

GINSIGN: GROUNDING NATURAL LANGUAGE INTO SYSTEM SIGNATURES FOR TEMPORAL LOGIC TRANSLATION

Anonymous authors

Paper under double-blind review

ABSTRACT

Natural language (NL) to temporal logic (TL) translation enables engineers to specify, verify, and enforce system behaviors without manually crafting formal specifications—an essential capability for building trustworthy autonomous systems. While existing NL-to-TL translation frameworks have demonstrated encouraging initial results, these systems either explicitly assume access to accurate atom grounding or suffer from low grounded translation accuracy. In this paper, we propose a framework for **Grounding Natural Language Into System Signatures** for Temporal Logic translation called GinSign. The framework introduces a grounding model that learns the *abstract task* of mapping NL spans onto a given system signature: given a lifted NL specification and a system signature \mathcal{S} , the classifier must assign each lifted atomic proposition to an element of the set of signature-defined atoms \mathcal{P} . We decompose the grounding task hierarchically—first predicting *predicate* labels, then selecting the appropriately typed *constant* arguments. Decomposing this task from a free-form generation problem into a structured classification problem permits the use of smaller masked language models and eliminates the reliance on expensive LLMs. Experiments across multiple domains show that frameworks which omit grounding tend to produce syntactically correct lifted LTL that is semantically nonequivalent to grounded target expressions, whereas our framework supports downstream model checking and achieves grounded logical-equivalence scores of 95.5%, a $1.4\times$ improvement over SOTA.

1 INTRODUCTION

Formal language specifications are foundational to a wide range of systems, including autonomous robots and vehicles (Tellex et al., 2020; Raman et al., 2013; Mallozzi et al., 2019; Harapanahalli et al., 2019), cyber-physical controllers (Konur, 2013; Abbas et al., 2013; Hoxha et al., 2018), and safety-critical software (Alur, 2015; Yoo et al., 2009; Bowen & Stavridou, 1993). Among these, temporal logic (TL) specification plays a central role in enabling formal verification of these systems (Watson & Scheidt, 2005; Bellini et al., 2000). Despite its power, TL specification is difficult and typically requires domain expertise (Yin et al., 2024; Cardoso et al., 2021; Thistle & Wonham, 1986). In practice, system requirements are often provided by stakeholders in natural language (NL), which is inherently ambiguous and lacks formal precision (Veizaga et al., 2021; Lamar, 2009; Lafi et al., 2021). To bridge this gap, there has been growing interest in using artificial intelligence methods to automatically translate natural language specifications into temporal logic (Fuggitti & Chakraborti, 2023; Chen et al., 2023).

Despite recent gains from deploying sequence-to-sequence (seq2seq) (Hahn et al., 2022; Pan et al., 2023; Hsiung et al., 2022) and large language models (LLMs) (Fuggitti & Chakraborti, 2023; Chen et al., 2023; Xu et al., 2024; Cosler et al., 2023), most NL-to-TL translation pipelines remain non-deployable in real-world systems. While lifted translation systems often yield well-formed and valid TL specifications, the atomic propositions (APs) over which the TL operates are never explicitly defined. This is a deliberate choice by SOTA methods which explicitly assume accurate groundings are readily available and the mappings are known (Hsiung et al., 2021; Chen et al., 2023). However, without a semantic definition for each AP, which is grounded in a system and the world in which the system operates, the resulting TL formulas cannot be interpreted on traces or state machines, making

054
055
056
057
058
059
060
061
062
063
064
065
066
067
068
069
070
071
072
073
074
075
076
077
078
079
080
081
082
083
084
085
086
087
088
089
090
091
092
093
094
095
096
097
098
099
100
101
102
103
104
105
106
107

