REASONSTL: A LEARNED FORMAL COMPILER FOR NATURAL LANGUAGE TO SIGNAL TEMPORAL LOGIC

Anonymous authors
Paper under double-blind review

ABSTRACT

Compiling natural-language requirements into Signal Temporal Logic (STL) requires exact recovery of temporal scope, predicate selection, logical structure, and numerical constraints. Errors in any of these decisions alter the executable semantics. We formulate this problem as direct compilation from a clear requirement and a declared predicate interface, evaluating both translation accuracy and end-to-end latency. We introduce REASONSTL, a locally deployed learned formal compiler that integrates semantic structure prediction, deterministic numerical tools, typed AST construction, and staged validation. Outcome-bounded process rewards tie stage-level credit to final-formula correctness, and prefix masking suppresses downstream credit after the first invalid step. The resulting trajectories expose tool execution, value placement, and reward assignment for inspection. We further introduce STL-BENCH, a bilingual, computation-aware benchmark with scenario-held-out evaluation and 500 independently authored expert requirements. REASONSTL-8B achieves 0.700/0.680 canonical accuracy on held-out English/Chinese scenarios and 0.780/0.752 expert semantic accuracy on Manual-500. On one H20, it attains 0.724 aggregate Manual-500 canonical accuracy with 3.88 s mean latency, compared with 0.524 and 3.87 s for Claude-Opus-4.7 under the reported serving configurations. These results establish a strong local accuracy–latency trade-off for open-loop NL-to-STL compilation.

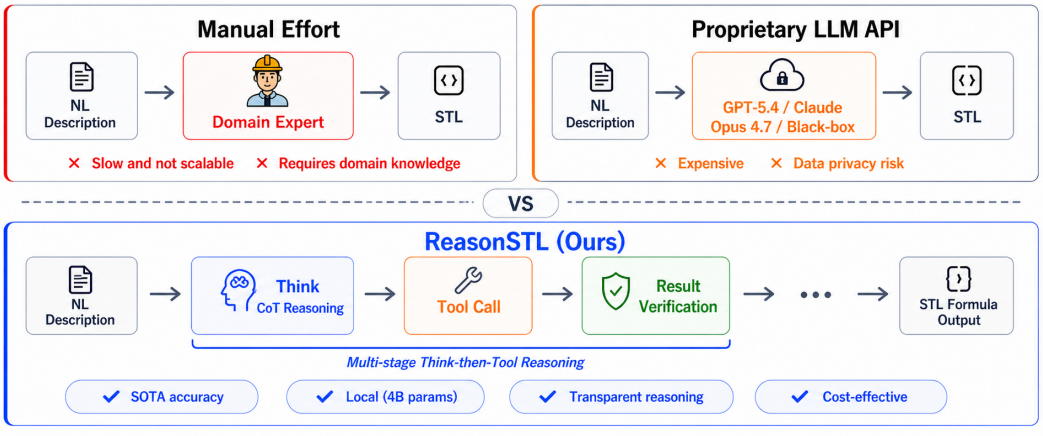


Figure 1: Comparison of three Approaches for translating NL descriptions into STL specifications.

1 INTRODUCTION

Signal Temporal Logic (STL) is a formal language for specifying temporal requirements over real-valued signals. It supports monitoring, verification, planning, and control in autonomous and cyber-physical systems (CPS) (Maler & Nickovic, 2004; Bartocci et al., 2018; Belta & Sadraddini, 2019). Engineers, however, commonly express requirements in natural language. Converting those requirements into STL remains labor-intensive because a valid formula must preserve logical scope,

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.

temporal intervals, signal identities, comparison directions, thresholds, and physical units. A semantically incorrect specification may remain syntactically valid and executable, leaving the error undetected.

We study the compilation stage of the specification pipeline, assuming a clear requirement and a declared set of measurable signals or predicates. The objective is accurate STL generation at low end-to-end latency. This setting separates three problems. Requirement elicitation determines user intent. Perception and state estimation determine physical-world truth. NL-to-STL compilation determines the temporal relations imposed over the declared interface. The compilation challenge persists under clear input. A translator may attach a duration to the wrong subformula, confuse a threshold state with a transition event, or preserve every numerical value while changing temporal scope.

Existing NL-to-STL systems usually treat the task as direct sequence prediction or final-answer optimization (He et al., 2022; Fang et al., 2025; 2026b). This design places semantic parsing, arithmetic, unit conversion, and serialization inside one unconstrained generation step. The resulting sequence can be grammatically valid even when a time interval attaches to the wrong operator or a converted quantity enters the wrong predicate. Proprietary models also face numerical errors and limited visibility into intermediate decisions. External serving adds deployment cost and data-privacy considerations.

A formal compiler should expose the decisions that determine executable meaning. It should separate semantic choices from deterministic computation, represent scope explicitly, reject interface violations before serialization, and preserve traceable links between normalized values and their destination fields. We implement these properties through a learned policy coupled to typed execution. The language model predicts semantic structure and selects declared predicates. Deterministic tools normalize quantities, a typed abstract syntax tree (AST) records operator scope, and validators check the resulting formal object.

We instantiate this formulation in REASONSTL. The model interleaves semantic reasoning with deterministic calls for duration parsing, unit conversion, arithmetic evaluation, and time-difference computation, then emits a typed STL AST. We formulate training as reinforcement learning over structured reasoning-and-tool trajectories. Our outcome-bounded process reward gives an intermediate step full credit when the step and its prefix are valid and the final formula is correct. A prefix mask removes policy credit from all stages following an invalid step. This reward makes the basis for intermediate credit explicit: local validity, prefix validity, and final-formula correctness. It trains the semantic decisions and tool calls under a common formal objective.

Our contributions are threefold. **i)** We formulate clear-input NL-to-STL compilation, which produces an executable STL specification from an unambiguous requirement and a declared predicate interface. Interactive clarification and environment-specific perception constitute separate pipeline stages. **ii)** We introduce REASONSTL, a learned formal compiler trained with outcome-bounded process-reward RL, which aligns local credit for typed tool use and tree construction with global formal correctness. **iii)** We characterize its accuracy–latency trade-off through scenario-held-out evaluation, independently authored Manual-500 requirements, expert semantic assessment, reward and component studies, failure analysis, and end-to-end serving measurements. These evaluations test both translation fidelity and the cost of local deployment.

2 RELATED WORK

Natural language to formal specifications. STL provides metric temporal operators over continuous signals and supports executable monitoring of cyber-physical systems (Maler & Nickovic, 2004; Bartocci et al., 2018). Early language-based methods learned mappings from instructions to LTL or related symbolic task representations (Gopalan et al., 2018; Brunello et al., 2019). NL2TL introduced a modular translation pipeline (Chen et al., 2023), while Lang2LTL connected language interpretation with grounded propositions for robotic tasks (Liu et al., 2023). NL2LTL provided a reusable translation package (Fuggitti & Chakraborti, 2023), and data-efficient training reduced the supervision required for robot task specification (Pan et al., 2023). Related work has also studied translation into first-order logic (Yang et al., 2024) and validation through model-checker feedback (Mendoza et al., 2024). These studies establish formal translation as a learnable problem.

Most focus on discrete propositions and do not jointly evaluate metric intervals, physical units, and numerical computation.

