From English to Eventually: LLMs' Efficacy in Translating Natural Language Requirements to Linear Temporal Logic (LTL)

No Author Given

No Institute Given

Abstract. Linear Temporal Logic (LTL) is a popular choice for capturing desirable requirements of software and systems to be verified, monitored, or analyzed using automated reasoning. Despite its popularity, expressing such requirements in LTL has been challenging due to the intricacies of LTL semantics. With the advent of Large Language Models (LLM), a promising approach to address this challenge is to use LLMs to automatically translate user-provided natural language descriptions of desired properties into their corresponding LTL representation. *In this paper, we set out to evaluate the efficacy of existing representative LLM approaches for translating assertive English sentences into their LTL counterparts.* Our evaluation uses both human-generated and synthetic ground-truth data and examines the LLMs' effectiveness across both syntactic and semantic dimensions of LTL. Our evaluation revealed three notable aspects: (1) LLMs, as expected, perform better on syntactic aspects of LTL than on semantic ones; (2) they generally perform better with more detailed prompts; and (3) reformulating the task as a Python code-completion problem substantially improves overall performance. We also discuss the challenges of conducting a fair evaluation and conclude with recommendations for future research.

1 Introduction

There is an ongoing interest of combining the capabilities of Large Language Models (LLMs) and automated reasoning (AR) [98, 52]. Such combination often exhibits a mutualistic symbiosis: (i) AR tools can not only enhance LLMs' symbolic reasoning capabilities but also reduce hallucinating behavior [103]; (ii) LLMs, on the other hand, can substantially improve the usability of AR tools by exposing a natural languages (`NL`) interface to the users of these tools [11]. To facilitate such mutualistic symbiosis, in many cases, LLMs have to translate `NL` assertive sentences to their formal logic representation. *In this paper, we evaluate the efficacy of existing, representative LLMs in translating assertive* `NL` *sentences to their corresponding propositional Linear Temporal Logic (LTL) formulas*; a task we refer to as `NL`$\rightarrow$`LTL`.

In the formal methods and requirements engineering community, `LTL` has gotten a lot of traction as a natural logic of choice for formally expressing guarantees, security and privacy policies, and requirements of systems, software, and

networks that are amenable to automatic analysis [82], runtime monitoring [8], testing [34], and synthesis [31]. Despite its popularity, manually expressing NL requirements in LTL can be challenging due to inherent ambiguity in natural languages [40] and intricacies in LTL semantics. This task is shown to be challenging even for experts, let alone regular users of formal analysis tools [40]. Hence, one of the main impediments towards wide adoption of many formal verification and automated reasoning tools that use LTL as a specification language [54] is the manual step of NL→LTL conversion.

To reduce users' burden, there have been research efforts to alleviate this error-prone, manual step [26, 55]. One such effort analyzes existing properties appearing in the literature and devises frequently-occurring property patterns that can help users systematically, albeit manually, translate their NL requirements in LTL [26]. Other efforts focus on developing automated tools that can translate restricted fragments of NL requirements to their LTL counterparts [9, 29]. Despite these efforts, there have an ongoing interest in translating free-form NL requirements to LTL, and other logics [65, 22]. LLMs hold the promise of automating the error-prone and challenging LTL specification process through NL→LTL. In addition to the proprietary, general-purpose LLMs [83, 84], there are efforts on fine-tuning some open-source models for the NL→LTL task [81]. *Although these approaches have shown promising results, what is missing in the literature is a systematic evaluation of these LLMs' efficacy for* NL→LTL.

This paper presents such an in-depth evaluation in which we collect NL→LTL ground-truth datasets from various sources including textbooks [63, 32, 50], papers [40], and manually constructed ones [14], and then evaluate representative general-purpose LLMs from two different perspectives: *syntax* and *semantics* of LTL. We note that there are some LLMs that carry out this NL→LTL translation through a collaborative human-in-the-loop process [12], whereas there are others, which perform the translation fully automatically, without any substantial human intervention [97]. *This paper primarily focuses on evaluating the latter.*

At a conceptual level, for evaluating an LLM's efficacy for a given ground-truth data entry $\langle s : \texttt{NL}, f_g : \texttt{LTL}\rangle$, one would first feed the NL s (along with the appropriate prompting text) to the LLM-under-test, and obtain the LTL representation f from the LLM. To evaluate whether the LLM-under-test's output LTL formula f matches with the ground-truth version, one can rely on logical equivalence checking: $f \equiv^{?} f_g$. Unfortunately, realizing this natural high-level approach has a major challenge: *ontological problem of choosing the same set of propositions* as the ground-truth. Concretely, checking whether $f \equiv^{?} f_g$ is not meaningful when f and f_g are using different sets of propositional variables. This problem is further exacerbated by fine-tuned LLMs having a restricted input-output interface ($\texttt{LLM}_{\texttt{ft}}^{\texttt{RI/O}}$) where they take an NL and just output a corresponding LTL formula. This precludes the possibility of using prompting text to provide a specific mapping of NL fragments to the appropriate propositional variables. As a result, these $\texttt{LLM}_{\texttt{ft}}^{\texttt{RI/O}}$s are not amenable to a fully automated and large scale evaluation. A natural follow up question is whether these $\texttt{LLM}_{\texttt{ft}}^{\texttt{RI/O}}$s do map the same NL fragment to the right corresponding propositional variables as in the

ground-truth but may be use a different variable name (*e.g.*, using p instead of q). If it is the case, it is conceivable to devise an automatic *variable renaming* scheme to solve the ontological problem discussed above.

As a follow-up, we constructed a NL to propositional logic (PL) ground-truth dataset (recall that, PL is a sub-logic of LTL), and manually evaluated the $\texttt{LLM}_{\texttt{ft}}^{\texttt{RI/O}}$-under-tests' efficacy of selecting the correct NL fragments to map to propositional variables. Unfortunately, we observed that these $\texttt{LLM}_{\texttt{ft}}^{\texttt{RI/O}}$s violate *the principle of maximal logical revelation* (*i.e.* , always translate to reveal as much of the logical structure as the target language supports) [40]. We observed the similar unsatisfactory performance from proprietary general-purpose LLMs (which do not have such restricted I/O interface) - referred to as $\texttt{LLM}_{\texttt{gp}}$ from now on - in selecting propositional variables while respecting the principle of maximal logical revelation, especially, when not explicitly instructed to do so in the prompt. Fortunately, one can rely on prompting text to provide the fixed mapping to the $\texttt{LLM}_{\texttt{gp}}$s to side-step this ontological problem, although many of them do not strictly adhere to the provided mapping when choosing the propositions.

In terms of features, we observe that almost all $\texttt{LLM}_{\texttt{ft}}^{\texttt{RI/O}}$s only support LTL with only future temporal operators. Although adding past temporal operators to the vanilla LTL (*i.e.* , containing next and until operators) does not increase the expressive power of LTL, it allows one to express certain formulas more succinctly with past operators (*i.e.* , since and yesterday). In addition, past-only fragment of LTL captures monitorable safety properties with standard semantics for finite traces, and hence is a popular choice for capturing security properties [37]. We also observed that almost $\texttt{LLM}_{\texttt{gp}}$s are better at translating to the future-only fragment of LTL compared to past-only fragment of LTL. Finally, $\texttt{LLM}_{\texttt{ft}}^{\texttt{RI/O}}$s, due to their restricted input-output interface, also do not allow us to carry out many of the syntactic aspects of our evaluation, and some semantic aspects of our evaluation. Overall, for $\texttt{LLM}_{\texttt{gp}}$s, we observe that they are substantially better at the syntactic aspects over the semantic aspects of LTL and performs substantially better when a detailed prompt covering all the nuanced aspects of the task is used, which matches the conventional wisdom.

The most notable finding of our evaluation is that reformulating the NL→LTL task to a Python code completion task substantially improves $\texttt{LLM}_{\texttt{gp}}$'s overall performance. Despite this substantial improvement in performance, one surprising fact was that in one of the experiments about LTL semantics, which essentially boils down to executing a Python program, the $\texttt{LLM}_{\texttt{gp}}$s, despite being equipped with Python interpreters, did not achieve full accuracy.

To summarize, the paper has the following contributions:

1. We develop the first systematic and in-depth evaluation of LLMs' efficacy on the NL→LTL translation from both syntactic and semantic aspects of LTL.
2. We identify rooms for improvement of $\texttt{LLM}_{\texttt{ft}}^{\texttt{RI/O}}$ for the NL→LTL task.
3. We observe that $\texttt{LLM}_{\texttt{gp}}$s substantially perform better when the NL→LTL task is reformulated as Python code completion and comprehension task.

4. We also present technical challenges with having a fair evaluation and then conclude with recommendations for future evaluations of LLMs, especially developed for the NL→LTL task.

Our artifacts are available in the following anonymous Github [2].

2 Research Questions and Their Motivations

We now outline the research questions driving our study and their motivations. Our evaluation goes beyond the standard end-to-end NL→LTL task, and for $\texttt{LLM}_{\texttt{gp}}$s also aims to gauge their proficiency in answering questions about syntactic (*e.g.*, well-formedness) and semantic (*e.g.*, trace classification) aspects of LTL.

Motivation for Evaluating LLMs' LTL Reasoning Capabilities. A natural question readers may have is that when there are symbolic reasoners available for carrying out different reasoning tasks (*e.g.*, model checkers, reactive synthesizer) what is the point of using LLMs for such semantic reasoning tasks (See Section B). There have been a growing interests in using LLMs for different temporal reasoning tasks (*e.g.*, satisfiability [62], model checking [7, 90], reactive synthesis [72]). Our evaluation aims to answer whether the LLMs have a basic understanding of the LTL semantics. In case LLMs cannot correctly answer simple LTL semantics-related queries (*e.g.*, trace characterization), then it is unlikely to faithfully perform more complex semantic reasoning, which have substantial theoretical complexity (*e.g.*, PSPACE-complete for LTL satisfiability).

A Syntactic Evaluation

$\mathbb{RQ}_{\mathbf{1}}^{\mathsf{syn}}$: *How effectively can LLMs extract atomic propositions from natural language?* Validating the output of these LLMs become a manual and timely ordeal without allowing users to explicitly provide any mapping from natural language sentence fragments to propositional variables. It is thus paramount to understand how effective are $\texttt{LLM}_{\texttt{ft}}^{\texttt{RI/O}}$s at identifying propositional variables given a natural language assertive sentence. Note that, for this experiment, we limit ourselves to only propositional logic formulas, ignoring temporal connectives, to validate the LLMs efficacy manually.

$\mathbb{RQ}_{\mathbf{2}}^{\mathsf{syn}}$: *How accurately can LLMs distinguish well-formed from malformed* LTL *formulas?* This question evaluates whether these LLMs' understanding of the syntactic structure of LTL formulas, including the correct use of connectives, modalities, and parentheses. LLMs as we know them have been trained on diverse data hence, we hypothesize they have internalized common patterns in the representation of LTL formulas. We specifically evaluate LLMs' ability to correctly identify whether a given formula is well-formed.

B Semantic Evaluation

$\mathbb{RQ}_{\mathbf{1}}^{\mathsf{sem}}$: *Can LLMs generate LTL formulas that are semantically equivalent to the intended ground truth?* Two formulas can be syntactically different (in representation) but semantically equivalent (in meaning). This research question

investigates whether LLMs can produce LTL formulas that are semantically indistinguishable from a reference formula, even when they differ in structure. We reduce equivalence checking between the LLM-predicted formula and the ground truth formula to a model checking problem that we solve with NuSMV.

$\mathbb{RQ}_{\mathbf{2}}^{\mathsf{sem}}$: *Can LLMs accurately assess whether a trace satisfies a given formula?* Beyond generating traces, LTL reasoning is determining whether a given trace satisfies a formula. This ability is foundational for applications in model checking and automated verification, where traces need to be validated against system specifications. We evaluate for a given trace an LLM's ability to accurately classify as satisfied or falsified for a given LTL formula.

$\mathbb{RQ}_{\mathbf{3}}^{\mathsf{sem}}$: *Can LLMs generate satisfiable traces for a given* LTL *formula?* Generating valid traces requires an understanding of LTL semantics, where traces are sequences of boolean valuations over time. For instance, given the formula $\mathbf{G}a$ (meaning "a is always true"), an LLM must generate an infinite trace where a holds at every step. Unlike pure sequence generation tasks, this requires LLMs to respect global constraints imposed by LTL formulas.