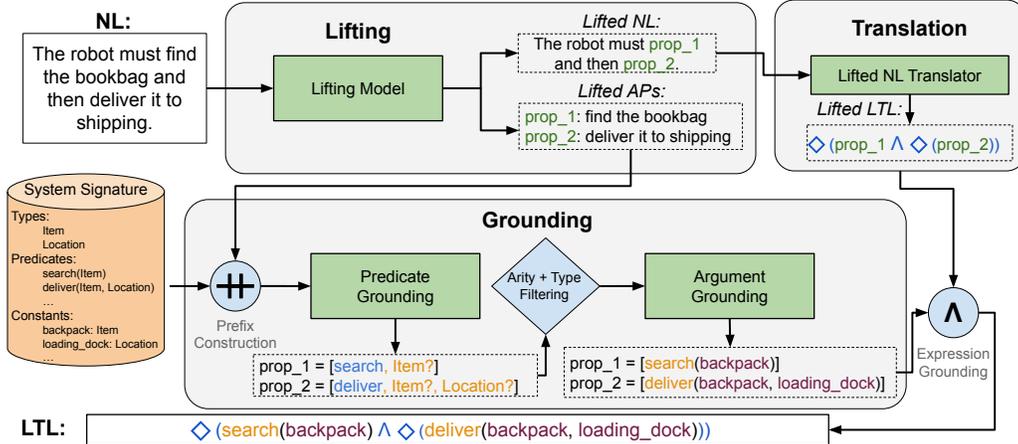


Figure 1: Overview of the GinSign Framework.

real-world deployment impossible. While there has been significant progress on *visual grounding* (linking natural language expressions to entities in perceptual scenes), these methods target perceptual reference resolution rather than the formal symbolic grounding considered in this work (Qiao et al., 2020). In our setting, Lang2LTL (Liu et al., 2023) employs an embedding-similarity approach to align lifted APs with semantic maps. However, this method undershoots state-of-the-art lifted translation accuracy by more than 20%, highlighting the challenge of achieving both accurate lifting and precise grounding simultaneously.

To illustrate this challenge, consider the NL translation shown in the top half of Figure 1. Many systems translate, “The robot must find the bookbag and then deliver it to shipping.”, to $\diamond(prop_1 \wedge \diamond(prop_2))$, which is logically coherent but semantically useless. Because the system’s underlying predicates (*search*, *deliver*) and constants (*backpack*, *loading_dock*) are not incorporated, the translation lacks operational meaning. This disconnect between syntax and semantics is a primary bottleneck for deploying NL-to-TL systems in grounded, dynamic environments. To address this gap, we posit that effective NL-to-TL translation for real-world deployment must ground all APs, which is shown in the bottom of Figure 1.

In this paper, we propose GinSign, a framework for **G**rounding Natural Language **I**nto System **S**ignatures for Temporal Logic. Our framework bridges the current disconnect between theoretical and practical application by both (1) inferring logical structure (temporal operators, boolean connectives), and (2) grounding NL spans to an inventory of atomic propositions with clear semantic grounding within a system. Our main contributions are as follows:

- We generalize the AP grounding task into a multi-step classification problem: given an NL string (lifted AP) and a system signature (potentially containing unseen classes), classify the NL span into a predicate and its arguments, grounded in the system signature.
- We propose a solution to the above problem using a hierarchical grounding framework. First, intermediate grounding is performed to classify the AP into a predicate on the system signature via an encoder-only model. Then, using prior knowledge of the predicate arity and types defined in the system signature, the same model is used to ground the typed arguments of that predicate.
- In experimental evaluation, GinSign outperforms LLM baselines in the grounded translation task by up to 1.4 \times . We establish a new baseline for end-to-end grounded translation on multiple datasets, presenting NL-to-TL translation that is executable on real traces.

The remainder of this paper is organized as follows. Section 2 formalizes NL-to-TL translation. Related work is surveyed in Section 3. Section 4 presents our grounding-first framework. Section 5 reports empirical results and analyzes common failure modes. Section 6 concludes this work with a discussion of limitations and directions for future work.