Natural language to STL. DeepSTL introduced a grammar-guided English-to-STL corpus and sequence-to-sequence baseline (He et al., 2022). Mao et al. applied Llama2 to STL translation and demonstrated the value of pretrained language models (Mao et al., 2024). KGST combines external knowledge, retrieval, and model specialization to improve translation quality (Fang et al., 2025). RESTL introduces curriculum learning and multi-aspect reinforcement learning for final STL generation (Fang et al., 2026b). Systematic decomposition has been used to resolve intermediate translation decisions (Mohammadinejad et al., 2024). Interactive systems instead refine ambiguous requirements through user feedback (Cosler et al., 2023; Fang et al., 2026a). REASONSTL studies a complementary setting in which the intended reading and predicate interface are fixed before compilation. It evaluates temporal structure, predicate selection, numerical normalization, and final STL construction under one executable contract.

Structured generation and tool use. Grammar-constrained decoding restricts outputs to valid structured languages (Geng et al., 2023). Such constraints improve syntactic validity, while temporal attachment and predicate semantics still require model decisions. ReAct interleaves language reasoning with external actions (Yao et al., 2022), and Toolformer trains language models to invoke external APIs (Schick et al., 2023). REASONSTL specializes this paradigm for formal compilation. The policy selects numerical operations and semantic operands, deterministic tools execute them, and each result is bound to a typed AST field. This structure makes tool execution and value incorporation independently verifiable.

Process-reward reinforcement learning. Process supervision assigns feedback to intermediate reasoning steps (Lightman et al., 2023), and process-reward learning extends this principle to broader reasoning tasks (Yao et al., 2026). Curriculum learning provides a general mechanism for organizing progressively difficult supervision (Bengio et al., 2009). DeepSeekMath introduced group-relative policy optimization for mathematical reasoning (Shao et al., 2024), while group-sequence optimization extends relative objectives across generated trajectories (Zheng et al., 2025). Search-R1 applies reinforcement learning to reasoning with external search tools (Jin et al., 2025). Formal compilation presents a different credit-assignment structure: a locally valid computation occurs inside an incorrect formula, and later decisions depend on an invalid prefix. REASONSTL therefore bounds stage rewards by final-formula correctness and masks process credit after the first invalid dependency.

3 PRELIMINARIES AND PROBLEM FORMULATION

3.1 SIGNAL TEMPORAL LOGIC

Let $\mathbf{s}_{0:T} = (s_0, \dots, s_T)$ be a finite discrete-time signal with $s_t \in \mathbb{R}^d$. An atomic predicate π^μ is induced by a real-valued function μ and holds at time t when $\mu(s_t) \geq 0$. We consider the STL grammar

$$\phi ::= \top \mid \pi^\mu \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathbf{U}_{[a,b]} \phi_2, \qquad 0 \leq a \leq b. \tag{1}$$

The satisfaction relation is defined recursively. In particular,

$$\mathbf{s}, t \models \pi^\mu \iff \mu(s_t) \geq 0, \tag{2}$$

$$\mathbf{s}, t \models \phi_1 \mathbf{U}_{[a,b]} \phi_2 \iff \exists t' \in [t+a, t+b] : \mathbf{s}, t' \models \phi_2 \wedge \forall t'' \in [t, t'),\ \mathbf{s}, t'' \models \phi_1. \tag{3}$$

Eventually and always are derived as $\mathbf{F}_{[a,b]}\phi := \top \mathbf{U}_{[a,b]} \phi$ and $\mathbf{G}_{[a,b]}\phi := \neg \mathbf{F}_{[a,b]} \neg\phi$. The former requires satisfaction at some point in the interval. The latter requires satisfaction throughout it. All intervals use the canonical time unit declared by the interface. Appendix A defines threshold-transition and past-time constructs and gives the complete typed AST convention.

3.2 CLEAR-INPUT NL-TO-STL COMPILATION

Let q be a natural-language requirement with one fixed intended reading. The declared interface $\mathcal{V}$ specifies every admissible signal or predicate together with its type and canonical unit. The compilation problem is to learn $f_\theta : (q, \mathcal{V}) \mapsto \hat{\phi}$, where $\hat{\phi} \in \Phi(\mathcal{V})$ is an executable STL formula whose leaves use only elements of $\mathcal{V}$. Correct compilation must recover the operator tree, temporal scope, predicate identities, comparison directions, normalized intervals, and normalized thresholds.

Table 1: Compilation obligations and their realization in the typed AST.

Requirement evidence	Compiler decision	AST realization
Temporal cue	Operator, anchor, and interval scope	Temporal node with normalized $[a, b]$
Logical relation	Conjunction, disjunction, implication, or negation	Typed parent-child structure
Signal description	Declared predicate and comparison direction	Interface-valid predicate leaf
Quantity and unit	Numerical operation and canonical value	Interval bound or threshold field
State or transition wording	Persistent state or threshold-crossing event	Predicate leaf or edge operator

Table 2: Evaluation partitions, isolation controls, and generalization targets. Manual-500 contains independently authored requirements.

Partition	Composition	Isolation control	Primary role
Full STL-BENCH	28,880 bilingual pairs; 6 domains, 33 scenarios, 41 signals	Validated and deduplicated before splitting	Training and controlled evaluation
Scenario-held-out	21/6/6 train/validation/test scenario IDs per language	All bilingual variants of held-out IDs excluded	Scenario and structure transfer
Manual-500	250 English and 250 Chinese expert-written pairs	Included only in testing	Authorship transfer and SemAcc

The contract assumes that requirement elicitation has resolved incompatible readings and missing quantities. It also assumes that the interface defines the measurable vocabulary needed by the requirement. REASONSTL selects and composes these declared predicates. A visual model, state estimator, or semantic adapter supplies their truth values in a downstream system. This separation fixes the semantic inputs to compilation and permits the same compiler to serve physical systems with compatible interfaces.

Table 1 links requirement evidence to the semantic decisions realized in the typed AST. The compiler must resolve temporal attachment, tree scope, duration roles, and the distinction between threshold states and transition events before serialization. Each training item has one frozen expert-authored canonical target $\phi^\star$. Canonical Accuracy (CanAcc) requires an exact AST match after deterministic normalization of operators, intervals, predicates, and thresholds; its scope excludes general STL equivalence. Model-blind expert Semantic Accuracy (SemAcc) accepts semantically correct non-canonical formulas and provides a reference-independent assessment. Format Accuracy (FmtAcc)

4 STL-BENCH

STL-BENCH evaluates the structural and numerical obligations defined in Section 3. It contains 28,880 bilingual NL-to-STL pairs across autonomous driving, robotics, industrial control, environmental monitoring, electrical systems, and aerospace. These domains comprise 33 CPS scenarios and 41 declared signals. Seventy-three percent of the samples require deterministic computation, including 7,337 duration normalizations, 5,161 unit conversions, and 4,527 arithmetic evaluations. The requirements cover temporal nesting, multiple predicates, relative-time anchoring, threshold events, unit conversion, duration normalization, and arithmetic expressions. Prior corpora target narrower settings. DeepSTL generates grammar-controlled English formulas over symbolic predicates with string output (He et al., 2022), and STL-DivEn broadens linguistic diversity through model-assisted augmentation and validation (Fang et al., 2025). STL-BENCH adds bilingual requirements, domain-grounded signals, annotated deterministic computation, and typed AST supervision under one declared interface.

