q$, its answer is correct, every applicable step is structurally valid, grounded, and semantically verified, the resulting ontology remains consistent, and the accepted state supports the answer.

Deterministic checks validate formatting, OWL syntax, vocabulary, grounding, and dependency order, while HermiT provides entailment and consistency judgments.

3 REALOGIC-FT

3.1 FRAMEWORK OVERVIEW

Figure 1 presents ReaLogic-FT, a progressive post-training framework that trains a language model to produce reasoner-verifiable structured proofs from natural-language problems. For each dataset

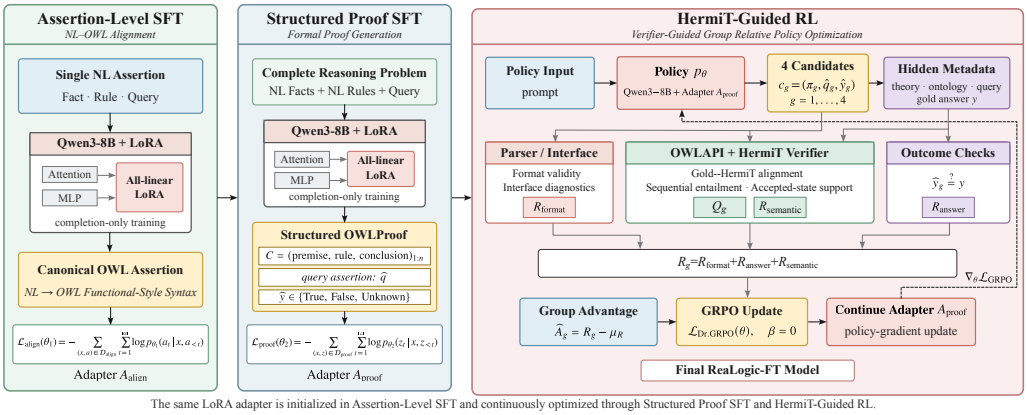

Figure 1: Overview of ReaLogic-FT, comprising assertion-level NL–OWL alignment, structured proof supervision, and HermiT-guided reinforcement learning.

pipeline, we instantiate the policy with Qwen3-8B Team (2025) and continuously train a single LoRA adapter over all attention and feed-forward projections Hu et al. (2022) through the three stages.

The first two stages separate acquisition of the formal language from proof composition; the third optimizes composition against executable logical criteria rather than a reference proof.

3.2 Verifier-Checkable Data Construction

We construct verifier-checkable data from ProofWriter Tafjord et al. (2021) and ProntoQA Saparov & He (2023). Although the two benchmarks differ in source schema and proof representation, each example is converted into a unified record containing a natural-language theory, an OWL ontology $\mathcal{O}$, a query assertion q and its negated assertion $\bar{q}$, a three-way label y, and, when available, a supporting derivation.

We parse, validate, canonicalize, and deduplicate all converted axioms under the supported $\mathcal{ALC}$ profileBaader et al. (2003); implementation details appear in Appendix A.1.4. For ProofWriter, rules outside the profile are removed only when the retained facts, rules, query, and proof form a self-contained projection that preserves the original label according to the retained reference proof; otherwise, the instance is rejected. Because this check does not query HermiT, a later HermiT audit identifies 37 ProofWriter test instances whose benchmark label disagrees with the projected ontology (Appendix A.4.1). ProofWriter already supplies open-world Unknown cases, whereas for ProntoQA we form Unknown queries from concept pairs with no directed subsumption path, retaining them only when neither q nor $\bar{q}$ is entailed.

We control leakage at the representation level: assertion pairs are globally deduplicated, and ProofWriter examples sharing the same canonicalized ABox and TBox are kept within a single split, so no ontology spans training and evaluation. The resulting records feed the three stages. We construct a difficulty-aware GRPO curriculumShao et al. (2024) that emphasizes longer proofs and Unknown cases; details appear in Appendix A.1.8.

3.3 Assertion-Level Supervised Fine-Tuning

Given these data, the first stage removes the formal-language barrier: the model learns the representation the verifier requires before it must solve complete reasoning problems. Each alignment example (x, a) maps a prompt containing one natural-language statement and its admissible ontology vocabulary to a canonical OWL assertion a. The model emits the assertion directly, and we

apply completion-only supervision:

$$\mathcal{L}_{\text{align}}(\theta_1) = -\sum_{(x,a)\in\mathcal{D}_{\text{align}}} \sum_{t=1}^{|a|} \log p_{\theta_1}(a_t \mid x, a_{<t}). \tag{1}$$

This objective concentrates supervision on the target OWL assertion rather than the fixed prompt tokens.

3.4 Structured Proof Supervised Fine-Tuning

Starting from the assertion-aligned adapter, the second stage operates on complete reasoning problems. For an input x containing natural-language facts, rules, a query, and the admissible ontology vocabulary, the target is $z = (\pi, q, y)$. Here, π is the reference structured proof in the format defined in the preliminaries, q is the gold canonical OWL query assertion, and $y \in \{\text{TRUE}, \text{FALSE}, \text{UNKNOWN}\}$ is the gold label. This representation follows prior work on explicit proof generation Saha et al. (2020); Tafjord et al. (2021), but expresses every proof element in a syntax that can be executed by an OWL reasoner.

We topologically order reference steps so that each proof is grounded in the sense of Section 2.2, and train the continued adapter with the completion-only objective:

$$\mathcal{L}_{\text{proof}}(\theta_2) = -\sum_{(x,z)\in\mathcal{D}_{\text{proof}}} \sum_{t=1}^{|z|} \log p_{\theta_2}(z_t \mid x, z_{<t}), \tag{2}$$

where θ_2 is initialized from θ_1. This stage jointly supervises query formalization, proof composition, and answer prediction. However, token-level imitation remains tied to one reference derivation: it can penalize semantically valid alternatives and provides no direct signal of logical soundness. The final stage addresses both limitations with verifier-derived rewards.

3.5 HermiT-Guided Reinforcement Learning

For each prompt x, the policy samples $G = 4$ candidate completions, $c_g = (\pi_g, \hat{q}_g, \hat{y}_g)$ for $g = 1, \ldots, G$, where π_g is the structured proof of candidate g, as defined in Section 2.2. As described above, the policy sees only x, while the theory, executable ontology, formal query, and gold answer remain hidden verifier metadata. The gold proof is neither exposed to the policy nor used for scoring.

Verifier-guided reward. We first parse each candidate and check its required tags, query match ($\hat{q}_g = q$), OWL syntax and symbol roles, grounding, and eight-step limit. These checks determine the validity of the model–reasoner interface and are retained for evaluation and diagnostics. Through the OWL API Horridge & Bechhofer (2011), HermiT Glimm et al. (2014) independently determines the semantic answer of the base ontology. Semantic rewards are computed only when this answer agrees with the benchmark label; when the labels disagree or verification fails, R_{semantic} is set to zero while R_{format} and R_{answer} are retained.

ReaLogic-FT verifies proof steps sequentially against an accepted state initialized with the input facts. Only HermiT-entailed conclusions are added to this state and may support subsequent steps. Let $J_{g,i}$ indicate that step i is accepted: it is structurally well formed and OWL-valid, cites a rule from the input TBox, uses only input facts or previously accepted conclusions as premises, and is verified by HermiT to entail its conclusion. Conclusion novelty and dataset-specific inference templates are recorded only for diagnostics and do not affect step acceptance or reward.

For a required proof with m_g generated steps, we define $Q_g = \frac{1}{m_g}\sum_{i=1}^{m_g} J_{g,i}$ when $m_g > 0$ and the accepted state supports the predicted answer, and $Q_g = 0$ otherwise. The accepted state supports $\hat{y}_g$ when it contains q for TRUE or $\bar{q}$ for FALSE. A proof is required exactly when the gold answer is TRUE or FALSE and is not directly supported by an input fact.

ReaLogic-FT separates benchmark supervision from semantic verification. The total reward is

$$R_g = R_{\text{format}} + R_{\text{answer}} + R_{\text{semantic}}, \tag{3}$$

where $R_{\text{format}} = +0.5$ when the output follows the required structure, including the prescribed tags, field organization, step order, and answer vocabulary, and $R_{\text{format}} = -0.5$ otherwise, while $R_{\text{answer}} = +2$ if $\hat{y}_g = y$ and -2 otherwise.

For examples requiring a proof, the semantic reward interpolates between the invalid- and fully valid-proof rewards according to Q_g, with $R_{\text{semantic}} = -2 + 4Q_g$.

For examples not requiring a proof, let k_g be the number of generated inference steps rejected by the sequential verifier. The semantic term penalizes such residual steps, with $R_{\text{semantic}} = \max(-0.75, -0.15k_g)$ if $k_g > 0$, and $R_{\text{semantic}} = 0$ otherwise.

When semantic verification is unavailable because of a benchmark–HermiT label mismatch or verifier failure, R_{semantic} is set to zero, while the format and answer terms are preserved. Thus, the gold label provides answer supervision, whereas HermiT evaluates grounded steps incrementally. This separation prevents solver feedback from replacing benchmark supervision and prevents rejected conclusions from supporting later reasoning.

Group-relative optimization. With these rewards in hand, we update the policy without a value critic: GRPO compares candidates generated for the same prompt Shao et al. (2024). We use its Dr. GRPO variant Liu et al. (2025): rewards are centered but not divided by their within-group standard deviation, with $\mu_R = \frac{1}{G}\sum_{j=1}^{G} R_j$ and $\widehat{A}_g = R_g - \mu_R$.

The resulting advantage assigns positive values to candidates whose verifier rewards exceed the group average and negative values otherwise.

The policy is then updated with the standard clipped surrogate Schulman et al. (2017). We write p_θ for the policy distribution, keeping π reserved for structured proofs, and let θ denote the current adapter parameters, initialized from θ_2. For token t of candidate g, the importance ratio is

$$\rho_{g,t}(\theta) = \frac{p_\theta(c_{g,t} \mid x, c_{g,<t})}{p_{\theta_{\text{old}}}(c_{g,t} \mid x, c_{g,<t})}, \tag{4}$$

where $c_{g,t}$ is the token at position t of candidate g, $c_{g,<t}$ its generated prefix, and $p_{\theta_{\text{old}}}$ the rollout policy. With clipping radius ϵ, the per-token contribution clips this ratio to $[1-\epsilon, 1+\epsilon]$ and takes the conservative bound,

$$\ell_{g,t}(\theta) = \min\left(\rho_{g,t}(\theta)\widehat{A}_g,\ \text{clip}(\rho_{g,t}(\theta), 1-\epsilon, 1+\epsilon)\,\widehat{A}_g\right). \tag{5}$$

Summing over tokens and candidates with maximum completion length $L_{\max}$ gives the Dr. GRPO loss

$$\mathcal{L}_{\text{Dr. GRPO}}(\theta) = -\frac{1}{GL_{\max}} \sum_{g=1}^{G} \sum_{t=1}^{|c_g|} \ell_{g,t}(\theta). \tag{6}$$

We set the reference-policy KL coefficient to $\beta = 0$; the rollout policy $p_{\theta_{\text{old}}}$ is the importance-sampling reference, not a separately regularizing model. Normalizing by the constant $GL_{\max}$ rather than the realized length $|c_g|$ follows Dr. GRPO and avoids favoring shorter completions Liu et al. (2025). Reinforcement learning updates the same adapter, yielding the final ReaLogic-FT policy.

4 Experiments

4.1 Experimental Setup

Datasets and tasks. We evaluate ReaLogic-FT on the verifier-checkable versions of ProofWriter and ProntoQA constructed in Section 3. The assertion-level test sets contain 2,338 ProofWriter and 655 ProntoQA NL–OWL pairs. Structured-proof evaluation uses 2,500 ProofWriter problems, balanced across depths 0, 1, 2, 3, and 5, and all 2,936 verifier-checkable ProntoQA test problems spanning one to five hops. The same instances are used for SFT and RL comparisons.

Comparison systems. We compare zero- and few-shot Qwen3-8B, Assertion-Level SFT under the same prompting conditions, Structured Proof SFT without demonstrations, Full ReaLogic-FT (Full), and an ablation removing R_{semantic} (w/o Sem.). RL is initialized from Structured Proof SFT, which serves as the pre-RL baseline.

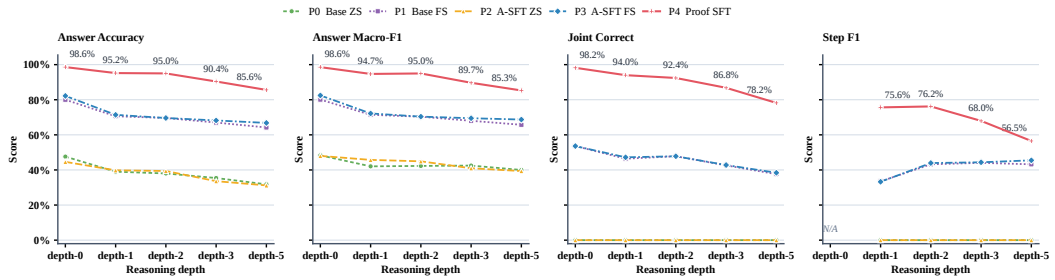

Figure 2: Structured-proof performance on ProofWriter across reasoning depth. Structured Proof SFT is compared with prompting and assertion-level alignment baselines. Step-F1 is undefined for depth-0 instances because they contain no gold inference steps.

Implementation details. We use Qwen3-8B in non-thinking mode with a separate rank-16 LoRA adapter for each dataset pipeline. Assertion-Level SFT uses three epochs and a learning rate of 2×10^{-5}, while Structured Proof SFT uses 1×10^{-5} for two ProofWriter epochs and three ProntoQA epochs. Dr. GRPO uses an effective prompt batch size of 4, four candidates per prompt, and a learning rate of 3×10^{-7}. The reported models use checkpoint 20 for ProofWriter and checkpoint 100 for ProntoQA. Assertion-Level SFT and RL use seeds 42, 43, and 44, whereas Structured Proof SFT results are reported for seed 42. Evaluation uses greedy decoding. Full implementation details are provided in Appendix A.5.

Evaluation metrics. For assertion alignment, we report *Exact Match* (EM) and *Strict Valid* (SV), where SV additionally requires valid OWL syntax, supported $\mathcal{ALC}$ constructs, and correct symbol roles. Structured Proof SFT is evaluated with three-way *Answer Accuracy* (Acc.), *Answer Macro-F1* (M-F1), *Joint Correct* (Joint), and *Step-level F1* (Step-F1); Joint additionally requires a correct query and a structurally valid proof, while Step-F1 compares predicted and reference proof steps. For HermiT-Guided RL, we report Acc., *Grounded Joint* (G-Joint), Step-F1, *Unknown F1* (U-F1), and *Strict Full Semantic Validity* (Full-Sem.). G-Joint adds theory grounding, U-F1 measures recognition of UNKNOWN queries, and Full-Sem. further requires sequential HermiT verification and verified answer support. Significance is assessed with paired bootstrap tests over test instances (10,000 replicates), resampled hierarchically over instances and training seeds when three seeds are available, with Holm correction within each analysis and dataset.

4.2 PROGRESSIVE SUPERVISED FINE-TUNING

Assertion-level formalization. Table 1 reports the formal-language alignment acquired in the first stage. Few-shot demonstrations substantially improve the base model, especially on ProntoQA, but still produce malformed or non-canonical predictions. Assertion-Level SFT reaches 100% EM and SV on both test sets across all three seeds. Paired bootstrap tests over the same evaluation instances confirm significant improvements over both zero-shot and few-shot prompting after Holm correction (all adjusted $p = 0.0008$; Appendix Table 10). Because few-shot prompting already reaches 95–98% EM, this stage serves mainly to align the model with the verifier's interface before it composes complete proofs.

Configuration	EM	SV
ProofWriter		
Base + ZS	83.23	92.56
Base + FS	95.30	97.48
Assertion-Level SFT	**100.00 ± 0.00**	**100.00 ± 0.00**
ProntoQA		
Base + ZS	58.17	85.34
Base + FS	98.32	98.32
Assertion-Level SFT	**100.00 ± 0.00**	**100.00 ± 0.00**

Table 1: Assertion-level NL–OWL results (%). ZS/FS denote zero-/few-shot prompting. SFT results are mean ± SD over seeds 42–44; baselines are single runs. Best results are bolded.

Structured proof generation. Table 2 shows that assertion-level alignment alone does not yield complete proofs: without demonstrations, both the base and assertion-aligned models obtain zero Joint and Step-F1. Structured Proof SFT raises Joint by 43.96 points on ProofWriter and 52.28 points on ProntoQA over the strongest few-shot baselines, while reaching answer accuracies above 92% on

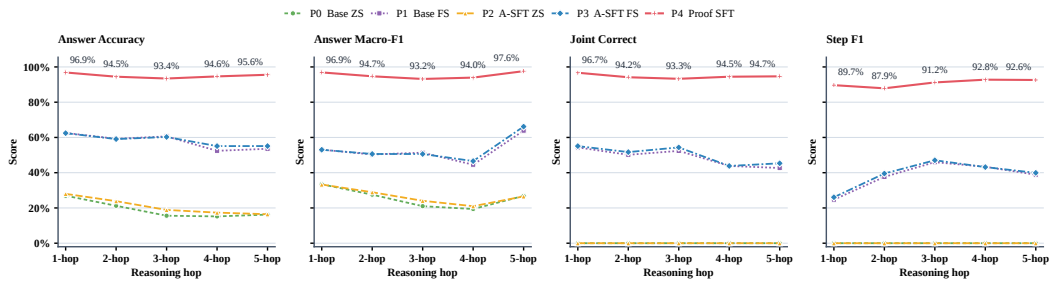

Figure 3: Structured-proof performance on ProntoQA across hop count, using the same systems and metrics as Figure 2.

