GenCTL:Constraint-Guided LLM Generation of Verifiable CTL Specifications from Natural Language Requirements

Ran Tao[1,2], Yanhong Huang[1,2,3]✉, Jianqi Shi[1,2,3], Yang Yang[3], and Fan Gao[1]

[1] Software Engineering Institute, East China Normal University, Shanghai, China
yhhuang@sei.ecnu.edu.scn
[2] National Trusted Embedded Software Engineering Technology Research Center, Shanghai, China
[3]Shanghai Fenglei Information Technology Co., Ltd., Shanghai, China

Abstract. Natural language (NL) requirements are widely used in system design, but their ambiguity makes them difficult to translate into formally verifiable specifications. Translating NL requirements into Computation Tree Logic (CTL) is challenging because the generated formulas must not only capture the intended meaning, but also remain compatible with the atomic propositions and constraints of concrete verification models. Although large language models (LLMs) provide a practical basis for NL-to-CTL (NL2CTL) translation, their outputs are often unstable and prone to parsing, typing, and grounding errors. To address this problem, we propose GENCTL, a prompt-based framework for model-aware NL2CTL translation without task-specific fine-tuning. GENCTL uses atomic-proposition (AP) grounding from nuXmv models as its core domain-specific mechanism for verifier-compatible generation, and combines retrieval-enhanced prompting, frequency-first candidate selection with length-normalized log-likelihood tie-breaking, and optional explanation-dictionary-based refinement to improve semantic guidance, output stability, and lightweight correction. On a generated NL–CTL dataset, the best automatically selected translation achieves 68% accuracy, which increases to 92% after one round of guided refinement. On 150 NL requirements constructed over three nuXmv models, AP-list prompting yields 139/150 directly checkable formulas, increasing to 149/150 after lightweight normalization. These results indicate that GENCTL improves the practical checkability of LLM-generated CTL specifications and provides a useful model-aware framework for aligning natural-language requirements with verifier-compatible formalization.

Keywords: Natural language to CTL · Large language models · Prompt Engineering · Model-aware grounding · Formal verification

1 Introduction

Computation Tree Logic (CTL) is widely used to specify and verify temporal properties of reactive and safety-critical systems, yet writing CTL formulas remains labor-intensive and error-prone, especially for users with limited formal-methods expertise. Since system requirements are usually expressed in natural language (NL), a practical gap remains between informal descriptions and formally checkable specifications. Recent advances in large language models (LLMs) and in-context learning make NL-to-formal translation increasingly feasible,[1,2] and recent studies have begun to investigate NL2CTL more directly through datasets and supervised generation frameworks [3,4].However, directly applying LLMs to NL2CTL remains difficult because the generated formulas are sensitive to prompt design, unstable across runs, and prone to parsing, typing, and grounding errors, including hallucinated or model-incompatible atomic propositions.

These difficulties become more serious when a concrete verification model is available, because the generated formula must also match the admissible identifiers and constraints of the target checker. Existing prompting techniques, including retrieval-based example construction and consistency-style multi-candidate decoding, can improve output relevance and robustness [5,6], but they do not by themselves ensure grounded and verifier-compatible CTL generation. Practical NL2CTL therefore requires coordinated support for semantic guidance, AP grounding, output stability, and downstream checkability.

To address these challenges, we propose GenCTL, a prompt-based framework for NL2CTL translation without task-specific fine-tuning, supporting both model-agnostic generation and model-aware generation under concrete verification models. Its main domain-specific mechanism is model-aware atomic-proposition (AP) grounding from nuXmv models, which constrains generation to verifier-usable identifiers. In addition, retrieval-enhanced few-shot prompting improves semantic guidance in model-agnostic settings, frequency-first multi-candidate selection improves robustness under stochastic decoding, and optional explanation-dictionary-based refinement supports lightweight correction of residual errors. Together, these components improve grounded identifier alignment and verifier compatibility in practical NL2CTL.

We evaluate GenCTL in both model-agnostic and model-aware settings. On a 100-instance generated NL–CTL set, the best automatically selected translation achieves 68% accuracy, increasing to 92% after one round of guided refinement. On 120 public Natural2CTL instances across three domains, dynamic prompting improves both AP grounding and semantic correctness over static prompting. On 150 NL requirements constructed over three nuXmv models, AP-list prompting yields 139/150 directly checkable formulas, increasing to 149/150 after lightweight normalization. Overall, the results show that retrieval-enhanced prompting mainly improves semantic guidance in model-agnostic settings, while model-aware AP grounding substantially improves direct checkability under concrete verifier constraints.

Contributions. The main contributions of this paper are as follows:

1. We present GENCTL, a prompt-based NL2CTL framework that supports both model-agnostic translation and model-aware generation under concrete nuXmv vocabularies, without task-specific fine-tuning.
2. We develop a model-aware AP-grounding procedure that extracts admissible Boolean-valued identifiers from nuXmv models and constrains CTL generation toward verifier-compatible outputs, while retrieval-enhanced prompting, frequency-first candidate selection, and optional explanation-dictionary-based refinement provide complementary support for semantic guidance, robustness, and lightweight correction.
3. We provide component-level evidence across model-agnostic and model-aware settings, showing that frequency-first selection improves robustness, retrieval-enhanced prompting improves AP recovery and semantic correctness on public Natural2CTL data, and AP-list grounding substantially improves nuXmv-level direct checkability.