The benchmark follows label-first construction. Each formal label fixes the operator tree, temporal intervals, predicates, units, and required computations before language realization. English and Chinese requirements are then produced against the same declared interface. The formal label therefore

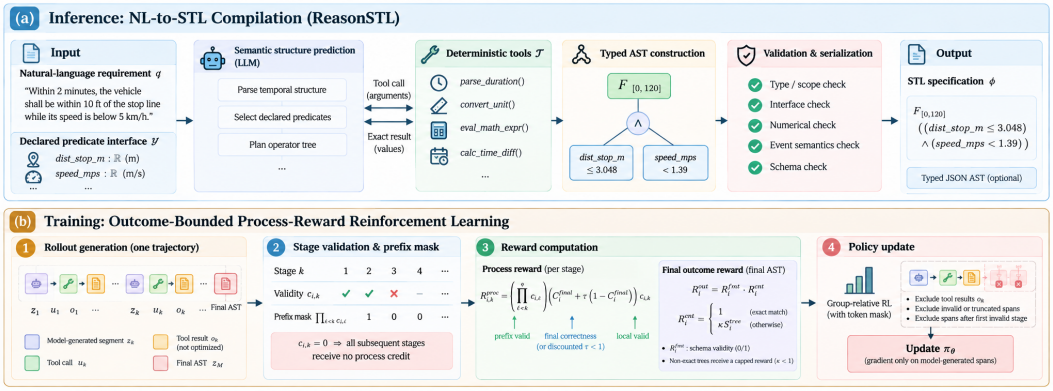


Figure 2: Inference and training framework of REASONSTL.

remains fixed during language realization. Rule-based validation checks schema, arity, intervals, signal names, tool execution, and alignment between computed values and their destination fields. Exact, symbolic, and high-similarity duplicates are removed before splitting.

The scenario-held-out split assigns 21/6/6 scenario IDs per language to train, validation, and test. It evaluates unseen combinations of engineering context, declared signals, and temporal structure. Manual-500 contains requirements independently written by six domain specialists without generation templates. A second specialist cross-checks each pair before freezing. The nearest generated-training neighbor has embedding similarity at most 0.20 for every Manual-500 item, with a mean nearest-neighbor similarity of 0.11. These partitions test complementary forms of generalization under one fixed compilation contract. Appendix B details construction, computation annotations, auditing, and split manifests.

5 REASONSTL: A LEARNED FORMAL COMPILER

REASONSTL implements the contract in Section 3 through typed generation, stage validation, and outcome-bounded process-reward training. Figure 2(a) traces compilation from input to STL. The policy predicts semantic structure, calls deterministic tools for numerical normalization, and emits a typed AST that a serializer turns into a formula. Figure 2(b) aligns training credit with the same typed stages. Each stage exposes an inspectable decision, so reward and validation attach to concrete compilation steps.

5.1 TYPED COMPILATION WITH DETERMINISTIC TOOLS

Given $(q, \mathcal{V})$, the policy produces $y = (z_1, u_1, o_1, \ldots, z_M)$, where z_m is a model-generated segment, u_m an optional tool call, and o_m its deterministic result. In Figure 2(a), the policy predicts temporal structure, declared predicates, and the operator tree, invoking tools for exact computation. Its final segment is a typed JSON AST; a deterministic serializer emits $\hat{\phi}$. The tool set is

$$\mathcal{T} = \{\texttt{parse_duration}, \texttt{convert_unit}, \texttt{eval_math_expr}, \texttt{calc_time_diff}\}. \quad (4)$$

The policy determines temporal structure, logical composition, predicates, tool choice, and arguments. Tools only execute numerical normalization. Returned values bind to interval fields or predicate thresholds, separating execution from incorporation checks. Before serialization, the AST exposes scope for type, arity, interval, and interface validation. Learned decisions, exact computations, and typed construction thus remain separately inspectable.

5.2 STAGE VALIDATION AND FAILURE CONTAINMENT

The obligations in Table 1 induce checks over the typed trajectory. Structural checks cover operator type, arity, interval order, and scope; interface checks cover declared signals, comparators, and units; tool checks cover schema, execution, and value placement. Edge operators require an explicit transition cue. These checks expose contract violations without deciding general STL equivalence.

Algorithm 1 Outcome-bounded compilation training

Require: Batch of $(q, \mathcal{V}, \phi^\star)$, old policy $\pi_{\theta_{\text{old}}}$, group size G, process factor τ, tree cap κ

for each $(q, \mathcal{V}, \phi^\star)$ **do**
  Sample G compiler rollouts from $\pi_{\theta_{\text{old}}}$
  for each rollout i **do**
    Parse stage spans and final AST $\hat{\phi}_i$; compute R_i^{fmt}
    Validate stages to obtain $\{c_{i,k}\}_{k=1}^{K_i}$
    Compute C_i^{final}, S_i^{tree}, and R_i^{out}
    Compute each $R_{i,k}^{\text{proc}}$ using Equation (5)
  end for
  Normalize outcome and aligned-stage rewards within the group
  Map advantages to stage and final-AST tokens; construct $M_{i,t}$
  Accumulate the masked group-relative objective
end for
Update π_θ with gradient clipping

During training, frozen annotations complete the stage indicators and identify the earliest invalid dependency. A valid conversion can retain local credit after a later structural error, while a malformed call and every subsequent stage receive none. At inference, reference-free checks precede deterministic serialization. Both phases share the typed execution path without treating validation as semantic proof.

5.3 Outcome-Bounded Process-Reward Reinforcement Learning

Figure 2(b) summarizes group-relative reinforcement learning. Rollout i contains K_i intermediate stages and final AST $\hat{\phi}_i$. For an annotated training item, validation assigns $c_{i,k} \in \{0, 1\}$ from tool execution, value incorporation, predicate alignment, and event selection. Let R_i^{fmt} indicate schema validity, $C_i^{\text{final}} = \mathbb{I}[\text{can}(\hat{\phi}_i) = \text{can}(\phi_i^\star)]$ an exact canonical match, and $S_i^{\text{tree}} \in [0, 1]$ tree similarity. We set $R_i^{\text{cnt}} = C_i^{\text{final}} + (1 - C_i^{\text{final}})\kappa S_i^{\text{tree}}$ for $0 < \kappa < 1$ and $R_i^{\text{out}} = R_i^{\text{fmt}} R_i^{\text{cnt}}$. Thus exact valid ASTs receive one, and non-exact trees remain capped at κ.

A valid local computation can supply a learning signal even when the final specification is incorrect. We bound that credit using final correctness and prefix validity:

$$R_{i,k}^{\text{proc}} = \left(\prod_{\ell < k} c_{i,\ell} \right) \left(C_i^{\text{final}} + \tau(1 - C_i^{\text{final}}) \right) c_{i,k}, \qquad 0 < \tau < 1, \tag{5}$$

where $c_{i,k}$ suppresses an invalid stage, the prefix product suppresses every subsequent stage, and τ retains discounted local credit when the final AST is non-exact. The tree cap κ separates non-exact outputs from exact matches. The process factor τ discounts valid local decisions in unsuccessful compilations. The prefix product prevents later stages from receiving process credit after an invalid step. Each factor thus corresponds to an explicit validation outcome.

Outcome and aligned-stage rewards are normalized separately within each rollout group. Tokens in intermediate stage k receive its process advantage. Final-AST tokens receive the outcome advantage. The policy update excludes deterministic tool results, invalid or truncated spans, and spans blocked by an earlier invalid stage. The binary token mask $M_{i,t}$ records these exclusions. This mapping applies each advantage to the model-generated decision span it evaluates and excludes environmental observations from the policy gradient. Algorithm 1 summarizes training; Appendix C defines the mask and full objective.