2 BACKGROUND

In this section, we provide preliminaries on linear temporal logic, we motivate the use of system signatures in the natural language grounding problem, we formulate the NL-to-TL translation task, and we discuss current SOTA NL-to-TL translation approaches.

2.1 LINEAR TEMPORAL LOGIC

The syntax of LTL is given by the following grammar:

$$\begin{aligned} \varphi ::= & \pi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \\ & \mid \bigcirc\varphi \mid \diamond\varphi \mid \square\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \end{aligned}$$

Where each atomic proposition belongs to a set of known symbols $\pi \in \mathcal{P}$. To verify a real system against an LTL requirement, one typically models the implementation as a finite-state Kripke structure $\mathcal{M} = (S, S_0, R, L)$, where S is the state set, $S_0 \subseteq S$ the initial states, $R \subseteq S \times S$ the total transition relation, and $L : S \rightarrow 2^{\mathcal{P}}$ the labeling function. Because every atomic proposition in φ is interpreted via L , *grounding* those propositions in the signature of \mathcal{M} is prerequisite to even running the model checker. In other words, the syntactic formula only becomes semantically testable once its APs are linked to concrete predicates over system states (Hsiung et al., 2021).

2.2 SYSTEM SIGNATURES

Well-designed automated systems and planners are typically grounded in a *planning domain definition language* (Ghallab et al., 1998), action vocabulary, or some other domain-specific semantic library for a system (Zhang et al., 2024; Oswald et al., 2024). We observe that these languages are realizations of many-sorted logical systems, and are therefore motivated to further apply this formalism in our efforts to ground TL specifications. Accordingly, we look to system signatures as the formal vocabulary that ties a grounded TL specification to the structure it specifies, as any well-formed system should have a system signature. Formally, a many-sorted system signature is defined as follows:

$$\mathcal{S} = \langle T, P, C \rangle$$

where T is a set of *type* symbols, P is a set of *predicate* symbols, and C is a set of *constant* symbols. System signatures are used to describe all of the non-logical terminals in a formal language (Finkbeiner & Zarba, 2006).

Each component of \mathcal{S} plays a distinct but interconnected role. Types $t \in T$ act as categories that restrict how constants and predicates may be combined—for example, distinguishing between arguments of type `location`, `agent`, or `item`. Constants $c \in C$ are the specific instances of these types, such as a particular `location` like `loading_dock`, or a particular `item` like `apple`. Predicates $p \in P$ then specify relations or properties defined over typed arguments: $p(t_1, \dots, t_m)$ requires arguments drawn from the corresponding type sets, yielding well-typed atoms like `deliver(apple, loading_dock)`. Thus, the connection between types, constants, and predicates informs the structure of possible grounding targets: constants instantiate types, and predicates bind these constants together into statements about the world.

2.3 NL-TO-TL TRANSLATION

In this section, we review the natural language to temporal logic (NL-to-TL) translation task. Prior work divides the task into the following three phases: lifting, translation, and grounding. To make this process concrete, we illustrate each step with the example specification: “Eventually pick up the package from room A.”

Lifting: We define *lifting* as the following classification problem. Given a sequence of tokens that constitute an NL specification \mathbb{S} , perform integer classification on each token as either a reference to some π , or not. Formally:

$$\lambda : \mathbb{S} \rightarrow \begin{cases} 0, & \mathbb{S}_i \text{ is not part of any } \pi \in \mathcal{P}, \\ n, & \mathbb{S}_i \text{ is part of } \pi_n. \end{cases}$$

The result of successful lifting is a mapping $\lambda: \mathbb{S} \rightarrow \{i | i \in \mathbb{Z}\}$ of lifted substrings to integer AP references that appear in the corresponding LTL expression. In LTL, each atomic proposition π is a boolean variable whose value is determined at evaluation time. The value is given by the presence of that AP on a trace. *Example.* In the sentence above, the phrase “pick up the package from room A” is identified as a reference to an atomic propositions. Lifting replaces them with placeholders, producing: “Eventually prop_1 .”