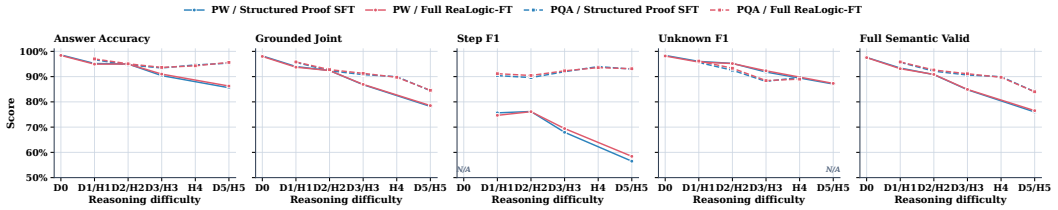

Figure 4: Structured Proof SFT and Full ReaLogic-FT performance across reasoning difficulty. Colors distinguish training stages, while line styles distinguish ProofWriter (D0–D5) and ProntoQA (H1–H5). ProofWriter has no depth-4 split; Step-F1 is undefined at depth 0, and U-F1 is undefined for ProntoQA at hop 5.

both datasets. These gains are significant after Holm correction; full bootstrap results appear in Appendix Table 12, while difficulty-stratified analyses are shown in Figures 2 and 3.

The breakdown shows different difficulty patterns across the two benchmarks. On ProofWriter, performance decreases as depth increases, indicating that composing longer derivations remains challenging. On ProntoQA, performance is more stable across hop counts, suggesting that the structured proof format is easier to maintain when the underlying reasoning pattern is more regular. On both datasets, the largest gains appear after structured proof supervision rather than assertion-level alignment alone. In a controlled initialization ablation under the original 1,024-token protocol (Appendix Table 14), assertion-aligned initialization improves all four metrics on ProntoQA but has mixed effects on ProofWriter. This comparison is only within the shared protocol; all main results use the final 768-token evaluation. The larger ProofWriter proof-supervision set may reduce the added benefit of assertion-level initialization, though formalization difficulty is a confound.

Configuration	Acc.	M-F1	Joint	Step-F1
ProofWriter				
Base + ZS	38.36	43.36	0.00	0.00
Base + FS	70.28	71.15	45.60	40.58
Assertion-Level SFT + ZS	37.72	44.01	0.00	0.00
Assertion-Level SFT + FS	71.64	72.67	45.96	41.88
Structured Proof SFT	**92.96**	**92.81**	**89.92**	**65.37**
ProntoQA				
Base + ZS	15.70	20.65	0.00	0.00
Base + FS	54.36	47.54	37.26	39.85
Assertion-Level SFT + ZS	17.10	21.83	0.00	0.00
Assertion-Level SFT + FS	55.48	48.27	38.83	40.90
Structured Proof SFT	**94.96**	**94.91**	**91.11**	**92.37**

Table 2: Structured proof results (%). Prompting baselines are deterministic single runs, and Structured Proof SFT is a single run with seed 42. Best results are shown in bold.

4.3 Verifier-Guided Reinforcement Learning

Overall performance and reward ablation. Table 3 compares Full ReaLogic-FT with Structured Proof SFT and the semantic-reward ablation. On ProofWriter, Full matches SFT on G-Joint and improves the other four metrics, with the largest gain in Step-F1 (65.37 to 66.36). On ProntoQA, Full improves all five reported means, with the largest gain in U-F1 (91.15 to 91.47). The improvements are modest, reflecting the strong Structured Proof SFT initialization and a small RL budget of 80

Config.	Acc.	G-Joint	Step-F1	U-F1	Full-Sem.
ProofWriter					
SFT	92.96	**89.92**	65.37	93.46	88.52
w/o Sem.	92.96 ± 0.22	89.71 ± 0.08	65.47 ± 1.02	93.42 ± 0.18	88.36 ± 0.14
Full	**93.12 ± 0.22**	**89.92 ± 0.21**	**66.36 ± 1.02**	**93.55 ± 0.21**	**88.60 ± 0.24**
ProntoQA					
SFT	94.96	91.11	92.37	91.15	90.97
w/o Sem.	94.81 ± 0.23	91.06 ± 0.23	92.27 ± 0.12	91.03 ± 0.37	90.89 ± 0.23
Full	**95.15 ± 0.16**	**91.33 ± 0.26**	**92.56 ± 0.06**	**91.47 ± 0.29**	**91.17 ± 0.28**

Table 3: HermiT-guided RL and semantic-reward ablation (%). SFT is a single run; RL results are mean ± s.d. over seeds 42–44.

and 400 training prompts on ProofWriter and ProntoQA, and none is statistically significant after Holm correction (Appendix Table 13).

In the ablation comparison, Full also exceeds w/o Sem. in mean performance on every metric across both datasets, with the largest reductions from removing R_{semantic} occurring in ProofWriter Step-F1 and ProntoQA U-F1. The accompanying decreases in G-Joint and Full-Sem. point in the same direction but likewise fall within the bootstrap confidence intervals. These aggregate results, however, average over instances with substantially different reasoning depths.

Performance across reasoning difficulty. Figure 4 shows that the effect of verifier-guided RL depends on reasoning difficulty. On ProofWriter, gains become more favorable at greater depths: at depth 5, Full improves Step-F1 by 1.92 points and Full-Sem. by 0.53 points. On ProntoQA, improvements are concentrated between one and three hops, with the largest two-hop gains in Step-F1 (0.83 points) and U-F1 (0.78 points); performance is slightly lower at four hops and changes modestly at five. Thus, verifier-guided RL has its largest mean effect on deeper ProofWriter instances but does not increase monotonically with ProntoQA hop count.

Residual errors. Accuracy still overstates proof quality after RL. The seed-42 Full models produce 232 (ProofWriter) and 231 (ProntoQA) hard outputs, which disagree with HermiT's answer or reach a correct answer without satisfying Full-Sem.; 80 and 113 of them pair a correct answer with an invalid proof. Answer errors concentrate at the open-world boundary: confusing UNKNOWN with a definite verdict accounts for 147 of 152 and 108 of 118 answer disagreements (Appendix Table 17).

5 CONCLUSION AND FUTURE WORK

Our results quantify the answer–proof gap: prompted models answer many problems correctly while producing no verifiable proof at all. Supervising derivations rather than answers closes much of that gap, and the verifier stage changes how proof quality is assessed: HermiT evaluates grounded inference steps incrementally without requiring the generated proof to imitate a single reference derivation. Across both datasets, the complete ReaLogic-FT reward achieves higher mean performance than its semantic-reward ablation on all reported metrics, although these gains are modest and do not reach statistical significance under the current training and evaluation budget. Near-ceiling performance on single-statement formalization in these controlled benchmarks suggests that the remaining challenge lies primarily in composing valid reasoning chains. This points toward verifier-guided search over a space of valid derivations that a single reference cannot enumerate.

Limitations. Both benchmarks are synthetic with controlled vocabularies, which enables exact verification but limits linguistic variety. HermiT certifies proofs against the given formalization, not the faithfulness of that formalization to the source text. We do not compare against inference-time solver pipelines such as Logic-LM Pan et al. (2023) and LINC Olausson et al. (2023), whose solvers return verdicts rather than scorable proofs, or against self-training methods such as STaR Zelikman et al. (2022); Appendix A.7 discusses further limitations.

AI USE STATEMENT

During the preparation of this paper, the authors used generative AI tools for literature discovery, assistance with code implementation and debugging, inspection of experimental outputs, and improvements to clarity and readability. AI-assisted references, code, analyses, and textual revisions were reviewed and verified by the authors through source checking, code testing, and examination of experimental results. Generative AI was not used to generate or clean datasets, define the formal framework, formulate mathematical or logical claims, construct proofs, or independently determine the methodology or conclusions. All content was reviewed and finalized by the authors, who take full responsibility for the accuracy, originality, and integrity of this work.

ETHICS STATEMENT

This work develops fine-tuning methods for improving the logical reasoning and proof-generation abilities of language models. Although more reliable reasoning may benefit model transparency and verification, strong benchmark performance should not be interpreted as guaranteeing correctness in real-world or high-stakes applications. ReaLogic-FT is intended to improve the verifiability of model-generated reasoning rather than enable autonomous decision-making. Our experiments use synthetic or publicly available logical-reasoning benchmarks and do not involve human participants or personal, private, or sensitive information. Nevertheless, the resulting models may inherit biases and other limitations from their pretrained base models. We therefore recommend independent verification and appropriate human oversight when applying these methods beyond controlled reasoning tasks.

REPRODUCIBILITY STATEMENT

Code, training configurations, model predictions, and the result files underlying the reported numbers will be released at `https://anonymous.4open.science/r/verifier-guided-reasoning-2BA0`. The release includes the Assertion-Level SFT, Structured Proof SFT, and HermiT-guided RL pipelines, together with the OWL API–HermiT backend and scripts used to produce the tables and figures. We also provide the data construction and filtering scripts needed to reproduce the verifier-checkable versions of ProofWriter and ProntoQA from the original datasets. The structured-proof interface, sequential accepted-state verification procedure, and complete reward definition are specified in the main paper and appendix. Prompt templates, curriculum design, evaluation metrics, and statistical tests are also documented in the appendix. The reported checkpoints, decoding settings, hyperparameters, random seeds, software versions, hardware environment, and measured computational costs are provided in Appendix A.5. Limitations concerning supported logic, reasoner cost, and evaluation scale are discussed in Appendix A.7.

REFERENCES

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 2 edition, 2006.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (eds.). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (eds.). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2 edition, 2007. doi: 10.1017/CBO9780511711787.

Luoxin Chen, Yichi Zhou, and Huishuai Zhang. Learning to Generate Formally Verifiable Step-by-Step Logic Reasoning via Structured Formal Intermediaries. *CoRR*, abs/2603.29500, 2026.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. Transformers as Soft Reasoners over Language. In *Proc. IJCAI'20*, pp. 3882–3890. ijcai.org, 2020.

Pavlos S. Efraimidis and Paul G. Spirakis. Weighted random sampling with a reservoir. *Inf. Process. Lett.*, 97(5):181–185, 2006.

Yu Feng, Nathaniel Weir, Kaj Bostrom, Sam Bayless, Darion Cassel, Sapana Chaudhary, Benjamin Kiesl-Reiter, and Huzefa Rangwala. VeriCoT: Neuro-symbolic Chain-of-Thought Validation via Logical Consistency Checks. *CoRR*, abs/2511.04662, 2025.

Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. HermiT: An OWL 2 Reasoner. *J. Autom. Reason.*, 53(3):245–269, 2014.

Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer (Second Edition). W3C recommendation, World Wide Web Consortium, 2012.

Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

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. In *Proc. ICLR'22*. OpenReview.net, 2022.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamile Lukosiute, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, Saurav Kadavath, Shannon Yang, Thomas Henighan, Timothy Maxwell, Timothy Telleen-Lawton, Tristan Hume, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Measuring Faithfulness in Chain-of-Thought Reasoning. *CoRR*, abs/2307.13702, 2023.

Joshua Ong Jun Leang, Giwon Hong, Wenda Li, and Shay B. Cohen. Theorem prover as a judge for synthetic data generation. In *Proc. ACL'25*, pp. 29941–29977. Association for Computational Linguistics, 2025.

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. *CoRR*, abs/2305.20050, 2023.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-Like Training: A Critical Perspective. *CoRR*, abs/2503.20783, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *Proc. ICLR'19*. OpenReview.net, 2019.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful Chain-of-Thought Reasoning. In *Proc. IJCNLP-AACL'23*, pp. 305–329. ACL, 2023.

Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition). W3C recommendation, World Wide Web Consortium, 2012.

Theo Olausson, Alex Gu, Benjamin Lipkin, Cedegao E. Zhang, Armando Solar-Lezama, Joshua B. Tenenbaum, and Roger Levy. LINC: A Neurosymbolic Approach for Logical Reasoning by Combining Language Models with First-Order Logic Provers. In *Proc. EMNLP'23*, pp. 5153–5176. ACL, 2023.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning. In *Findings of EMNLP'23*, volume EMNLP 2023 of *Findings of ACL*, pp. 3806–3824. ACL, 2023.

Debjit Paul, Robert West, Antoine Bosselut, and Boi Faltings. Making Reasoning Matter: Measuring and Improving Faithfulness of Chain-of-Thought Reasoning. In *Findings of EMNLP'24*, volume EMNLP 2024 of *Findings of ACL*, pp. 15012–15032. ACL, 2024.

Yu Pei, Yongping Du, and Xingnan Jin. FoVer: First-Order Logic Verification for Natural Language Reasoning. *Trans. Assoc. Comput. Linguistics*, 13:1340–1359, 2025.

Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. PRover: Proof Generation for Interpretable Reasoning over Rules. In *Proc. EMNLP'20*, pp. 122–136. ACL, 2020.

Swarnadeep Saha, Prateek Yadav, and Mohit Bansal. multiPRover: Generating Multiple Proofs for Improved Interpretability in Rule Reasoning. In *Proc. NAACL-HLT'21*, pp. 3662–3677. ACL, 2021.

Abulhair Saparov and He He. Language Models Are Greedy Reasoners: A Systematic Formal Analysis of Chain-of-Thought. In *Proc. ICLR'23*. OpenReview.net, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *CoRR*, abs/2402.03300, 2024.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. ProofWriter: Generating Implications, Proofs, and Abductive Statements over Natural Language. In *Findings of ACL'21*, volume ACL-IJCNLP 2021 of *Findings of ACL*, pp. 3621–3634. ACL, 2021.

Qwen Team. Qwen3 Technical Report. *CoRR*, abs/2505.09388, 2025.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language Models Don't Always Say What They Think: Unfaithful Explanations in Chain-of-Thought Prompting. In *Proc. NeurIPS'23*, 2023.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In *Proc. ACL'24*, pp. 9426–9439. ACL, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *Proc. ICLR'23*. OpenReview.net, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Proc. NeurIPS'22*, 2022.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and Monte-Carlo tree search. In *Proc. ICLR'25*. OpenReview.net, 2025.

Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. Faithful Logical Reasoning via Symbolic Chain-of-Thought. In *Proc. ACL'24*, pp. 13326–13365. ACL, 2024.

Jundong Xu, Hao Fei, Huichi Zhou, Xin Quan, Qijun Huang, Shengqiong Wu, William Yang Wang, Mong-Li Lee, and Wynne Hsu. LogicReward: Incentivizing LLM Reasoning via Step-Wise Logical Supervision. In *Proc. ICLR'26*, 2026.

Hui Yang, Jiaoyan Chen, and Uli Sattler. Large Language Model for OWL Proofs. In *Proc. WWW'26*, pp. 3952–3963. ACM, 2026a.

Kaiyu Yang, Jia Deng, and Danqi Chen. Generating Natural Language Proofs with Verifier-Guided Search. In *Proc. EMNLP'22*, pp. 89–105. ACL, 2022.

Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. Harnessing the Power of Large Language Models for Natural Language to First-Order Logic Translation. In *Proc. ACL'24*, pp. 6942–6959. ACL, 2024.

Zhuojie Yang, Wentao Wan, and Keze Wang. ORACLE: optimizing reasoning abilities of large language models via constraint-led synthetic data elicitation. In *Proc. AAAI'26*, pp. 34378–34386. AAAI Press, 2026b.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. SatLM: Satisfiability-Aided Language Models Using Declarative Prompting. In *Proc. NeurIPS'23*, 2023.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping Reasoning With Reasoning. In *Proc. NeurIPS'22*, 2022.

A Appendix

A.1 Dataset Construction and Processing

This section describes how we construct verifier-checkable training and evaluation data from ProofWriter and ProntoQA. The two source benchmarks differ in their schemas, proof annotations, difficulty definitions, and answer spaces. We therefore normalize them into a common representation, project their logical content into the supported $\mathcal{ALC}$ fragment, validate every formal expression, reconstruct executable proof chains, and derive task-specific views for the three training stages of ReaLogic-FT.

A.1.1 Source Benchmarks and Task Formulation

ProofWriter. ProofWriter is a synthetic rule-based reasoning benchmark containing natural-language facts, rules, queries, answer labels, and supporting proofs Tafjord et al. (2021). Its instances are grouped by reasoning depth, including depth-0, depth-1, depth-2, depth-3, and depth-5. ProofWriter already provides True, False, and open-world Unknown instances. However, not every source rule is expressible in the strict $\mathcal{ALC}$ profile adopted by ReaLogic-FT. We therefore audit each instance and construct a self-contained $\mathcal{ALC}$ projection whenever the retained facts, rules, query, and proof preserve the original answer. Instances for which no such projection can be constructed are discarded.

ProntoQA. ProntoQA contains synthetic ontology-style reasoning problems organized by hop count, from 1-hop to 5-hop Saparov & He (2023). Each problem contains natural-language concept assertions, concept-inclusion rules, a query, and a supporting derivation. We map its source answer labels to True and False and additionally construct Unknown instances under open-world semantics, as described in Section A.1.6.

Unified three-way task. For both benchmarks, we represent a reasoning problem by a natural-language theory and query, together with an executable ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$. The TBox $\mathcal{T}$ contains general concept inclusions (GCIs), while the ABox $\mathcal{A}$ contains concept and role assertions about named individuals. Let q denote the formal query assertion and $\bar{q}$ its negated assertion. In the processed data, ProntoQA queries are class assertions, whereas ProofWriter retains both class assertions and positive object-property assertions. We assign the three-way label

$$y(\mathcal{O}, q) = \begin{cases} \textsc{True}, & \mathcal{O} \models q, \\ \textsc{False}, & \mathcal{O} \models \bar{q}, \\ \textsc{Unknown}, & \mathcal{O} \not\models q \ \wedge \ \mathcal{O} \not\models \bar{q}. \end{cases} \tag{7}$$

This definition follows the open-world assumption: failure to derive a query does not by itself imply that the query is false Hitzler et al. (2012). Inconsistent ontologies are rejected because they do not admit a meaningful three-way entailment decision.