5.4 A Complete Compilation Trace

Figure 2(a) follows "within 2 minutes, the ground vehicle shall be within 10 ft of the stop line while its speed is below 5 km/h," given `dist_stop_m` and `speed_mps`. The policy builds an eventuality over their conjunction; tools return 120 s, 3.048 m, and 1.39 m/s. Validation precedes serialization to

$$\mathbf{F}_{[0,120]}\big((\texttt{dist_stop_m} \leq 3.048) \wedge (\texttt{speed_mps} < 1.39)\big). \tag{6}$$

Figure 2(b) trains the same trajectory: valid stages receive prefix- and outcome-bounded credit, the first invalid stage blocks later credit, and non-exact trees remain below exact matches. Tool results receive no policy gradient. Inference reuses the typed tool and serialization path without reward computation. Appendices C and E give implementation details and errors.

6 EXPERIMENTS

We evaluate translation fidelity, the contribution of compiler components, and end-to-end serving cost. Canonical and expert semantic accuracy assess generalization; training controls examine reward design; serving measurements quantify the local accuracy–latency trade-off. A mismatch audit identifies the remaining semantic limitations.

6.1 EXPERIMENTAL SETUP

We evaluate Qwen3-4B and Qwen3-8B local models on DeepSTL and STL-BENCH. *Direct* uses task-specific training with direct output generation. *ReasonSTL* uses the complete reasoning-and-tool compiler. Proprietary models serve as prompted reference systems. They receive the same task definition, declared signal interface, JSON schema, tools, canonicalization instructions, and an expanded demonstration set. Their weights remain fixed. Local same-backbone variants support component attribution.

All outputs follow the same extraction, JSON repair, parsing, schema validation, canonicalization, and scoring path. Repair is restricted to serialization and preserves operators, predicates, intervals, and subtrees. Local models use deterministic decoding. For every reported system, three STL-qualified domain experts independently assess all 500 anonymized Manual-500 predictions under model-blind evaluation. They resolve disagreements by consensus, accept equivalent noncanonical formulas, and retain all 250 items per language in the SemAcc denominator. The evaluators are independent of the paper authors and Manual-500 writers. A separate three-annotator audit labels all canonical mismatches of REASONSTL-4B. Training uses eight H20 GPUs; local serving uses BF16 vLLM on one H20 with concurrency eight.

6.2 ACCURACY AND GENERALIZATION

DeepSTL isolates symbolic English translation and lacks the computation annotations required by the complete compiler, so this evaluation uses Direct-4B. Table 3 reports every evaluated system in the main paper. Direct-4B reaches 0.8143 CanAcc and 0.8257 FmtAcc. It exceeds Claude-Opus-4.7 by 15.4 percentage points in CanAcc and the prior reinforcement-learning system RESTL by 21.6 points. This result establishes a strong local translation baseline before typed reasoning and tools are introduced.

Table 3: Complete DeepSTL results. Scores are fractions; bold marks the best result.

Method	CanAcc	FmtAcc
Direct-4B	**.8143**	**.8257**
Claude-Opus-4.7	.6603	.6691
RESTL (Fang et al., 2026b)	.5985	.6327
DeepSeek-V4	.5685	.5758
GPT-5.4	.5150	.5220
Kimi-K2.6	.5048	.5113
KGST (Fang et al., 2025)	.4538	.4939
DeepSTL (He et al., 2022)	.2002	.2916
Qwen3-4B	.1131	.1631

The broader evaluation tests scenario transfer and independently authored requirements. Table 4 reports every evaluated local and hosted system, including format accuracy where available. The scenario entries use English CanAcc/FmtAcc followed by Chinese CanAcc/FmtAcc; Manual-500 entries use English/Chinese scores.

On scenario-held-out STL-BENCH, REASONSTL-4B improves over Direct-4B by 9/14 percentage points in English/Chinese CanAcc. The 8B compiler reaches 0.700/0.680 CanAcc and 0.820/0.780 FmtAcc, improving over Direct-8B by 22/24 and 11.6/2.0 percentage points, respectively. It also exceeds the prompted hosted systems, reaching more than twice the scenario CanAcc of DeepSeek-V4 and Claude-Opus-4.7 under the same interface and tools.

On Manual-500, REASONSTL-4B reaches 0.556/0.528 CanAcc against 0.432/0.400 for Direct. The 8B compiler reaches 0.740/0.708 CanAcc and 0.780/0.752 SemAcc. Expert review accepts valid noncanonical formulas and recovers an additional 4.0/4.4 percentage points. Its SemAcc exceeds

Table 4: Complete scenario-held-out and Manual-500 results as fractions. Scenario entries report CanAcc/FmtAcc for English, then Chinese. Manual entries report English/Chinese scores. Bold marks the best reported result; – indicates an unreported result.

System	Scenario CanAcc/FmtAcc	Manual CanAcc	Manual SemAcc
Qwen3-4B	.070/.110; .090/.130	–	–
Direct-4B	.420/.500; .330/.410	.432/.400	.440/.440
REASONSTL-4B	.510/.560; .470/.490	.556/.528	.592/.572
Direct-8B	.480/.704; .440/.760	.504/.480	.532/.480
REASONSTL-8B	**.700/.820**; **.680/.780**	**.740/.708**	**.780/.752**
DeepSeek-V4	.310/.330; .270/.350	.528/.484	.580/.560
GPT-5.4	.220/.240; .190/.200	–	–
Claude-Opus-4.7	.260/.280; .270/.280	.544/.504	.592/.580
Kimi-K2.6	.170/.200; .190/.200	–	–

Table 5: End-to-end serving measurements. Mean and p95 are seconds; Manual CanAcc aggregates English and Chinese.

System	Deploy	QPS	Mean	p95	Tokens/query	Manual CanAcc
Direct-4B	H20	5.90	1.32	2.08	99	.416
REASONSTL-4B	H20	2.20	3.62	4.94	382	.542
Direct-8B	H20	4.90	1.71	2.53	103	.492
REASONSTL-8B	H20	2.00	3.88	5.51	402	.724
DeepSeek-V4	API	0.80	8.30	10.12	688	.506
Claude-Opus-4.7	API	2.20	3.87	5.45	572	.524

both hosted systems by at least 0.18/0.17 in English/Chinese. Every Manual-500 item has training-neighbor similarity at most 0.20, with a mean of 0.11.

6.3 COMPONENT EVIDENCE AND EFFICIENCY

Table 6 reports method outcomes directly; Appendix D records all SFT and RL update counts. The SFT cold start reaches 0.345/0.270 CanAcc, and Direct reaches 0.420/0.330. Reasoning without tools obtains 0.446/0.372, while tools with final reward obtain 0.482/0.438. Among matched-budget reward controls, outcome bounding raises 0.469/0.423 to 0.501/0.461, and prefix masking reaches 0.510/0.470. The final-reward control remains below the full method despite a larger training budget. Under matched training, typed JSON improves CanAcc over flat STL by 2.0/5.0 percentage points. Sensitivity tests vary CanAcc by at most 0.006 over the tested τ, κ, and tolerance settings in Appendix D.

Table 6: Component results on scenario-held-out STL-BENCH.

Variant	EN	ZH
SFT only	.345	.270
Direct	.420	.330
Think, no tools	.446	.372
Tools + final reward	.482	.438
Unbounded process reward	.469	.423
Bounded, no prefix mask	.501	.461
Full	**.510**	**.470**
Flat STL, matched	.490	.420

Table 5 measures generation, tool execution, parsing, validation, and serialization. Relative to Direct, compilation improves Manual CanAcc from 0.416 to 0.542 at 4B and from 0.492 to 0.724 at 8B. Mean local latency rises from 1.32 to 3.62 s and from 1.71 to 3.88 s, respectively. REASONSTL-4B matches Claude-Opus-4.7's measured throughput of 2.20 queries/s. The 8B compiler has similar mean latency and improves Manual CanAcc by 20.0 percentage points. Cross-infrastructure measurements characterize the reported accuracy–latency trade-off without isolating hardware efficiency.