2 Related Work

Early approaches to translating natural language (NL) requirements into temporal logic mainly relied on controlled languages, hand-crafted rules, and specification patterns. These methods provide interpretability and syntactic control, but usually require substantial manual effort and transfer poorly across domains. More recent work increasingly formulates NL-to-temporal-logic translation as a data-driven or LLM-assisted task. Most existing studies focus on linear temporal logic (LTL), including datasets, neural translation models, and systems that combine templates, learned representations, or planners to generate executable temporal specifications[7,8,9,10].Although these results demonstrate the promise of LLM-based formal-language generation, CTL remains more challenging because its branching-time semantics and path quantifiers make the output more dependent on the target verification model.

Prompting and in-context learning provide an effective alternative to task-specific fine-tuning for structured generation[1,2].Prior work shows that prompt design, demonstration selection, and retrieval-based prompt construction can substantially affect performance by improving the relevance of in-context examples[11,5]Related techniques, including chain-of-thought prompting, automatic prompt engineering, and self-consistency decoding, have also been shown to improve reasoning quality and output robustness[12,13,14,6]. For formal-specification generation, however, semantic plausibility alone is insufficient: outputs must also satisfy syntactic, typing, and tool-compatibility constraints. This motivates combining retrieval-enhanced prompting with explicit grounding constraints and candidate-selection strategies.

Work specifically targeting NL2CTL remains limited. Early studies derived CTL formulas from restricted English or grammar-based interfaces, offering strong syntactic control but limited portability[15,16,17].More recent efforts include the Natural2CTL benchmark[3] and supervised encoder–decoder approaches for NL2CTL generation[4].Although these methods improve empirical performance, they still suffer from syntactic, semantic, and grounding errors, especially when generated formulas must

match the vocabulary of a concrete verification model. Prior CTL-oriented systems such as GLAsT and EASE further suggest that effective translation depends on aligning textual descriptions with design artefacts and verification variables [16,17]. Motivated by this observation, our work focuses on verifier-compatible NL2CTL under concrete models and studies how prompt-based generation can be guided by model-derived atomic-proposition vocabularies without task-specific fine-tuning. In GenCTL, model-aware atomic-proposition grounding from nuXmv models serves as the main domain-specific mechanism for improving compatibility with the target verifier, while retrieval-enhanced prompting, multi-candidate selection, and explanation-dictionary-based refinement provide complementary support for semantic guidance, robustness, and lightweight correction.

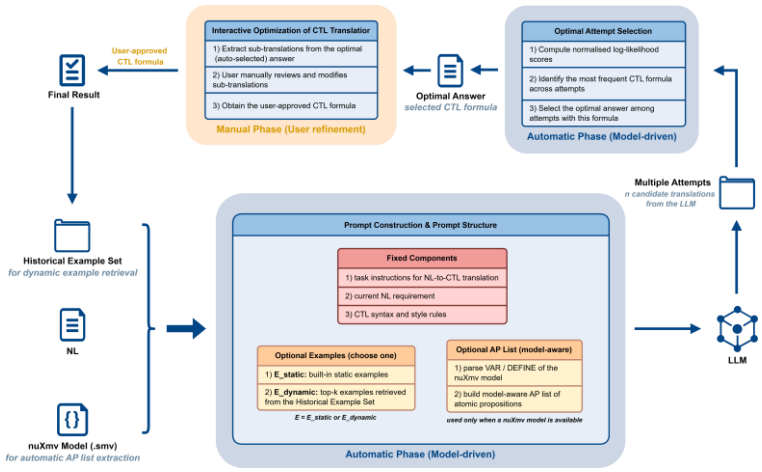

Fig. 1. Overview of GENCTL. The framework is organized around model-aware grounding and includes prompt construction, candidate selection, and optional interactive refinement for practical NL2CTL generation.

3 Method

Fig. 1 illustrates the overall organization of GenCTL. The framework is organized around model-aware atomic-proposition grounding, which provides the main domain-specific mechanism for guiding NL2CTL generation toward verifier-compatible outputs under concrete nuXmv models. Given a natural language requirement, GenCTL constructs a structured prompt and, in model-agnostic settings, incorporates retrieval-enhanced NL–CTL examples to provide semantically relevant context and support the recovery of appropriate atomic propositions from domain-consistent demonstrations. When a concrete verification model is available, model-aware AP grounding constrains generation to identifiers admissible in the target nuXmv model. GenCTL further uses multi-candidate generation with frequency-first ranking to improve output stability under stochastic decoding, and supports optional interactive refinement through an

explanation dictionary to correct residual translation errors. User-approved NL–CTL pairs can also be retained as historical examples for subsequent prompt construction.

Structure of prompts	
Task Introduction	Instruction that defines the NL2CTL translation task and assigns the LLM the role of a CTL expert. It requires the model to output, for each NL requirement, a CTL formula in a predefined JSON format together with a step-by-step explanation of the translation process.
CTL Translation Rules	Specification of the allowed CTL syntax and operator semantics. The prompt reminds the model of the meaning of temporal and path operators (e.g., X for "next", U for "until", G for "globally", F for "finally", A/E for universal/existential paths) and constrains the output to well-formed CTL formulas that use only these operators, Boolean connectives (|, &, ~, ->, <->) and atomic propositions, without any free-form natural-language text.
Current NL Requirement	The natural-language requirement to be translated (current input sentence).
Few-shot Examples	Either (i) built-in static NL–CTL examples, or (ii) top-k NL–CTL examples retrieved from the historical example set using embedding similarity. Each example includes the NL requirement, its CTL formula and an explanation dictionary.
Model-aware AP List (Optional)	List of admissible atomic propositions extracted from the nuXmv model (e.g., from VAR and DEFINE sections). The prompt instructs the LLM to only use these APs (and Boolean combinations thereof) when constructing CTL formulas.