A.1.2 Unified Data Representation

All constructed datasets are stored in JSONL format, with one independent JSON object per line. The unified schema preserves the source identity and difficulty group while attaching the formal information required for supervised training, semantic verification, and reproducible evaluation. Table 4 summarizes the principal fields.

Field	Description
`id`	Unique identifier of the processed example.
`source_id`	Identifier of the corresponding example in the source benchmark.
`dataset`	Source benchmark, either `ProofWriter` or `ProntoQA`.
`subset`	Reasoning-difficulty group: ProofWriter depth or ProntoQA hop count.
`split`	Dataset partition: `train`, `dev`, or `test`.
`theory`	Natural-language facts and rules, together with their formal identifiers and OWL translations.
`ontology`	Executable formal theory represented by canonicalized TBox and ABox axioms. The RL view additionally stores a base IRI and a canonical ontology fingerprint.
`ontology_vocabulary`	Admissible classes, object properties, and individuals for the example. It is used to prevent ungrounded symbols in model outputs.
`query`	Natural-language query, canonical OWL assertion q, and negated assertion $\bar{q}$.
`answer`	Gold three-way label in $\{\text{TRUE}, \text{FALSE}, \text{UNKNOWN}\}$.
`proof` / `gold_reasoning` / `gold_reasoning_steps`	Dataset-specific reference derivation containing supporting input facts and/or ordered premise–rule–conclusion steps. It is empty for UNKNOWN instances.
`prompt`	Raw user instruction stored in the dataset. The Qwen chat template is applied at load time before the instruction is presented to the policy.
`completion` / `target`	Canonical assertion for assertion-level SFT, or the structured proof, query assertion, and answer for structured proof SFT.
`statement_family`	Logical family of an assertion-level example, such as a positive fact, negative fact, concept inclusion, or rule with a negated consequent.

Table 4: Principal fields in the constructed ReaLogic-FT datasets. Not every field is exposed to the policy at every training stage. In particular, the formal ontology, gold answer, and reference derivation are hidden from the policy during reinforcement learning and are retained only as verifier metadata.

Assertion-level example. Listing 1 shows an abridged assertion-alignment record, where the target is a single canonical OWL axiom rather than a free-form explanation.

Listing 1: Abridged assertion-level NL–OWL record.

```
{
  "dataset": "ProntoQA",
  "split": "train",
  "task": "nl_to_strict_alc_assertion",
  "statement_family": "rule_negative_consequent",
  "input": "Each zumpus is not dull.",
  "target": "SubClassOf(:zumpus ObjectComplementOf(:dull))",
  "ontology_vocabulary": {
    "classes": [":dull", ":zumpus"],
    "object_properties": [],
    "individuals": [":Fae"]
  }
}
```

Structured-proof example. Listing 2 presents an abridged example derived from a ProofWriter instance. The reference proof explicitly records the premises, cited rule, and conclusion of every non-direct inference step.

Listing 2: An abridged structured-proof record from ProofWriter.

```
{
  "dataset": "ProofWriter",
  "subset": "depth-2",
  "theory": {
    "facts": [
      {
        "text": "Erin is rough.",
```

```
        "assertion": "ClassAssertion(:rough :Erin)"
      }
    ],
    "rules": [
      {
        "text": "Rough things are quiet.",
        "assertion": "SubClassOf(:rough :quiet)"
      },
      {
        "text": "If something is quiet then it is not nice.",
        "assertion": "SubClassOf(:quiet ObjectComplementOf(:nice))"
      }
    ]
  },
  "query": {
    "text": "Erin is nice.",
    "assertion": "ClassAssertion(:nice :Erin)",
    "negated_assertion":
      "ClassAssertion(ObjectComplementOf(:nice) :Erin)"
  },
  "answer": "False",
  "proof": [
    {
      "premises": ["ClassAssertion(:rough :Erin)"],
      "rule": "SubClassOf(:rough :quiet)",
      "conclusion": "ClassAssertion(:quiet :Erin)"
    },
    {
      "premises": ["ClassAssertion(:quiet :Erin)"],
      "rule": "SubClassOf(:quiet ObjectComplementOf(:nice))",
      "conclusion":
        "ClassAssertion(ObjectComplementOf(:nice) :Erin)"
    }
  ]
}
```


A.1.3 Assertion-Level NL–OWL Data Construction

The assertion-level dataset teaches the model to translate one natural-language fact, rule, or query into one canonical assertion in OWL Functional-Style Syntax. We extract candidate statements from the training portions of the source benchmarks and preserve their source identifiers, statement types, ontology vocabularies, and formal targets.

Each candidate pair (x, a) consists of a natural-language statement x and its formal assertion a. The supported top-level axiom types are

$$\begin{aligned} &\mathrm{ClassAssertion}(C, a), \\ &\mathrm{ObjectPropertyAssertion}(r, a, b), \\ &\mathrm{SubClassOf}(C, D), \end{aligned} \tag{8}$$

where C and D are class expressions in the selected $\mathcal{ALC}$ profile, r is a role name, and a, b are individuals. The assertions are serialized in OWL Functional-Style Syntax Motik et al. (2012).

Before an example is admitted, its target is parsed, recursively validated, and canonically re-serialized as described in Section A.1.4. We then normalize the natural-language input and deduplicate examples by the pair

$$\big(\mathrm{NormNL}(x), \mathrm{CanonOWL}(a)\big). \tag{9}$$

The retained examples are grouped by logical statement family and stratified into training, development, and test partitions using an 80/10/10 ratio. This statement-level split is constructed only from the source training data and is used to measure formal-language acquisition independently of the complete reasoning benchmarks.

Algorithm 1 AST-Based OWL Assertion Validation

Require: Candidate OWL string s, vocabulary $\mathcal{V}$, whitelist $\mathcal{W}$
Ensure: Canonical assertion a, or REJECT

```
Tokenize s by lexical analysis
Parse the tokens into an AST T by recursive descent
if parsing fails or unconsumed tokens remain then
    return REJECT
end if
if the root type of T is not in W then
    return REJECT
end if
Recursively check constructor arity and argument roles
if a node violates the selected ALC grammar then
    return REJECT
end if
Extract classes, properties, and individuals from T
if any symbol is absent from V then
    return REJECT
end if
a ← CanonicalSerialize(T)
return a
```

A.1.4 AST PARSING AND WHITELIST VALIDATION

String matching alone is insufficient for validating nested OWL expressions. We therefore parse each candidate assertion into an abstract syntax tree (AST) using lexical analysis and recursive descent Aho et al. (2006). For example,

$$\mathrm{SubClassOf}(\,\mathrm{ObjectIntersectionOf}(\mathsf{green}, \mathrm{ObjectComplementOf}(\mathsf{quiet})), \mathsf{furry}) \tag{10}$$

is represented as a `SubClassOf` root whose first child is a nested intersection expression and whose second child is the atomic class `:furry`.

The validator uses separate whitelists for top-level axiom types, class constructors, object properties, and individual positions. It checks:

1. whether the root is an admitted axiom type;
2. whether every constructor has the required arity;
3. whether each child occurs in the correct argument role;
4. whether all classes, object properties, and individuals belong to the example-specific ontology vocabulary;
5. whether every nested class expression belongs to the supported $\mathcal{ALC}$ grammar;
6. whether the parser consumes the complete token sequence without unmatched parentheses or trailing content.

This recursive validation prevents syntactically plausible but structurally invalid expressions, such as using an individual in a class position or supplying an incorrect number of arguments to a constructor.

Algorithm 1 summarizes the procedure. Only accepted trees are canonically re-serialized. The canonical representation is used for deduplication, ontology fingerprints, structural grouping, and equality checks throughout training and evaluation.

A.1.5 STRUCTURED PROOF DATA CONSTRUCTION

The structured-proof view is constructed from complete reasoning problems. For each source instance, we first build a unified formal record containing the natural-language theory, executable ontology, ontology vocabulary, formal query q, negated assertion $\bar{q}$, three-way answer y, and reference derivation when one is available.

A structured proof is represented as

$$\pi = (F_0, s_1, \ldots, s_m), \qquad s_i = (P_i, r_i, c_i), \tag{11}$$

where $F_0 \subseteq \mathcal{A}$ contains optional input facts explicitly recorded by the proof, either as direct support or as premises for subsequent inference steps. P_i is the set of premises cited by step i, r_i is a rule from the TBox, and c_i is the step conclusion. We reconstruct reference steps in topological order so that

$$P_i \subseteq \mathcal{A} \cup \{c_1, \ldots, c_{i-1}\}. \tag{12}$$

Thus, every premise is either an original input assertion or a conclusion derived by an earlier step.

For a directly supported query, F_0 contains q or $\bar{q}$ and the proof contains no non-direct step. For a non-direct TRUE example, the final derivation supports q; for a non-direct FALSE example, it supports $\bar{q}$. An UNKNOWN example has an empty reasoning block because neither outcome is derivable.

Strict-$\mathcal{ALC}$ projection of ProofWriter. Some ProofWriter rules fall outside the selected logical profile. Rather than removing such rules independently, we audit the facts, rules, query, and proof jointly. Unsupported rules are removed only if the remaining content forms a self-contained subproblem whose reference derivation is complete and whose three-way answer agrees with the original instance. Otherwise, the entire example is rejected. The retained proof is then rebuilt from its premise dependencies and revalidated against the projected ontology. This prevents an unsupported rule from being deleted while leaving behind an incomplete proof or a changed answer.

The resulting supervised target contains three components:

$$z = (\pi, \hat{q}, \hat{y}), \tag{13}$$

where π is the reference structured OWL proof, q is the canonicalized query assertion, and $y \in \{\text{TRUE}, \text{FALSE}, \text{UNKNOWN}\}$ is the gold three-way label.

A.1.6 CONSTRUCTION OF UNKNOWN INSTANCES

ProofWriter already provides open-world UNKNOWN examples. ProntoQA, in contrast, requires additional construction to support the same three-way task. We generate these examples deterministically from the ontology structure rather than asking a language model to create them.

For each ProntoQA ontology, we construct a directed concept graph $G = (V, E)$, where an edge $(C, D) \in E$ represents a concept inclusion $C \sqsubseteq D$. If an individual a is asserted to belong to a source concept C, we select a target concept D for which the graph contains no directed subsumption path supporting $D(a)$. This produces a candidate query $q = D(a)$. Graph non-reachability is used only as a candidate-generation criterion; it is not treated as sufficient evidence of UNKNOWN.

For final validation, we construct the negated assertion $\bar{q} = (\neg D)(a)$ and retain the example only if the ontology is consistent and HermiT verifies

$$\mathcal{O} \not\models q \qquad \text{and} \qquad \mathcal{O} \not\models \bar{q}. \tag{14}$$

This second-stage semantic check removes candidates whose entailments are not captured by simple graph reachability. Duplicate queries and queries already present in the source ontology are also removed.

Algorithm 2 summarizes the construction. The current processed ProntoQA data contain no 5-hop/UNKNOWN stratum; the curriculum construction therefore assigns this stratum a zero quota (Section A.1.8).

A.1.7 DEDUPLICATION, SPLITTING, AND LEAKAGE PREVENTION

We apply leakage controls at both the statement and ontology levels.

Assertion-level deduplication. Assertion-alignment pairs are globally deduplicated using Equation 9. Whitespace-normalized inputs paired with identical canonical OWL targets therefore form one example. Inputs associated with multiple distinct targets are treated as ambiguous and excluded. The retained pairs are stratified by statement family before the 80/10/10 split so that less frequent logical constructions are represented in all partitions whenever their counts permit.

Algorithm 2 Construction of ProntoQA Unknown Instances

Require: Consistent ontology $\mathcal{O}$ and concept graph G
Ensure: Set $\mathcal{U}$ of verified Unknown examples
$\mathcal{U} \leftarrow \emptyset$
for each asserted source pair (C, a) **do**
  for each candidate target concept D **do**
    if $D = C$ or D is reachable from C in G **then**
      continue
    end if
    $q \leftarrow D(a)$
    $\bar{q} \leftarrow \neg D(a)$
    if $\mathcal{O} \not\models q$ and $\mathcal{O} \not\models \bar{q}$ **then**
      Add $(\mathcal{O}, q, \bar{q}, \text{UNKNOWN})$ to $\mathcal{U}$
    end if
  end for
end for
Remove duplicate and source-overlapping queries
return $\mathcal{U}$

Ontology-level grouping. For structured proof supervision and evaluation, we preserve source identifiers and canonicalize the corresponding ABox and TBox. In particular, ProofWriter examples sharing the same canonicalized ontology are assigned to a single partition. Consequently, different queries over the same theory cannot be distributed across training and evaluation. This is stricter than query-level deduplication because it prevents the model from observing the same logical world with a different target query during training.

Projection consistency. A ProofWriter projection is retained only if its facts, rules, query, proof, and answer remain self-contained after unsupported constructs are removed. We reject examples with missing proof dependencies, invalid symbols, unparseable OWL, inconsistent ontologies, unsupported query forms, or labels that are not preserved according to the retained reference proof. Label preservation is thus established from the reference proof rather than by HermiT; the HermiT audit in Section A.4.1 identifies 37 ProofWriter test instances whose benchmark label disagrees with the projected ontology, which are retained with semantic credit disabled.

Policy–verifier separation. The structured SFT view exposes a dataset-specific prompt together with its reference completion. The RL view reuses the corresponding prompt unchanged, including the allowed ontology vocabulary, while withholding the reference completion from the policy. The gold answer, executable ontology, canonical formal query and its negated assertion, and reference proof remain hidden verifier metadata. In particular, the reference proof is neither presented to the policy nor used for string-overlap reward. This separation prevents the RL stage from reducing to another form of reference-proof imitation.

A.1.8 VERIFIER-GUIDED RL CURRICULUM CONSTRUCTION

The raw RL views are derived only from the training partitions of the structured-proof data. Each record retains the natural-language prompt together with its formal ontology, query, answer, and reference derivation for offline construction and auditing. The policy loader, however, removes the reference derivation and all verifier metadata before optimization, exposing only the prompt to the model.

We construct fixed GRPO curriculum pools by stratifying the raw RL records jointly by reasoning difficulty and three-way answer label. ProofWriter is stratified by reasoning depth, whereas ProntoQA is stratified by hop count. Table 5 reports the resulting quotas for the 10,000-instance ProofWriter pool and the 5,000-instance ProntoQA pool.

Within each stratum, we apply weighted reservoir sampling without replacement Efraimidis & Spirakis (2006). For candidate i, let m_i be its number of gold inference steps, a_i the number of theory

Dataset	Difficulty	True	False	Unknown
ProofWriter	depth-0	60	60	80
	depth-1	90	90	120
	depth-2	450	450	600
	depth-3	900	900	1,200
	depth-5	1,500	1,500	2,000
ProntoQA	1-hop	42	41	167
	2-hop	83	84	333
	3-hop	167	166	667
	4-hop	208	209	833
	5-hop	1,000	1,000	0

Table 5: Difficulty–label quotas of the fixed GRPO curriculum pools. ProntoQA assigns zero quota to the absent 5-hop/UNKNOWN stratum.

axioms, and d_i indicate a directly supported non-Unknown query. We define the capped theory-complexity term and sampling weight as

$$c_i = \frac{\min(a_i, 30)}{15}, \qquad w_i = \begin{cases} 1 + c_i, & y_i = \text{UNKNOWN}, \\ 1 + c_i + 4m_i + 2m_i^2 + 2(1 - d_i), & \text{otherwise}. \end{cases} \tag{15}$$

Thus, entailed and contradicted instances with longer proofs receive larger sampling weights, while Unknown instances are weighted by theory size because they have empty reference proofs. Given $u_i \sim \mathrm{Uniform}(0, 1)$, the reservoir priority is

$$\kappa_i = \frac{\log u_i}{w_i}, \tag{16}$$

and the highest-priority items are retained up to the quota of each stratum. Sampling uses seed 42, after which the selected records are deterministically shuffled. Duplicate identifiers are rejected, and every selected record is revalidated for schema consistency, supported OWL syntax, and ontology-symbol roles. The curriculum manifest records the source and output SHA-256 hashes, stratum quotas, seed, and validation status.

A.1.9 DATASET STATISTICS

The construction pipeline produces four task-specific views:

1. *Unified Formal Records*, which preserve the natural-language theory together with its executable ontology, query, answer, and derivation;
2. *Assertion-Alignment Pairs*, used for statement-level NL–OWL supervision;
3. *Structured Proof Demonstrations*, used for complete proof-generation supervision;
4. *Verifier-Guided RL Instances*, which expose only the prompt to the policy and retain the formal record as hidden verifier metadata.

Table 6 reports the training views supplied to the three post-training stages. The assertion-alignment training sets contain 18,637 ProofWriter pairs and 5,216 ProntoQA pairs. After ontology-level leakage control, the structured-proof training sets contain 253,042 and 23,496 instances, respectively. Removing the reference completions produces raw RL views with the same training cardinalities. The procedure in Section A.1.8 then selects duplicate-free curriculum pools of 10,000 ProofWriter instances and 5,000 ProntoQA instances. These pool sizes denote the available RL training data rather than the number of policy-update steps.