$\mathbb{RQ}_{\mathbf{4}}^{\mathsf{sem}}$: *Can LLMs generate semantically equivalent* ***Past***-LTL *formulas from natural language descriptions?* This question explores whether LLMs can correctly handle past temporal reasoning. Although past temporal operators (*e.g.*, historically) can be viewed as mirrors of their future counterpart (*e.g.*, always) and do not add expressiveness to LTL, correctly reasoning about them can be tricky and would require a deeper semantic understanding. We test if LLMs can accurately translate NL descriptions with past temporal aspects to semantically equivalent Past-LTL formulas.

3 Prompting Categories

In our evaluation, we consider three categories of prompts: two are natural language prompts with varying level of details whereas the last one formulates the different tasks into a Python code completion and/or comprehension task.

Minimal Prompt. This prompt is considered a lower bound in our study due to its succinctness and not having any specialized instruction and formal context. The models are simply instructed to respond to LTL queries with brief instructions, minimal symbol definitions, and no formal grammar or semantics. The goal of this strategy is to test whether general-purpose LLMs can truly infer and yield accurate results even with minimal information and guidance from their pre-trained knowledge. We will refer this approach as Minimal.

Detailed Prompt. This is an enriched instructional prompt with a comprehensive background on benchmark syntax rules comprising of the full BNF grammar, semantic definitions, operator constraints, valid symbol usage, and trace syntax conventions. This approach stems from the hypothesis that LLMs benefit from rich contextual information when dealing with formal logic domains which we are going to refer to as Detailed. The detailed prompt strategy hypothesizes that providing extensive domain knowledge upfront will enable better reasoning by establishing clear definitions for all temporal operators while providing

formal semantics for trace evaluation by offering concrete examples of operator behavior which turn reduces ambiguity through mathematical precision.

Python Code Completion Prompt. LLMs have demonstrated exceptional proficiency in code generation and completion tasks, a capability stemming from their extensive pretraining on vast code corpora, with Python being at the forefront. In this strategy which we refer to as PYTHON, we reduce the NL→LTL task to either a code completion or comprehension task. We define the abstract syntax tree (AST) of an LTL formula as an algebraic data type (implemented with dataclasses) and a trace as finite list of states, which are mapping of atomic propositions to Boolean values. We define the semantics of LTL with a recursive function that pattern matches the constructor and recursive evaluates the formula. As an example, for the trace generation task, we require LLMs to fill up the trace for the given formula AST for which the evaluation function will return true. Although there are other approaches for converting the different tasks to a python code completion/comprehension task, we argue that this is the most natural and direct approach for encoding. Although PYTHON requires more input setup, the goal is to improve the overall performance on semantically intensive tasks by aligning them to LLM's code reasoning and generation capabilities.

4 Methodology

This section presents the datasets, the experimental settings, evaluation metrics, and validation methodology we used to answer the research questions.

A Datasets

We curated multiple ground-truth datasets from diverse sources to enable comprehensive evaluation. We note that evaluating data contamination precisely is beyond this work's scope, but we provide dataset details to enable future researchers to investigate this concern.

$\mathbb{DS}_{\mathbf{1}}^{\mathsf{tricky}}$: This dataset is from the work by Greenman *et. al* in their *Little Tricky Logic* study [40] which includes three tasks: paraphrasing LTL to English, translating English to LTL, and judging whether a trace satisfies a given formula. It captures practical challenges in understanding and applying LTL. This dataset plays an integral role in our study of NL→LTL. Each of the 306 entries in this dataset includes a column for natural language specification, the ground truth LTL formula, and a mapping of atomic propositions identified in the natural language to the variables used in the logical formula.

$\mathbb{DS}_{\mathbf{2}}^{\mathsf{syntax}}$: We randomly generated both well- and ill-formed formulas by randomly walking a context-free grammar the syntax of LTL. Well-formed formula abides by the syntactic representation of an LTL formula, while an ill-formed formula would be missing a parenthesis, a misplaced modality, an atomic variable or a combination of any of these. This dataset contains 299 randomly generated formulas, of which 150 were well-formed and 149 were non-well-formed. The complexity of the formulas (calculated with AST depth) ranges from level 0

to 6, with 0 being a basic atomic proposition and level 6 comprising operator nesting, incorporating a diverse range of propositional, unary temporal, and binary temporal operators.
$\mathbb{DS}_{\mathbf{3}}^{\mathsf{prop}}$: This data from textbooks [86, 94, 47], real-world scenarios, and problem sets. A total of 144 entries containing the natural language, its propositional logic formula, and the mapping of the extracted `NL` to atomic propositions were in the curated dataset. We use this data solely for answering $\mathbb{RQ}_{\mathbf{1}}^{\mathsf{syn}}$.
$\mathbb{DS}_{\mathbf{4}}^{\mathsf{trace}}$: This dataset was constructed from the `LTL` formulas in the $\mathbb{DS}_{\mathbf{1}}^{\mathsf{tricky}}$ dataset. We generated traces that would formalize as our ground truth dataset using `NuSMV` for each `LTL` formula creating both satisfying (positive) and violating (negative). These ground truth traces were then randomized to ensure counterbalancing during the evaluation against the LLM-generated response. This dataset is used for trace generation and trace classification with the goal to support evaluation scenarios that require reasoning about trace satisfaction.
$\mathbb{DS}_{\mathbf{5}}^{\mathsf{pastltl}}$: We adapted $\mathbb{DS}_{\mathbf{1}}^{\mathsf{tricky}}$ in the creation of this dataset by rephrasing `NL` originally using future to use past temporal expressions. It introduces Past `LTL` operators and supports assessment of LLMs' translation to past-time temporal logic. Having a total of 294 records, just like the future dataset, each entry in this dataset includes the natural language specification, the ground truth Past `LTL` formula, and a mapping of atomic propositions.
$\mathbb{DS}_{\mathbf{6}}^{\mathsf{book}}$: This dataset contains `LTL` formulas from textbooks on logic and formal specifications [63, 32, 50, 14] and contains `NL` statements involving logical and temporal reasoning about complex real-world specifications into formal logic. We collated a total of 141 records of natural language and their respective `LTL` formulas and derived their atomic proposition mappings with the complexity of the generated `LTL` formula dependent on the natural language description.

B Prompting Techniques

Zero-Shot Prompting. This approach relies solely on the model's knowledge acquired during its pretraining phase. The usage of this approach is to analyze the intrinsic understanding of `LTL` without any guidance or cues within the prompt itself with the goal of evaluating its base performance. In our work, the models are merely asked to "*Generate the answer only, without any explanation.*"
Few-Shot Prompting. The models are provided a small diverse set of annotated examples specific to the task at hand. This approach aims to test whether cues in terms of structured examples improve the model's performance. To address concerns about few-shot sensitivity, we fixed the same set of examples across all experiments to ensure reproducibility. We note that systematic sensitivity analysis over different few-shot sets is deferred to future work.
Zero-Shot Self-Refine Prompting. This is a progression of the zero-shot prompt. The initial response from zero-shot prompting is followed by an iterative process where the model is prompted to refine its own output if their initial-provided response is an aberration from what they had intended to provide. The LLM is prompted again with its initial response and asked to improve or refine the

answer. For example, our prompt will be of the following form: "*Your initial response was: {initial_ response}. Now, refine your answer.*" This iterative process allows us to observe if models can self-correct errors and towards the goal of a more accurate interpretations of LTL statements. This approach is adapted from the work of Madaan et al. [64]. While there exist refinement with external feedback, we explore the capabilities of LLMs on their own parametric knowledge.

C Experiments

Each experiment evaluates a specific task using a representative dataset. For Future LTL Synthesis ($\mathbb{RQ}_{\mathbf{1}}^{\mathsf{sem}}$), we test on both Dataset 1 and Dataset 6 to assess generalization across independent ground-truth sources described in Table 1.

For each model-dataset-prompting combination, the LLM receives the input (NL, formula, or trace) along with the prompt and generates an output. The output is automatically verified against ground truth using the metrics described below or manually inspected when automatic verification is infeasible.

$\mathbb{E}_{\texttt{nl2pl}}(NL) \rightarrow$ **(phrase: AP)**: Given NL text, models extract atomic propositions (phrases from NL mapped to formal variables). Evaluation uses Levenshtein distance (for structural variation while preserving meaning) and Jaccard similarity (for lexical overlap). This prerequisite task supports investigating the maximal logical revelation problem.

$\mathbb{E}_{\texttt{wff}}(formula) \rightarrow$ **(Yes/No)**: Given an LTL formula, models classify it as well-formed or ill-formed. Evaluation uses accuracy, precision, recall, and F1-score with emphasis on F1 due to class balance.

$\mathbb{E}_{\texttt{nl2ltl}}(NL, AP) \rightarrow$ **(LTL Formula)**: Given NL and fixed AP mappings, models generate future-time LTL formulas. All three prompting strategies are employed. For Python, models output Python AST representations. Semantic equivalence is verified via NuSMV with entailment classification as secondary metric.

$\mathbb{E}_{\texttt{tracechar}}(LTL, Trace) \rightarrow$ **(Yes/No)**: Given LTL formula and execution trace, models classify whether the trace satisfies the formula. Evaluation uses accuracy and F1-score. For Python, formulas are in Python AST format.

$\mathbb{E}_{\texttt{tracegen}}(formula) \rightarrow (\tau^+/\tau^-)$: Models generate satisfying (positive) and violating (negative) traces for a given LTL formula. Positive/negative order is randomized per formula. Generated traces are verified for correctness using NuSMV.

$\mathbb{E}_{\texttt{nl2pltl}}(NL, AP) \rightarrow$ **(Past LTL Formula)**: Mirroring future LTL generation, models generate past-time LTL formulas given NL and AP mappings. Evaluation uses semantic equivalence verification via NuSMV.

D General LLMs Models ($\texttt{LLM}_{\texttt{gp}}$)

We assessed several state-of-the-art general-purpose LLMs: `GPT-3.5-Turbo`, `GPT-4o`, `GPT-4o-mini`, `Gemini-1.5−Pro`, `Gemini-1.5−Flash`, `Gemini-2.5−Flash` and `Claude-3.5−Sonnet`. These LLMs were chosen for their strong generative abilities and the diversity in their size, architecture, and training regimes, offering a broad view of how parametric knowledge and design impact performance [63, 12, 80, 102] on LTL-related tasks. For cost-effectiveness in token usage, coupled with the number of

Table 1: Task-Evaluation Mapping Across Experiments, Research Questions, Datasets, and Input-Output per experiments

Task	Experiment	RQ	Dataset	Size	Input → Output	Metrics
Phrase Extraction	$\mathbb{E}_{\text{nl2pl}}$	$\mathbb{RQ}_{\mathbf{1}}^{\text{syn}}$	$\mathbb{DS}_{\mathbf{3}}^{\text{prop}}$	144	nl → (phrase: var)	Jaccard, Levenshtein
Well-formedness Check	$\mathbb{E}_{\text{wff}}$	$\mathbb{RQ}_{\mathbf{2}}^{\text{syn}}$	$\mathbb{DS}_{\mathbf{2}}^{\text{syntax}}$	299	formula → Yes/No	Accuracy, F1
Future LTL Synthesis	$\mathbb{E}_{\text{nl2ltl}}$	$\mathbb{RQ}_{\mathbf{1}}^{\text{sem}}$	$\mathbb{DS}_{\mathbf{1}}^{\text{tricky}}$, $\mathbb{DS}_{\mathbf{6}}^{\text{book}}$	306, 141	(nl, ap) → ϕ_{future}	Semantic Equivalence
Trace Classification	$\mathbb{E}_{\text{tracechar}}$	$\mathbb{RQ}_{\mathbf{2}}^{\text{sem}}$	$\mathbb{DS}_{\mathbf{4}}^{\text{trace}}$	306	(formula, trace) → Yes/No	Accuracy, F1
Trace Generation	$\mathbb{E}_{\text{tracegen}}$	$\mathbb{RQ}_{\mathbf{3}}^{\text{sem}}$	$\mathbb{DS}_{\mathbf{4}}^{\text{trace}}$	306	formula → τ^+/τ^-	Trace Satisfaction
Past LTL Synthesis	$\mathbb{E}_{\text{nl2pltl}}$	$\mathbb{RQ}_{\mathbf{4}}^{\text{sem}}$	$\mathbb{DS}_{\mathbf{5}}^{\text{past}}$	294	(nl, ap) → ϕ_{past}	Semantic Equivalence

experiments in this study, we employed cost-effective models in the Google Gemini family as well as that of Anthropic. `Gemini-1.5−Flash`, `Gemini-2.5−Flash`, `Gemini-1.5−Pro` and `Claude-3.5−Sonnet` provided substantial cost savings without compromising output quality, making them the preferred choice of our evaluation pipeline. (See Appendix C for detailed models.)