Fig. 2. Prompt construction

3.1 Prompt Construction and Example Retrieval

Fig. 2shows the prompt template used in GENCTL. For an input NL requirement, the prompt is defined as

$$P = \langle I, R, E_{\text{prompt}}, M \rangle \quad (1)$$

where I is the task instruction, R specifies CTL syntax and generation constraints, E_{prompt} is a few-shot example block, and M is an optional model-aware atomic-proposition (AP) list. GenCTL builds E_{prompt} from either curated seed examples or dynamically retrieved historical NL–CTL pairs, and appends M when model-aware constraints are available.

In model-agnostic settings, the retrieved examples serve not only as semantically relevant demonstrations, but also as a source of domain-consistent atomic-proposition usage. To improve both in-context relevance and AP recovery, GenCTL retrieves semantically similar examples from a historical NL–CTL repository. This repository is built incrementally from NL–CTL pairs that have been confirmed and stored during prior user interactions, allowing subsequent prompting to benefit from previously validated examples.

Algorithm 1 Hierarchical symbol extraction for AP grounding from nuXmv .smv models

```
Require: SMV model file 𝓜, root module name root
Ensure: AP candidate list S_AP with fully qualified identifiers
1: 𝒯 ← Preprocess(𝓜) ▷remove comments and normalize whitespace
2: 𝓜 ← SplitModules(𝒯)
3: D ← ∅                                   ▷local declarations
4: I ← ∅                                   ▷module instantiations
5: for all module m ∈ 𝓜 do
6: (D_m, I_m) ← ParseModule(m)  ▷extract declarations and instances
7:     D ← D ∪ D_m
8:     I ← I ∪ I_m
9: end for
10: G ← BuildInstanceGraph(I)
11: C ← CollectContexts(G, root)  ▷prefix contexts reachable from root
12: S ← ∅
13: for all (name,desc,m,kind) ∈ D do
14:     for all prefix ∈ C(m) do
15:         full ← ConcatPrefix(prefix,name)
16:         S ← S ∪ {(full,desc,m,kind)}
17:   end for
18: end for
19: S ← Deduplicate(S).  ▷remove duplicates by (full,m,kind)
20: S_AP ← FilterBooleanSymbols(S)  ▷retain symbols directly usable as CTL APs
21: return S_AP
```

Each stored NL requirement is encoded offline, and the input requirement is matched to the top-k most similar examples using sentence embeddings and cosine similarity[18].The retrieved NL–CTL pairs and their explanation dictionaries are then inserted into E_{prompt}, consistent with prior findings that example selection strongly affects in-

context performance[1,11].In the experiments reported in this paper, we fix $k = 10$ unless otherwise stated, while curated seed examples are used when no historical repository is available.

3.2 Model-Aware AP Grounding

Among the components of GenCTL, model-aware AP grounding serves as the main domain-specific mechanism for constraining generation under concrete verifier vocabularies. When a nuXmv model is available, GenCTL augments the prompt with an AP list M derived from the smv specification. This grounding step is intended to reduce undefined-identifier errors, improve vocabulary consistency with the target model, and increase downstream checkability in nuXmv.

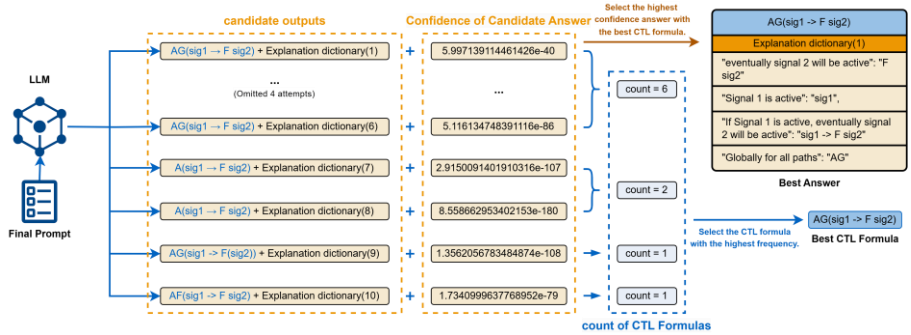

Fig. 3. Two-stage candidate selection in GENCTL based on output frequency and length-normalized log-likelihood.

As shown in Algorithm 1, the method constructs a hierarchical symbol table by parsing variable and definition declarations, resolving module instantiations, and propagating qualified identifiers along the instance hierarchy. This propagation step is necessary because symbols in nuXmv may be introduced through nested module instances, and exposing only local declarations would miss the fully qualified identifiers that are actually usable in CTL formulas. The extracted symbols are then filtered to retain Boolean-valued expressions suitable for CTL generation, excluding declarations that are syntactically present in the model but are not directly usable as atomic propositions.

In this way, AP grounding constrains the LLM to a model-consistent vocabulary and primarily improves verifier compatibility, although semantic correctness still depends on whether the selected propositions adequately capture the NL intent.

3.3 Frequency-First Candidate Selection

Because stochastic decoding may produce different CTL formulas and explanation dictionaries from the same prompt, GenCTL adopts a two-stage selection strategy (**Fig. 3**) to improve output stability while preserving a representative explanation for subsequent refinement.