For development and assertion-level evaluation, the ProofWriter assertion-alignment view contains 2,326 development and 2,338 test pairs, while the corresponding ProntoQA view contains 651 and

Dataset	Assertion SFT	Proof SFT	GRPO pool
ProofWriter	18,637	253,042	10,000
ProntoQA	5,216	23,496	5,000

Table 6: Numbers of training instances supplied to the three post-training stages. The GRPO column reports the size of the difficulty-aware curriculum pool.

Difficulty	ProofWriter	ProntoQA
0 / 1-hop	500	668
1 / 2-hop	500	650
2 / 3-hop	500	609
3 / 4-hop	500	559
5 / 5-hop	500	450
Total	2,500	2,936

Table 7: Difficulty composition of the final held-out reasoning evaluation sets.

655. The structured-proof development sets contain 36,815 ProofWriter instances and 2,923 ProntoQA instances. Final reasoning performance is measured on the fixed held-out sets summarized in Table 7, rather than on the full 72,581-instance ProofWriter structured-proof test view.

The ProofWriter reasoning evaluation set contains 500 instances from each of depth-0, depth-1, depth-2, depth-3, and depth-5. The ProntoQA evaluation set contains 668, 650, 609, 559, and 450 instances from 1-hop through 5-hop, respectively, for a total of 2,936 instances.

In Table 7, the first numeric column uses ProofWriter reasoning depth and the second uses ProntoQA hop count. The rows are aligned only to present the two distributions compactly and do not imply that the two difficulty definitions are equivalent. These labels are retained in the evaluation records for stratified analysis; the corresponding depth and hop fields in the training records are used for curriculum construction.

A.2 Formal Fragment and Symbolic Verification

Each reasoning instance is represented by an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, a query assertion q, and its negated assertion $\bar{q}$. The underlying logical vocabulary consists of a set of concept names N_C, role names N_R, and individuals N_I. Concept names denote sets of individuals, role names denote binary relations between individuals, and individual names denote particular domain elements. Following the standard definition of $\mathcal{ALC}$ Baader et al. (2007), the supported concepts are generated by

$$C, D ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists r.C \mid \forall r.C. \tag{17}$$

where $A \in N_C$ and $r \in N_R$. The constructors $\neg$, $\sqcap$, $\sqcup$, $\exists$, and $\forall$ respectively denote concept complement, conjunction, disjunction, existential restriction, and universal restriction.

The TBox $\mathcal{T}$ contains general concept inclusions (GCIs) of the form $C \sqsubseteq D$, which encode the rule component of an instance. The ABox $\mathcal{A}$ contains assertions about named individuals, including concept assertions $C(a)$ and role assertions $r(a, b)$, where $a, b \in N_I$. In the OWL serialization used by ReaLogic-FT, these forms principally correspond to `SubClassOf`, `ClassAssertion`, and `ObjectPropertyAssertion`, respectively.

For a class query $q = C(a)$, its negated assertion is $\bar{q} = (\neg C)(a)$. In OWL Functional-Style Syntax, these assertions are represented as `ClassAssertion(C:a)` and `ClassAssertion(ObjectComplementOf(C):a)`. ProntoQA contains only this class-query form. ProofWriter additionally retains positive object-property queries $q = r(a, b)$, serialized as `ObjectPropertyAssertion(:r:a:b)`. For verifier-side three-way checking, the negated assertion of such a query is constructed as `NegativeObjectPropertyAssertion(:r:a :b)`. This negative object-property assertion is a valid OWL 2 axiom used as hidden verifier metadata, but it lies outside the selected model-facing strict-$\mathcal{ALC}$ output profile. Under the open-world assumption, the three-way answer is defined by Eq. equation 7.

This definition assumes that $\mathcal{O}$ is consistent. In particular, failure to derive q does not imply $\bar{q}$; an instance is labeled UNKNOWN only when neither assertion is entailed Hitzler et al. (2012).

A.2.1 OWL FUNCTIONAL-STYLE SYNTAX

ReaLogic-FT serializes logical assertions using OWL Functional-Style Syntax Motik et al. (2012). An expression begins with the name of an OWL axiom or class constructor, followed by its arguments in parentheses. Arguments may be atomic classes, roles, individuals, or recursively nested class expressions. For example:

Listing 3: Examples of OWL Functional-Style assertions and concept-inclusion axioms.

```
ClassAssertion(:Student :Alice)

SubClassOf(
  :Student
  :Person
)

SubClassOf(
  :Researcher
  ObjectSomeValuesFrom(:worksOn :Project)
)

SubClassOf(
  ObjectIntersectionOf(
    :green
    ObjectComplementOf(:quiet)
  )
  :furry
)
```

The first expression states that the individual `:Alice` is asserted to belong to the class `:Student`. The second encodes the subclass axiom `:Student` $\sqsubseteq$ `:Person`. The third corresponds to the existential restriction `:Researcher` $\sqsubseteq \exists$`:worksOn.:Project`, stating that every researcher works on at least one project. The final expression illustrates the recursive composition of class conjunction and complement in the supported $\mathcal{ALC}$ fragment.

The validator admits the full constructor set shown in Equation 17; however, the retained benchmark instances instantiate only the task-specific subset present in ProofWriter and ProntoQA, primarily concept names, complement, conjunction, and positive role assertions.

The representation follows the structural conventions enforced during assertion-level data construction. Positive and negative class facts are written as `ClassAssertion(:Class :Individual)` and `ClassAssertion(ObjectComplementOf(:Class) :Individual)`. Positive role facts use `ObjectPropertyAssertion(:Role :Subject :Object)`, while concept-inclusion rules use `SubClassOf(Antecedent Consequent)`. Multiple antecedent conditions are grouped using `ObjectIntersectionOf`, and negative conditions or consequents are represented using `ObjectComplementOf`. The prompt asks the model to translate one natural-language statement using only the supplied ontology vocabulary. The supervised target and strict output parser further require the completion to contain one canonical OWL Functional-Style expression without additional text.

The nested functional structure admits a direct abstract-syntax-tree representation: each axiom or constructor becomes an internal node, and its classes, roles, individuals, or nested expressions become ordered children. This property supports recursive parsing, constructor-arity checking, symbol-role validation, vocabulary grounding, and canonical re-serialization. It also allows the resulting expressions to be transferred to the OWL API and HermiT without converting free-form reasoning text into a second intermediate notation Horridge & Bechhofer (2011); Glimm et al. (2014). ReaLogic-FT uses only the task-specific subset described above rather than the full expressivity of OWL 2.

A.2.2 HERMIT REASONING PRINCIPLES

HermiT is an OWL 2 DL reasoner based on hypertableau reasoning Glimm et al. (2014). At a high level, it transforms ontology axioms into an internal representation and attempts to construct a model satisfying them. If the construction produces an unavoidable clash, the tested set of axioms is inconsistent. HermiT determines whether the axioms are satisfiable, equivalently, whether they admit a model; an unavoidable clash establishes inconsistency. HermiT supports ontology-consistency checking, concept satisfiability, instance checking, classification, and entailment queries.

The semantic verifier primarily uses two operations. First, for an ontology $\mathcal{O}$ and candidate assertion α, entailment checking determines whether

$$\mathcal{O} \models \alpha. \tag{18}$$

Second, consistency checking determines whether

$$\mathrm{Consistent}(\mathcal{O}) = 1. \tag{19}$$

For three-way classification, HermiT independently tests q and $\bar{q}$. A consistent ontology receives the label TRUE when q is entailed, FALSE when $\bar{q}$ is entailed, and UNKNOWN when neither is entailed. If both are entailed or the ontology is inconsistent, the instance is treated as semantically invalid rather than assigned one of the three labels.

For a generated proof step $s_i = (P_i, r_i, c_i)$, the cited premises P_i and rule r_i are assembled into a local ontology $\mathcal{O}_i^{\mathrm{local}}$, together with the required vocabulary declarations. Let $\bar{c}_i$ denote the negated assertion of the generated conclusion c_i. The semantic validity indicator for the step is

$$H_i = \mathbb{I}\Big[\mathrm{Consistent}\big(\mathcal{O}_i^{\mathrm{local}}\big) \wedge \mathcal{O}_i^{\mathrm{local}} \models c_i \wedge \mathcal{O}_i^{\mathrm{local}} \not\models \bar{c}_i\Big]. \tag{20}$$

This local test ensures that the conclusion follows from the premises and rule explicitly cited by the model, rather than merely following from unrelated axioms elsewhere in the original ontology. Deterministic checks performed before HermiT invocation additionally require every premise to be an input fact or an earlier conclusion, every cited rule to originate from the input TBox, and every symbol to occur in the provided ontology vocabulary.

A.2.3 HERMIT IN REALOGIC-FT

HermiT is used at several points in the ReaLogic-FT pipeline. During data construction, HermiT checks executable-ontology consistency and audits benchmark labels against the formal entailment relation. For ProntoQA, graph-based non-reachability is used only to propose candidate UNKNOWN queries; a candidate is retained only if HermiT confirms

$$\mathrm{Consistent}(\mathcal{O}) = 1, \qquad \mathcal{O} \not\models q, \qquad \mathcal{O} \not\models \bar{q}. \tag{21}$$

This additional test prevents the absence of a graph path from being incorrectly treated as sufficient evidence of semantic unknowability.

For a generated proof $\pi = (F_0, s_1, \ldots, s_m)$, ReaLogic-FT applies deterministic checks covering output tags, query match, OWL syntax and symbol roles, ontology-vocabulary membership, premise grounding, rule provenance, dependency order, and the eight-step limit. HermiT independently determines the semantic answer of the base ontology. A candidate is eligible for semantic reward only when this answer agrees with the benchmark label; otherwise, R_{semantic} is set to zero while the benchmark-based reward is preserved. Conclusion novelty and dataset-specific inference templates are retained only as diagnostic signals.

Proof steps are verified sequentially against an accepted state initialized with the input facts. A step $s_i = (P_i, r_i, c_i)$ is eligible for HermiT verification only if it is structurally well formed and OWL-valid, cites a rule from the input TBox, and uses only input facts or previously accepted conclusions as premises. HermiT then verifies

$$P_i \cup \{r_i\} \models c_i. \tag{22}$$

Only a conclusion that passes both the deterministic checks and HermiT entailment verification is added to the accepted state and may support subsequent steps. The accepted state supports a TRUE prediction when it contains q and a FALSE prediction when it contains $\bar{q}$. For Full-Sem. evaluation,

a valid UNKNOWN output must contain no reasoning items and the base ontology must entail neither q nor $\bar{q}$. Unknown predictions are handled by the shared reward components rather than a separate Unknown-specific term.

During HermiT-guided reinforcement learning, the policy receives only the natural-language problem, ontology vocabulary, and required output format. The executable ontology, canonical query and opposite query, gold answer, and other verifier metadata remain hidden from the policy. Candidate completions are parsed into structured proofs and evaluated using the deterministic and HermiT-based checks above. Together, the deterministic and semantic checks determine step acceptance, accepted-state answer support, and R_{semantic}. The reference proof is neither supplied to HermiT as a reward target nor used for reward-time string matching, allowing an alternative derivation to receive positive feedback when it is grounded, semantically valid, and supports the correct answer.

The Python training and evaluation pipeline communicates with a Java reasoning backend implemented using the OWL API Horridge & Bechhofer (2011). The backend parses or constructs the requested OWL axioms, creates the corresponding ontology, invokes HermiT, and returns structured consistency and entailment results to the Python verifier. Natural-language statements are never passed directly to HermiT; all semantic requests operate on the OWL Functional-Style representation. The experiments use HermiT version `1.4.5.519` with OWL API version `5.1.9`.

HermiT is used during dataset validation, reinforcement-learning reward computation, and offline evaluation. It is not required for policy generation at inference time: the trained model directly produces the structured proof, query assertion, and three-way answer. An external reasoner may subsequently audit the generated proof, but this verification is separable from the model's generation procedure.

A.3 PROMPT TEMPLATES AND OUTPUT FORMATS

This section presents the prompt templates and output schemas used across the three post-training stages, together with the prompts used for the assertion-level base-model baselines. ProofWriter and ProntoQA share the Assertion-Level SFT template but use dataset-specific Structured Proof SFT templates. HermiT-guided RL reuses the corresponding structured-proof prompt unchanged. No separately authored system message is used. During SFT, the raw user prompt and reference assistant completion are formatted through the non-thinking Qwen chat template; during RL, the same template is applied to the user prompt with an assistant-generation marker. Model-specific control tokens and chat-role delimiters are omitted below because they are inserted automatically by the tokenizer.

A.3.1 ASSERTION-LEVEL SFT PROMPT

Assertion-Level SFT translates one natural-language fact, rule, or query into one OWL Functional-Style axiom. ProofWriter and ProntoQA use the same prompt template. Each instance explicitly supplies the admissible classes, object properties, and individuals, followed by the natural-language statement to be translated. The prompt itself is deliberately concise; the requirement of producing one canonical OWL expression is additionally enforced by the supervised target and strict output parser.

User prompt. Listing 4 presents the exact Assertion-Level SFT prompt template. Bracketed uppercase expressions denote fields populated during data construction.

Listing 4: User prompt template for assertion-level SFT.

```
Translate the following natural language statement into
OWL Functional-Style Syntax.

Use only the ontology vocabulary listed below.

Classes:
[ALLOWED_CLASSES]

Object properties:
[ALLOWED_OBJECT_PROPERTIES]
```

```
Individuals:
[ALLOWED_INDIVIDUALS]

Statement:
[NATURAL_LANGUAGE_STATEMENT]

Assertion:
```

Assistant completion. The completion contains only the canonical OWL expression. Listing 5 shows an example where the natural-language rule "Each zumpus is not dull" is paired with its corresponding OWL representation.

Listing 5: Example assistant completion for assertion-level SFT.

```
SubClassOf(:zumpus ObjectComplementOf(:dull))
```

The assistant completion does not repeat the vocabulary, statement, task label, or any natural-language explanation. This narrow interface isolates NL–OWL alignment from logical inference and concentrates completion-only supervision on the target formal expression.

A.3.2 Assertion-Level Base-Model Baseline Prompts

For the assertion-level base-model evaluation, ProofWriter and ProntoQA use the same matched prompt templates. The zero-shot and few-shot conditions share the task instructions, output constraints, translation rules, and instance-specific suffix. The few-shot condition additionally inserts eleven fixed demonstrations whose symbols are disjoint from the vocabulary of the evaluated instance.

For compact presentation, the exact prompts are divided across the following listings. The zero-shot prompt concatenates Listings 6 and 7. The few-shot prompt inserts Listing 8 between the shared instructions and the instance-specific suffix. The division into listings is only for page layout; the corresponding parts are concatenated in the stored prompt.

Listing 6: Shared assertion-level base-model prompt.

```
You are translating one natural-language statement into OWL Functional-
    Style Syntax.

Task:
Translate only the final statement into exactly one OWL Functional-
    Style axiom or assertion.

Output requirements:
1. Output exactly one OWL Functional-Style expression.
2. Output only the expression. Do not explain.
3. Do not use Markdown, code fences, labels, or quotation marks.
4. Do not perform logical inference. Translate only the given statement
    .
5. For the final statement, use only symbols from the final ontology
    vocabulary.
6. Preserve the order of conditions in the natural-language statement.

Translation rules:
1. A positive class fact uses:
   ClassAssertion(:Class :Individual)

2. A negative class fact uses:
   ClassAssertion(ObjectComplementOf(:Class) :Individual)

3. A positive relation fact uses:
   ObjectPropertyAssertion(:Role :Subject :Object)

4. A negative relation fact uses:
```

```
   NegativeObjectPropertyAssertion(:Role :Subject :Object)

5. A positive atomic rule uses:
   SubClassOf(:Antecedent :Consequent)

6. A rule with a negative consequent uses:
   SubClassOf(:Antecedent ObjectComplementOf(:Consequent))

7. A rule with multiple conditions uses ObjectIntersectionOf
   in the antecedent:
   SubClassOf(
     ObjectIntersectionOf(CONDITION_1 CONDITION_2)
     :Consequent
   )

8. A negative antecedent condition uses ObjectComplementOf:
   ObjectComplementOf(:Class)

9. If a symbol appears in both Classes and Individuals,
   it may be used as a nominal individual inside
   ObjectOneOf(...).
```

Listing 7: Instance-specific suffix shared by the assertion-level zero-shot and few-shot baseline prompts.

```
Now translate the final statement.

Final ontology vocabulary:

Classes:
[ALLOWED_CLASSES]

Object properties:
[ALLOWED_OBJECT_PROPERTIES]

Individuals:
[ALLOWED_INDIVIDUALS]

Final statement:
[NATURAL_LANGUAGE_STATEMENT]

Assertion:
```

Listing 8: Few-shot demonstrations for assertion-level translation.

```
The examples below illustrate syntax and translation patterns only.
Their symbols are demonstration symbols and must not be copied into
the final answer.

Example 1: positive class assertion
Statement:
ExampleAlice is cobalt_example.

Assertion:
ClassAssertion(:cobalt_example :ExampleAlice)

Example 2: negative class assertion
Statement:
ExampleAlice is not quiet_example.

Assertion:
ClassAssertion(
  ObjectComplementOf(:quiet_example)
  :ExampleAlice
)
```

```
Example 3: positive object-property assertion
Statement:
ExampleAlice observes_example ExampleBob.

Assertion:
ObjectPropertyAssertion(
  :observes_example
  :ExampleAlice
  :ExampleBob
)

Example 4: negative object-property assertion
Statement:
ExampleAlice does not observe_example ExampleBob.

Assertion:
NegativeObjectPropertyAssertion(
  :observes_example
  :ExampleAlice
  :ExampleBob
)

Example 5: positive atomic rule
Statement:
Every scarlet_example thing is gentle_example.

Assertion:
SubClassOf(
  :scarlet_example
  :gentle_example
)

Example 6: atomic rule with a negative consequent
Statement:
Every scarlet_example thing is not gentle_example.

Assertion:
SubClassOf(
  :scarlet_example
  ObjectComplementOf(:gentle_example)
)

Example 7: intersection rule
Statement:
If something is scarlet_example and is large_example,
then it is gentle_example.

Assertion:
SubClassOf(
  ObjectIntersectionOf(
    :scarlet_example
    :large_example
  )
  :gentle_example
)

Example 8: intersection rule with a negative antecedent
Statement:
If something is not scarlet_example and is large_example,
then it is gentle_example.

Assertion:
SubClassOf(
  ObjectIntersectionOf(
```

```
    ObjectComplementOf(:scarlet_example)
    :large_example
  )
  :gentle_example
)

Example 9: intersection rule with two negative antecedents
Statement:
If something is not scarlet_example and is not large_example,
then it is gentle_example.

Assertion:
SubClassOf(
  ObjectIntersectionOf(
    ObjectComplementOf(:scarlet_example)
    ObjectComplementOf(:large_example)
  )
  :gentle_example
)

Example 10: intersection rule with a negative consequent
Statement:
If something is scarlet_example and is large_example,
then it is not gentle_example.

Assertion:
SubClassOf(
  ObjectIntersectionOf(
    :scarlet_example
    :large_example
  )
  ObjectComplementOf(:gentle_example)
)

Example 11: intersection rule with a negative antecedent
and a negative consequent
Statement:
If something is not scarlet_example and is large_example,
then it is not gentle_example.

Assertion:
SubClassOf(
  ObjectIntersectionOf(
    ObjectComplementOf(:scarlet_example)
    :large_example
  )
  ObjectComplementOf(:gentle_example)
)
```

A.3.3 STRUCTURED PROOF SFT PROMPT

Structured Proof SFT operates on complete reasoning instances. Both dataset-specific prompts expose an allowed ontology vocabulary, natural-language facts and rules, a natural-language query, and the required structured output. They share the same top-level `reasoning`, `query_assertion`, and `answer` organization, but differ in their proof-body schemas. ProofWriter records supporting input assertions in an `initial_facts` block and provides a separate direct-fact instruction. ProntoQA omits this block and allows source facts to appear directly as step premises.