E Evaluation Metrics

We conducted all experiments on a local Apple M2 Pro system using the LLMs' APIs, Scikit-Learn for classification metrics, and `NuSMV` for formal verification. $\mathbb{E}_{\texttt{wff}}$ and $\mathbb{E}_{\texttt{tracechar}}$ was evaluated by employing accuracy, precision, recall, and F1-score: Precision $= \frac{TP}{TP+FP}$, Recall $= \frac{TP}{TP+FN}$, F1 $= \frac{2 \cdot P \cdot R}{P+R}$. F1 is emphasized in part due to class imbalance while accuracy measures overall correctness. Atomic proposition extraction is evaluated via Levenshtein distance and Jaccard similarity. The former is employed in cases where the predicted atomic proposition structure deviates from the ground truth while still retaining the meaning, while the latter measures lexical overlap by comparing the set of words in the extracted phrase and the ground truth. For semantic equivalence (*i.e.* , $\forall \sigma, \sigma \models \phi_{\text{pred}} \iff \sigma \models \phi_{\text{gt}}$) and entailment checking (*i.e.* , $\forall \sigma, \sigma \models \phi_{\text{pred}} \Rightarrow \sigma \models \phi_{\text{gt}}$ or $\forall \sigma, \sigma \models \phi_{\text{gt}} \Rightarrow \sigma \models \phi_{\text{pred}}$), we use `NuSMV`. Trace satisfiability is checked via model satisfaction $\tau \models \phi$ whereas trace generation for a formula ϕ also relies on `NuSMV` (*i.e.* , asserting $\neg\phi$ to be true in the most general model where propositions are assigned true/false non-deterministically).

5 Results

In this section, we present the experimental results from evaluating the performance of various LLMs' (generalized and fine-tuned) prediction (`Pred`) against the ground-truth (`GT`) by running the experiments presented in Section C using the three prompting strategies discussed in Section B.

$\mathbb{RQ}_{\mathbf{1}}^{\text{syn}}$: ***How effectively can LLMs extract atomic propositions from natural language?***

Our experimental results clearly demonstrate that a well-detailed prompt provided within the prompt significantly impacts the performance of task like atomic

proposition extraction, which in this study we consider it an integral part of the end-to-end pipeline of the LTL formula generation. Specifically, the DETAILED consistently yielded superior performance compared to the MINIMAL across various evaluation metrics. Table 2 depicts a summary of the aggregate performance improvements observed when transitioning from the MINIMAL to the DETAILED across all evaluated models. An average improvement of 16.5% was achieved in overall atomic proposition extraction efficacy. Two metrics were used for lexical

Table 2: Atomic Proposition Extraction: Aggregate Performance Summary Across All Models

Metric	**Minimal**	**Detailed**	Δ
Accuracy	52.19%	65.31%	+13.12%
Precision	58.45%	72.03%	+13.58%
Recall	59.64%	67.79%	+8.15%
F1-Score	57.96%	69.26%	+11.30%
Jaccard Sim.	0.83	0.86	+0.03
Levenshtein	3.66	3.05	–0.61

match in this study, with a higher Jaccard Similarity indicating and a lower Levenshtein Distance indicating the model's ability to correctly match the generated response with the ground truth. Hence, a 3.61% increase from 0.83 to 0.86 demonstrates that the DETAILED led to slightly more accurate identification and extraction of the relevant propositional phrases from the natural language input. Likewise, for Levenshtein Distance, a -16.67% decrease of 0.61 from 3.66 to 3.05 indicates that the phrases extracted by models using the DETAILED required fewer edits to match the ground truth.

One validation criterion was adherence to the principle of maximum revelation. We observed that many of these LLMs fail to follow the principle even in textbook cases. As an example, according to principle of maximum revelation the following English sentence "*Mary will* not *join the team and will* not *play the flute*" will be translated to the following propositional logic formula $\neg p \wedge \neg q$ where p is the propositional variable signifying the event "Mary will join the team" and q is "Mary will play the flute". Many of the LLMs, however, considered a whole sentence as a single propositional variable without exposing the logical negation in the sentence.

$\mathbb{RQ}_2^{\mathsf{syn}}$: *How accurately can LLMs distinguish well-formed from malformed LTL formulas?*

Our analysis of the benchmark results shows, again, that DETAILED achieved an average improvement of 10.86% in accuracy, 12.07% in precision, and 17.87% in F1-score. `GPT-4o` leads with MINIMAL, while `Gemini-1.5-Flash` excels with DETAILED. `Gemini-1.5-Flash` shows the largest accuracy gain of 31.82%. The error reduction rate of DETAILED was 20.4% across all models. Table 3 shows the accuracy per model for DETAILED and MINIMAL with both `GPT-4o` and

GPT-3.5-Turbo performing slightly better in this task. These results are truly

Table 3: Performance Comparison for Well-Formed Formula Classification ($\mathbb{E}_{\texttt{wff}}$): Minimal Prompt (MINIMAL) vs. Detailed Prompt (DETAILED)

Model	Prompt	Acc. (MINIMAL)	Acc. (DETAILED)	Δ Acc
Claude-Sonnet	Few-Shot	54.52	80.00	+25.48 ↑
Gemini 1.5	Few-Shot	52.51	84.33	+31.82 ↑
GPT-3.5-Turbo	Few-Shot	62.88	61.00	−1.88 ↓
GPT-4o	Few-Shot	74.58	66.00	−8.58 ↓
GPT-4o-mini	Few-Shot	58.19	65.67	+7.48 ↑

a testament to detailed prompting and their impact in our case, the task of identify whether a formula well-formed or not. The lack of perfect scores proves that our execution was not without fail. Our error analysis further revealed these errors stemmed from either syntactic misunderstandings, invalid identification of atomic propositions, and invalid operator identification. In MINIMAL, the most frequent error was from models rejecting valid formulas due to unfamiliar variable names. For instance, Claude-3.5−Sonnet incorrectly marked the formula X(H(X(O(wykepokyro)))) as non-WFF, stating that "wykepokyro" was not a valid LTL symbol. Such failures reveal a bias toward conventional atomic propositions where single letters or familiar words are mostly used in the representation of an atomic proposition, despite the fact that LTL allows arbitrary atomic symbols. A more dire scenario was the inconsistent enforcement of operator arity. Claude-3.5−Sonnet, for instance, misclassified the malformed formula (((S¬(TRUE)) | TRUE)SY(G(TRUE))) as well-formed, overlooking that the binary 'S' (Since) operator lacked a left-hand operand. Au contraire, it rejected the valid formula, claiming that 'Y' and 'H' were not valid LTL operators, despite previously accepting them. Being able to identify a modality in one formula and blatantly rejecting the same modality in another instance reflects a contradiction that emphasizes a lack of deeper syntactic reasoning and reveals LLMs' inconsistent judgment of LTL formulas, particularly those involving Past modalities.

$\mathbb{RQ}_1^{\text{sem}}$: *Can LLMs generate LTL formulas that are semantically equivalent to the ground truth?*

Our evaluations was in three folds: semantic equivalence analysis, logical entailment checking, and structural discrepancy comparison between ground truth and predicted formulas using the $\mathbb{DS}_1^{\text{tricky}}$ dataset. The observed lower percentage (-6%) for the syntactic correctness rate of PYTHON as depicted in Figure 1 stems from the fact that its output was explicitly requested in a Python Abstract Syntax Tree (AST) structure, rather than the standard LTL syntax that the Syntactic Correctness Rate metric is typically designed to evaluate. PYTHON prompting strategy still appears to be the strongest contender for NL→LTL.

Semantic Equivalence: The first analysis tackled the strictest criterion of logical equivalence of the ground truth with the predicted LLM response, ranges from a low 19.09% for GPT-3.5-Turbo under Zero-Shot Self-Refine to a high of

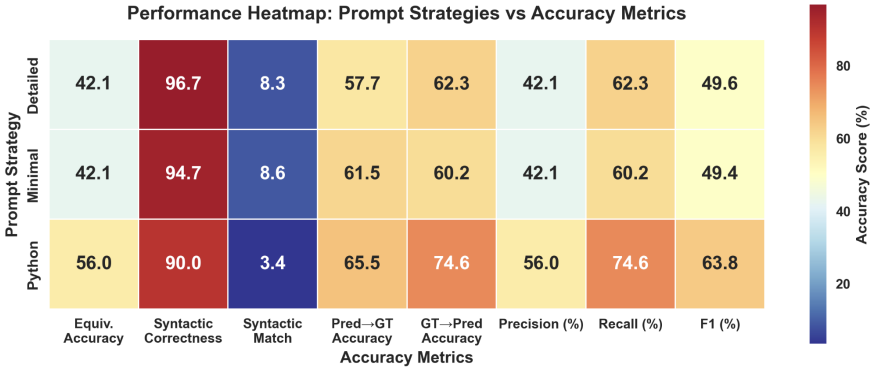

Fig. 1: Comparative Heatmap of Future LTL ($\mathbb{E}_{\texttt{nl2ltl}}$) for the different prompting strategies using $\mathbb{DS}_{\mathbf{1}}^{\text{tricky}}$

64.75% for `Claude-3.5−Sonnet`, under `Zero-Shot`. Given the low performance of `GPT-3.5-Turbo` it is our recommendation that they are used in the `NL→LTL` task. The overall semantic equivalence results indicate that, while LLMs can generate LTL, achieving precise semantic equivalence remains a significant challenge. Although the percentage change between DETAILED→MINIMAL is infinitesimal 0.1%, that of PYTHON →MINIMAL and PYTHON → DETAILED averaged an increment of 32.9% and 32.8% respectively. (See Appendix A) for more fine-grained results.

Entailment Analysis: We employ separately a less stricter logical equivalence by checking for both forward ($\varphi_{\text{GT}} \models \varphi_{\text{Pred}}$) and backward ($\varphi_{\text{Pred}} \models \varphi_{\text{GT}}$) entailment. The PYTHON consistently emerged as the best-performing prompting approach, significantly improving the models' ability to capture the full semantic completeness GT→Pred of the ground truth, showing statistically significant gains of +23.8% over MINIMAL and +19.8% over DETAILED strategies. While also leading in precision Pred→GT, these gains were not statistically significant. The DETAILED Strategy surprisingly showed no significant improvement over MINIMAL, and even a slight decrease in precision, indicating that simply adding more detail is not always beneficial for semantic accuracy.

Syntactic Equivalence: A notable observation is the consistently high syntactic correctness rate, with most models achieving 97% to 100%. This indicates that LLMs are highly proficient at generating syntactically valid LTL formulas when provided with a detailed prompt. However, the much lower semantic equivalence accuracy (stricter version) highlights a substantial gap that models can produce syntactically equivalent LTL that is nonetheless semantically incorrect. Table 1 summarizes our results. Our inspection of the incorrect outputs revealed the recurring issues while also distinguishing acceptable structural variations from true semantic errors. Models frequently misplace temporal operators, fail to preserve scope in nested expressions, or produce verbose representations for constraints. For example, the input "*No more than one thread can have the lock.*"

is best expressed as $\neg(x1 \wedge x2)$, but the model instead produces a longer disjunction enumerating all exclusive cases. In other cases, phrases like "*after the last state in which x1 holds*" are often misunderstood, leading to logical mismatches.

$\mathbb{RQ}_2^{\text{sem}}$: *Can LLMs accurately assess the satisfiability of a trace against a given formula?*

The experimental results for the trace characterization task are summarized in Figure 2. This task at the core was a classification problem, where the model checks whether a given formula satisfies or falsifies a given formula. Overall, the PYTHON strategy emerges as the best-performing approach for this task, achieving an average Accuracy of 74.457% and this is a significant +15.39% averaged increment. This represents a statistically significant improvement of +22.4% over the MINIMAL strategy of 60.807% and this is an +8.197% increment. Although, the DETAILED strategy produces a 65. 51% and this is a +10. 17% improvement over MINIMAL, it is not statistically significant and performs -12.0% worse than the Python Strategy.