Stage 1: Formula selection. In the current implementation, candidates are grouped by exact formula strings rather than semantic equivalence. This choice provides a simple and reproducible approximation to output agreement, and the results in Section4.2 show that the resulting frequency-first strategy is empirically effective for candidate selection in our NL2CTL setting, although semantically equivalent but syntactically different formulas are treated as distinct candidates. Given n sampled answers$\{a_i\}_{i=1}^{n}$ with CTL formulas ϕ_i, we group answers by identical formulas and select the most frequent one:

$$\phi^* = \arg\max_{\phi} |\{i \mid \phi_i = \phi\}|.$$

Algorithm 2 Optional Interactive Refinement with Explanation Dictionaries

Require: Natural language requirement N_{input}, large language model LLM, maximum refinement rounds r_{max}
Ensure: Final CTL translation T_{final}
$(T_0, ED_0) \leftarrow \mathrm{LLM}(N_{input})$
Present T_0 and ED_0 to the user
$T_{final} \leftarrow T_0$
$r \leftarrow 0$
while refinement is requested and $r < r_{max}$ do
$ED_{r+1} \leftarrow \mathrm{Edit}(ED_r)$
$(T_{r+1}, ED'_{r+1}) \leftarrow \mathrm{LLM}(N_{input}, ED_{r+1})$
Present $\mathrm{T_{r+1}}$ and $\mathrm{ED'_{r+1}}$ to the user
$T_{final} \leftarrow T_{r+1}$
$r \leftarrow r + 1$
end while
return T_{final}

For a generated answer $a = (y_1, \ldots, y_{|a|})$, we define the length-normalized log-likelihood as

$$\mathrm{LL}_{\mathrm{norm}}(a) = \frac{1}{|a|} \sum_{t=1}^{|a|} \log p(y_t \mid y_{<t}, P),$$

where P denotes the prompt and $|a|$ is the number of generated tokens in the answer. If multiple formulas have the same frequency, ties are broken by the highest $\mathrm{LL}_{\mathrm{norm}}$ among their corresponding answer instances.

Stage 2: Representative answer selection. Let $\hat{A} = \{a_i \mid \phi_i = \phi^*\}$ be the subset of answers generating ϕ^*.We then select

$$a^* = \arg\max_{a_i \in \hat{A}} \mathrm{LL}_{\mathrm{norm}}(a_i).$$

The final output is (ϕ^*, a^*), where ϕ^* is the selected formula and a^* is the representative answer whose explanation dictionary is retained for optional subsequent refinement.

In this way, output frequency serves as the primary signal for consensus across stochastic samples, while LL_{norm} provides a secondary confidence signal for tie-breaking and representative-answer selection.

3.4 Interactive Refinement

After automatic selection, GenCTL supports an optional human-in-the-loop refinement stage through an explanation dictionary that maps salient NL phrases to CTL sub-formulas. For each requirement, the model outputs (i) a CTL formula, (ii) a brief explanation, and (iii) an editable explanation dictionary that links salient NL fragments to candidate CTL sub-formulas. Since LLM-generated formal specifications are often only partially incorrect [19], the refinement process focuses on revising problematic sub-translations rather than rewriting the full formula, after which the model regenerates the CTL translation under the updated constraints. This mechanism is intended to improve transparency and controllability by preserving correct portions of the generated specification while allowing localized corrections; the overall refinement procedure is summarized in Algorithm 2 .

4 Experiments

We evaluate GENCTL in both model-agnostic and model-aware settings. Using curated NL–CTL pairs and requirements constructed over three nuXmv models, we assess translation accuracy, AP grounding, and verifiability under different prompting and selection strategies.

4.1 Experimental Setup and Datasets

We evaluate GENCTL in two settings: *model-agnostic*, where NL–CTL pairs are translated without a concrete transition system, and *model-aware*, where generated CTL formulas must be grounded in a given nuXmv model.

Model-agnostic dataset (D1). D1 consists of both generated and public NL–CTL pairs. For RQ0 and RQ1, we construct a 100-instance generated NL–CTL set following the data-construction procedure of prior NL2CTL work [4]. For RQ2, we use 120 public instances from Natural2CTL [3]across three domains: tactical control systems, embedded RTOS, and general software requirements (40 per domain). D1 is used to evaluate candidate selection, interactive refinement, and retrieval-enhanced prompting in the absence of an explicit system model. The generated subset is used for controlled analysis of candidate selection and guided refinement, whereas the public Natural2CTL portion is used to evaluate retrieval-enhanced prompting on domain-specific requirements.

Model-aware dataset (D2). D2 contains 150 NL requirements constructed over three nuXmv case-study models: a traffic-light controller, ABP4, and UMS. For each model, we create 50 requirements grounded in its variables and behaviours, each paired with a semantically valid and directly checkable reference CTL formula.

Decoding settings. Unless otherwise stated, all experiments use the same system prompt with ten few-shot demonstrations. For each input, we sample $\mathbf{n = 10}$ candidate formulas with temperature 1.0 and select the final output using frequency-first self-consistency, with length-normalised log-likelihood for tie-breaking.

Model versions. We use GPT-4.1 (gpt-4.1), GPT-3.5 Turbo (gpt-3.5-turbo-0125), and Claude 3.7 Sonnet (claude-3-7-sonnet-20250219) through their respective APIs. All model calls used in the experiments were accessed between June and July 2025. We report these model identifiers and the access window for reproducibility; subsequent API updates may nevertheless introduce version drift.