Every non-direct step contains an index, one or more premises, one cited `SubClassOf` rule, and one class-assertion conclusion. For both datasets, an UNKNOWN prediction requires an empty `reasoning` block. The formal query assertion and three-way answer follow the proof block, and no explanation may appear outside the required tags.

User prompts. Listings 9 and 10 show the ProofWriter and ProntoQA templates, respectively. Bracketed uppercase expressions represent instance-specific content inserted during data construction.

Listing 9: ProofWriter Structured Proof SFT user prompt template.

```
Solve the ProofWriter logical reasoning problem using its strict ALC
    projection and return a structured OWL reasoning chain.

Rules outside the selected ALC profile have already been removed. Use
    only the facts, rules, and ontology vocabulary listed below. Do not
    add explanations outside the required tags.

Classes:
[ALLOWED_CLASSES]

Object properties:
[ALLOWED_OBJECT_PROPERTIES]

Individuals:
[ALLOWED_INDIVIDUALS]

Facts:
[NL_FACTS_AS_BULLETS]

Rules:
[NL_RULES_AS_BULLETS]

Query:
[NATURAL_LANGUAGE_QUERY]

Required output format:
<reasoning>
<initial_facts>
ZERO_OR_MORE_OWL_ASSERTIONS
</initial_facts>
<step index="1">
<premises>
ONE_OR_MORE_OWL_ASSERTIONS
</premises>
<rule>
ONE_OWL_SUBCLASS_AXIOM
</rule>
<conclusion>
ONE_OWL_CLASS_ASSERTION
</conclusion>
</step>
</reasoning>
<query_assertion>
ONE_OWL_ASSERTION
</query_assertion>
<answer>True|False|Unknown</answer>

For a direct fact, include it in <initial_facts> and emit no <step>.
    For Unknown, return an empty <reasoning> block.

Response:
```

Listing 10: ProntoQA Structured Proof SFT user prompt template.

```
Solve the ProntoQA logical reasoning problem and return a structured
    OWL reasoning chain.

Use only the ontology vocabulary listed below. Every premise, rule,
    conclusion, and query assertion must use OWL Functional-Style Syntax
    . Do not add explanations outside the required tags.
```

```
Classes:
[ALLOWED_CLASSES]

Object properties:
[ALLOWED_OBJECT_PROPERTIES]

Individuals:
[ALLOWED_INDIVIDUALS]

Facts:
[NL_FACTS_AS_BULLETS]

Rules:
[NL_RULES_AS_BULLETS]

Query:
[NATURAL_LANGUAGE_QUERY]

Required output format:
<reasoning>
<step index="1">
<premises>
ONE_OR_MORE_OWL_ASSERTIONS
</premises>
<rule>
ONE_OWL_SUBCLASS_AXIOM
</rule>
<conclusion>
ONE_OWL_CLASS_ASSERTION
</conclusion>
</step>
</reasoning>
<query_assertion>
ONE_OWL_CLASS_ASSERTION
</query_assertion>
<answer>True|False|Unknown</answer>

For Unknown, return an empty <reasoning> block.

Response:
```

In the actual ProofWriter data, the first two sentences specify that rules outside the selected $\mathcal{ALC}$ profile have already been removed. The two templates expose the same categories of problem information but retain dataset-specific instructions and proof-body schemas. These differences are preserved throughout both supervised proof training and reinforcement learning.

Assistant completion. The following ProofWriter example illustrates a one-step proof for the query "Erin is quiet" given the fact "Erin is rough" and the rule "Rough things are quiet," as shown in Listing 11.

Listing 11: Example ProofWriter structured-proof completion in which one cited rule derives the query assertion from an input fact.

```
<reasoning>
<initial_facts>
ClassAssertion(:rough :Erin)
</initial_facts>
<step index="1">
<premises>
ClassAssertion(:rough :Erin)
</premises>
<rule>
SubClassOf(:rough :quiet)
```

```
</rule>
<conclusion>
ClassAssertion(:quiet :Erin)
</conclusion>
</step>
</reasoning>
<query_assertion>
ClassAssertion(:quiet :Erin)
</query_assertion>
<answer>True</answer>
```

For a directly supported ProofWriter query, the completion records the supporting assertion in `initial_facts` and contains no `step` block, as shown in Listing 12.

Listing 12: ProofWriter direct-proof output format. The supporting input assertion is recorded in `initial_facts`, and no inference step is generated.

```
<reasoning>
<initial_facts>
ClassAssertion(:quiet :Erin)
</initial_facts>
</reasoning>
<query_assertion>
ClassAssertion(:quiet :Erin)
</query_assertion>
<answer>True</answer>
```

For an UNKNOWN query, the reasoning block must be empty, as shown in Listing 13.

Listing 13: UNKNOWN output format shared by both datasets. The reasoning block remains empty because neither the query nor its negated assertion is derivable.

```
<reasoning>
</reasoning>
<query_assertion>
ClassAssertion(:quiet :Erin)
</query_assertion>
<answer>Unknown</answer>
```

HermiT-guided RL reuses the corresponding dataset-specific Structured Proof SFT prompt shown in Listings 9 and 10. The prompt text is copied unchanged into each RL instance and is formatted with the same non-thinking Qwen chat template. Thus, the transition from supervised proof learning to reinforcement learning introduces no additional policy-facing instruction or output-schema shift; only the source of the optimization signal changes.

The policy-visible prompt contains the task instruction, allowed ontology vocabulary, natural-language facts and rules, natural-language query, and required output schema. The gold completion is removed when the RL view is constructed. The executable ontology, canonical formal query, negated-assertion query, and gold answer are retained separately as hidden verifier metadata used by reward computation. The reference proof is retained only for data provenance and offline evaluation and is neither presented to the policy nor used by the RL reward. The ontology vocabulary remains visible because it constrains symbol generation, whereas the executable ontology constructed from the formalized theory remains hidden.

For each RL instance, the policy returns the same three top-level components required by the corresponding Structured Proof SFT prompt: a dataset-specific structured proof block, a canonical OWL query assertion, and one answer from $\{\text{TRUE}, \text{FALSE}, \text{UNKNOWN}\}$. Candidate proofs are evaluated against the hidden ontology and theory metadata rather than against the reference proof text. Consequently, an alternative proof can receive full semantic credit when its cited premises and rules are grounded in the input theory and its conclusions are validated by HermiT.

Policy–verifier separation. Let x denote the visible prompt and m the hidden verifier metadata. Their contents are summarized as

$$x = (V, \mathcal{F}_{\mathrm{NL}}, \mathcal{R}_{\mathrm{NL}}, q_{\mathrm{NL}}, \mathcal{S}_{\mathrm{out}}), \tag{23}$$

$$m = (\mathcal{O}, q, \bar{q}, y, \pi^*), \tag{24}$$

where V is the allowed vocabulary, $\mathcal{F}_{\mathrm{NL}}$ and $\mathcal{R}_{\mathrm{NL}}$ are the natural-language facts and rules, q_{NL} is the natural-language query, and $\mathcal{S}_{\mathrm{out}}$ is the dataset-specific output schema. The reference proof π^* is retained for data provenance and offline evaluation but is not supplied to the policy or used by the RL reward. Reward computation uses the remaining formal metadata, including $\mathcal{O}$, q, $\bar{q}$, and y.

A.3.4 Output Parsing and Failure Handling

Assertion-level completions are parsed as single OWL Functional-Style expressions. The parser requires complete token consumption, so a valid expression followed by an explanation, Markdown fence, label, or second expression does not satisfy the strict interface. The parsed abstract syntax tree is checked for its top-level axiom type, constructor arity, recursive structure, symbol roles, and membership in the supplied ontology vocabulary. Accepted expressions are canonically re-serialized before exact-match and validity evaluation.

Structured-proof and RL completions are parsed into $(\pi, \hat{q}, \hat{y})$, where π is the structured proof recorded in the reasoning block, $\hat{q}$ is the content of the unique `query_assertion` block, and $\hat{y}$ is the content of the unique `answer` block. The shared parser can extract an optional `initial_facts` block followed by an ordered sequence of `step` blocks. Canonical ProofWriter targets use this block to record supporting input assertions, whereas canonical ProntoQA targets omit it and cite input facts directly as step premises. Every step must contain exactly one `premises` block, one `rule` block, and one `conclusion` block. Step indices must follow the generated proof order, and each premise must resolve to an input fact or the conclusion of an earlier step.

An output fails the format interface if a required top-level field is missing, a singleton field is duplicated, tags are malformed or improperly nested, the answer is not one of `True`, `False`, and `Unknown`, the query assertion cannot be parsed, or non-whitespace explanatory text appears outside the required structure. A nonempty proof accompanying an UNKNOWN answer may remain syntactically parseable, but it violates the required Unknown-output semantics and receives the corresponding reasoning penalty. Direct proofs and non-direct proofs are distinguished through the presence of inference steps rather than through an additional natural-language label.

The verifier accepts at most eight inference steps. A candidate exceeding this limit fails the interface check and is not treated as a valid structured proof. For RL candidates, the format component is

$$R_{\mathrm{format}} = \begin{cases} +0.5, & \text{all interface checks pass,} \\ -0.5, & \text{otherwise.} \end{cases} \tag{25}$$

Syntactically well-formed output is therefore necessary but not sufficient for a positive format reward: the query field, answer field, OWL expressions, symbol roles, grounding dependencies, and proof-length constraint must also satisfy their respective checks.

Generation length is bounded by the decoding configuration rather than by the parser. The parser does not repair or complete a response truncated at the token limit. A truncated completion is processed as generated; if it lacks a required closing tag or any required field, it fails the corresponding format check.

A.3.5 Comparison of the Three Prompt Interfaces

Table 8 summarizes the information exposed to the model and the expected output at each stage.

The interfaces become progressively more demanding while preserving the formal representation acquired in earlier stages. Assertion-Level SFT isolates formal-language acquisition by mapping one statement to one OWL expression. Structured Proof SFT reuses these expressions within dataset-specific proof structures and additionally requires query formalization and three-way classification. For each dataset, HermiT-guided RL preserves the Structured Proof SFT prompt unchanged but removes the reference completion from the optimization target, replacing proof imitation with gold-answer and verifier-based semantic rewards. This continuity avoids an unnecessary prompt-format shift while allowing the policy to explore alternative logically valid proofs.

Stage	Policy-visible input	Expected output	Supervision or feedback
Assertion-Level SFT	One natural-language statement and its allowed ontology vocabulary	Exactly one canonical OWL Functional-Style expression	Token-level completion supervision using the reference OWL expression
Structured Proof SFT	Natural-language facts, rules, query, allowed vocabulary, and a dataset-specific output schema	A dataset-specific structured proof, formal query assertion, and three-way answer	Token-level completion supervision using one reference structured proof
HermiT-Guided RL	The unchanged dataset-specific prompt from Structured Proof SFT	The corresponding structured proof, formal query assertion, and three-way answer	Format and gold-answer supervision with sequential HermiT-based semantic feedback

Table 8: Comparison of the prompt and output interfaces used by the three stages of ReaLogic-FT.

A.4 HermiT-Guided Reward and Evaluation

This section specifies how a sampled candidate is verified and converted into a scalar reward. We first describe the deterministic and semantic checks applied to each candidate, then give the complete reward definition, and finally state the Dr. GRPO objective used to update the policy.

A.4.1 Candidate Verification Pipeline

For each RL prompt x, the policy generates $G = 4$ candidate completions $c_1, \ldots, c_G$. Each completion is parsed into a structured proof $\pi_g = (F_{0,g}, s_{g,1}, \ldots, s_{g,m_g})$, a query assertion $\hat{q}_g$, and an answer $\hat{y}_g \in \{\textsc{True}, \textsc{False}, \textsc{Unknown}\}$. The policy observes only the problem prompt and permitted ontology vocabulary; the executable ontology, canonical query and its opposite, gold answer, and verifier metadata remain hidden.

Verification begins with deterministic interface checks. The parser validates the required tags, field uniqueness, step order, answer vocabulary, and eight-step limit. Each generated OWL expression is checked for syntax, constructor arity, symbol roles, and ontology-vocabulary membership. A valid interface additionally requires $\hat{q}_g = q$, every recorded initial fact to occur in the input theory, and every cited rule to originate from the input TBox. These checks are retained for evaluation and diagnostics.

HermiT first determines the semantic answer of the base ontology. Step-level semantic verification is performed only when this answer agrees with the benchmark label. The verifier then initializes an accepted state with the input facts and processes the generated steps sequentially. A step

$$s_{g,i} = (P_{g,i}, r_{g,i}, c_{g,i}) \tag{26}$$

is eligible for HermiT verification when it is structurally well formed and OWL-valid, $r_{g,i}$ belongs to the input TBox, and every premise in $P_{g,i}$ is either an input fact or a previously accepted conclusion. HermiT accepts the step when

$$P_{g,i} \cup \{r_{g,i}\} \models c_{g,i}. \tag{27}$$

Let

$$J_{g,i} = \mathbb{I}\left[S_{g,i} \wedge O_{g,i} \wedge R_{g,i} \wedge P_{g,i}^{\mathrm{acc}} \wedge H_{g,i}\right], \tag{28}$$

where $S_{g,i}$ denotes structural validity, $O_{g,i}$ OWL validity, $R_{g,i}$ rule provenance, $P_{g,i}^{\mathrm{acc}}$ accepted-state premise grounding, and $H_{g,i}$ HermiT entailment. Only conclusions with $J_{g,i} = 1$ are added to the accepted state and may support subsequent steps.

Answer support is evaluated against the input facts and accepted conclusions. A TRUE prediction is supported when this state contains q, and a FALSE prediction is supported when it contains $\bar{q}$. For Full-Sem. evaluation, a valid UNKNOWN output must contain no reasoning items, and the base ontology must entail neither q nor $\bar{q}$. No whole-chain ontology is constructed from rejected conclusions, and the complete ReaLogic-FT reward does not perform the legacy consistency check over all generated conclusions.

The gold label determines answer correctness, whereas HermiT supplies the base semantic label and incremental step-entailment judgments. Reference proofs are not used in reward computation, al-

lowing an alternative derivation to receive full semantic credit when its steps are grounded, accepted sequentially, and support the correct answer.

If the benchmark and HermiT labels disagree, or if HermiT fails or times out, semantic verification is marked unavailable and R_{semantic} is set to zero; the format and benchmark-answer rewards are preserved. Candidates with missing required verifier metadata cannot be semantically verified and are recorded as invalid verification instances.

Semantic-verification coverage. We distinguish semantic unavailability from benchmark–HermiT disagreement. On ProntoQA, HermiT returned a semantic label agreeing with the benchmark for all 2,936 held-out instances. On ProofWriter, semantic labels were available for 2,494 of 2,500 instances; 2,457 agreed with the benchmark and 37 (1.48% of semantically available instances) disagreed. Semantic labels were unavailable for six additional ProofWriter instances; these cases were excluded from semantic scoring while retaining benchmark-based evaluation. These counts are identical across model seeds because they depend only on the input ontology. No HermiT timeout, exception, or service error occurred during the reported evaluations. For disagreement cases, benchmark-based evaluation and supervision are retained, while semantic reward and Full-Sem. credit are disabled.

A.4.2 Complete Reward Definition

The complete ReaLogic-FT reward for candidate g consists of three components:

$$R_g = R_{\text{format}} + R_{\text{answer}} + R_{\text{semantic}}. \tag{29}$$

The first two components provide benchmark-based supervision, while the semantic component evaluates generated reasoning through sequential HermiT verification.