Translation: Given a natural-language specification s , the goal is to produce a temporal-logic formula φ that preserves the intended behavior: $f: s \rightarrow \varphi$. Let $\mathcal{P} = \{\pi_1, \dots, \pi_m\}$ be the finite set of atomic propositions, each with a semantic interpretation over traces (e.g., $\llbracket \pi_i \rrbracket \subseteq \Sigma^\omega$). A TL formula is built from \mathcal{P} using boolean and temporal operators (e.g., $\diamond, \square, \bigcirc, \mathcal{U}$). For $f(s)$ to be actionable, every π_i appearing in φ must be mapped to a meaning defined in the system signature \mathcal{S} , outlined in section 2.2. *Example.* The lifted string “Eventually prop_1 ” is translated into the LTL formula $\varphi = \diamond \text{prop}_1$.

Grounding: Let the lifted specification contain k placeholder atoms $\{\text{prop}_1, \dots, \text{prop}_k\}$. Grounding is defined as a total function

$$g_{\mathcal{S}}: \{\text{prop}_1, \dots, \text{prop}_k\} \rightarrow \underbrace{\{p \mid p \in P\} \cup \{p(c_1, \dots, c_m) \mid p \in P, c_i \in C\}}_{\mathcal{P}_{\mathcal{S}}},$$

which assigns to every placeholder either (i) a predicate $p \in P$ (nullary case) or (ii) a fully instantiated atom $p(c_1, \dots, c_m)$ whose arguments c_i are constants of the appropriate type. The image set $\mathcal{P}_{\mathcal{S}}$ thus represents the *grounded* atomic-proposition vocabulary permitted by \mathcal{S} . For the remainder of the paper, we will refer to constants as *arguments*, as they are used exclusively as arguments accepted by predicates in P . *Example.* Given the system signature $\mathcal{S} = \langle T = \{\text{room}, \text{object}\}, P = \{\text{pick_up}(\text{obj}, \text{room})\}, C = \{\text{roomA}, \text{package1}\}\rangle$, the grounding step resolves $\text{prop}_1 \mapsto \text{pick_up}(\text{package1}, \text{roomA})$. The final grounded formula is: $\varphi = \diamond \text{pick_up}(\text{package1}, \text{roomA})$.

3 RELATED WORK

Current frameworks for neural NL-to-TL translation either attempt translation in one shot using an LLM prompting approach, or divide the task into multiple steps, including lifting, verification, and human-in-the-loop feedback. An overview is provided in Table 1.

NL2LTL: (Fuggitti & Chakraborti, 2023) introduce a Python package that implements a few-shot, template-based approach to NL-to-LTL translation. Users of this package are required to supply LTL templates and example translations in order to construct the prompt.

NL2TL: (Chen et al., 2023) introduce a framework that decomposes the end-to-end translation task into 1) lifting with a GPT model and 2) translation with a seq2seq model. Most critically, this approach reports that lifting is an effective method for reducing translation complexity. We continue to exploit this fact in our framework, yet, we find this lifted translation can not be *verified* until the lifted APs have been grounded—a step omitted from this framework, as shown in the table.

Lang2LTL: (Liu et al., 2023) propose a modular pipeline for language grounding and translation into LTL. Their framework separates the task into referring expression recognition with an LLM, grounding through embedding similarity against a semantic database. Yet it presumes access to proposition embeddings from a semantic map and does not extend this mechanism to arbitrary system signatures, which we perform in our work.

Framework	Lifting	Translation	Grounding
LLM-Baseline	×	✓	×
NL2LTL (Fuggitti & Chakraborti, 2023)	×	✓	×
NL2TL (Chen et al., 2023)	✓	✓	×
Lang2LTL (Liu et al., 2023)	✓	✓	✓
GinSign (ours)	✓	✓	✓

Table 1: Overview of each framework’s support for lifting, grounding, and translation.