4.2 RQ0: Effect of Candidate Ranking in the Model-Agnostic Setting

We first examine whether candidate ranking improves NL2CTL performance in the model-agnostic setting. This experiment is conducted on the 100 generated NL–CTL pairs described in Section4.1, using the same static prompt with ten few-shot examples and no user interaction. For each input, the LLM generates $n = 10$ candidate CTL formulas, and the final output is selected by a specified ranking strategy. Unless otherwise stated, the default sampling temperature is 1.0.

We begin by comparing four models under the default automatic setting, namely static few-shot prompting with multi-candidate generation and *Freq-first* selection using length-normalized log-likelihood tie-breaking. As shown in **Table 1**,GPT-4.1 achieves the best accuracy at 68% (68/100), followed by Claude3.7 Sonnet at 56%, the T5-large reference baseline from prior NL2CTL work [4] at 38%, and GPT-3.5 Turbo at 17%. The T5 result is included as a supervised reference baseline rather than as a fully re-trained in-domain comparison under the present setting. These results indicate that model choice has a substantial effect on NL2CTL accuracy even under the same prompting and ranking protocol.

We next isolate the effect of candidate ranking using GPT-4.1outputs and four selection methods: *Freq-first* (with ties broken by $\mathrm{LL}_{\mathrm{norm}}$), $\mathrm{LL}_{\mathrm{norm}}$-*only*, *First*, and *Random*. Here, *First* serves as a no-reranking baseline that directly takes the first sampled candidate, while *Random* serves as an uninformed selection baseline over the same candidate pool. **Fig. 4** shows that *Freq-first* performs best, achieving 68%, compared with 53% for $\mathrm{LL}_{\mathrm{norm}}$-*only*, 47% for *First*, and 48% for *Random*. This result supports frequency-first selection with length-normalized log-likelihood tie-breaking as the default strategy in the following experiments.

To examine whether the default temperature setting is overly aggressive for formal-logic generation, we further repeated the same GPT-4.1 experiment with temperatures 0.2 and 0.5 while keeping the prompt, the ten few-shot examples, the number of sampled candidates ($n = 10$), and the ranking rule unchanged. The resulting accuracies were 62% and 68%, respectively, compared with 68% at temperature 1.0. These results suggest that overly conservative decoding can reduce useful candidate diversity, while the main conclusion on the effectiveness of *Freq-first* ranking remains unchanged across a moderate-to-high temperature range.

4.3 RQ1: Impact of User Interaction

We next evaluate the effect of human-in-the-loop interaction on NL2CTL accuracy in the model-agnostic setting, using the same 100 generated NL–CTL pairs as in Section4.2.Since GPT-4.1 achieves the best initial performance, we use it as the base model. For each input, the initial CTL formula is selected from $n = 10$ candidates using *Freq-first* with length-normalized log-likelihood

Table 1. Translation accuracy on the 100-instance generated NL–CTL set.

Model/Setting	Accuracy
T5-large (reference baseline)	38/100 (38%)
Claude 3.7 Sonnet+static prompt + Freq-first	56/100 (56%)
GPT-3.5 Turbo + static prompt + Freq-first	17/100 (17%)
GPT-4.1 + static prompt + Freq-first	68/100 (68%)
GPT-4.1 + guided refinement	92/100 (92%)

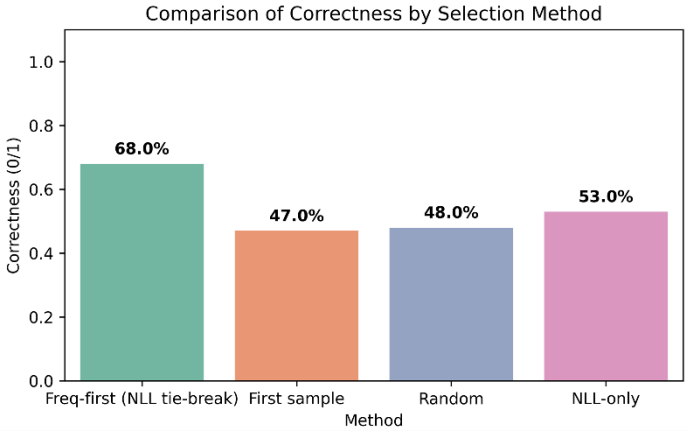

Fig. 4. Correctness of different candidate selection methods on the generated NL–CTL set. FREQ-FIRST with length-normalized log-likelihood tie-breaking achieves the best performance.

tie-breaking, and the associated explanation dictionary is then presented for optional refinement. We allow at most one refinement round per requirement, in which a participant with basic familiarity with formal methods edits problematic dictionary entries and the model regenerates the final formula.

As shown in **Table 1**, accuracy improves from 68% before interaction to 92% after one refinement round, yielding a gain of 24 percentage points. This result suggests that even limited user intervention can effectively correct residual translation errors that remain after automatic candidate selection.

This experiment should be interpreted as a single-participant feasibility case study rather than a controlled user study. The refinement was carried out by one participant with basic familiarity with formal methods, but without expert-level experience. The

edits were typically local: across the 100 instances, the participant revised an average of 1.7 explanation-dictionary entries per requirement (median = 1, range = 0–4), and the average refinement time was 1.3 minutes per requirement. These observations suggest that the improvement mainly comes from targeted correction of residual sub-translation errors rather than extensive manual rewriting of the full CTL formula. Accordingly, RQ1 should be interpreted as a feasibility-oriented case study.