Format and answer rewards. Let $I_g = 1$ when the candidate satisfies the required output structure, including the required tags, field organization, step order, and answer vocabulary, and let $I_g = 0$ otherwise. With gold answer y,

$$\begin{aligned} R_{\text{format}} &= \begin{cases} +0.5, & I_g = 1, \\ -0.5, & I_g = 0, \end{cases} \\ R_{\text{answer}} &= \begin{cases} +2, & \hat{y}_g = y, \\ -2, & \hat{y}_g \neq y. \end{cases} \end{aligned} \tag{30}$$

Query identity, OWL validity, grounding, and the step limit are retained as interface-validity checks but are not folded into R_{format}.

Semantic reward. A proof is required when $y \in \{\text{TRUE}, \text{FALSE}\}$ and the corresponding target assertion is not directly present in the input facts. For a required proof with m_g generated steps, let

$$Q_g = \begin{cases} \dfrac{1}{m_g} \displaystyle\sum_{i=1}^{m_g} J_{g,i}, & m_g > 0 \text{ and the accepted state supports } \hat{y}_g, \\ 0, & \text{otherwise.} \end{cases} \tag{31}$$

where $J_{g,i} = 1$ exactly when step i is accepted by the sequential verifier. Thus, rejected conclusions cannot support either later steps or the final answer.

For examples requiring a proof, the semantic reward interpolates between the invalid- and fully valid-proof rewards:

$$R_{\text{semantic}} = -2 + 4Q_g. \tag{32}$$

For examples not requiring a proof, let

$$k_g = m_g - \sum_{i=1}^{m_g} J_{g,i} \tag{33}$$

be the number of generated inference steps rejected by the sequential verifier. The semantic reward is then

$$R_{\text{semantic}} = \begin{cases} \max(-0.75, -0.15k_g), & k_g > 0, \\ 0, & k_g = 0. \end{cases} \tag{34}$$

This branch applies to directly supported TRUE or FALSE examples and to UNKNOWN examples. Accordingly, UNKNOWN predictions receive no separate class-specific reward and are supervised through R_{answer}.

Semantic scoring is enabled only when HermiT successfully determines the base-ontology label and that label agrees with the benchmark answer. If the labels disagree or semantic verification fails, R_{semantic} is set to zero, while R_{format} and R_{answer} are preserved.

A.4.3 DR. GRPO OPTIMIZATION OBJECTIVE

GRPO compares multiple completions sampled for the same prompt without training a separate value critic Shao et al. (2024). We use the Dr. GRPO variant Liu et al. (2025), which centers rewards within each candidate group but does not divide them by their within-group standard deviation. For $G = 4$ candidates, the group mean and centered advantage are

$$\mu_R = \frac{1}{G}\sum_{j=1}^{G} R_j, \qquad \widehat{A}_g = R_g - \mu_R. \tag{35}$$

Here, $\widehat{A}_g$ represents the relative quality of candidate g among completions generated for the same prompt. If all candidates receive the same reward, their centered advantages are zero.

For token $c_{g,t}$ with generated prefix $c_{g,<t}$, the importance ratio between the updated policy and the rollout policy is

$$\rho_{g,t}(\theta) = \frac{p_\theta(c_{g,t} \mid x, c_{g,<t})}{p_{\theta_{\text{old}}}(c_{g,t} \mid x, c_{g,<t})}. \tag{36}$$

The ratio is clipped using radius ϵ:

$$\widetilde{\rho}_{g,t}(\theta) = \text{clip}(\rho_{g,t}(\theta), 1-\epsilon, 1+\epsilon)\,, \tag{37}$$

and the conservative token-level surrogate is

$$\ell_{g,t}(\theta) = \min\Big(\rho_{g,t}(\theta)\widehat{A}_g, \widetilde{\rho}_{g,t}(\theta)\widehat{A}_g\Big)\,. \tag{38}$$

Let $M_{g,t}$ be the completion-token mask and $L_{\max}$ the fixed maximum completion length. The implemented Dr. GRPO objective is

$$\mathcal{L}_{\text{Dr. GRPO}}(\theta) = -\frac{1}{GL_{\max}}\sum_{g=1}^{G}\sum_{t=1}^{L_{\max}} M_{g,t}\ell_{g,t}(\theta). \tag{39}$$

Unlike normalization by each completion's realized length, the fixed denominator $GL_{\max}$ avoids introducing a completion-length-dependent normalization factor. In the reported configuration, $G = 4$, $L_{\max} = 768$, $\epsilon = 0.2$, reward standard-deviation scaling is disabled, and one optimization iteration is performed per rollout batch. Completions discarded by the configured truncation mask do not contribute to the loss.

We set the reference-policy KL coefficient to $\beta = 0$. Consequently, $p_{\theta_{\text{old}}}$ in Equation 36 is the policy that generated the current rollout, not a separately regularizing reference model. Reinforcement learning continues from the Structured Proof SFT adapter and updates the same LoRA parameters using the verifier-derived group-relative advantages.

A.5 IMPLEMENTATION AND REPRODUCIBILITY DETAILS

This section reports the settings needed to reproduce the reported models: the backbone and adapter configuration, the final training hyperparameters of each stage, the evaluation and decoding protocol, the computing environment and its measured cost, and the seed and reporting conventions used throughout.

Stage	Data	Init.	Budget / checkpoint	LR	Batch/Acc.	Length
Assertion SFT	ProofWriter	Qwen3-8B	≤ 3 epochs	2×10^{-5}	$8/2$	1,024
Assertion SFT	ProntoQA	Qwen3-8B	≤ 3 epochs	2×10^{-5}	$8/2$	1,024
Structured-Proof SFT	ProofWriter	Assertion adapter	≤ 2 epochs	1×10^{-5}	$8/2$	1,024
Structured-Proof SFT	ProntoQA	Assertion adapter	≤ 3 epochs	1×10^{-5}	$8/2$	1,024
HermiT-Guided RL	ProofWriter	Structured-proof adapter	20 updates	3×10^{-7}	$1/4$	1,024+768
HermiT-Guided RL	ProntoQA	Structured-proof adapter	100 updates	3×10^{-7}	$1/4$	1,024+768

Table 9: Final training configurations. SFT budgets are maximum epoch counts, with the checkpoint attaining the lowest development loss restored at the end of training. The RL rows report the checkpoints used for the final evaluation. Batch/Acc. denotes per-device batch size and gradient accumulation steps; RL length is shown as maximum prompt length plus maximum completion length.

A.5.1 Backbone, LoRA, and Stage-wise Initialization

All experiments use Qwen3-8B as the frozen backbone Team (2025). We train separate dataset-specific model chains for ProofWriter and ProntoQA. Within each chain, a single LoRA adapter Hu et al. (2022) is updated progressively across the three post-training stages. Assertion-Level SFT initializes a new adapter from the backbone; Structured Proof SFT reloads this adapter as trainable; HermiT-Guided RL subsequently reloads the Structured Proof SFT adapter. No new adapter is initialized between consecutive stages.

The adapter uses rank $r = 16$, scaling factor $\alpha = 32$, dropout 0.05, and no bias parameters. It is attached to `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`, covering the linear projections in both attention and feed-forward modules. The backbone is loaded without weight quantization, training uses bfloat16 precision and gradient checkpointing, and Qwen3 thinking mode is disabled when formatting both training and evaluation conversations.

A.5.2 Final Training Hyperparameters

Table 9 summarizes the principal settings used in the reported models. Batch entries give the per-device batch size followed by the number of gradient-accumulation steps. Because all experiments use one GPU, the corresponding effective batch sizes are 16 for both SFT stages and 4 for RL.

Assertion-Level SFT Assertion-Level SFT optimizes only completion tokens; prompt tokens are masked from the language-modeling loss. We use AdamW Loshchilov & Hutter (2019) with a cosine learning-rate schedule, a warmup ratio of 0.03, zero weight decay, a per-device evaluation batch size of 16, and a maximum sequence length of 1,024 tokens. Training runs for at most three epochs. ProofWriter is evaluated and checkpointed every 500 optimizer steps, whereas ProntoQA is evaluated and checkpointed every 100 steps. In both cases, the checkpoint with the lowest development loss is restored. Early stopping uses a patience of two evaluation events and a minimum improvement threshold of 10^{-5}.

Structured Proof SFT Structured Proof SFT continues optimizing the complete adapter produced by Assertion-Level SFT, with the same target modules and LoRA configuration. Prompt tokens remain excluded from the loss. We use AdamW, a cosine schedule, a learning rate of 10^{-5}, a warmup ratio of 0.03, zero weight decay, a per-device training batch size of 8, two gradient-accumulation steps, and a per-device evaluation batch size of 16. ProofWriter is trained for at most two epochs and evaluated every 2,000 optimizer steps with early-stopping patience two. ProntoQA is trained for at most three epochs and evaluated every 250 steps with patience three. Both use an early-stopping threshold of 10^{-5} and restore the checkpoint with the lowest development loss.

HermiT-Guided RL HermiT-Guided RL starts from the corresponding Structured Proof SFT adapter. We report the checkpoints after 20 optimizer updates for ProofWriter and 100 updates for ProntoQA. We use AdamW with a cosine schedule, a learning rate of 3×10^{-7}, a warmup ratio of 0.03, zero weight decay, a maximum gradient norm of 0.5, a per-device batch size of 1, and four gradient-accumulation steps. Each prompt is associated with a group of $G = 4$ sampled candidates. Rollout sampling uses temperature 0.8, nucleus probability 0.95, top-k sampling with $k = 50$, and a repetition penalty of 1.05. The maximum prompt and completion lengths are 1,024 and 768 tokens,

Dataset	Reference Setup	Metric	Candidate	Reference	Δ (pp)	95% CI (pp)	Holm p
ProofWriter	Base + ZS	Exact Match	100.00	83.23	16.77	$[15.27, 18.26]$	0.0008
	Base + ZS	Strict Valid	100.00	92.56	7.44	$[6.42, 8.51]$	0.0008
	Base + FS	Exact Match	100.00	95.30	4.70	$[3.85, 5.56]$	0.0008
	Base + FS	Strict Valid	100.00	97.48	2.52	$[1.92, 3.17]$	0.0008
ProntoQA	Base + ZS	Exact Match	100.00	58.17	41.83	$[38.17, 45.65]$	0.0008
	Base + ZS	Strict Valid	100.00	85.34	14.66	$[12.06, 17.40]$	0.0008
	Base + FS	Exact Match	100.00	98.32	1.68	$[0.76, 2.75]$	0.0008
	Base + FS	Strict Valid	100.00	98.32	1.68	$[0.76, 2.75]$	0.0008

Table 10: Paired bootstrap significance tests for Assertion-Level SFT (%). The candidate is Assertion-Level SFT in every comparison. All three SFT seeds attain identical per-instance EM and SV outcomes and are therefore treated as a fixed candidate, while the prompting baselines are deterministic single runs. Bootstrap resampling is paired over the fixed evaluation instances using 10,000 replicates. ZS and FS denote zero-shot and few-shot prompting, respectively. Δ is the candidate–reference difference in percentage points, and Holm correction is applied across the four comparisons within each dataset.

respectively. Dr. GRPO uses clipping radius $\epsilon = 0.2$, one policy iteration per batch, no within-group standard-deviation scaling, and no reference-policy KL penalty ($\beta = 0$). Truncated completions are masked from the objective, model dropout is disabled, and vLLM is not used. The HermiT service uses a 20-second request timeout, a 2 GB Java heap, and a maximum of eight generated reasoning steps.

A.5.3 Evaluation and Decoding Configuration

All reported test predictions use greedy decoding with `do_sample=false`; therefore, rollout sampling parameters used during RL training do not affect evaluation. Qwen3 thinking mode is disabled, decoder-only batches use left padding, and generation terminates at the tokenizer end-of-sequence token or the applicable length limit. Assertion-level evaluation uses batch size 16 and at most 128 new tokens. Structured-proof prompting, Structured Proof SFT, and HermiT-Guided RL are all evaluated with batch size 8 and at most 768 new tokens, matching the RL completion limit; only the initialization ablation in Table 14 uses the original limit of 1,024 new tokens.

Zero-shot baselines receive the same task interface and ontology vocabulary as the trained models. Few-shot baselines prepend four fixed demonstrations selected only from the training split. The same demonstrations are used for the corresponding model comparisons, and their identifiers are explicitly excluded from the evaluation instances. ProofWriter evaluation uses 2,500 examples, with 500 examples at each retained reasoning depth, while ProntoQA evaluation uses all 2,936 held-out examples across one to five hops. No test example is used for checkpoint or hyperparameter selection.

Joint correctness metrics. For Structured Proof SFT, *Joint Correct* (Joint) requires a correct answer and query together with a well-formed, well-typed proof whose steps satisfy the dataset-specific inference templates. For HermiT-Guided RL, *Grounded Joint* (G-Joint) instead requires a correct answer and query, a valid OWL interface, structural answer support, and a dependency-ordered proof whose initial facts and cited rules are grounded in the input theory. G-Joint does not assess HermiT entailment, which is captured separately by Full-Sem.

A.5.4 Hyperparameter Development and Selection

Most hyperparameters were shared across ProofWriter and ProntoQA, including the LoRA rank and scaling factor, target modules, SFT batch configuration, and sequence-length limits. The learning rates for Assertion-Level SFT and Structured Proof SFT were set to 2×10^{-5} and 1×10^{-5}, respectively, with checkpoint selection based solely on development loss. For RL, preliminary engineering runs used a learning rate of 5×10^{-7} to validate memory consumption, generation, reward integration, and checkpointing, whereas the reported configurations used 3×10^{-7}. We use fixed dataset-

Dataset	Stage	Steps	Train (h)	Eval. (s)	HermiT phase	HermiT calls	Final / Step	Verifier share
ProofWriter	Assertion-Level SFT	–	0.62	N/A	–	0	0 / 0	0.00%
ProntoQA	Assertion-Level SFT	–	0.26	N/A	–	0	0 / 0	0.00%
ProofWriter	Structured Proof SFT	–	27.99	3291.02	Evaluation	3,438	2,609 / 829	0.28%
ProntoQA	Structured Proof SFT	–	7.07	5799.82	Evaluation	9,547	3,086 / 6,461	0.35%
ProofWriter	HermiT-Guided RL	20	0.07	N/A	Training	148	80 / 68	N/A
ProntoQA	HermiT-Guided RL	100	0.52	N/A	Training	1,245	400 / 845	N/A

Table 11: Representative per-run training and verification costs for the final configurations. Training time reports the runtime up to the selected checkpoint. Evaluation time covers complete deterministic generation and metric computation, including HermiT verification. Final and Step denote verifier invocations for base-ontology answer determination and individual-step entailment verification, respectively. Verifier share is the fraction of total evaluation runtime attributable to recorded HermiT latency. RL invocation counts are reconstructed exactly from candidate-level attribution records up to the reported checkpoint; training-time verifier latency was not recorded, so the corresponding runtime share is unavailable.

specific reporting checkpoints: checkpoint 20 for ProofWriter and checkpoint 100 for ProntoQA. The same checkpoint is used across all RL seeds and reward configurations within each dataset, ensuring a compute-matched comparison without configuration-specific checkpoint selection.

A.5.5 Computing Infrastructure and Software Environment

Training and evaluation were conducted in an AutoDL container running Ubuntu 22.04.1 LTS on a 64-bit Linux kernel. Each experiment used a single NVIDIA A800 PCIe GPU with 80 GB memory; although the host exposed two A800 GPUs, no distributed or multi-GPU training was used. The allocated instance provided 14 virtual CPU cores from an Intel Xeon Gold 6348 processor at 2.60 GHz and 120 GB system memory. The host used NVIDIA driver 580.82.07, which reports compatibility with CUDA 13.0, while the installed CUDA toolkit was 11.8 (V11.8.89). The PyTorch environments likewise used CUDA runtime 11.8 and cuDNN 8.7.0.

The SFT environment used Python 3.10.20, PyTorch 2.1.2 with CUDA 11.8, Transformers 4.52.4, PEFT 0.11.1, TRL 0.8.6, and Accelerate 1.8.1. The RL environment used Python 3.10.20, PyTorch 2.1.2 with CUDA 11.8, Transformers 4.52.4, PEFT 0.15.2, TRL 0.17.0, and Accelerate 1.8.1. The Java verification service ran on OpenJDK 17.0.19 and was built with Apache Maven 3.6.3. Its source and target compatibility levels were set to Java 8, and it used HermiT 1.4.5.519 together with OWL API 5.1.9.

A.5.6 Measured Computational Cost

Table 11 reports representative per-run training and verification costs for the three post-training stages. All runs use one NVIDIA A800 PCIe GPU with 80 GB memory. The reported training time is the `train_runtime` saved by the Hugging Face Trainer and may exclude process initialization, checkpoint loading, and other non-trainer overhead. The entries should therefore be interpreted as audited per-run measurements rather than cumulative costs over all random seeds. Assertion-Level SFT and Structured Proof SFT do not invoke HermiT during optimization; the HermiT measurements associated with Structured Proof SFT come from the subsequent verifier-aware evaluation. During RL, HermiT is invoked online to compute semantic rewards.

The two Assertion-Level SFT runs require 0.88 recorded GPU-hours per seed in total, while the seed-42 Structured Proof SFT runs require 35.06 trainer GPU-hours across the two datasets, excluding evaluation and process overhead. Assertion-Level SFT is run with seeds 42, 43, and 44 and reported using the mean and sample standard deviation over these three seeds. Structured Proof SFT is reported using seed 42.