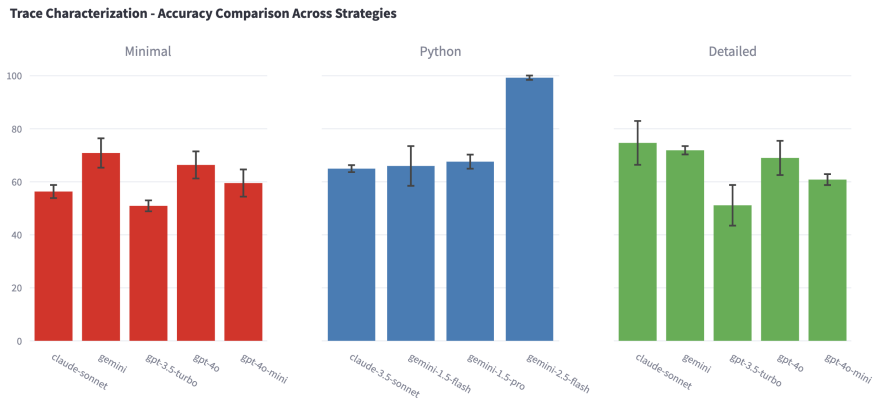

Fig. 2: Performance of LLMs on Trace Characterization ($\mathbb{E}_{\texttt{tracechar}}$) by the Prompting Strategies

We observed several patterns from the trace characterization results that failed the satisfability check. These errors were consistent with specific formulas having a bias of predicting a trace as "`Positive`" and also having difficulty with LTL formulas with complex temporal nesting. These errors gives us a glimpse of the tendency for LLMs to overgeneralize familiar-looking formulas or internalized traces instead of logical sequence-to-sequence reasoning. Some models also appear to be have difficulty with deeply nested temporal constructs like $\texttt{X}(\texttt{X}(\texttt{X}(\texttt{X}(\texttt{F}(x_1)))))$, predicting `Negative` despite the trace being satisfiable. Most models perform considerably better on simpler formulas such as $\texttt{F}(x_1)$ or $\texttt{G}(x_1)$, but exhibit significant flaws when multiple operators are involved. Some fail-

ures appear to stem from traces resembling memorized patterns, suggesting a tendency to match known templates rather than engage in symbolic reasoning.

$\mathbb{RQ}_3^{\text{sem}}$: ***Can LLMs generate valid traces for a given LTL formula?***
This research question evaluates LLMs' ability to generate valid positive (satisfying) and negative (violating) execution traces for a given LTL formula. This task directly assesses the models' understanding of LTL semantics in a generative context, which is crucial for applications in automated verification and counterexample generation. Some relevant trends include: DETAILED prompt strategy generally yields the highest performance across most models in with `Claude-3.5−Sonnet` under `Few-Shot` prompt with an accuracy of 74.51%. PYTHON

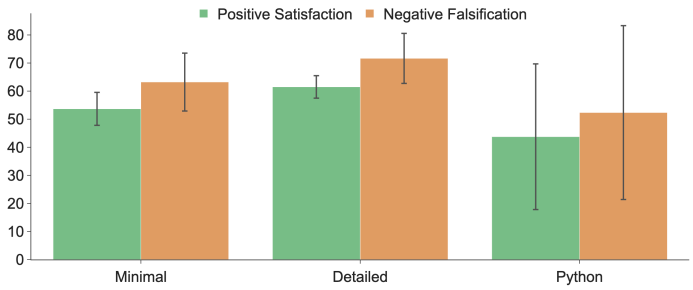

Fig. 3: Satisfaction and Falsification of Positive and Negative trace under the three prompting strategies.

models like `Gemini-2.5−Flash` achieve very strong results of 75.49% in accuracy under `Zero-Shot Self-Refine`. Figure 3 depicts the satisfaction and falsification of positive and negative traces respectively. This indicates that while the Python approach can unlock high performance, it is highly dependent on the specific model's architecture and its ability to handle the nuances of generating complex Python object instantiations for traces.

$\mathbb{RQ}_4^{\text{sem}}$: ***Can LLMs generate semantically equivalent Past-LTL formulas from natural language descriptions?***
Across all configurations, `Claude-3.5−Sonnet` Sonnet and `Gemini-2.5−Flash` consistently outperformed earlier versions and other models. In the DETAILED prompt setting, `Claude-3.5−Sonnet` using `Zero-Shot` achieved the highest syntactic correctness rate of 99.65% and precision of 69.68%, while `Gemini-1.5−Flash` under `Few-Shot` yielded the highest F1 score of 76.04%. However, these results dropped notably under the Minimal prompt format, underscoring the impact of rich natural language grounding on logical accuracy. Figure 4 shows that despite the lowest syntactic match rate of 0.6% among all the other strategies, Python employed in the task of translation managed to have a much higher semantic equivalence. Under the PYTHON prompt strategy where LTL formulas were framed in code-like syntax, `Gemini-2.5−Flash` reached the best performance overall. Its `Zero-Shot Self-Refine` variant achieved an F1 score of

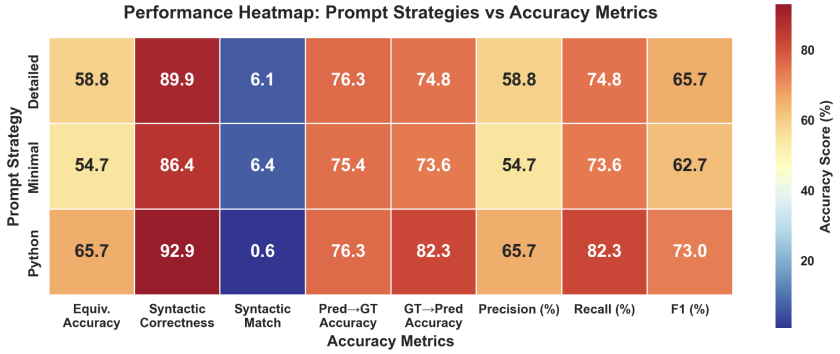

Fig. 4: Comparative Heatmap of LLM-generated Past LTL ($\mathbb{E}_{\texttt{nl2pltl}}$) across the prompting strategies

79.23%, syntactic correctness of 96.84%, and equivalence accuracy of 72.96%, indicating a strong alignment between generated and ground-truth formulas both syntactically and semantically. Interestingly, `GPT-3.5-Turbo` showed significant degradation with refinement, especially under the MINIMAL and DETAILED conditions, suggesting limitations in its ability to self-correct LTL translations. In contrast, models like `Claude-3.5−Sonnet` and `Gemini-1.5−Flash` benefitted from `Zero-Shot Self-Refine`. (See Appendix B) for detailed results.

Our evaluation of errors encountered in MINIMAL revealed several recurring challenges. The most error-prone sentence across all models was "*It previously held that if x1 was true then x2 was true before that.*" This structure, involving nested past implications, led to confusion in operator precedence and correct scoping of the P and S operators. Models frequently mishandle implications between past modalities. We noticed, `Claude-3.5−Sonnet` often confused Y (Yesterday) and H (Historically), contributing to wrong semantics despite well-formed syntax. Another interesting observation, specifically predominant in `GPT-3.5-Turbo`, was the use of "=" to assign one proposition to another, which demanded a bi-implication "↔" symbol. For example, given a natural language requiring the translation, the correct LTL representation is $\mathcal{H}(a \leftrightarrow b) \rightarrow \mathcal{H}(c \leftrightarrow d)$. However, the generated result was $\mathcal{H}(a = b) \rightarrow \mathcal{H}(c = d)$.

$\mathbb{RQ}_1^{\text{sem}}$: *Can LLMs generate LTL formulas that are semantically equivalent to the ground truth?*

Across all prompting strategies, PYTHON prompting led to the best performance, particularly for `Claude-3.5−Sonnet` under `Zero-Shot` with the highest F1 of 77% and semantic equivalence of 72.13%. Closely following suit is `Gemini-2.5−Flash` with an F1 score of 73.44% and semantic equivalence rate of 66.67%. Under DETAILED prompts, `Claude-3.5−Sonnet` under `Zero-Shot` achieved an F1 of 72.35% and equivalence accuracy of 64.75%. Under MINIMAL prompts, F1 dropped to 51.23%, and equivalence accuracy to 42.7% (See Figure 5).

Analysis of errors revealed several recurring patterns. Models often struggled with nested or repeated temporal expressions. For example, translating

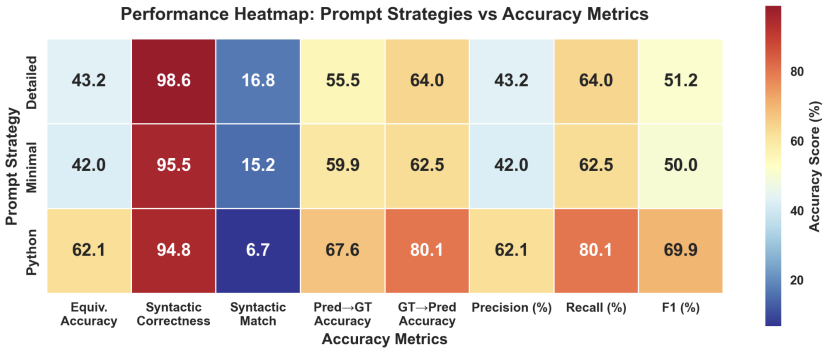

Fig. 5: Comparative Heatmap of Future LTL ($\mathbb{E}_{\texttt{nl2ltl}}$) for the different prompting strategies using $\mathbb{DS}^{\texttt{book}}_{\mathbf{6}}$

"*infinitely often, send implies infinitely often receive*" ideally yields $G(F(x_1)) \rightarrow G(F(x_2))$, but models frequently produced $G(F(x_1) \rightarrow F(x_2))$, failing to capture the global implication of eventualities. Common issues encountered include but not limited to, operator precedence, the confusion of G operator with F where models occasionally replace G with F or vice versa, leading to altered semantics and spacing or paranthesis where minor syntax deviations are typically caught during parsing. Across all models, the formula *"The machine provides beer infinitely often after initially providing soda three times in a row"* was the most error-prone, suggesting that reasoning about sequences of past events and conditional past temporal relationships is particularly challenging for these models.

Table 4 summarizes our experimental results based on trends and peaks identified. Figure 6 succinctly summarizes and shows the robustness of Python's transferable code-reasoning capabilities. Our preliminary Dashboard playground can be found here [3].

6 Related Work

Large Language Models (LLMs). LLMs like GPT-4o [77], LLaMA [91], and T5 [74] demonstrate impressive generalization across diverse tasks [46, 16]. Larger model size [53], in-context learning [80], and chain-of-thought prompting [95] contribute to their improved performance. However, LLMs often struggle with temporal semantics, scoping, and logical soundness [71, 96]. To mitigate these limitations partially, techniques like instruction tuning, RLHF [12], and self-refinement [63] have been adapted. Synthetic datasets [51] commonly improve task-specific performance.

Prompting. Prompting plays a crucial role in helping align LLMs with structured tasks. Few-shot cases and chain-of-thought prompting have proven effective to enhance logical and sequential reasoning [95] as seen in our study. Likewise, Human-in-the-loop corrections [25] offer an iterative refinement process, supporting specification synthesis without full model retraining.

Table 4: **Summary of Key Findings**

RQ	Task	$\mathcal{M}$ (Minimal Prompt)	$\mathcal{D}$ (Detailed Prompt)	$\mathcal{P}$ (Python Prompt)
RQ^1_{syn}	Atomic Proposition Extraction	J = **0.83** (Claude-3.5−Sonnet) L_d = **3.66** (Claude-3.5−Sonnet)	J = **0.86** (+3.61%) (Gemini-1.5−Flash) L_d = **3.05** (-16.67%) (Gemini-1.5−Flash)	N/A
RQ^2_{syn}	LTL Well-Formedness Validation	Acc=**52.19%** (Avg.) F_1 = **57.96%** (Avg.)	Acc=**84.33%** (Δ_{acc} = +10.86%) (Gemini-1.5−Flash) F_1 = **85.08%** (Δ_{F_1} = +17.87%) (Gemini-1.5−Flash) ϵ_r = **20.4%** (Total Reduction)	N/A
RQ^1_{sem}	NL→Future LTL Generation	Equiv = **65.56%** (Gemini-1.5−Flash) F_1 = **72.27%** (Gemini-1.5−Flash)	Equiv = **64.75%** (Claude-3.5−Sonnet) F_1 = **72.35%** (Claude-3.5−Sonnet)	Equiv = **72.13%** (Claude-3.5−Sonnet) F_1 = **77.80%** (Claude-3.5−Sonnet) $\phi_{GT} \models \phi_{pred}$: +**23.8%** (vs $\mathcal{M}$, Stat. Sig.)
RQ^2_{sem}	Trace Satisfiability Classification	Acc=**76.00%** (Gemini-1.5−Flash)	Acc=**80.72%** (Claude-3.5−Sonnet) Δ_{acc} = +7.7% (vs $\mathcal{M}$)	Acc=**100.00%** (Gemini-2.5−Flash) Δ_{acc} = +**22.4%** (vs $\mathcal{M}$, Stat. Sig.)
RQ^3_{sem}	Trace Generation	Acc=**68.14%** (Gemini-1.5−Flash)	Acc=**74.51%** (Claude-3.5−Sonnet) (Claude Few-Shot)	Acc=**74.67%** (Gemini-2.5−Flash) (< 20% for Few-Shot)
RQ^4_{sem}	NL→Past LTL Generation	Equiv = **65.56%** (Gemini-1.5−Flash) F_1 = **72.27%** (Gemini-1.5−Flash)	Equiv = **69.96%** (Gemini-1.5−Flash) F_1 = **76.04%** (Gemini-1.5−Flash) Syn_{corr} = **99.65%** (Claude-3.5−Sonnet)	Equiv = **72.96%** (Gemini-2.5−Flash) F_1 = **79.23%** (Gemini-2.5−Flash) Syn_{corr} = **96.84%** (Gemini-2.5−Flash)

$\mathcal{M}$ = MINIMAL; $\mathcal{D}$ = DETAILED; $\mathcal{P}$ = PYTHON; J = Jaccard Similarity; L_d = Levenshtein Distance; Δ = Absolute Change; ϵ_r = Error Reduction; Acc = Accuracy; Equiv = Equivalence Accuracy; F_1 = F1-Score; Syn_{corr} = Syntactic Correctness Rate; $\phi_{\text{GT}} \models \phi_{\text{pred}}$ = Ground Truth entails Prediction (Semantic Completeness); N/A = Not Applicable; Stat. Sig. = Statistically Significant. Performance numbers represent the **best observed result** for a model within that specific strategy and task, unless explicitly stated as an average or a range.