Table 2. Strict AP grounding on Natural2CTL(40 per domain).

Domain	Method	Exact set	Global hit
TCS	Static	0.0%	0.0%
	Dynamic	10.0%	20.3%
RTOS	Static	2.5%	6.5%
	Dynamic	5.0%	32.3%
GSR	Static	7.5%	9.2%
	Dynamic	7.5%	34.5%
Overall	Static	3.3%	4.0%
	Dynamic	7.5%	26.9%

Automatic re-sampling control. As a no-feedback control, we also increased the number of sampled candidates from $\mathbf{n = 10}$ to $\mathbf{n = 20}$ and $\mathbf{n = 30}$ under the same prompt and selection setting. The corresponding accuracies were 67% and 69%, compared with 68% for the default $\mathbf{n = 10}$ setting. Thus, simply increasing the automatic sampling budget does not reproduce the large gain obtained through one round of interactive refinement (92%), suggesting that the improvement in RQ1 cannot be explained solely by additional inference-time computation.

4.4 RQ2: Dynamic Prompting on Domain-Specific Public Data

We next evaluate dynamic prompting on domain-specific public requirements from Natural2CTL[3].The evaluation set contains 120 NL–CTL pairs from three domains, namely tactical control systems (TCS), embedded RTOS, and general software requirements (GSR), with 40 instances per domain. For each domain, we additionally sample 70 disjoint in-domain NL–CTL pairs as a retrieval pool. We compare *static prompting*, which uses a fixed few-shot context, with *dynamic prompting*, which retrieves the top-k most similar examples to build an input-specific prompt. Unless otherwise stated, $k = 10$, and the remaining decoding settings follow RQ0.

AP grounding. We evaluate AP grounding using both strict and soft matching, with the main soft-matching results reported at $\boldsymbol{\tau = 0.7}$. As shown in **Table 2** and **Table 3**, dynamic prompting consistently improves AP grounding across all domains. The overall global AP hit rate increases from 4.0% (12/297) to 26.9% (80/297), while the overall soft AP F1 rises from 0.109 to 0.299. **Fig. 5** further shows that this advantage remains stable across different thresholds.

Semantic correctness. We further assess whether the generated CTL formulas preserve the intended meaning of the NL requirements. Two annotators with CTL/model-checking background independently label outputs, counting logically equivalent formulas as correct; disagreements are resolved by discussion. The agreement is

substantial, with Cohen's $\kappa = 0.726$. **Table 4** shows that dynamic prompting improves semantic correctness in all three domains, increasing

Table 3. Soft AP matching (macro F1 at $\tau = 0.7$).

Domain	Static	Dynamic
TCS	0.019	0.269
RTOS	0.149	0.337
GSR	0.160	0.292
Overall	0.109	0.299

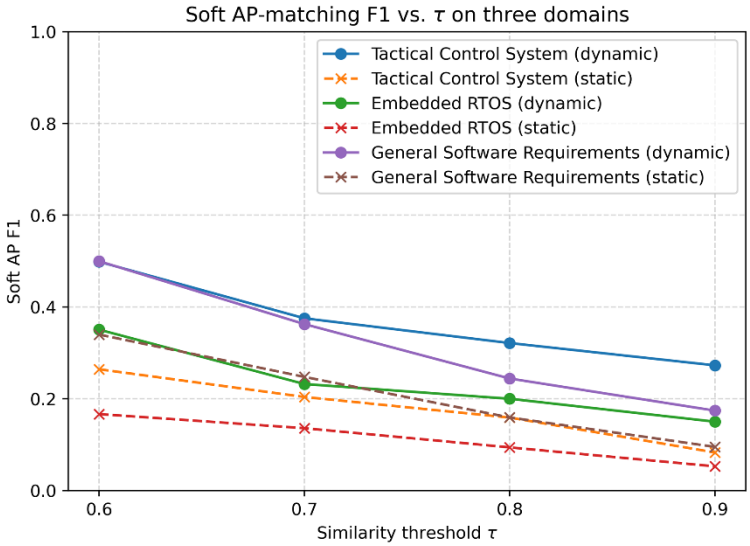

Fig. 5. Soft AP-matching F1 under different similarity thresholds τ on three domains. Dynamic prompting consistently outperforms static prompting.

the overall number of correct formulas from 25/120 to 54/120. These results suggest that retrieval-augmented, domain-consistent examples provide more effective in-context guidance for both AP grounding and semantic adequacy.

4.5 RQ3: Model-Aware NL2CTL with nuXmv Models

We finally evaluate model-aware NL2CTL on D2, where generated CTL formulas must be directly checkable on concrete `nuXmv` models[20].We compare three prompting modes:*Pure-NL*,which uses only the NL requirement; *Full-SMV*, which provides the full `.smv` model as context; and *AP-List*, which supplies only the extracted atomic-proposition list. For each requirement, we generate $n = 10$ candidates and select the final output using *Freq-first* with length-normalized log-likelihood tie-breaking. We

evaluate both nuXmv-level verifiability and semantic correctness in order to characterise the trade-off between model compatibility

Table 4. Semantic correctness of generated CTL formulas on Natural2CTL.

Domain	Static	Dynamic	Improvement
TCS (40)	7/40 (17.5%)	19/40 (47.5%)	+12
RTOS (40)	12/40 (30.0%)	15/40 (37.5%)	+3
GSR (40)	6/40 (15.0%)	20/40 (50.0%)	+14
Overall (120)	25/120 (20.8%)	54/120 (45.0%)	+29

Table 5. Verifiability and grounding errors on the model-aware dataset D2 (150 NL properties).