HermiT-Guided RL is run with seeds 42, 43, and 44, with all three runs initialized from the same seed-42 Structured Proof SFT checkpoint. Measured up to the selected checkpoints of 20 updates on ProofWriter and 100 updates on ProntoQA, the runs require 0.07 and 0.52 trainer hours while making 148 and 1,245 verifier invocations, respectively. With an effective batch of four prompts,

these checkpoints correspond to 80 and 400 training prompts. The complete RL experiment matrix contains the complete ReaLogic-FT reward and the w/o Sem. ablation for three seeds on both datasets, amounting to 12 training runs; per-run costs are given in Table 11. The RL results should therefore be interpreted as evidence under a small post-training budget rather than as a convergence study over optimization length.

A.5.7 Random Seeds and Determinism

Assertion-Level SFT is run with training seeds 42, 43, and 44 for both ProofWriter and ProntoQA, whereas the reported Structured Proof SFT results use seed 42. The verifier-guided curricula are constructed once with seed 42 and shared across RL runs. Full ReaLogic-FT and the w/o Sem. ablation use RL optimization and data-sampling seeds 42, 43, and 44, while all RL runs start from the fixed seed-42 Structured Proof SFT checkpoint for the corresponding dataset. Evaluation uses greedy decoding and is therefore not affected by token-sampling randomness.

These settings make data selection, curriculum construction, and evaluation repeatable, but we do not claim bitwise reproducibility across hardware or software installations because GPU kernels may be nondeterministic. The SFT seed controls the Trainer-side sources of randomness, while exact reproduction may additionally depend on the timing of adapter initialization in the training launcher.

A.5.8 Number of Runs and Reporting Protocol

For each dataset, Assertion-Level SFT is trained independently with seeds 42, 43, and 44, and the main paper reports the mean and sample standard deviation over these three runs. Structured Proof SFT is trained with seeds 42, 43, and 44, but only the seed-42 run is evaluated under the final 768-token protocol; the seed-43 and seed-44 runs were evaluated only under the earlier 1,024-token limit and are therefore not reported.

For HermiT-Guided RL, the seed-42 Structured Proof SFT checkpoint is held fixed as the common pre-RL initialization. Full ReaLogic-FT and the w/o Sem. ablation are trained independently with RL seeds 42, 43, and 44. Following development-set selection, we report checkpoint 20 for ProofWriter and checkpoint 100 for ProntoQA, using the mean and sample standard deviation over the three RL seeds. Consequently, RL variability measures randomness in verifier-guided optimization conditional on a fixed Structured Proof SFT initialization; it does not include variation from the preceding Structured Proof SFT stage.

Prompting baselines require no training and are evaluated once. Every reported evaluation is computed from one complete pass over the designated held-out set using greedy decoding. The bootstrap analyses resample evaluation examples and, where three-seed results are available, account hierarchically for variation across training seeds.

A.6 Additional Experimental Results

This section reports the analyses referenced by the main paper: significance tests for the two supervised stages, the assertion-level initialization ablation, the complete reward ablation with its difficulty-wise breakdown, and a qualitative failure analysis.

Multiple-comparison protocol. We report three significance analyses that answer distinct questions: whether assertion-level alignment improves over prompting, whether structured proof supervision improves over the preceding conditions, and whether verifier-guided reinforcement learning improves over its pre-RL checkpoint and over the semantic-reward ablation. Holm correction is applied within each analysis and within each dataset, so that the correction family matches the set of comparisons a reader would examine together when interpreting that analysis. Because the three analyses use different candidates, references, and evaluation subsets, we do not pool them into a single family; conversely, all comparisons belonging to one analysis are corrected jointly rather than reported per test. The difficulty-wise results are reported descriptively, without significance tests.

A.6.1 Statistical Analysis of Progressive Supervision

Bootstrap Significance Tests for Assertion-Level SFT We apply the paired bootstrap procedure to compare Assertion-Level SFT with the zero-shot and few-shot base-model conditions on the assertion-level task. The eight comparisons within each dataset are jointly corrected using the Holm procedure. Table 10 shows that Assertion-Level SFT significantly outperforms every prompting condition on both Exact Match and Strict Valid, with all Holm-adjusted p values at 0.0008 and all confidence intervals excluding zero. The margins are largest against zero-shot prompting, where Exact Match improves by 16.77 points on ProofWriter and 41.83 points on ProntoQA. Few-shot prompting narrows the gap considerably, particularly on ProntoQA, where the base model already reaches 98.32% on both metrics; the remaining 1.68-point improvements nonetheless exclude zero. Because the fine-tuned model reaches 100% on both metrics and both datasets, these tests establish that the formal-language barrier is fully removed at the assertion level rather than merely reduced.

Bootstrap Significance Tests for Structured Proof SFT We apply the same paired bootstrap procedure to compare Structured Proof SFT with all four pre-structured-proof conditions. The sixteen comparisons within each dataset are jointly corrected using the Holm procedure. Table 12 shows significant gains across answer accuracy, Macro-F1, Joint Correct, and Step-F1 for every reference condition. Zero-shot inference produces no jointly correct structured proofs for either the base model or the assertion-aligned model, despite their nonzero answer scores. Assertion-level alignment therefore teaches the formal representation but does not by itself establish the complete proof-generation behavior required by the task. Few-shot demonstrations partially recover this behavior, yielding Joint Correct scores of 45.60–45.96 on ProofWriter and 37.26–38.83 on ProntoQA. Structured Proof SFT raises these scores to 89.92 and 91.11, respectively. The corresponding Step-F1 improvements range from 23.49 to 24.78 percentage points on ProofWriter and from 51.48 to 52.53 percentage points on ProntoQA. Together, these results support the intended stage-wise division of labor: assertion-level supervision establishes the formal language, while structured proof supervision teaches the model to compose that language into complete derivations and aligned answers.

Bootstrap Significance Tests for HermiT-Guided RL We further assess the reliability of the differences between Full ReaLogic-FT and the pre-RL or reward-ablated configurations. For each dataset, we evaluate Acc., G-Joint, Step-F1, U-F1, and Full-Sem. using 10,000 hierarchical paired bootstrap replicates over evaluation instances and the three RL training seeds; corresponding seed identities are paired when comparing Full with w/o Sem., while Structured Proof SFT is treated as a fixed deterministic run. Holm correction is applied across the ten comparisons within each dataset. As shown in Table 13, the point estimates consistently favor Full: it matches or improves upon Structured Proof SFT and achieves higher mean values than w/o Sem. on every reported metric across both datasets. However, all 95% confidence intervals include zero, and no comparison reaches statistical significance after Holm correction. We therefore interpret the results as a consistent directional pattern in favor of semantic verification, while recognizing that additional training seeds and larger-scale evaluation would be needed to estimate the magnitude and robustness of these gains more precisely.

A.6.2 Ablation of Assertion-Level Initialization

Experimental setup. We examine whether assertion-level alignment provides a useful initialization for subsequent structured proof learning. For each dataset, we train Structured Proof SFT under two configurations. *Assertion Adapter* initializes the policy from the LoRA adapter obtained by Assertion-Level SFT, whereas *Base Model* attaches a fresh LoRA adapter directly to Qwen3-8B and omits the assertion-level stage. The two configurations use identical structured-proof training and development data, LoRA settings, optimization hyperparameters, checkpoint-selection criteria, and evaluation procedures; only the adapter initialization differs. Evaluation is performed on the same 2,500 ProofWriter and 2,936 ProntoQA instances used in the main experiments.

This auxiliary ablation was conducted under the original evaluation protocol with a maximum completion length of 1,024 tokens. Both initialization conditions were evaluated using the same test instances, decoding procedure, and metric implementation, so their within-table comparison remains

Dataset	Reference Setup	Metric	Candidate	Reference	Δ (pp)	95% CI (pp)	Holm p
ProofWriter	Base + ZS	Accuracy	92.96	38.36	54.60	$[52.52, 56.68]$	0.0032
	Base + ZS	Macro-F1	92.81	43.36	49.45	$[47.49, 51.42]$	0.0032
	Base + ZS	Joint	89.92	0.00	89.92	$[88.68, 91.08]$	0.0032
	Base + ZS	Step-F1	65.37	0.00	65.37	$[61.67, 68.91]$	0.0032
	Base + FS	Accuracy	92.96	70.28	22.68	$[20.80, 24.64]$	0.0032
	Base + FS	Macro-F1	92.81	71.15	21.66	$[19.82, 23.53]$	0.0032
	Base + FS	Joint	89.92	45.60	44.32	$[42.16, 46.56]$	0.0032
	Base + FS	Step-F1	65.37	40.58	24.78	$[20.67, 28.96]$	0.0032
	Assertion-Level SFT + ZS	Accuracy	92.96	37.72	55.24	$[53.16, 57.32]$	0.0032
	Assertion-Level SFT + ZS	Macro-F1	92.81	44.01	48.80	$[46.83, 50.76]$	0.0032
	Assertion-Level SFT + ZS	Joint	89.92	0.00	89.92	$[88.68, 91.08]$	0.0032
	Assertion-Level SFT + ZS	Step-F1	65.37	0.00	65.37	$[61.67, 68.91]$	0.0032
	Assertion-Level SFT + FS	Accuracy	92.96	71.64	21.32	$[19.44, 23.24]$	0.0032
	Assertion-Level SFT + FS	Macro-F1	92.81	72.67	20.14	$[18.36, 22.01]$	0.0032
	Assertion-Level SFT + FS	Joint	89.92	45.96	43.96	$[41.76, 46.20]$	0.0032
	Assertion-Level SFT + FS	Step-F1	65.37	41.88	23.49	$[19.30, 27.76]$	0.0032
ProntoQA	Base + ZS	Accuracy	94.96	15.70	79.26	$[77.72, 80.76]$	0.0032
	Base + ZS	Macro-F1	94.91	20.65	74.26	$[72.57, 75.95]$	0.0032
	Base + ZS	Joint	91.11	0.00	91.11	$[90.09, 92.13]$	0.0032
	Base + ZS	Step-F1	92.37	0.00	92.37	$[91.40, 93.32]$	0.0032
	Base + FS	Accuracy	94.96	54.36	40.60	$[38.66, 42.54]$	0.0032
	Base + FS	Macro-F1	94.91	47.54	47.38	$[45.93, 48.81]$	0.0032
	Base + FS	Joint	91.11	37.26	53.85	$[51.91, 55.82]$	0.0032
	Base + FS	Step-F1	92.37	39.85	52.53	$[50.96, 54.09]$	0.0032
	Assertion-Level SFT + ZS	Accuracy	94.96	17.10	77.86	$[76.33, 79.39]$	0.0032
	Assertion-Level SFT + ZS	Macro-F1	94.91	21.83	73.09	$[71.47, 74.75]$	0.0032
	Assertion-Level SFT + ZS	Joint	91.11	0.00	91.11	$[90.09, 92.13]$	0.0032
	Assertion-Level SFT + ZS	Step-F1	92.37	0.00	92.37	$[91.40, 93.32]$	0.0032
	Assertion-Level SFT + FS	Accuracy	94.96	55.48	39.48	$[37.53, 41.42]$	0.0032
	Assertion-Level SFT + FS	Macro-F1	94.91	48.27	46.64	$[45.24, 48.03]$	0.0032
	Assertion-Level SFT + FS	Joint	91.11	38.83	52.28	$[50.34, 54.22]$	0.0032
	Assertion-Level SFT + FS	Step-F1	92.37	40.90	51.48	$[49.91, 53.01]$	0.0032

Table 12: Paired bootstrap significance tests for Structured Proof SFT (%). Bootstrap resampling is performed over test instances from the fixed evaluation sets, with the seed-42 Structured Proof SFT run serving as the candidate in every comparison. ZS and FS denote zero-shot and few-shot prompting, respectively. Candidate and Reference give the corresponding metric values, and Δ denotes their difference in percentage points. Joint denotes Joint Correct and evaluates the structural proof requirements without the additional theory-grounding requirement imposed by G-Joint. Confidence intervals are obtained from 10,000 paired bootstrap replicates. Holm p reports the adjusted p-value after correction over sixteen tests within each dataset.

controlled. The results should not be compared numerically with the main Structured Proof SFT table, which uses the final unified 768-token evaluation protocol.

Dataset	Reference	Metric	Candidate	Reference	Δ (pp)	95% CI (pp)	Holm p
ProofWriter	Structured Proof SFT	Acc.	93.12	92.96	0.16	$[-0.31, 0.67]$	1.00
	Structured Proof SFT	G-Joint	89.92	89.92	0.00	$[-0.45, 0.51]$	1.00
	Structured Proof SFT	Step-F1	66.36	65.37	0.99	$[-0.96, 3.03]$	1.00
	Structured Proof SFT	U-F1	93.55	93.46	0.09	$[-0.34, 0.55]$	1.00
	Structured Proof SFT	Full-Sem.	88.60	88.52	0.08	$[-0.37, 0.59]$	1.00
	w/o Sem.	Acc.	93.12	92.96	0.16	$[-0.16, 0.49]$	1.00
	w/o Sem.	G-Joint	89.92	89.71	0.21	$[-0.12, 0.57]$	1.00
	w/o Sem.	Step-F1	66.36	65.47	0.89	$[-0.73, 2.77]$	1.00
	w/o Sem.	U-F1	93.55	93.42	0.13	$[-0.18, 0.45]$	1.00
	w/o Sem.	Full-Sem.	88.60	88.36	0.24	$[-0.08, 0.60]$	1.00
ProntoQA	Structured Proof SFT	Acc.	95.15	94.96	0.19	$[-0.18, 0.59]$	1.00
	Structured Proof SFT	G-Joint	91.33	91.11	0.22	$[-0.24, 0.69]$	1.00
	Structured Proof SFT	Step-F1	92.56	92.37	0.19	$[-0.25, 0.63]$	1.00
	Structured Proof SFT	U-F1	91.47	91.15	0.32	$[-0.41, 1.13]$	1.00
	Structured Proof SFT	Full-Sem.	91.17	90.97	0.19	$[-0.28, 0.69]$	1.00
	w/o Sem.	Acc.	95.15	94.81	0.34	$[-0.11, 0.85]$	1.00
	w/o Sem.	G-Joint	91.33	91.06	0.26	$[-0.30, 0.87]$	1.00
	w/o Sem.	Step-F1	92.56	92.27	0.30	$[-0.09, 0.70]$	1.00
	w/o Sem.	U-F1	91.47	91.03	0.44	$[-0.39, 1.34]$	1.00
	w/o Sem.	Full-Sem.	91.17	90.89	0.27	$[-0.27, 0.89]$	1.00

Table 13: Hierarchical paired bootstrap comparisons between Full ReaLogic-FT and the reference configurations (%). Candidate denotes Full ReaLogic-FT in every comparison, and w/o Sem. denotes Full without R_{semantic}. Bootstrap resampling is paired over the same evaluation instances and hierarchically accounts for the three RL training seeds, using 10,000 replicates; Structured Proof SFT is treated as a fixed deterministic run. Holm correction is applied across the ten comparisons within each dataset. Every 95% confidence interval includes zero, and no comparison is significant after Holm correction.

Configuration	Acc.	M-F1	Joint	Step-F1
ProofWriter				
Assertion Adapter	92.44	92.29	89.00	**64.77**
Base Model	**93.00**	**92.88**	**89.68**	64.10
ProntoQA				
Assertion Adapter	**94.99**	**94.74**	**94.69**	**91.49**
Base Model	93.87	93.63	93.29	91.24

Table 14: Assertion-level initialization ablation under the original 1,024-token evaluation protocol (%). *Assertion Adapter* initializes Structured Proof SFT from the assertion-aligned adapter, whereas *Base Model* trains a fresh LoRA adapter directly from Qwen3-8B. Both configurations are single runs with seed 42 and are evaluated under the same protocol, making the within-table comparison controlled. These values are not directly comparable with the main Structured Proof SFT results, which use the final unified 768-token evaluation protocol. Joint denotes the structural Joint Correct metric and does not impose the additional theory-grounding requirement of G-Joint. Best results within each dataset are shown in bold.

Results and interpretation. Under this controlled 1,024-token auxiliary protocol, Table 14 shows different initialization effects across the two datasets. On ProofWriter, direct initialization from the base model produces similar performance: it improves Accuracy, Macro-F1, and Joint by 0.56, 0.59, and 0.68 percentage points, respectively, while reducing Step-F1 by 0.67 points. On ProntoQA, omitting assertion-level initialization reduces all four metrics, by 1.12 points in Accuracy, 1.11 points in Macro-F1, 1.40 points in Joint, and 0.25 points in Step-F1. These results indicate a clearer initialization effect on ProntoQA within this ablation, but their absolute metric values should not be compared with those in the main 768-token evaluation.

This cross-dataset contrast should nevertheless be interpreted cautiously. ProofWriter provides 253,042 structured-proof training instances, compared with 23,496 for ProntoQA, a 10.8-fold difference. The datasets also differ in formalization difficulty: before any fine-tuning, the base model obtains 83.23% zero-shot assertion Exact Match on ProofWriter but only 58.17% on ProntoQA. Consequently, the stronger initialization effect observed on ProntoQA may reflect both its smaller structured-proof supervision set and its more difficult formalization interface.

A.6.3 Difficulty-Wise Reward Ablation

The aggregate results may conceal differences across reasoning difficulty. We therefore report each reward configuration separately by ProofWriter reasoning depth and ProntoQA reasoning hop. Figures 5 and 6 visualize the principal trends, while Tables 15 and 16 report the complete set of answer- and proof-level metrics. Structured Proof SFT is reported as a single seed-42 run, whereas Full and w/o Sem. are reported as mean ± sample standard deviation over RL seeds 42, 43, and 44.