Traditional NLP for NL-to-TL. Earlier approaches employed in natural language to temporal logic relied on handcrafted grammars and pattern matching [26, 56]. While these systems could be effective in constrained domains like robotics [57, 27], they typically required strict templates [73] and struggled to generalize to more open or ambiguous natural language.

LLMs for Code Generation and Completion. Models like Codex [17], CodeT5 [93], and StarCoder [59] demonstrate strong capabilities in producing syntactically correct and semantically meaningful code. These models excel in tasks that require maintaining program structure, variable consistency, and logic flow, skills closely related to formal specification generation. Techniques such as syntax-constrained decoding [99], AST-based modeling [30], and repair-oriented prompting [33] have been proposed to improve correctness and interpretability.

7 Discussion and Recommendations

Data contamination. One main challenge of any effort of benchmarking LLMs is that the testing data may end up spilling into the LLMs' training data, especially because these LLMs often train over all data in the Internet. Unfortunately, we cannot guarantee that our testing data did not spill out to these LLMs' training data, especially because we stored and shared the data in Github for reproducibility. Even if the testing data were contaminated, our results are still valuable and can be viewed as upper-bound on the LLM's efficacy.

Requirements to logic. We consider small fragments of `NL` text capturing requirements as inputs to LLMs. Software requirements in practice, however, can be very large. In that sense, our evaluation can be seen as a lower-bound on the overall `NL` requirement to formal specification capabilities of LLMs. Supporting this would require addressing the usual challenges, including token limits.

Impact of Data Regularity. A vital observation from our study is that the $\mathbb{DS}_{\mathbf{6}}^{\mathsf{book}}$ dataset collated from older, structured textbooks and learning materials led to a significantly higher semantic equivalence than the $\mathbb{DS}_{\mathbf{1}}^{\mathsf{tricky}}$ dataset, which was designed specifically to test human understanding of `LTL`. With the latter formed to expose edge and overlooked cases in `NL`→`LTL`, models struggled to produce semantically correct formulas. These results suggest that the confusions experienced by humans were equally experienced by these LLMs.
Prompting impacts. Our results show that prompting style plays an integral role in improving LLM performance. We consistently observed that `Few-Shot` dominated both `Zero-Shot` and `Zero-Shot Self-Refine` where `Zero-Shot Self-Refine` performing better than `Zero-Shot`. Providing more detailed prompts improved models' performance. Reformulating the tasks to Python code completion or comprehension substantially improves performance over `NL` prompts.
Threat to Validity. The reported results are based only on the versions of the LLMs and our dataset. As $\texttt{LLM}_{\texttt{gp}}$s undergo constant improvement, the results may not be reproducible for their future versions. Finally, although we try to collect and construct fair datasets, they may not necessarily reflect how normal users would use LLMs for the `NL`→`LTL` task.
Recommendations. - As the $\texttt{LLM}_{\texttt{gp}}$s are trained on data available on the Internet, to maintain a fair evaluation, we should consider using private ground-truth testing data that are not made available to the LLMs for training purposes.
- For fair comparisons, future fine-tuned models should take `NL` fragment to atomic proposition mapping as optional input.
- Reformulating any task at hand to be performed by an LLM to a code completion/comprehension task can substantially improve the overall task performance.

8 Conclusion

This paper evaluates existing LLMs' efficacy for the `NL`→`LTL` task from several angles. Our evaluation reveals that recent efforts on using LLMs for different temporal semantic reasoning tasks are not well-motivated, at least from our experiments, and such semantic reasoning tasks should be delegated to the appropriate dedicated symbolic reasoners. One surprising finding of this work is that by posing the `NL`→`LTL` task as a Python code completion or comprehension task substantially improves LLMs' capabilities. To allow a fair and large-scale evaluation and comparison with prior efforts, future efforts on fine-tuning LLMs for `NL`→`LTL` should accept `NL` to proposition mapping as input, side-stepping the ontological problem. In addition, they should keep the testing data private, and make it available only upon request to avoid data contamination. Finally, future work should also investigate whether real software requirements can be formalized into a set of `LTL` formulas instead of a single formula.

Bibliography

[1] Achhab, M.A., Hammad, A., Mountassir, H.: Verifying ltl properties on hierarchical systems: Application to aircraft autopilot. In: Proceedings of the Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. p. 28–35. ISOLA '06, IEEE Computer Society, USA (2006). https://doi.org/10.1109/ISoLA.2006.10, https://doi.org/10.1109/ISoLA.2006.10

[2] Anonymous: Anonymous artifact. https://github.com/icsekdk/promptEVAL (2025)

[3] Anonymous: Nl to ltl dashboard playground. https://appexperimentdashboardpy-2qq8aouv7wkun3k9yzctqw.streamlit.app/ (2025)

[4] Anthropic: Introducing Claude 3.5 Sonnet - frontier intelligence at 2x the speed. https://www.anthropic.com/news/claude-3-5-sonnet (2024), [Accessed 27-01-2025]

[5] Backasch, R., Hochberger, C., Weiss, A., Leucker, M., Lasslop, R.: Runtime verification for multicore soc with high-quality trace data **18**(2) (Apr 2013). https://doi.org/10.1145/2442087.2442089, https://doi.org/10.1145/2442087.2442089

[6] Baier, C., Katoen, J., Larsen, K.: Principles of Model Checking. The MIT Press, MIT Press (2008), https://books.google.com/books?id=5dvxCwAAQBAJ

[7] Batsakis, S., Tachmazidis, I., Mantle, M., Papadakis, N., Antoniou, G.: Model checking using large language models—evaluation and future directions. Electronics **14**(2) (2025). https://doi.org/10.3390/electronics14020401, https://www.mdpi.com/2079-9292/14/2/401

[8] Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for ltl and tltl. ACM Trans. Softw. Eng. Methodol. **20**(4) (Sep 2011). https://doi.org/10.1145/2000799.2000800, https://doi.org/10.1145/2000799.2000800

[9] Bauer, A., Leucker, M., Streit, J.: Salt—structured assertion language for temporal logic. In: Liu, Z., He, J. (eds.) Formal Methods and Software Engineering. pp. 757–775. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

[10] Benedetti, M., Cimatti, A.: Bounded model checking for past ltl. In: Tools and Algorithms for the Construction and Analysis of Systems: 9th International Conference, TACAS 2003 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003 Warsaw, Poland, April 7–11, 2003 Proceedings 9. pp. 18–33. Springer (2003)

[11] Besold, T.R., d'Avila Garcez, A., Bader, S., Bowman, H., Domingos, P., Hitzler, P., Kuehnberger, K.U., Lamb, L.C., Lowd, D., Lima, P.M.V., de Penning, L., Pinkas, G., Poon, H., Zaverucha, G.: Neural-symbolic

learning and reasoning: A survey and interpretation (2017), https://arxiv.org/abs/1711.03902

[12] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners (2020), https://arxiv.org/abs/2005.14165

[13] Brunello, A., Montanari, A., Reynolds, M.: Synthesis of ltl formulas from natural language texts: State of the art and research directions. In: Time (2019), https://api.semanticscholar.org/CorpusID:203912123

[14] Buzhinsky, I.: Formalization of natural language requirements into temporal logics: a survey. In: 2019 IEEE 17th International Conference on Industrial Informatics (INDIN). vol. 1, pp. 400–406 (2019). https://doi.org/10.1109/INDIN41052.2019.8972130

[15] Cai, M., Vasile, C.I.: Safety-critical learning of robot control with temporal logic specifications (2022), https://arxiv.org/abs/2109.02791

[16] Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P.S., Yang, Q., Xie, X.: A survey on evaluation of large language models **15**(3) (Mar 2024). https://doi.org/10.1145/3641289, https://doi.org/10.1145/3641289

[17] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)

[18] Chen, Y., Gandhi, R., Zhang, Y., Fan, C.: Nl2tl: Transforming natural languages to temporal logics using large language models (2024), https://arxiv.org/abs/2305.07766

[19] Christiano, P.F., Leike, J., Brown, T., Martic, M., Legg, S., Amodei, D.: Deep reinforcement learning from human preferences. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf

[20] Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: A new symbolic model verifier. In: Computer Aided Verification: 11th International Conference, CAV'99 Trento, Italy, July 6–10, 1999 Proceedings 11. pp. 495–499. Springer (1999)

[21] Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: a new symbolic model checker. International Journal on Software Tools for Technology Transfer **2**(4), 410–425 (Mar 2000). https://doi.org/10.1007/s100090050046, https://doi.org/10.1007/s100090050046

[22] Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge, MA, USA (2000)

[23] Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string metrics for matching names and records. In: KDD Workshop on Data Cleaning and Object Consolidation (2003), https://www.cs.cmu.edu/afs/cs/Web/People/wcohen/postscript/kdd-2003-match-ws.pdf

[24] Coogan, S., Arcak, M.: Freeway traffic control from linear temporal logic specifications. In: 2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS). pp. 36–47 (2014). https://doi.org/10.1109/ICCPS.2014.6843709

[25] Cosler, M., Hahn, C., Mendoza, D., Schmitt, F., Trippel, C.: nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In: Computer Aided Verification: 35th International Conference, CAV 2023, Paris, France, July 17–22, 2023, Proceedings, Part II. p. 383–396. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-37703-7_18, https://doi.org/10.1007/978-3-031-37703-7_18

[26] Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering. p. 411–420. ICSE '99, Association for Computing Machinery, New York, NY, USA (1999). https://doi.org/10.1145/302405.302672, https://doi.org/10.1145/302405.302672

[27] Dzifcak, J., Scheutz, M., Baral, C., Schermerhorn, P.: What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In: 2009 IEEE International Conference on Robotics and Automation. pp. 4163–4168 (2009). https://doi.org/10.1109/ROBOT.2009.5152776

[28] Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., Moreschini, P.: Assisting requirement formalization by means of natural language translation. Formal Methods in System Design **4**(3), 243–263 (1994). https://doi.org/10.1007/BF01384048, https://doi.org/10.1007/BF01384048

[29] Farrell, M., Luckcuck, M., Monahan, R., Reynolds, C., Sheridan, O.: Adventures in FRET and Specification, p. 106–123. Springer Nature Switzerland (Oct 2024). https://doi.org/10.1007/978-3-031-75380-0_7, http://dx.doi.org/10.1007/978-3-031-75380-0_7

[30] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, M.G., Qin, B., Liu, T., Lu, S., Zhou, M.: Codebert: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020. pp. 1536–1547 (2020)

[31] Finkbeiner, B.: Synthesis of reactive systems. In: Esparza, J., Grumberg, O., Sickert, S. (eds.) Dependable Software Systems Engineering. NATO Science for Peace and Security Series, D: Information and Communication Security, vol. 45, pp. 72–98. IOS Press (2016)

[32] Fisher, M.: An Introduction to Practical Formal Methods Using Temporal Logic. Wiley (2011), https://books.google.com/books?id=zl6OLZv7d1kC

[33] Franceschi, A., Corradini, F., Polini, A., Re, B.: Repairing code with large language models: a comparative study on the effectiveness of prompt engineering. Empirical Software Engineering **28**(6), 1–28 (2023)