Table 7: Non-exclusive audit of canonical mismatches from the 4B full compiler. Shares are computed over all canonical mismatches; one prediction may receive multiple tags.

Audit category	**Share**	**Observed failure type**
Temporal scope or nesting	33%	Incorrect attachment, anchor, or persistence scope
Operator or event selection	21%	State/event confusion or wrong temporal operator
Predicate alignment	18%	Signal, comparator, or interface mismatch
Numerical or tool specification	17%	Conversion, argument, or value-placement error
Alternative or reference-related	8%	Valid noncanonical form or reference issue
Residual JSON or schema invalidity	3%	Unrecoverable serialization or typing failure

6.4 Failure Anatomy

Table 7 identifies temporal scope and nesting as the most frequent mismatch tag. Consider: "Whenever an alarm occurs, the valve must open within 5 seconds and remain open for 20 seconds." The reference is $\mathbf{G}(a \rightarrow \mathbf{F}_{[0,5]}\mathbf{G}_{[0,20]}v)$. A representative prediction is $\mathbf{G}(a \rightarrow \mathbf{F}_{[0,5]}v)$. The prediction preserves the trigger, declared predicate, and deadline. It drops the nested persistence obligation and reduces a 20-second invariant to one-time satisfaction. The formula remains syntactically valid and retains the same predicates. Inspecting its tree scope reveals the missing persistence constraint.

Residual JSON or schema invalidity accounts for 3% of mismatches. Raw JSON, post-repair JSON, and final schema validity reach 97.4%, 99.8%, and 99.5%. Serialization is therefore a minor error source. Temporal attachment and nesting remain the most frequent semantic errors despite high parse validity. Structural validity alone does not establish fidelity to the requirement. Appendix E provides relative-time, event-selection, numerical, and alternative-valid cases.

6.5 Scope, Validity, and Deployment

The results apply to the clear-input contract: intent and the measurable interface are fixed before ReasonSTL runs. Ambiguity requires prior elicitation, and predicate truth requires downstream sensing or state estimation. CanAcc may reject valid alternatives, so full-set expert SemAcc supplies a reference-independent complement, though formal equivalence proof remains outside the protocol. The evidence covers six CPS domains, two languages, and two local model sizes; broader interfaces and scales require further evaluation. Deployment additionally requires monitor adapters, system-model consistency checks, and application-specific validation.

7 Conclusion

We presented ReasonSTL, a learned formal compiler that translates clear natural-language requirements over declared predicate interfaces into executable STL through semantic structure prediction, deterministic numerical tools, typed AST construction, and staged validation. Its outcome-bounded process-reward objective credits valid intermediate decisions, suppresses credit after invalid dependencies, and ties local supervision to final-formula correctness. We also introduced STL-Bench, a bilingual computation-aware benchmark spanning six CPS domains, numerical constraints, and compositional temporal structures. ReasonSTL improves over direct generation at both evaluated model scales and achieves the strongest scenario-held-out and Manual-500 results among the evaluated systems. Serving measurements further establish a favorable accuracy–latency trade-off for local deployment. Temporal attachment and nesting remain the primary failure modes, motivating future work on broader semantic equivalence checking, wider predicate-interface coverage, and adaptive tool invocation.

AI USE STATEMENT

Generative AI tools were used to realize natural-language requirements from prespecified formal labels during synthetic benchmark construction, as described in Appendix B, and to assist with manuscript organization and language editing. Their use excluded benchmark-label selection, reported-measurement generation, and final scientific decision-making. The generated benchmark items underwent deterministic validation and human auditing. The authors reviewed all AI-assisted text, verified technical claims against experimental records and cited literature, and take responsibility for the final content.

ETHICAL CONSIDERATIONS

This work studies formal-specification translation and uses no personally identifiable information. Generated specifications may be incorrect despite passing structural validation. They should therefore be independently checked before use in safety-critical systems. We follow the licenses and access conditions of all models, APIs, datasets, and software artifacts used in this study.

REPRODUCIBILITY

The supplementary material provides the benchmark splits, Manual-500 protocol, generation templates, validators and tools, baseline prompts, evaluation scripts, training configurations, raw outputs, and JSON-to-STL serializer. An anonymized implementation accompanies the submission.

REFERENCES

Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In *Lectures on Runtime Verification: Introductory and Advanced Topics*, pp. 135–175. Springer, 2018.

Calin Belta and Sadra Sadraddini. Formal methods for control synthesis: An optimization perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):115–140, 2019.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

Andrea Brunello, Angelo Montanari, and Mark Reynolds. Synthesis of ltl formulas from natural language texts: State of the art and research directions. In *26th International symposium on temporal representation and reasoning (TIME 2019)*, pp. 17–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019.

Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. Nl2tl: Transforming natural languages to temporal logics using large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 15880–15903, 2023.

Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *International Conference on Computer Aided Verification*, pp. 383–396. Springer, 2023.

Yue Fang, Zhi Jin, Jie An, Hongshen Chen, Xiaohong Chen, and Naijun Zhan. Enhancing transformation from natural language to signal temporal logic using llms with diverse external knowledge. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10446–10458, 2025.

Yue Fang, Zhi Jin, Jie An, Hongshen Chen, Xiaohong Chen, and Naijun Zhan. Clarifystl: An interactive llm agent framework for stl transformation through requirements clarification. *arXiv preprint arXiv:2605.01209*, 2026a.

Yue Fang, Zhi Jin, Jie An, Hongshen Chen, Xiaohong Chen, and Naijun Zhan. Restl: Reinforcement learning guided by multi-aspect rewards for signal temporal logic transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30682–30689, 2026b.

Francesco Fuggitti and Tathagata Chakraborti. Nl2ltl–a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 16428–16430, 2023.

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

Nakul Gopalan, Dilip Arumugam, Lawson LS Wong, and Stefanie Tellex. Sequence-to-sequence language grounding of non-markovian task specifications. In *Robotics: Science and Systems*, volume 2018, 2018.

Jie He, Ezio Bartocci, Dejan Ničković, Haris Isakovic, and Radu Grosu. Deepstl: from english requirements to signal temporal logic. In *Proceedings of the 44th International Conference on Software Engineering*, pp. 610–622, 2022.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. Lang2ltl: Translating natural language commands to temporal robot task specification. *arXiv preprint arXiv:2302.11649*, 2023.

Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *International symposium on formal techniques in real-time and fault-tolerant systems*, pp. 152–166. Springer, 2004.

Yuchen Mao, Tianci Zhang, Xu Cao, Zhongyao Chen, Xinkai Liang, Bochen Xu, and Hao Fang. Nl2stl: Transformation from logic natural language to signal temporal logics using llama2. In *2024 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, pp. 469–474. IEEE, 2024.

Daniel Mendoza, Christopher Hahn, and Caroline Trippel. Translating natural language to temporal logics with large language models and model checkers. In *2024 Formal Methods in Computer-Aided Design (FMCAD)*, pp. 1–11. IEEE, 2024.

Sara Mohammadinejad, Sheryl Paul, Yuan Xia, Vidisha Kudalkar, Jesse Thomason, and Jyotirmoy V Deshmukh. Systematic translation from natural language robot task descriptions to stl. In *International Conference on Bridging the Gap between AI and Reality*, pp. 259–276. Springer, 2024.