Method	Verifiable	Undefined AP	Type errors	Other syntax
Pure-NL	6	144	0	0
Full-SMV	141	3	1	5
AP-List	139	1	10	0

and semantic flexibility. For clarity, **Table 7** consolidates the main findings of **Table 5** and **Table 6** into a single view of failure modes and the semantics–verifiability trade-off.

Verifiability and failure modes.**Table 5** reports direct verifiability and error breakdowns. Pure-NL performs poorly, with only 6/150 formulas accepted by nuXmv; its failures are dominated by undefined atomic propositions (144 cases), indicating that unconstrained generation often produces predicates tha3 are semantically plausible but absent from the target model vocabulary. In contrast, both model-aware settings greatly improve checkability: Full-SMV yields 141 verifiable formulas, while AP-List yields 139 and reduces undefined-identifier errors to a single case. This shows that model-aware prompting primarily improves verifier compatibility by constraining the generated vocabulary.

Most remaining failures in Full-SMV and AP-List are no longer grounding failures, but type- or syntax-level issues. In particular, a simple enum-negation pattern, where!x = v should be normalised to !(x = v), accounts for a large fraction of the residual non-verifiable cases. After this lightweight post-processing, verifiable formulas increase to 142/150 for Full-SMV and 149/150 for AP-List, while Pure-NL remains at 6/150. These results suggest that once grounding is enforced, the remaining gap is largely due to surface-form compatibility rather than missing model variables.

Semantic correctness and trade-off.**Table 6** summarises semantic correctness on the three case-study models. Pure-NL achieves the highest overall semantic accuracy at 99/150 (66.0%), whereas Full-SMV and AP-List both obtain 89/150 (59.3%). These results do not support a uniform improvement in semantic quality from model-aware grounding. Rather, the results indicate a trade-off: unconstrained generation is more likely to produce semantically plausible pred

Table 6. Semantic correctness on the model-aware dataset D2 (50 NL properties per model).

Model	Pure-NL	Full-SMV	AP-List
Traffic	37/50 (74%)	32/50 (64%)	31/50 (62%)
ABP4	33/50 (66%)	26/50 (52%)	30/50 (60%)
UMS	29/50 (58%)	31/50 (62%)	28/50 (56%)
Overall	99/150 (66%)	89/150 (59.3%)	89/150 (59.3%)

Table 7. Summary of failure modes and the semantics–verifiability trade-off on D2, compiled from Table 5 and Table 6, together with the lightweight-normalisation results reported in the text.

Method	Verifiable	Verifiable + norm.	Undefined AP	Type / syntax	Semantic correctness
Pure-NL	6	6	144	0	99/150 (66.0%)
Full-SMV	141	142	3	6	89/150 (59.3%)
AP-List	139	149	1	10	89/150 (59.3%)

icates, but many such predicates are not part of the actual nuXmv vocabulary and hence cannot be checked.

Between the two model-aware variants, Full-SMV performs best on UMS, while AP-List slightly outperforms it on ABP4. More importantly, AP-List achieves semantic correctness comparable to Full-SMV while almost eliminating undefined identifiers and keeping the prompt substantially more compact. In this sense, AP-List provides the most practical balance among semantic adequacy, grounding accuracy, and verification compatibility.

Overall, the RQ3 results indicate that the main benefit of model-aware grounding lies in improving direct checkability, rather than in uniformly increasing semantic accuracy. Pure-NL often generates semantically reasonable but unverifiable formulas, whereas AP-List preserves comparable semantic adequacy to Full-SMV while yielding the strongest practical compatibility with nuXmv. As in RQ2, semantic correctness is annotated by two CTL/model-checking annotators under a blinded protocol, yielding substantial agreement (Cohen's $\kappa = 0.71$).

5 Conclusion

This paper presented GenCTL, a prompt-based framework for NL2CTL that supports both model-agnostic translation and model-aware generation with LLMs. Rather than relying on task-specific fine-tuning,GenCTL combines model-aware atomic-proposition grounding with retrieval-enhanced prompting, frequency-first candidate selection, and optional explanation-dictionary-based refinement to improve grounded identifier alignment, output stability, and practical verifier compatibility. Taken together, these components contribute to different aspects of practical NL2CTL: retrieval-enhanced prompting mainly improves semantic guidance and AP recovery, frequency-first selection improves robustness under stochastic decoding, and model-aware grounding strengthens verifier-level compatibility under concrete model constraints.

Experiments in both model-agnostic and model-aware settings demonstrate the practical value of this design. In model-agnostic settings, frequency-first selection improves robustness over alternative ranking rules, retrieval-enhanced prompting improves AP grounding and semantic correctness on public Natural2CTL data, and guided refinement further improves accuracy in a single-participant feasibility setting. In model-aware settings, AP-list prompting substantially improves direct checkability on concrete nuXmv models, while RQ3 further shows that this benefit mainly lies in verifier compatibility rather than in uniformly improving semantic accuracy. Overall, the results suggest that prompt-based NL2CTL can be made more practical by combining semantic guidance, grounded identifier alignment, and checker-level compatibility.