[34] Fraser, G., Wotawa, F.: Using ltl rewriting to improve the performance of model-checker based test-case generation. In: Proceedings of the 3rd International Workshop on Advances in Model-Based Testing. p. 64–74. A-MOST '07, Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1291535.1291542, https://doi.org/10.1145/1291535.1291542

[35] Fuggitti, F., Chakraborti, T.: Nl2ltl – a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas. Proceedings of the AAAI Conference on Artificial Intelligence **37**(13), 16428–16430 (Jul 2024). https://doi.org/10.1609/aaai.v37i13.27068, https://ojs.aaai.org/index.php/AAAI/article/view/27068

[36] Garreta, R., Moncecchi, G., Hauck, T., Hackeling, G.: scikit-learn : Machine Learning Simplified: Implement scikit-learn into every step of the data science pipeline. Packt Publishing (2017), https://books.google.com/books?id=sEFPDwAAQBAJ

[37] Geatti, L., Gigante, N., Montanari, A., Venturato, G.: Past Matters: Supporting LTL+Past in the BLACK Satisfiability Checker. In: Combi, C., Eder, J., Reynolds, M. (eds.) 28th International Symposium on Temporal Representation and Reasoning (TIME 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 206, pp. 8:1–8:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.TIME.2021.8, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TIME.2021.8

[38] Google: Gemini 1.5 Pro - our best model for reasoning across large amounts of information. https://deepmind.google/technologies/gemini/pro/ (2025), [Accessed 27-01-2025]

[39] Goranko, V., Rumberg, A.: Temporal Logic. In: Zalta, E.N., Nodelman, U. (eds.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, Summer 2024 edn. (2024)

[40] Greenman, B., Saarinen, S., Nelson, T., Krishnamurthi, S.: Little tricky logic: Misconceptions in the understanding of ltl. The Art, Science, and Engineering of Programming **7**(2) (Oct 2022). https://doi.org/10.22152/programming-journal.org/2023/7/7, http://dx.doi.org/10.22152/programming-journal.org/2023/7/7

[41] Gunter, E.L.: From natural language to linear temporal logic: Difficulties of capturing natural language specifications in formal languages for automatic analysis. In: Monterey Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (sep 2003)

[42] Hahn, C., Schmitt, F., Kreber, J.U., Rabe, M.N., Finkbeiner, B.: Teaching temporal logics to neural networks. arXiv preprint arXiv:2003.04218 (2020)

[43] Harris, C.B., Harris, I.G.: Generating formal hardware verification properties from natural language documentation. In: Proceedings of the 2015

IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015). pp. 49–56 (2015). https://doi.org/10.1109/ICOSC.2015.7050777

[44] He, J., Bartocci, E., Ničković, D., Isakovic, H., Grosu, R.: Deepstl: from english requirements to signal temporal logic. In: Proceedings of the 44th International Conference on Software Engineering. pp. 610–622 (2022)

[45] He, J., Bartocci, E., Ničković, D., Isakovic, H., Grosu, R.: Deepstl: from english requirements to signal temporal logic. In: Proceedings of the 44th International Conference on Software Engineering. p. 610–622. ICSE '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3510003.3510171, https://doi.org/10.1145/3510003.3510171

[46] Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., Wang, H.: Large language models for software engineering: A systematic literature review. ACM Trans. Softw. Eng. Methodol. **33**(8) (Dec 2024). https://doi.org/10.1145/3695988, https://doi.org/10.1145/3695988

[47] Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press (2004), https://books.google.com/books?id=eUggAwAAQBAJ

[48] Jackermeier, M., Abate, A.: Deepltl: Learning to efficiently satisfy complex ltl specifications (2024), https://arxiv.org/abs/2410.04631

[49] Jawahar, G., Sagot, B., Seddah, D.: What does BERT learn about the structure of language? In: ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics. Florence, Italy (Jul 2019), https://inria.hal.science/hal-02131630

[50] Jeffrey, R., Burgess, J.: Formal Logic: Its Scope and Limits. Hackett Publishing Company (2006), https://books.google.com/books?id=iqvsjhvZCgcC

[51] Jordon, J., Szpruch, L., Houssiau, F., Bottarelli, M., Cherubin, G., Maple, C., Cohen, S.N., Weller, A.: Synthetic data – what, why and how? (2022), https://arxiv.org/abs/2205.03257

[52] Kambhampati, S.: Can large language models reason and plan? Annals of the New York Academy of Sciences **1534**(1), 15–18 (Mar 2024). https://doi.org/10.1111/nyas.15125, http://dx.doi.org/10.1111/nyas.15125

[53] Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., Amodei, D.: Scaling laws for neural language models (2020), https://arxiv.org/abs/2001.08361

[54] Konnov, I.: Edmund m. clarke, thomas a. henzinger, helmut veith, and roderick bloem (eds): Handbook of model checking: Springer international publishing ag, cham, switzerland, 2018, xxiv+1210 pp, isbn 978-3-319-10574-1 (hardcover, 2.13 kg), isbn 978-3-319-10575-8 (ebook, pdf). https://doi.org/10.1007/978-3-319-10575-8. Form. Asp. Comput. **31**(4), 455–456 (Aug 2019). https://doi.org/10.1007/s00165-019-00486-z, https://doi.org/10.1007/s00165-019-00486-z

[55] Konrad, S., Cheng, B.: Real-time specification patterns. In: Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005. pp. 372–381 (2005). https://doi.org/10.1109/ICSE.2005.1553580

[56] Konrad, S., Cheng, B.H.C.: Automated analysis of natural language properties for uml models. In: Bruel, J.M. (ed.) Satellite Events at the MoDELS 2005 Conference. pp. 48–57. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

[57] Kress-Gazit, H., Fainekos, G.E., and, G.J.P.: Translating structured english to robot controllers. Advanced Robotics **22**(12), 1343–1359 (2008). https://doi.org/10.1163/156855308X344864, https://doi.org/10.1163/156855308X344864

[58] Latvala, T., Biere, A., Heljanko, K., Junttila, T.: Simple is better: Efficient bounded model checking for past ltl. In: Verification, Model Checking, and Abstract Interpretation: 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005. Proceedings 6. pp. 380–395. Springer (2005)

[59] Li, R., Allal, L.B., Benhalloum, A., Allamanis, M., Bibi, A., Cassano, F., Chausson, C., Dey, D., Drori, I., et al.: Starcoder: may the source be with you! Transactions on Machine Learning Research (2023), https://openreview.net/forum?id=GZwT6KAF0Y, openReview preprint

[60] Liu, J.X., Yang, Z., Idrees, I., Liang, S., Schornstein, B., Tellex, S., Shah, A.: Grounding complex natural language commands for temporal tasks in unseen environments (2023), https://arxiv.org/abs/2302.11649

[61] Liu, J.X., Yang, Z., Schornstein, B., Liang, S., Idrees, I., Tellex, S., Shah, A.: Lang2LTL: Translating natural language commands to temporal specification with large language models. In: Workshop on Language and Robotics at CoRL 2022 (2022), https://openreview.net/forum?id=VxfjGZzrdn

[62] Luo, W., Wan, H., Zhang, D., Du, J., Su, H.: Checking ltl satisfiability via end-to-end learning. ASE '22, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3551349.3561163, https://doi.org/10.1145/3551349.3561163

[63] Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B.P., Hermann, K., Welleck, S., Yazdanbakhsh, A., Clark, P.: Self-refine: Iterative refinement with self-feedback. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems. vol. 36, pp. 46534–46594. Curran Associates, Inc. (2023), https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf

[64] Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B.P., Hermann, K., Welleck, S., Yazdanbakhsh, A., Clark, P.: Self-refine: iterative refinement with self-feedback. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. NIPS '23, Curran Associates Inc., Red Hook, NY, USA (2023)

[65] Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. pp. 152–166. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[66] Manas, K., Zwicklbauer, S., Paschke, A.: Tr2mtl: Llm based framework for metric temporal logic formalization of traffic rules. In: 2024 IEEE Intelligent Vehicles Symposium (IV). p. 1206–1213. IEEE (Jun 2024). https://doi.org/10.1109/iv55156.2024.10588650, http://dx.doi.org/10.1109/IV55156.2024.10588650

[67] Mendoza, D., Hahn, C., Trippel, C.: Translating natural language to temporal logics with large language models and model checkers. In: Narodytska, N., Rümmer, P. (eds.) Proceedings of the 24th Conference on Formal Methods in Computer-Aided Design – FMCAD 2024. Lecture Notes in Computer Science, vol. 15423, pp. 119–129. TU Wien Academic Press (2024). https://doi.org/10.34727/2024/isbn.978-3-85448-065-5_17

[68] Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., Gao, J.: Large language models: A survey (2024), https://arxiv.org/abs/2402.06196

[69] Minsky, Y., Madhavapeddy, A., Hickey, J.: Real World OCaml: Functional Programming for the Masses. Real World OCaml, O'Reilly Media (2013), https://books.google.com/books?id=nabtAQAAQBAJ

[70] Mirzadeh, I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., Farajtabar, M.: Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models (2024), https://arxiv.org/abs/2410.05229

[71] Morishita, T., Morio, G., Yamaguchi, A., Sogawa, Y.: Enhancing reasoning capabilities of llms via principled synthetic logic corpus. In: Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., Zhang, C. (eds.) Advances in Neural Information Processing Systems. vol. 37, pp. 73572–73604. Curran Associates, Inc. (2024), https://proceedings.neurips.cc/paper_files/paper/2024/file/8678da90126aa58326b2fc0254b33a8c-Paper-Conference.pdf

[72] Murphy, W., Holzer, N., Koenig, N., Cui, L., Rothkopf, R., Qiao, F., Santolucito, M.: Guiding llm temporal logic generation with explicit separation of data and control (2024), https://arxiv.org/abs/2406.07400

[73] Nelken, R., Francez, N.: Automatic translation of natural language system specifications into temporal logic. In: Alur, R., Henzinger, T.A. (eds.) Computer Aided Verification. pp. 360–371. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)

[74] Ni, J., Ábrego, G.H., Constant, N., Ma, J., Hall, K.B., Cer, D., Yang, Y.: Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models (2021), https://arxiv.org/abs/2108.08877

[75] Niwattanakul, S., Singthongchai, J., Naenudorn, E., Wanapu, S.: Using of jaccard coefficient for keywords similarity. In: Using of Jaccard Coefficient for Keywords Similarity (2013), https://api.semanticscholar.org/CorpusID:14040055

[76] OpenAI: GPT-4o mini: advancing cost-efficient intelligence - a small model with superior textual intelligence and multimodal reasoning. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ (2024), [Accessed 27-01-2025]

[77] OpenAI: Hello GPT-4o - our new flagship model that can reason across audio, vision, and text in real time. https://openai.com/index/hello-gpt-4o/ (2024), [Accessed 27-01-2025]

[78] Pan, J., Chou, G., Berenson, D.: Data-efficient learning of natural language to linear temporal logic translators for robot task specification (2023), https://arxiv.org/abs/2303.08006

[79] Patil, R., Gudivada, V.: A review of current trends, techniques, and challenges in large language models (llms). Applied Sciences **14**(5) (2024). https://doi.org/10.3390/app14052074, https://www.mdpi.com/2076-3417/14/5/2074

[80] Perez, E., Kiela, D., Cho, K.: True few-shot learning with language models. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems. vol. 34, pp. 11054–11070. Curran Associates, Inc. (2021), https://proceedings.neurips.cc/paper_files/paper/2021/file/5c04925674920eb58467fb52ce4ef728-Paper.pdf

[81] Pirozelli, P., José, M.M., de Tarso P. Filho, P., Brandão, A.A.F., Cozman, F.G.: Assessing logical reasoning capabilities of encoder-only transformer models (2024), https://arxiv.org/abs/2312.11720

[82] Pnueli, A.: The temporal logic of programs. Foundations of Computer Science p. 46–57 (Sep 1977). https://doi.org/10.1109/SFCS.1977.32, https://ieeexplore.ieee.org/document/4567924/

[83] Rahman, A., Mahir, S.H., Tashrif, M.T.A., Aishi, A.A., Karim, M.A., Kundu, D., Debnath, T., Moududi, M.A.A., Eidmum, M.Z.A.: Comparative analysis based on deepseek, chatgpt, and google gemini: Features, techniques, performance, future prospects (2025), https://arxiv.org/abs/2503.04783

[84] Ray, P.P.: Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. Internet of Things and Cyber-Physical Systems **3**, 121–154 (2023). https://doi.org/https://doi.org/10.1016/j.iotcps.2023.04.003, https://www.sciencedirect.com/science/article/pii/S266734522300024X