Jiayi Pan, Glen Chou, and Dmitry Berenson. Data-efficient learning of natural language to linear temporal logic translators for robot task specification. *arXiv preprint arXiv:2303.08006*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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

Yu'an 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 *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6942–6959, 2024.

Jiarui Yao, Ruida Wang, and Tong Zhang. Prl: Process reward learning improves llms' reasoning ability and broadens the reasoning boundary. *arXiv preprint arXiv:2601.10201*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Table 8: Fields of the typed STL AST and their roles in serialization.

Field	Type	Role
`STL`	object	Root containing the complete formula
`Operation`	string	Temporal or Boolean operator
`Time`	list/null	Interval $[a, b]$ for bounded temporal operators
`Leftaction`	object/string/null	Left child of a binary operator
`Rightaction`	object/string/null	Unary child or right child
`SubQueries`	list	Children of commutative Boolean operators
Predicate leaf	string	Declared signal, comparator, and threshold

A Formal Compilation Interface

This section specifies the source semantics, declared predicate interface, typed target representation, and canonical matcher used by the learned compiler.

STL semantics. Section 3 defines the discrete-time trace, atomic predicates, bounded until, and the derived eventually and always operators. We retain those conventions and specify the threshold-transition and past-time constructs used by the benchmark.

For an atomic predicate π^μ, $\mathrm{rise}(\pi^\mu, t) := \neg\pi^\mu[t-1] \wedge \pi^\mu[t]$ and $\mathrm{fall}(\pi^\mu, t) := \pi^\mu[t-1] \wedge \neg\pi^\mu[t]$. These operators require an explicit false-to-true or true-to-false transition. A state description or a change in signal magnitude alone does not establish a Boolean threshold crossing. The past-time operators historically and once satisfy $\mathbf{H}_{[a,b]}\phi$ when ϕ holds at every $t' \in [t-b, t-a]$ and $\mathbf{O}_{[a,b]}\phi$ when it holds at some t' in that interval.

Declared predicate interface. The interface $\mathcal{V}$ in Section 3 fixes admissible signal or predicate names, types, and canonical units. A system designer, ontology, visual model, state estimator, or LLM-based semantic adapter may supply this vocabulary and its interpretation. ReasonSTL compiles the requirement q over the supplied interface.

Typed AST. STL formulas are represented as JSON trees. Internal nodes encode temporal or Boolean operators and leaves encode normalized predicates. The release schema uses `Eventually` as the unique label for $\mathbf{F}$.

```
{
  "STL": {
    "Operation": "Eventually",
    "Time": [0, 1800],
    "Leftaction": null,
    "Rightaction": {
      "Operation": "and",
      "SubQueries": [
        "altitude>3048.0",
        "speed>=102.89"
      ]
    }
  }
}
```

This tree serializes to $\mathbf{F}_{[0,1800]}((\text{altitude} > 3048.0) \wedge (\text{speed} \geq 102.89))$. Table 8 summarizes the main fields.

Canonicalization and metrics. Canonicalization normalizes operator aliases, numerical fields, temporal intervals, and predicate triples. The recursive matcher compares operators, intervals, predicates, and child subtrees. Children of commutative `and` and `or` nodes are aligned independently of order. Numerical fields use tolerance 0.1 to absorb post-conversion rounding. CanAcc requires an exact canonical tree match; FmtAcc applies the same matcher after replacing predicate contents, temporal bounds, and thresholds with placeholders. Invalid JSON, invalid arity, unknown declared predicates, type errors, and invalid intervals receive zero. Partial tree similarity supplies dense training reward. Evaluation assigns a binary score to every item.

Table 9: Domain and scenario coverage of STL-BENCH.

Domain	Representative scenarios	Signals
Autonomous driving	AEB, ACC, lane keeping, parking, traction control	speed, acceleration, brake, distance
Robotics	pick-and-place, welding, assembly, navigation	position, torque, force, velocity
Industrial control	reactor, CNC, conveyor, press, boiler	pressure, temperature, flow, strain
Environmental	climate, water, air, greenhouse, noise	humidity, CO_2, pH, concentration
Electrical	battery, motor, grid, solar, signal processing	voltage, current, frequency, power
Aerospace	altitude hold, landing, survey, satellite, delivery	altitude, airspeed, pitch, yaw

Table 10: Deterministic tool annotations: arguments, returned values, and destination fields in the AST.

Tool	Input	Output	AST destination
`parse_duration`	"30 minutes"	1800	Temporal interval
`convert_unit`	$(10000, \mathrm{ft}, \mathrm{m})$	3048.0	Predicate threshold
`convert_unit`	$(5, \mathrm{km/h}, \mathrm{m/s})$	1.39	Predicate threshold
`eval_math_expr`	$2 \star 900$	1800	Interval bound
`calc_time_diff`	08:00 to 08:15	900	Temporal interval

Training uses deterministic canonical matching for reward computation. We do not treat this matcher as a general temporal-logic equivalence procedure. CanAcc measures canonical-reference recovery, while SemAcc provides a separate reference-independent estimate at test time.

B STL-BENCH CONSTRUCTION AND EVALUATION PROTOCOL

Coverage. Table 9 expands the six-domain coverage summarized in Section 4: 28,880 English and Chinese pairs, 33 scenarios, and 41 canonical signals. The two languages share an interface but use independently realized requirements.

Label-first construction. Each template fixes the scenario, admissible signals, operator tree, temporal intervals, thresholds, units, and required computations before language realization. DeepSeek-Chat produces engineering-style requirements conditioned on the fixed formal object. Samples are organized into six complexity levels using tree depth, operator count, predicate count, and tool-chain length; the sampling ratios from levels 1 to 6 are 25%, 25%, 20%, 15%, 10%, and 5%.

Computation annotations. Seventy-three percent of the benchmark requires at least one deterministic computation. The released annotations contain 7,337 duration-normalization calls, 5,161 unit conversions, and 4,527 arithmetic evaluations, together with time-difference calls where applicable. Each annotation records the tool, arguments, expected result, and destination field in the final AST. Table 10 gives representative calls.

Validation and deduplication. Validators check language length, JSON syntax, required fields, AST well-formedness, arity, interval validity, predicate syntax, declared-signal use, tool executability, unit conversion, arithmetic, and consistency between tool results and final fields. Samples that cannot be repaired deterministically are removed. Exact and symbolic duplicates are filtered by normalized text, AST hashes, and predicate assignments. Qwen3-Embedding-4B then clusters requirements above cosine similarity 0.9; each cluster retains the clearer or structurally richer samples.

Splits. The standard split is stratified by language, domain, complexity, and tool type. The main evaluation additionally holds out complete scenarios. For each language, 21/6/6 of the 33 scenario IDs form train, validation, and test; all bilingual variants of held-out IDs are excluded from joint training. Fixed seeds and split manifests are included in the release.

Manual-500. Six domain specialists independently authored 500 pairs, with 250 English and 250 Chinese items. They received the target domain, declared predicate interface, and task definition. Generation templates and benchmark examples were withheld. The two language pools were authored independently. A second specialist cross-checked each NL-to-STL pair, and ambiguous pairs were revised before the set was frozen. Manual-500 remained excluded from training, validation, prompt selection, and checkpoint selection. The maximum embedding similarity from any Manual-500 item to generated training data is 0.20, and the average nearest-neighbor similarity is 0.11.