This study has several limitations. The current evaluation remains limited in scale, including a generated NL–CTL set for RQ0/RQ1 and a manually constructed model-aware benchmark over three nuXmv models. As a result, the generated subset of D1 may still reflect construction bias from prior NL2CTL data-generation procedures, and D2 should be viewed as a controlled case-study benchmark over concrete models rather than a broad independent benchmark of general NL2CTL capability. The interactive refinement experiment is a single-participant feasibility study rather than a controlled user study. In addition, the current candidate selection strategy is string-based and does not account for semantically equivalent but syntactically different CTL formulas. Finally, although model-aware grounding substantially improves direct checkability, it may reduce semantic flexibility in some cases, and the overall performance still depends on stochastic LLM decoding. Future work will consider larger and more independent benchmarks, equivalence-aware candidate aggregation, additional model checkers and temporal logics, and stronger automatic semantic validation methods.

Acknowledgments

This work is supported by National Natural Science Foundation of China Projects (No.92370201), and Shanghai Fenglei Information Technology National SRDI Key “Little Giant” Program.

References

1. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Amodei, D., et al.: Language Models are Few-Shot Learners. In: Advances in Neural Information Processing Systems 33, pp. 1877–1901 (2020)
2. Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Chang, B., Sun, X., Li, L., Sui, Z.: A Survey on In-Context Learning. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp. 1107–1128 (2024). https://aclanthology.org/2024.emnlp-main.64.pdf
3. Zrelli, R., Amaral Misson, H., Ben Attia, M., Gohring de Magalhães, F., Shabah, A., Nicolescu, G.: Natural2CTL: A Dataset for Natural Language Requirements and Their CTL Formal Equivalents. In: International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 205–216. Springer Nature Switzerland, Cham (2024)

4. Zhao, M., Tao, R., Huang, Y., et al.: NL2CTL: Automatic Generation of Formal Requirements Specifications via Large Language Models. In: International Conference on Formal Engineering Methods, pp. 1–17. Springer, Singapore (2024)
5. Rubin, O., Herzig, J., Berant, J.: Learning to Retrieve Prompts for In-Context Learning. In: Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 2655–2671 (2022). https://doi.org/10.18653/v1/2022.naacl-main.191
6. Wang, X., Wei, J., Schuurmans, D., Le, Q.V., Chi, E.H., Narang, S., Chowdhery, A., Zhou, D.: Self-Consistency Improves Chain of Thought Reasoning in Language Models. In: The Eleventh International Conference on Learning Representations (ICLR) (2023). https://openreview.net/forum?id=1PL1NIMMrw
7. Chen, Y., Gandhi, R., Zhang, Y., Fan, C.: NL2TL: Transforming Natural Languages to Temporal Logics Using Large Language Models. arXiv preprint arXiv:2305.07766 (2023). https://arxiv.org/abs/2305.07766
8. Wang, J., Sundarsingh, D.S., Deshmukh, J.V., Kantaros, Y.: ConformalNL2LTL: Translating Natural Language Instructions into Temporal Logic Formulas with Conformal Correctness Guarantees. arXiv preprint arXiv:2504.21022 (2025). https://arxiv.org/abs/2504.21022
9. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971 (2023). https://arxiv.org/abs/2302.13971
10. Rabiei, B., Dai, Z., Pilla, S.L.S.R., Dong, Q., Atanasov, N., et al.: LTLCodeGen: Code Generation of Syntactically Correct Temporal Logic for Robot Task Planning. arXiv preprint arXiv:2503.07902 (2025). https://arxiv.org/abs/2503.07902
11. Liu, J., Shen, D., Zhang, Y., Dolan, B., Carin, L., Chen, W.: What Makes Good In-Context Examples for GPT-3? In: Proceedings of the 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures (DeeLIO 2022), pp. 100–114 (2022). https://aclanthology.org/2022.deelio-1.10.pdf
12. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., et al.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: Advances in Neural Information Processing Systems 35, pp. 24824–24837 (2022)
13. Zhou, Y., Muresanu, A.I., Han, Z., Paster, K., Pitis, S., Chan, H., Ba, J.: Large Language Models Are Human-Level Prompt Engineers. In: Proceedings of the 11th International Conference on Learning Representations (ICLR) (2023). https://openreview.net/forum?id=92gvk82DE-
14. Zhang, Z., Zhang, A., Li, M., Smola, A.J.: Automatic Chain of Thought Prompting in Large Language Models. In: Proceedings of the International Conference on Learning Representations (ICLR 2023) (2023). https://arxiv.org/abs/2210.03493
15. Holt, A., Klein, E.: A Semantically-Derived Subset of English for Hardware Verification. In: Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, pp. 451–456 (1999). doi: 10.3115/1034678.1034747
16. Harris, C.B., Harris, I.G.: GLAsT: Learning Formal Grammars to Translate Natural Language Specifications into Hardware Assertions. In: Proceedings of the 2016 Conference on Design, Automation & Test in Europe (DATE 2016), pp. 966–971 (2016)
17. Krishnamurthy, R., Hsiao, M.S.: EASE: Enabling Hardware Assertion Synthesis from English. In: Rules and Reasoning: Third International Joint Conference, RuleML+RR 2019, Bolzano, Italy, September 16–19, 2019, Proceedings, pp. 82–96 (2019). doi: 10.1007/978-3-030-31095-0_6
18. Reimers, N., Gurevych, I.: Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural

Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 3982–3992 (2019). https://aclanthology.org/D19-1410
19. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., et al.: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research 21(140), 1–67 (2020)
20. Cavada, R., et al.: The nuXmv Symbolic Model Checker. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification (CAV 2014). LNCS, vol. 8559, pp. 334–342 (2014). doi: 10.1007/978-3-319-08867-9_22