[85] Romera-Paredes, B., Torr, P.: An embarrassingly simple approach to zero-shot learning. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 2152–2161. PMLR, Lille, France (07–09 Jul 2015), https://proceedings.mlr.press/v37/romera-paredes15.html

[86] Rosen, K.: Discrete Mathematics and Its Applications. McGraw-Hill Education (2011), https://books.google.com/books?id=QlWQzgEACAAJ

[87] Rozier, K.Y.: Linear temporal logic symbolic model checking. Computer Science Review **5**(2), 163–203 (2011).

https://doi.org/https://doi.org/10.1016/j.cosrev.2010.06.002, https://www.sciencedirect.com/science/article/pii/S1574013710000407

[88] Rozier, K.Y., Vardi, M.Y.: Symbolic ltl compilation for model checking. In: Grace Hopper Celebration of Women in Computing 2007 (2007)

[89] Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P.H., Hospedales, T.M.: Learning to compare: Relation network for few-shot learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)

[90] Tang, W., Belle, V.: Ltlbench: Towards benchmarks for evaluating temporal logic reasoning in large language models (2024), https://arxiv.org/abs/2407.05434

[91] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., Lample, G.: Llama: Open and efficient foundation language models (2023), https://arxiv.org/abs/2302.13971

[92] Wang, C., Ross, C., Kuo, Y.L., Katz, B., Barbu, A.: Learning a natural-language to ltl executable semantic parser for grounded robotics (2021), https://arxiv.org/abs/2008.03277

[93] Wang, Y., Wang, W., Joty, S., Lin, S.C., Ng, H.T.: Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 8693–8708 (2021)

[94] Wasilewska, A.: Logics for Computer Science: Classical and Non-Classical. Springer International Publishing (2018), https://books.google.com/books?id=o_9ctgEACAAJ

[95] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. In: Proceedings of the 36th International Conference on Neural Information Processing Systems. NIPS '22, Curran Associates Inc., Red Hook, NY, USA (2022)

[96] Xiong, S., Payani, A., Kompella, R., Fekri, F.: Large language models can learn temporal reasoning (2024), https://arxiv.org/abs/2401.06853

[97] Xu, Y., Feng, J., Miao, W.: Learning from failures: Translation of natural language requirements into linear temporal logic with large language models. In: 2024 IEEE 24th International Conference on Software Quality, Reliability and Security (QRS). pp. 204–215 (2024). https://doi.org/10.1109/QRS62785.2024.00029

[98] Yao, Y., Li, Z., Zhao, H.: Beyond chain-of-thought, effective graph-of-thought reasoning in language models (2024), https://arxiv.org/abs/2305.16582

[99] Yin, P., Neubig, G.: A syntactic neural model for general-purpose code generation. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 440–450 (2018)

[100] Yujian, L., Bo, L.: A normalized levenshtein distance metric. IEEE Transactions on Pattern Analysis and Machine Intelligence **29**(6), 1091–1095 (2007). https://doi.org/10.1109/TPAMI.2007.1078

[101] Zhezherau, A., Yanockin, A.: Hybrid training approaches for llms: Leveraging real and synthetic data to enhance model performance in domain-specific applications (2024), https://arxiv.org/abs/2410.09168
[102] Zhong, R., Lee, K., Zhang, Z., Klein, D.: Adapting language models for zero-shot learning by meta-tuning on dataset and prompt collections (2021), https://arxiv.org/abs/2104.04670
[103] Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q., Chi, E.: Least-to-most prompting enables complex reasoning in large language models (2023), https://arxiv.org/abs/2205.10625

Appendix

A Prompting Templates

This appendix provides full prompt templates used for the MINIMAL, DETAILED, and PYTHON prompting strategies. We will focus on the prompt employed in the task of translating the natural language to future LTL.

Base Prompt for NL-to-Future LTL under MINIMAL strategy

```
def generate_base_prompt(natural_language, atomic_propositions):
    """Generate the base prompt for any approach."""
    return (
        "You are a Linear Temporal Logic (LTL) Parser. Your task is to convert a given "
        "Natural Language statement to an LTL formula, using the provided mapping of "
        "natural language phrases to atomic propositions.\n\n"
        "LTL Symbols:\n"
        "- AND: &\n"
        "- OR: |\n"
        "- NOT: !\n"
        "- IMPLIES: ->\n"
        "- BIIMPLICATION: <->\n"
        "- NEXT: X\n"
        "- EVENTUALLY: F\n"
        "- ALWAYS: G\n"
        "- UNTIL: U\n\n"
        f"Natural Language statement: {natural_language}\n"
        f"Atomic Propositions mapping: {atomic_propositions}\n\n"
    )
```

Base Prompt for NL-to-Future LTL under DETAILED strategy

```
def generate_base_prompt(natural_language, atomic_propositions):
    """Generate the base system prompt for LTL conversion task"""
    return f"""You are a teacher who is proficient in propositional linear temporal logic (LTL).

In propositional linear temporal logic, you have three elements:
- propositional variables;
- logical connectives/operators;
- linear temporal connectives/operators.

Logical connectives/operators in propositional linear temporal logic include:
logical AND or conjunction (represented as &),
logical OR or disjunction (represented as |),
logical Not or negation (represented as !),
logical implication or entailment (represented as ->),
```

```
logical equivalence or bi-implication (represented as <->).

Linear temporal logic connectives/operators in propositional linear temporal logic include:
Next or tomorrow (represented as X),
Eventually or future (represented as F),
Globally or henceforth (represented as G),
Until (represented as U)
Yesterday or last (represented as Y),
Once (represented as O),
Historically (represented as H),
Since (represented as S)


Here is a BNF grammar for the syntax of propositional linear temporal logic formulas.

<formula> ::= <ap> | <TRUE> | <FALSE> | <formula> "&" <formula> | <formula> "|" <formula> |
"!" <formula> | "(" <formula> ")" | <formula> "->" <formula> | <formula> "<->" <formula>
| <formula> "U" <formula> | "F" <formula> | "G" <formula> | "X" <formula> |
<formula> "S" <formula> | "O" <formula> | "H" <formula> | "Y" <formula>

<TRUE> ::= "true" | "True" | "TRUE"
<FALSE> ::= "false" | "False" | "FALSE"

<ap> ::= It is the set of propositional logic variables which should start with any letters (small or capital) followed by an alphanumeric string.
Simply put, <ap> ::= [a-zA-Z][a-zA-Z0-9]* in regular expression.

The semantics of a propositional linear temporal logic formula are defined with respect to a linear trace \sigma and a position i in the trace. The position i is a non-negative number (i.e., 0 or any positive whole number).

Each element of a trace is a substitution (a mapping from propositional variables to either true or false).

In short, a trace \sigma is a sequence of states, where each state assigns truth values to propositional variables.

Given a trace \sigma, a position i in \sigma where the temporal logic formula is being evaluated, we can have the following semantics:

(1) \sigma and i always satisfy true
(2) \sigma and i falsify false
```

```
(3) In \sigma and i, a proposition a is true if and only if the current substitution \sigma[i] satisfies a
(4) In \sigma and i, p & q is true if and only if both p and q are true in \sigma and i
(5) In \sigma and i, p | q is true if and only if one of p or q are true in \sigma and i
(6) In \sigma and i, !p is true if and only if p is false in \sigma and i
(7) In \sigma and i, p -> q is true if and only if when p is true then q is true in \sigma and i
(8) In \sigma and i, p <-> q is true if and only if both p and q are both true or p and q are both false in \sigma and i
(9) In \sigma and i, X p is true if and only if p is true in \sigma at position i + 1
(10) In \sigma and i, Y p is true if and only if i > 0 and p is true in \sigma at position i - 1
(11) In \sigma and i, O p is true if and only if p is true in \sigma at position i or in \sigma at any position lower than i
(12) In \sigma and i, F p is true if and only if p is true in \sigma at position i or in \sigma at any position greater than i
(13) In \sigma and i, H p is true if and only if p is true in \sigma at position i and in \sigma at all positions lower than i
(14) In \sigma and i, G p is true if and only if p is true in \sigma at position i and in \sigma at all positions greater than i
(15) In \sigma and i, p S q is true if and only if q is true in \sigma at position i and p is true in \sigma at all positions from some position j <= i to position i

We say a trace \sigma satisfies a propositional LTL formula f if and only if \sigma satisfies f in the 0th position of \sigma.

You are now trying to convert a natural language English text into a propositional linear temporal logic formula.

The input for this task will have two parts: natural language English sentences followed by natural language to propositional variable mapping.

The input mapping from propositional variables to the natural language fragment having the form (variable_name -> "English sentence fragment") dictates the meaning of the propositional variable.

For this task, you can only use the propositional variable given to you as part of the task input. Do not introduce any new propositional variables other than what is given to you.

When generating the answer for the given natural language text to convert to LTL, stop providing additional explanations. Your output should only contain the formula in a single line.

```

```
Now convert the following:

Natural Language: {natural_language}

Proposition Mapping: {atomic_propositions}

When converting the natural language sentences in English to LTL, you cannot use any of the past temporal operators. The LTL formula you should output can only contain the following operators/connectives: &, |, !, ->, <->, X, F, U, G, Y, O, H, S

Very Important Syntax Rules

You must use only the following symbols in your formula:
- & for logical AND
- | for logical OR
- ! for logical NOT
- -> for implication
- <-> for bi-implication
- F, G, X, U for temporal operators

Your output must be a **single-line formula** using only the allowed syntax above, and must strictly follow the BNF grammar provided.
Convert the natural language to LTL formula:

"""
```

Base Prompt for NL-to-Future LTL under PYTHON strategy

```python
def generate_base_prompt(natural_language, atomic_propositions):
    return f"""
You are a teacher who is proficient in propositional linear temporal logic (LTL) and Python.

You are given the following Python class structure that defines how LTL formulas should be represented:

```python
from dataclasses import dataclass
from typing import *

class Formula:
    pass

@dataclass
class AtomicProposition(Formula):
    name : str

@dataclass
```
```

```
class Literal(Formula):
    name : str

@dataclass
class LNot(Formula):
    Formula: Formula

@dataclass
class LAnd(Formula):
    left: Formula
    right: Formula

@dataclass
class LOr(Formula):
    left: Formula
    right: Formula

@dataclass
class LImplies(Formula):
    left: Formula
    right: Formula

@dataclass
class LEquiv(Formula):
    left: Formula
    right: Formula

@dataclass
class Since(Formula):
    a : Formula
    b : Formula

@dataclass
class Until(Formula):
    a : Formula
    b : Formula

@dataclass
class Next(Formula):
    Formula: Formula

@dataclass
class Always(Formula):
    Formula: Formula

@dataclass
class Eventually(Formula):
    Formula: Formula

@dataclass
```

```
class Once(Formula):
    Formula: Formula

@dataclass
class Historically(Formula):
    Formula: Formula

@dataclass
class Yesterday(Formula):
    Formula: Formula


FormulaType = Union[AtomicProposition, Literal, LNot, LAnd, LOr,
↪  LImplies, LEquiv, Since, Until, Next, Always, Eventually, Once,
↪  Historically, Yesterday]

type varToValMapping = tuple[str, bool]
type state = list[varToValMapping]
type trace = list[state]

class OptionType:
    pass

@dataclass
class ReallyNone(OptionType):
    pass

@dataclass
class Some(OptionType):
    value: bool

myOptionType = Union[ReallyNone, Some]

def isPropositionTrueInTracePosition(p : AtomicProposition, t: trace,
↪  pos: int) -> myOptionType:
    if pos < 0 or pos >= len(t):
        return ReallyNone()
    state_at_pos = t[pos]
    for var, val in state_at_pos:
        if var == p.name:
            return Some(val)
    return ReallyNone()

def evalFormula(f : Formula, t: trace, pos: int) -> myOptionType:
    match f:
        case AtomicProposition(name):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            return isPropositionTrueInTracePosition(f, t, pos)
        case Literal(name):
```

```
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            if name == "True":
                return Some(True)
            elif name == "False":
                return Some(False)
            else:
                return ReallyNone()
        case LNot(inner):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            inner_eval = evalFormula(inner, t, pos)
            match inner_eval:
                case Some(val):
                    return Some(not val)
                case ReallyNone():
                    return ReallyNone()
        case LAnd(left, right):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            left_eval = evalFormula(left, t, pos)
            right_eval = evalFormula(right, t, pos)
            match left_eval, right_eval:
                case (Some(lval), Some(rval)):
                    return Some(lval and rval)
                case (ReallyNone(), _):
                    return ReallyNone()
                case (_, ReallyNone()):
                    return ReallyNone()
        case LOr(left, right):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            left_eval = evalFormula(left, t, pos)
            right_eval = evalFormula(right, t, pos)
            match left_eval, right_eval:
                case (Some(lval), Some(rval)):
                    return Some(lval or rval)
                case (ReallyNone(), _):
                    return ReallyNone()
                case (_, ReallyNone()):
                    return ReallyNone()
        case LImplies(left, right):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            left_eval = evalFormula(left, t, pos)
            right_eval = evalFormula(right, t, pos)
            match left_eval, right_eval:
                case (Some(lval), Some(rval)):
                    return Some((not lval) or rval)
                case (ReallyNone(), _):
```

```
                return ReallyNone()
            case (_, ReallyNone()):
                return ReallyNone()
    case LEquiv(left, right):
        if pos < 0 or pos >= len(t):
            return ReallyNone()
        left_eval = evalFormula(left, t, pos)
        right_eval = evalFormula(right, t, pos)
        match left_eval, right_eval:
            case (Some(lval), Some(rval)):
                return Some(lval == rval)
            case (ReallyNone(), _):
                return ReallyNone()
            case (_, ReallyNone()):
                return ReallyNone()
    case Since(a, b):
        if pos < 0 or pos >= len(t):
            return ReallyNone()
        foundB = False
        i = pos
        while i >= 0 :
            eval_result = evalFormula(b, t, i)
            if isinstance(eval_result, ReallyNone):
                return ReallyNone()
            if isinstance(eval_result, Some) and eval_result.value:
                foundB = True
                break
            i -= 1
        if not foundB:
            return Some(False)
        j = i + 1
        while j <= pos:
            eval_result = evalFormula(a, t, j)
            if isinstance(eval_result, ReallyNone):
                return ReallyNone()
            if isinstance(eval_result, Some) and not
            ↪  eval_result.value:
                return Some(False)
            j += 1
        return Some(True)
    case Until(a, b):
        if pos < 0 or pos >= len(t):
            return ReallyNone()
        foundB = False
        i = pos
        while i < len(t) :
            eval_result = evalFormula(b, t, i)
            if isinstance(eval_result, ReallyNone):
                return ReallyNone()
            if isinstance(eval_result, Some) and eval_result.value:
```

```
                    foundB = True
                    break
                i += 1
            if not foundB:
                return Some(False)
            j = pos
            while j < i:
                eval_result = evalFormula(a, t, j)
                if isinstance(eval_result, ReallyNone):
                    return ReallyNone()
                if isinstance(eval_result, Some) and not
                ↪  eval_result.value:
                    return Some(False)
                j += 1
            return Some(True)

        case Next(inner):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            if pos + 1 < len(t):
                return evalFormula(inner, t, pos + 1)
            else:
                return ReallyNone()
        case Always(inner):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            for i in range(pos, len(t)):
                eval_result = evalFormula(inner, t, i)
                if isinstance(eval_result, ReallyNone):
                    return ReallyNone()
                if isinstance(eval_result, Some) and not
                ↪  eval_result.value:
                    return Some(False)
            return Some(True)
        case Eventually(inner):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            for i in range(pos, len(t)):
                eval_result = evalFormula(inner, t, i)
                if isinstance(eval_result, ReallyNone):
                    return ReallyNone()
                if isinstance(eval_result, Some) and eval_result.value:
                    return Some(True)
            return Some(False)
        case Once(inner):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            for i in range(0, pos+1):
                eval_result = evalFormula(inner, t, i)
                if isinstance(eval_result, ReallyNone):
```

```
                    return ReallyNone()
                if isinstance(eval_result, Some) and eval_result.value:
                    return Some(True)
            return Some(False)
        case Historically(inner):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            for i in range(0, pos+1):
                eval_result = evalFormula(inner, t, i)
                if isinstance(eval_result, ReallyNone):
                    return ReallyNone()
                if isinstance(eval_result, Some) and not
                ↪  eval_result.value:
                    return Some(False)
            return Some(True)
        case Yesterday(inner):
            if pos < 0 or pos >= len(t):
                return ReallyNone()
            if pos >= 1:
                return evalFormula(inner, t, pos - 1)
            else:
                return Some(False)
        case _:
            return ReallyNone()```

  Your task is to fill up the value of the  variable formulaToFind in
  ↪  the code such that  if the user chooses a value for
  ↪  traceGivenAsInput, the program will print "TRUE" if and only if
  ↪  the user-chosen value for traceGivenAsInput satisfies the formula.
For choosing the value for `formulaToFind`, you are given the
following natural language description along with a mapping from
natural language fragment to variable names to use.
You should restrict yourself to only using those variable names
given to you in the mapping, and nothing else.
Input:
 Natural Language: "{natural_language}"
 Atomic Propositions: {atomic_propositions}

 You MUST use ONLY the Python class constructors provided below
(AtomicProposition, Eventually, Always, LAnd, LOr, LNot, LImplies,
↪  LEquiv, Next, Until).
You MUST only use the variables provided in the Atomic Propositions
↪  mapping.

You MUST return ONLY a single line of valid Python code like this
formulaToFind = <your formula here>
"""
```

A Additional Results

This section provides detailed per-model and per-task results complementing the summary statistics presented in Section 5. All data is available in the interactive dashboard and GitHub repository.

A NL-to-Future LTL Results

Table 5, 6 and 7 presents semantic equivalence, entailment, and syntactic correctness results for all models under MINIMAL, DETAILED and PYTHON prompt.

Table 5: NL-to-Future LTL: Minimal Prompting Strategy (All Models)

Model	Equiv_Acc	GT_to_Pred	Pred_to_GT	Syn_Corr	Precision	Recall	F1
claude-sonnet	44.23	64.62	60.94	93.75	44.23	64.62	52.52
gemini-1.5-flash	49.42	64.48	67.31	95.72	49.42	64.48	55.95
gpt-3.5-turbo	26.19	52.78	43.80	96.38	26.19	52.78	35.01
gpt-4o-mini	36.23	61.23	56.12	98.36	36.23	61.23	45.53
gpt-4o	52.43	67.79	70.52	96.05	52.43	67.79	59.13

Table 6: NL-to-Future LTL: Detailed Prompting Strategy (All Models)

Model	Equiv_Acc	GT_to_Pred	Pred_to_GT	Syn_Corr	Precision	Recall	F1
claude-sonnet	56.06	71.63	69.07	98.68	56.06	71.63	62.89
gemini-1.5-flash	46.32	62.34	63.32	93.75	46.32	62.34	53.15
gpt-3.5-turbo	9.79	49.48	20.65	86.18	9.79	49.48	16.35
gpt-4o-mini	39.02	64.11	49.66	100.00	39.02	64.11	48.52
gpt-4o	53.61	70.10	67.35	99.01	53.61	70.10	60.75

Table 7: NL-to-Future LTL: Python Prompting Strategy (All Models)

Model	Equiv_Acc	GT_to_Pred	Pred_to_GT	Syn_Corr	Precision	Recall	F1
gemini-1.5-pro	58.00	80.00	64.82	83.71	58.00	80.00	67.25
gemini-1.5-flash	45.39	69.74	56.32	93.81	45.39	69.74	54.99
gemini-2.5-flash	60.34	75.52	71.09	97.07	60.34	75.52	67.08
claude-sonnet	58.97	72.76	71.92	97.39	58.97	72.76	65.14

The consistent 20–30% improvement from Minimal to Python across all models demonstrates the effectiveness of code-based reformulation, even for models like GPT-3.5-turbo that show poor absolute performance.

B NL-to-Past LTL Results

Table 8 presents semantic equivalence, entailment, and syntactic correctness results for all models under minimal prompting. Table 9 shows the same metrics under detailed prompting. Table 10 presents results under Python code-based prompting.

Table 8: NL-to-Past LTL: Minimal Prompting Strategy (All Models)

Model	Equiv_Acc	GT_to_Pred	Pred_to_GT	Syn_Corr	Precision	Recall	F1
claude-sonnet	56.72	75.21	74.04	85.92	56.72	75.21	64.67
gemini-1.5-flash	65.56	80.50	80.99	88.03	65.56	80.50	72.27
gpt-3.5-turbo	42.60	65.92	69.54	86.97	42.60	65.92	51.76
gpt-4o-mini	54.44	72.98	72.24	88.73	54.44	72.98	62.36
gpt-4o	66.67	80.30	81.37	95.42	66.67	80.30	72.85

Table 9: NL-to-Past LTL: Detailed Prompting Strategy (All Models)

Model	Equiv_Acc	GT_to_Pred	Pred_to_GT	Syn_Corr	Precision	Recall	F1
claude-sonnet	66.43	80.87	81.59	98.94	66.43	80.87	72.94
gemini-1.5-flash	69.96	83.27	80.54	96.83	69.96	83.27	76.04
gpt-3.5-turbo	36.55	61.42	60.87	79.58	36.55	61.42	45.83
gpt-4o-mini	63.78	79.53	76.68	92.25	63.78	79.53	70.79
gpt-4o	64.75	78.78	81.23	99.65	64.75	78.78	71.08

Table 10: NL-to-Past LTL: Python Prompting Strategy (All Models)

Model	Equiv_Acc	GT_to_Pred	Pred_to_GT	Syn_Corr	Precision	Recall	F1
gemini-1.5-pro	69.78	82.67	80.26	82.11	69.78	82.67	75.68
gemini-1.5-flash	55.78	80.08	64.02	94.74	55.78	80.08	65.75
gemini-2.5-flash	71.64	84.33	82.96	96.14	71.64	84.33	77.47
claude-sonnet	63.47	81.18	76.64	97.89	63.47	81.18	71.24

In this figure, it is evident that Python reformulation outperforms minimal and detailed prompting across all tasks, with the most significant improvements on NL-to-LTL translation using textbook data. This suggests that code-based reformulation is particularly effective for complex, formally-structured specifications found in academic sources. The consistency of Python's advantage across independent datasets (Little Tricky Logic and textbooks) indicates the effect is robust and not an artifact of dataset selection. This finding supports our hypothesis that leveraging LLMs' native code-reasoning capabilities transfers effectively to formal logic specification tasks.

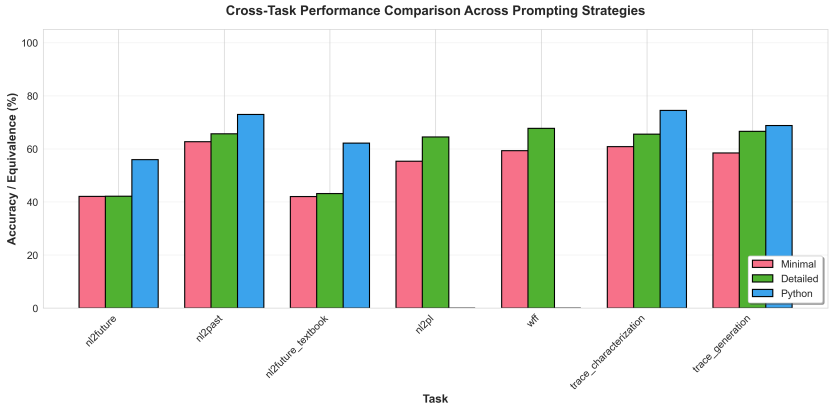

Fig. 6: Cross-task performance between MINIMAL, DETAILED and PYTHON Approaches

C Model Details and Versions

We selected models representing the current state-of-the-art across three major providers: Anthropic, Google, and OpenAI. Model selection prioritized diversity in architecture, training data, and model scale to capture a broad spectrum of LLM capabilities and design choices. These models were chosen for three reasons: (1) *Accessibility*: all are available via public APIs, enabling reproducible evaluation; (2) *Diversity*: they represent different architectures, training objectives (e.g., constitutional AI vs. reinforcement learning from human feedback), and release dates; (3) *Practical relevance*: these are the models most commonly used in industry and research for formal specification tasks at the time of evaluation (early 2024).

Table 11: LLM Models Evaluated With Details and Access Information

Model	**Provider**	**Version/Checkpoint**	**API Endpoint**
Claude-Sonnet	Anthropic	claude-3-sonnet-20240229	claude.anthropic.com
Claude-3.5-Sonnet	Anthropic	claude-3-5-sonnet-20241022	claude.anthropic.com
Gemini-1.5-pro	Google	gemini-1.5-pro-001	google.generativeai.google.com
Gemini-1.5-flash	Google	gemini-1.5-flash-001	google.generativeai.google.com
Gemini-2.5-flash	Google	gemini-2.5-flash-001	google.generativeai.google.com
GPT-4	OpenAI	gpt-4-0613	api.openai.com
GPT-3.5-turbo	OpenAI	gpt-3.5-turbo-0613	api.openai.com