Human evaluation. For every reported system, SemAcc covers all 500 frozen Manual-500 predictions (250 English and 250 Chinese). Three STL-qualified domain experts independently judge each anonymized prediction under system-blind evaluation and resolve disagreements by consensus. The experts are independent of the paper authors and Manual-500 writers. A prediction is correct when its STL semantics faithfully express the complete requirement under the declared signal interface. Equivalent noncanonical formulas are accepted, and every item remains in the denominator. A separate three-annotator protocol labels every canonical mismatch of the 4B full model using non-exclusive failure categories. The semantic evaluation and mismatch audit address distinct questions and maintain separate records.

C REASONSTL COMPILER IMPLEMENTATION

This section specifies the operational details behind Figure 2: tool execution, stage validation, reward construction, token masking, and training configuration. The checks enforce the declared compilation contract; they do not constitute a proof of unrestricted semantic equivalence.

Tool execution. A model-emitted `<tool_call>` block is parsed against the typed tool schema. Well-formed calls are executed by the environment and returned through `<tool_result>`. Invalid names, malformed arguments, and execution errors invalidate the corresponding stage. During training, the frozen annotation additionally determines whether a result is incorporated into its designated AST field. Tool-result tokens are environmental observations and are excluded from policy-gradient updates.

```
<think>
Normalize the 2-minute deadline and both physical quantities.
</think>
<tool_call>parse_duration("2 minutes")</tool_call>
<tool_result>120</tool_result>
<tool_call>convert_unit(10, "ft", "m")</tool_call>
<tool_result>3.048</tool_result>
<tool_call>convert_unit(5, "km/h", "m/s")</tool_call>
<tool_result>1.39</tool_result>
```

Stage validation. Table 11 lists the evidence used by local checks. Type, schema, execution, interface-membership, and numerical-consistency checks are reference-free. Predicate alignment, value destination, and event selection use the frozen item annotation when constructing training rewards. At inference, only checks available from the input, declared interface, generated trace, and tool results are applied. These checks can detect specified contract violations, while semantic equivalence to an unknown target remains outside their scope.

Reward construction. We use the format indicator R_i^{fmt}, canonical-match indicator C_i^{final}, and tree similarity S_i^{tree} from Section 5.3. Tree similarity is zero when no valid AST is available. The content and outcome rewards enter advantage normalization as

$$R_i^{\text{cnt}} = C_i^{\text{final}} + (1 - C_i^{\text{final}})\kappa S_i^{\text{tree}}, \qquad R_i^{\text{out}} = R_i^{\text{fmt}} R_i^{\text{cnt}}, \qquad 0 < \kappa < 1. \tag{7}$$

Equation (5) defines stage credit from local validity, prefix validity, and final correctness. Outcome advantages are normalized over all G rollouts; process advantages are normalized only among rollouts with an aligned stage k, denoted $\mathcal{G}_k$:

$$A_i^{\text{out}} = \frac{R_i^{\text{out}} - \text{mean}_j R_j^{\text{out}}}{\text{std}_j R_j^{\text{out}} + \epsilon}, \qquad A_{i,k}^{\text{proc}} = \frac{R_{i,k}^{\text{proc}} - \text{mean}_{j \in \mathcal{G}_k} R_{j,k}^{\text{proc}}}{\text{std}_{j \in \mathcal{G}_k} R_{j,k}^{\text{proc}} + \epsilon}. \tag{8}$$

Table 11: Stage checks used to construct the training validity indicators in Section 5.3.

Stage	Validation	Example
Tool selection	Function belongs to $\mathcal{T}$	duration parser for "2 minutes"
Arguments	Arguments satisfy the tool schema	valid quantity and unit strings
Execution	Tool terminates successfully	no conversion failure
Incorporation	Result occupies its annotated AST field	120 in eventuality interval
Predicate selection	Signal is declared; training target aligns comparator	declared distance signal
Event selection	Edge form requires an explicit transition cue	`Rise` only for a transition

Table 12: Training and decoding settings for the local compiler models.

Hyperparameter	Value
Backbones	Qwen3-4B and Qwen3-8B
Fine-tuning	Full, input embeddings frozen
Hardware	8×H20 training; 1×H20 evaluation
SFT learning rate / effective batch	2×10^{-5} / 64
SFT maximum length / steps	2048 / 300
RL learning rate / batch / group	3×10^{-6} / 16 / 8
RL maximum tool rounds / tokens	5 / 2048
Train / test temperature	1.0 / 0.0
Top-p / gradient clipping	0.95 / 1.0
Process factor τ / tree cap κ	0.5 / 0.3
Numerical tolerance	0.1
Direct / full RL steps	3K / 5K

For model-generated token t, $A_{i,t} = A_{i,k}^{\text{proc}}$ when t belongs to intermediate stage k, and $A_{i,t} = A_i^{\text{out}}$ when t belongs to the final-AST span. The binary mask $M_{i,t}$ is zero for deterministic tool results, malformed or truncated spans, invalid stages, and all stage spans after the first invalid stage; it is one for the remaining model-generated tokens. The masked group-relative likelihood-ratio objective is

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{T_i}\sum_{t=1}^{T_i} M_{i,t}\frac{\pi_\theta(y_{i,t} \mid q, \mathcal{V}, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid q, \mathcal{V}, y_{i,<t})}A_{i,t}\right]. \tag{9}$$

We use group-normalized advantages and gradient clipping. The objective omits PPO-style ratio clipping.

In Algorithm 1, the implementation aligns corresponding validated stages across rollouts and applies $M_{i,t}$ before accumulating this objective.

Training configuration. A short SFT cold start teaches the AST schema and tool-call protocol before RL. Table 12 reports the configuration used for the reported local models. The two tested model sizes support a capacity comparison; characterizing scaling behavior requires additional sizes and training budgets.

D Complete Experimental Evidence

This section records the training budgets, sensitivity checks, and implementation conditions that support the complete comparison tables in the main paper.

Main comparison tables. Table 3 reports the complete DeepSTL ranking, and Table 4 reports every evaluated local and hosted system on scenario-held-out STL-BENCH and Manual-500. The main paper includes these complete comparisons so that all headline claims can be assessed without consulting the appendix.

SFT and RL controls. Table 13 records the training budgets underlying the outcome-only Table 6 in the main paper. SFT only evaluates the cold-start checkpoint. Different RL lengths support system-level comparisons; the 5K-step rows provide matched-budget reward and representation controls.

Table 13: Training controls on scenario-held-out STL-BENCH. SFT and RL columns report update counts; CanAcc is a fraction.

Variant	SFT	RL	EN CanAcc	ZH CanAcc
SFT only	300	0	.345	.270
Direct	300	3K	.420	.330
Think, no tools	300	5K	.446	.372
Tools + final reward	300	8K	.482	.438
Unbounded process reward	300	5K	.469	.423
Bounded, no prefix mask	300	5K	.501	.461
Full	300	5K	**.510**	**.470**
Flat STL, matched	300	5K	.490	.420

Representation controls. Under matched local training, typed JSON improves CanAcc from 0.490/0.420 for flat STL to 0.510/0.470. Typed JSON also supplies the structure required for validation, deterministic serialization, tool binding, and stage-level supervision. The measured accuracy effect is specific to the evaluated local models and training configuration.