ProofWriter Results by Reasoning Depth Table 15 and Figure 5 show that ProofWriter performance generally decreases as reasoning depth increases. Structured Proof SFT remains highly competitive at shallow depths, whereas the mean effect of verifier-guided RL becomes more favorable on the depth-3 and depth-5 subsets. Relative to Structured Proof SFT, Full improves Acc., G-Joint, Step-F1, U-F1, and Full-Sem. at depth 5 by 0.67, 0.33, 1.92, 0.24, and 0.53 percentage points, respectively. Full also exceeds w/o Sem. on all five metrics at this depth. The largest mean gain occurs in Step-F1, suggesting that semantic feedback may be particularly helpful for longer derivations, although the magnitude of the improvement varies across metrics.

Config.	Ckpt.	Depth	N	Acc.	G-Joint	Step-F1	U-F1	Full-Sem.
SFT	–	depth-0	500	**98.60**	**98.20**	N/A	**98.39**	**97.60**
	–	depth-1	500	**95.20**	**94.00**	**75.65**	96.03	**93.40**
	–	depth-2	500	**95.00**	**92.40**	**76.19**	**95.24**	90.80
	–	depth-3	500	90.40	86.80	67.95	91.81	84.80
	–	depth-5	500	85.60	78.20	56.51	87.07	76.00
w/o Sem.	20	depth-0	500	98.40 ± 0.20	98.00 ± 0.20	N/A	98.16 ± 0.23	97.47 ± 0.12
	20	depth-1	500	$\mathbf{95.20 \pm 0.20}$	93.87 ± 0.31	75.51 ± 0.39	$\mathbf{96.08 \pm 0.12}$	93.27 ± 0.31
	20	depth-2	500	94.53 ± 0.12	92.00 ± 0.20	75.36 ± 0.89	94.74 ± 0.10	90.47 ± 0.31
	20	depth-3	500	90.80 ± 0.60	$\mathbf{86.93 \pm 0.46}$	$\mathbf{69.44 \pm 2.17}$	92.08 ± 0.44	$\mathbf{84.93 \pm 0.46}$
	20	depth-5	500	85.87 ± 0.83	77.73 ± 0.42	56.48 ± 1.17	87.14 ± 0.62	75.67 ± 0.46
Full	20	depth-0	500	98.40 ± 0.00	98.00 ± 0.00	N/A	98.16 ± 0.00	97.53 ± 0.12
	20	depth-1	500	94.93 ± 0.31	93.73 ± 0.12	74.67 ± 0.59	95.76 ± 0.31	93.13 ± 0.12
	20	depth-2	500	$\mathbf{95.00 \pm 0.20}$	$\mathbf{92.40 \pm 0.20}$	76.11 ± 0.29	95.18 ± 0.21	$\mathbf{90.87 \pm 0.31}$
	20	depth-3	500	$\mathbf{91.00 \pm 0.53}$	$\mathbf{86.93 \pm 0.31}$	$\mathbf{69.44 \pm 1.06}$	$\mathbf{92.31 \pm 0.45}$	$\mathbf{84.93 \pm 0.31}$
	20	depth-5	500	$\mathbf{86.27 \pm 0.76}$	$\mathbf{78.53 \pm 0.64}$	$\mathbf{58.43 \pm 1.66}$	$\mathbf{87.31 \pm 0.74}$	$\mathbf{76.53 \pm 0.64}$

Table 15: ProofWriter reward ablation results by reasoning depth (%). SFT denotes Structured Proof SFT, Full denotes the complete ReaLogic-FT reward, and w/o Sem. denotes Full without the semantic reward R_{semantic}. SFT is evaluated once, while the RL configurations are reported as mean ± sample standard deviation over seeds 42, 43, and 44. The highest reported value at each depth is shown in bold. Step-F1 is undefined for depth-0 instances because they contain no gold inference steps.

ProntoQA Results by Reasoning Hop Table 16 and Figure 6 show a non-monotonic relationship between verifier-guided RL and reasoning hop count. Full obtains the highest reported values on all five metrics at one and two hops. At three hops, Full leads on G-Joint, Step-F1, and Full-Sem., whereas w/o Sem. is slightly higher on Acc. and U-F1. Structured Proof SFT remains strongest at four hops. At five hops, Full obtains the highest Acc., G-Joint, and Step-F1 and matches the SFT Full-Sem. point estimate. Relative to w/o Sem. at five hops, Full improves Acc., G-Joint, Step-F1, and Full-Sem. by 0.59, 0.44, 0.32, and 0.52 percentage points, respectively. U-F1 is undefined at five hops because this subset contains no gold UNKNOWN instances. These results show that the effect of semantic reward varies across ProntoQA hop levels rather than increasing monotonically with reasoning length.

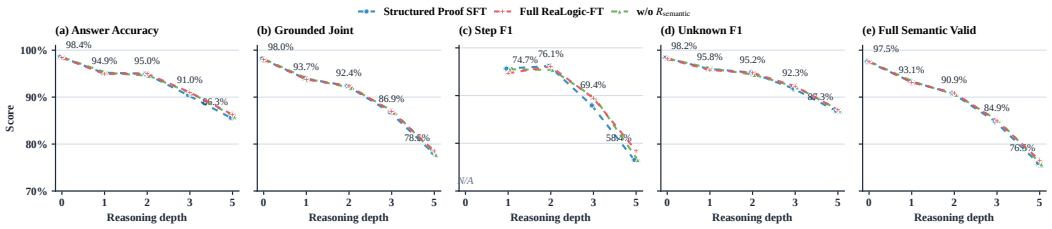

Figure 5: ProofWriter reward ablation results across reasoning depths. The comparison shows how removing the semantic reward affects answer-level and verifier-based proof metrics as the required derivations become longer.

Config.	Ckpt.	Hop	N	Acc.	G-Joint	Step-F1	U-F1	Full-Sem.
SFT	–	1-hop	668	96.71	95.66	90.40	95.59	95.66
	–	2-hop	650	94.62	92.31	89.62	92.54	92.15
	–	3-hop	609	93.43	90.80	91.95	88.10	90.64
	–	4-hop	559	**94.63**	**89.98**	**93.98**	**89.69**	**89.98**
	–	5-hop	450	95.33	84.44	92.97	N/A	**84.00**
w/o Sem.	100	1-hop	668	96.71 ± 0.26	95.51 ± 0.30	89.99 ± 1.05	95.77 ± 0.27	95.51 ± 0.30
	100	2-hop	650	94.51 ± 0.49	92.31 ± 0.46	89.94 ± 0.90	92.57 ± 0.94	92.15 ± 0.46
	100	3-hop	609	$\mathbf{93.87 \pm 0.41}$	91.24 ± 0.25	92.12 ± 0.38	$\mathbf{88.99 \pm 0.64}$	91.08 ± 0.25
	100	4-hop	559	93.74 ± 1.12	89.68 ± 0.85	93.57 ± 0.72	88.13 ± 2.27	89.68 ± 0.85
	100	5-hop	450	95.04 ± 1.22	84.15 ± 0.93	92.81 ± 0.60	N/A	83.48 ± 0.93
Full	100	1-hop	668	$\mathbf{97.01 \pm 0.40}$	$\mathbf{95.81 \pm 0.65}$	$\mathbf{91.16 \pm 1.47}$	$\mathbf{96.16 \pm 0.73}$	$\mathbf{95.81 \pm 0.65}$
	100	2-hop	650	$\mathbf{95.08 \pm 0.31}$	$\mathbf{92.77 \pm 0.27}$	$\mathbf{90.45 \pm 0.29}$	$\mathbf{93.32 \pm 0.44}$	$\mathbf{92.62 \pm 0.27}$
	100	3-hop	609	93.65 ± 0.41	$\mathbf{91.30 \pm 0.72}$	$\mathbf{92.29 \pm 0.16}$	88.43 ± 0.67	$\mathbf{91.13 \pm 0.72}$
	100	4-hop	559	94.28 ± 0.72	89.74 ± 0.37	93.56 ± 0.31	89.01 ± 1.29	89.74 ± 0.37
	100	5-hop	450	$\mathbf{95.63 \pm 0.34}$	$\mathbf{84.59 \pm 0.51}$	$\mathbf{93.13 \pm 0.27}$	N/A	$\mathbf{84.00 \pm 0.59}$

Table 16: ProntoQA reward ablation results by reasoning hop (%). SFT denotes Structured Proof SFT, Full denotes the complete ReaLogic-FT reward, and w/o Sem. denotes Full without the semantic reward R_{semantic}. SFT is evaluated once, while the RL configurations are reported as mean $\pm$ sample standard deviation over seeds 42, 43, and 44. The highest reported value at each hop is shown in bold. U-F1 is undefined for the five-hop subset because it contains no gold UNKNOWN instances.

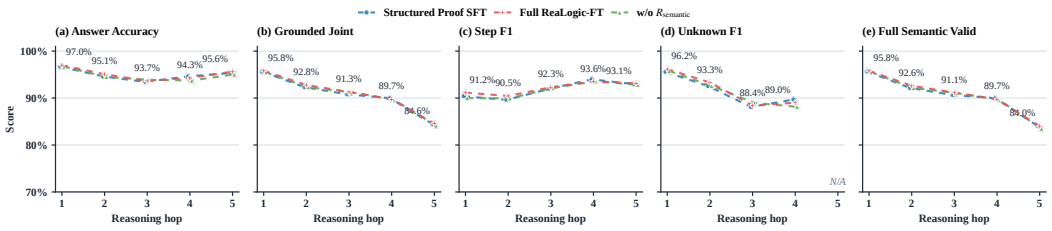

Figure 6: ProntoQA reward ablation results across reasoning hops. The comparison shows how removing the semantic reward affects answer-level and verifier-based proof metrics at different reasoning difficulties.

A.6.4 QUALITATIVE EXAMPLES AND FAILURE ANALYSIS

Table 17 reports dataset-level diagnostic counts for the seed-42 ReaLogic-FT models, using checkpoint 20 for ProofWriter and checkpoint 100 for ProntoQA. For conciseness, we present qualitative examples only from ProntoQA. The examples illustrate a verifier-confirmed recovery, correct open-world abstention, a missed derivation, and a correct answer accompanied by an invalid proof. They are diagnostic cases rather than estimates of failure frequency or cross-seed variability.

Dataset	Hard outputs	Answer ≠ HermiT	Correct answer, invalid proof	False Unknown	Missed Unknown
ProofWriter	232	152	80	69	78
ProntoQA	231	118	113	66	42

Table 17: Non-exclusive diagnostic counts for the seed-42 ReaLogic-FT models, evaluated at checkpoint 20 for ProofWriter and checkpoint 100 for ProntoQA. A hard output either disagrees with the HermiT-derived answer or has the correct benchmark answer without satisfying Strict Full-Sem. False Unknown and Missed Unknown are subsets of answer disagreement; therefore, the columns overlap and need not sum to the number of hard outputs. Rows for which semantic verification is unavailable are excluded.

Verifier-confirmed recovery. Listing 14 shows a four-hop example for which Structured Proof SFT abstains despite the existence of a derivation. ReaLogic-FT instead produces the complete chain from `lempus(Sally)` to `opaque(Sally)`. The deterministic verifier confirms that every cited premise and rule is grounded in the input theory, and HermiT accepts all four steps sequentially. The final conclusion enters the accepted state and supports the TRUE answer.

Listing 14: Abridged ProntoQA example `4hop_example460_in_context_example5`. ReaLogic-FT recovers a four-step proof missed by Structured Proof SFT. The displayed steps abbreviate the corresponding OWL Functional-Style assertions.

```
Relevant fact:
ClassAssertion(:lempus :Sally)

Relevant rules:
SubClassOf(:lempus :impus)
SubClassOf(:impus :wumpus)
SubClassOf(:wumpus :gorpus)
SubClassOf(:gorpus :opaque)

Query:
ClassAssertion(:opaque :Sally)

Structured Proof SFT:
<reasoning></reasoning>
<answer>Unknown</answer>

ReaLogic-FT:
Step 1:
  lempus(Sally) => impus(Sally)
Step 2:
  impus(Sally) => wumpus(Sally)
Step 3:
  wumpus(Sally) => gorpus(Sally)
Step 4:
  gorpus(Sally) => opaque(Sally)
<answer>True</answer>
```

Correct open-world abstention. Listing 15 illustrates a correct UNKNOWN prediction. The theory states that Alex is an `impus` and a `vumpus`. Its only rule involving `liquid` derives `not liquid` from `brimpus`, but the theory provides no path from either input fact to `brimpus`; it also contains no derivation of `liquid`. ReaLogic-FT therefore returns an empty reasoning block. HermiT independently confirms that the base ontology entails neither polarity, and the output satisfies both G-Joint and Full-Sem.

Listing 15: Abridged ProntoQA example `4hop_example452_test_example_unknown0`. ReaLogic-FT correctly abstains because neither the query nor its opposite follows from the input theory.

```
Relevant facts:
ClassAssertion(:impus :Alex)
```

```
ClassAssertion(:vumpus :Alex)

Only rule involving liquid:
SubClassOf(
  :brimpus
  ObjectComplementOf(:liquid)
)

No fact or rule derives:
ClassAssertion(:brimpus :Alex)
ClassAssertion(:liquid :Alex)

Query:
ClassAssertion(:liquid :Alex)

ReaLogic-FT:
<reasoning></reasoning>
<answer>Unknown</answer>

HermiT audit:
  O does not entail liquid(Alex)
  O does not entail not liquid(Alex)
```

Failure case I: false Unknown. For the query "Rex is earthy," ReaLogic-FT returns an empty reasoning block and UNKNOWN. However, the input fact that Rex is a `numpus` supports the five-step chain

$$\texttt{numpus} \rightarrow \texttt{brimpus} \rightarrow \texttt{zumpus} \rightarrow \texttt{impus} \rightarrow \texttt{yumpus} \rightarrow \texttt{not earthy}.$$

The correct answer is therefore FALSE. HermiT independently assigns the same semantic label, showing that this is a model error rather than a benchmark–verifier disagreement. The output receives the incorrect-answer penalty and, because a non-direct proof is required but absent, the minimum semantic proof reward with $Q_g = 0$. No separate Unknown-specific reward is used.

Failure case II: correct answer with rejected reasoning. For the query "Fae is not dull," the model correctly predicts FALSE and generates a chain ending in `dull(Fae)`. Its first step, however, cites the unsupported rule

$$\texttt{lempus} \rightarrow \texttt{lorpus},$$

whereas the input theory contains

$$\texttt{jompus} \rightarrow \texttt{lorpus}.$$

The sequential verifier rejects the first conclusion. Because `lorpus(Fae)` is not added to the accepted state, every subsequent step is blocked by an unavailable premise. The final answer is correct, but the output fails both G-Joint and Full-Sem. This case demonstrates why answer accuracy alone cannot establish proof faithfulness.

Together, these examples distinguish final-answer correctness from grounded and semantically verified reasoning. They also illustrate the role of sequential verification: accepted conclusions may support later steps, whereas rejected conclusions cannot propagate through the proof. This motivates reporting Acc., G-Joint, Step-F1, U-F1, and Full-Sem. separately.

A.7 LIMITATIONS AND FUTURE WORK

Expressivity of the target logic. ReaLogic-FT operates on the strict $\mathcal{ALC}$ profile described in Section A.1.4. This choice keeps entailment decidable and lets HermiT certify every step, but it excludes constructs that appear in richer ontologies, such as role hierarchies, transitive roles, cardinality restrictions, and datatypes. Instances of ProofWriter whose rules fall outside the profile are rejected rather than approximated, so the reported results characterize the supported fragment rather than unrestricted natural-language reasoning. Extending the framework to $\mathcal{SROIQ}$ or to selected OWL 2 profiles would widen coverage, at the cost of higher reasoning complexity in the reward loop.

Proof search. The policy emits a single derivation per rollout, and the complete ReaLogic-FT reward evaluates that derivation as generated. No search is performed over alternative orderings or intermediate lemmas. Beam search over step candidates, or tree search that expands a proof prefix and scores continuations with the step-level verifier, would let the model reach derivations it currently cannot. Both require invoking the reasoner on partial proofs, which the present interface evaluates only after a complete candidate has been produced.

Verification cost. Verification requires determining the semantic answer of the base ontology and checking entailment for each eligible proof step, so its cost grows with both the number of rollouts and the derivation length. This currently bounds the practical values of G and the proof-length limit. Caching entailment results across candidates that share prefixes, and batching reasoner calls, would make longer derivations and larger groups affordable.

Benchmark scope. Both benchmarks are synthetically generated with controlled vocabularies, which makes exact verification possible but limits linguistic variety. Applying the pipeline to human-written theories, where formalization itself becomes ambiguous, would test whether assertion-level alignment remains as reliable as it is here.

Scope of auditability. The reasoner checks each generated step against a given formal theory: the executable ontology and canonical query are hidden verifier metadata constructed during data preparation. ReaLogic-FT therefore certifies that a proof is valid with respect to this formalization, not that the formalization faithfully represents the source text. Extending auditing to settings without a trusted formalization, for example by verifying the model's own formalization of the full theory, remains future work.

Comparison to other neuro-symbolic and self-training approaches. All reported comparisons are internal to the three post-training stages. We do not compare against inference-time solver pipelines such as Logic-LM Pan et al. (2023) and LINC Olausson et al. (2023). In both systems the language model acts as a semantic parser and an external solver returns only a verdict over TRUE, FALSE, and UNKNOWN, so no proof is produced that our step-level and semantic-validity criteria could score, and only answer-level metrics would be directly comparable. We also do not compare against self-training alternatives such as STaR Zelikman et al. (2022), which filters sampled rationales by final-answer correctness and is itself an approximation to a policy-gradient objective; relative to that binary signal, our reward supplies finer-grained verifier feedback on grounding, step entailment, and accepted-state answer support. Direct empirical comparison with both families remains future work.