Sensitivity. The full 4B configuration obtains 0.510/0.470. For $\tau = 0.3/0.5/0.7$, CanAcc is 0.506/0.467, 0.510/0.470, and 0.504/0.466. For $\kappa = 0.2/0.3/0.4$, it is 0.504/0.464, 0.510/0.470, and 0.506/0.466. Frozen rescoring at numerical tolerance 0.05/0.10/0.15 produces 0.509/0.469, 0.510/0.470, and 0.510/0.470. Embedding-pruning thresholds 0.85/0.90/0.95 produce 0.501/0.459, 0.510/0.470, and 0.506/0.465. The main conclusion is stable within these local ranges.

End-to-end serving cost. Table 5 in the main paper reports the complete end-to-end measurements, including prompt processing, generation, tools, parsing, AST construction, validation, and serialization. Local systems use one H20 with concurrency eight. Hosted measurements include provider-specific infrastructure effects and do not support hardware-normalized comparisons.

E SUCCESSFUL AND FAILED COMPILATION TRACES

Successful end-to-end trace. The vehicle example in Section 5.4 uses the tool calls shown in Appendix C. Duration parsing returns 120 s; unit conversion returns 3.048 m and 1.39 m/s. The compiler places these values in the eventuality interval and the distance and speed predicates, respectively. Schema validation precedes serialization into $\mathbf{F}_{[0,120]}((\texttt{dist_stop_m} \leq 3.048) \wedge (\texttt{speed_mps} < 1.39))$.

At training time, a valid tool stage receives full process credit when its prefix is valid and the final AST is exact. If duration parsing fails, the prefix mask removes process credit from all subsequent stages. A rollout with valid computations and a non-exact final tree receives discounted process credit and capped outcome reward. At inference, reward computation is disabled. The compiler executes the same tools, applies the reference-free checks described above, and returns the serialized STL. Passing these checks establishes typed contract compliance, not semantic equivalence to an unavailable reference.

Mismatch audit. Table 7 in the main paper reports all non-exclusive categories from three annotators' cross-check of every canonical mismatch of the 4B full model. The cases below explain the semantic distinctions behind these tags.

Numerical normalization and incorporation. For "within 30 minutes, the UAV must climb above 10,000 ft," with altitude declared in meters, the intended formula is $\mathbf{F}_{[0,1800]}(\text{altitude_m} > 3048)$. A representative error is $\mathbf{F}_{[0,30]}(\text{altitude_m} > 10000)$. The prediction copies both surface numbers and violates the canonical units for time and altitude. Correct execution alone is insufficient: the normalized values must also appear in their intended AST fields.

Relative-time anchoring. Consider "after the UAV enters the inspection zone, it must reach altitude above h within 2 minutes, and after reaching that altitude, keep speed below v for the next 30 seconds." The intended formula is $\mathbf{G}(\text{enter_zone} \rightarrow \mathbf{F}_{[0,120]}((\text{altitude} > h) \wedge \mathbf{G}_{[0,30]}(\text{speed} < v)))$. A common error is $\mathbf{G}(\text{enter_zone} \rightarrow (\mathbf{F}_{[0,120]}(\text{altitude} > h) \wedge \mathbf{G}_{[0,30]}(\text{speed} < v)))$.

Both durations are correct. The erroneous nesting anchors the second interval at zone entry; the requirement anchors it at altitude attainment. This case isolates semantic attachment from numerical calculation.

Threshold state versus transition event. For "whenever the current is above 75 amperes, the voltage must have remained stable over the preceding 10 seconds," the antecedent is the threshold state $\mathrm{current} > 75$. The prediction $\mathrm{Rise}(\mathrm{current} > 75)$ restricts the obligation to the instant of a false-to-true crossing. It is strictly weaker over traces that remain above the threshold. Edge operators require an explicit reference to entering, leaving, crossing, or another transition event.

Predicate alias. A requirement referring to "rotational speed" may have canonical predicate $\mathrm{rpm} > X$. A model may emit $\mathrm{rotational_speed} > X$. Under a schema that declares rpm, the prediction violates the compilation contract and is rejected. A downstream interface that explicitly aliases both names may align their semantics. This case motivates separate evaluation of predicate-interface compliance and formula semantics.

Alternative-valid formula. CanAcc may reject a formula that expresses the requirement through an equivalence outside deterministic normalization. Canonicalization already aligns the children of commutative `and` and `or` nodes independently of order, so a formula that only reorders these children still matches the frozen target. Equivalences such as reassociation and distribution lie beyond the normalized set. The mismatch audit assigns such cases to the alternative-valid category, and SemAcc assesses their semantic correctness.

F Compiler Boundaries and Reproducibility

Task boundary. The clear-input contract begins after requirement elicitation. Omitted thresholds, incompatible readings, and undefined signals must be resolved before compilation. This separation supports independent measurement of translation fidelity and places ambiguity handling in a complementary workflow stage.

Executable grounding belongs to the surrounding system. ReasonSTL selects and composes declared predicates. Sensors, perception models, state estimators, or simulators define their truth values. A task-level predicate can enter the AST once the downstream interface supplies its interpretation. This modular contract defines the information required to apply the compiler in another physical environment with a compatible interface; such transfer is not evaluated here.

Evaluation boundary. Each item has one canonical target. Deterministic matching enables stable training and reproducible comparison, although it may reject semantically valid alternatives. SemAcc complements this metric with expert judgments over the complete Manual-500 prediction set for every reported system. Formal equivalence proof remains outside the evaluation protocol. Broader equivalence coverage could combine rule normalization, bounded trace testing, robustness comparison, and solver-based checks with deterministic, auditable supervision.

STL-Bench uses label-first templates and model-assisted language realization at scale. Scenario-held-out testing, Manual-500, embedding checks, deterministic validation, and human auditing reduce generator artifacts. The reported results apply to the declared domains, interfaces, STL dialect, and languages. Unrestricted open-domain generalization and exhaustive label correctness require broader evidence.

Efficiency and deployment. Explicit reasoning and tool execution increase latency over direct generation. ReasonSTL-4B reduces throughput from 5.90 to 2.20 queries/s and improves scenario CanAcc by 9/14 percentage points relative to Direct-4B. Adaptive tool invocation and shorter compilation traces may improve this trade-off. The two evaluated local model sizes support a capacity comparison. Establishing a scaling law requires additional sizes and training budgets.

Schema validity provides structural assurance. Deployment additionally requires a serializer or adapter for the target monitor, consistency checks against the system model, and independent validation appropriate to the application. ReasonSTL supplies the formalization front end within this broader CPS assurance pipeline.

Artifacts. The anonymized supplementary package contains fixed benchmark splits, Manual-500, construction templates, validators, tool implementations, baseline prompts and demonstrations, de-

coding settings, raw predictions, evaluation scripts, training configurations, and the JSON-to-STL serializer. The release records the exact model and endpoint identifiers used for every reported run and preserves automatic scores separately from expert judgments.

G LIMITATIONS AND FUTURE WORK

Limitations. REASONSTL assumes a clear requirement and a declared predicate interface. Requirement elicitation, predicate grounding, and signal estimation remain responsibilities of the surrounding system. Its validators detect structural, interface, numerical, and execution errors. General semantic equivalence remains outside their scope. Evaluation covers six CPS domains, two languages, and two local model sizes. Broader domains, STL dialects, interfaces, and model scales require further study. Explicit reasoning and tool execution also increase latency relative to direct generation.

Future work. Future work will extend equivalence checking through broader normalization, bounded trace testing, robustness comparison, and solver-based analysis. Adaptive tool invocation and shorter compilation trajectories may reduce inference cost. Integrating predicate-grounding modules could connect declared interfaces to perception and state estimation. Further evaluation should examine additional languages, domains, STL operators, model scales, and deployment environments.