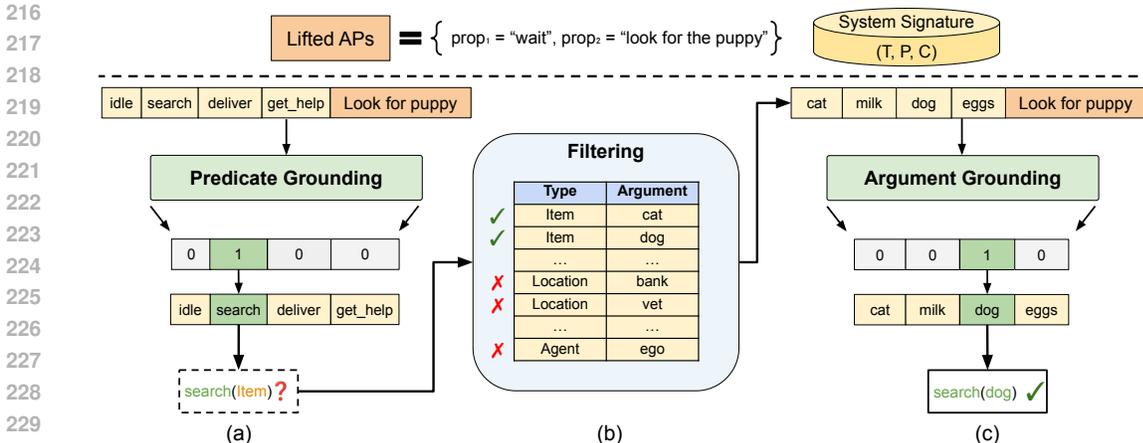


Figure 2: An overview of our grounding components. Given n lifted NP APs, we convert the system signature into a prefix using Algorithm 1. The lifted NL is first combined with the predicates prefix to ground the predicate to a known action (a). Since each predicate requires an argument, we filter out non-candidate arguments by type (b). We then combine the lifted NL with the arguments prefix to classify the correct argument (c). Both predicate and argument grounding use the same token classification BERT model, which processes any prefix and lifted NL.

4 METHODOLOGY

In this section, we introduce GinSign, an end-to-end *grounded* NL-to-TL translation framework. GinSign accepts two input components: a system signature, and a natural language specification. We leverage the compositional approach and decompose the NL to TL translation task into lifting, translation, and grounding. We perform the lifting and translation using a BERT (Devlin et al., 2019) and T5 (Raffel et al., 2020) model as in previous works. Our methodology focuses on the grounding of the APs obtained from the lifting into the defined state space. The grounded APs will then be inserted into the temporal logic formula obtained from the lifted translation.

The input to the grounding module is the NL string associated with each extracted AP. The output is the grounding of this NL string into the state space defined using a system signature. The GinSign framework performs the grounding using a hierarchical approach consisting of predicate grounding and argument grounding. An overview of the framework is shown in Figure 2 and the details are provided in Section 4.1. Both of the grounding steps are formulated as an *abstract grounding task*, which we solve using a BERT model. The details are provided in Section 4.2.

4.1 HIERARCHICAL GROUNDING

In this section, we first describe the hierarchical grounding strategy of GinSign with Figure 2. Next, we provide the details of the predicate grounding and the argument grounding.

Hierarchical strategy: We design a hierarchical grounding strategy because of the dependency between the predicates and arguments. Each predicate has a fixed number of arguments with specified type. The hierarchical decomposition can by design ensure that the correct number of arguments of the right type are assigned to each predicate. The two grounding steps are defined, as follows:

1. **Predicate grounding:** predicts a predicate $p \in P$ for each placeholder, reducing the argument search space to $C_t = \{c \in C \mid \text{type}(c) = t\}$ (Figure 2 (a)).
2. **Argument grounding:** dependent on p , chooses concrete constant(s) from C_t and produces the final atom $p(c_1, \dots, c_m)$, thereby completing g_S (Figure 2 (c)).

This hierarchy turns a single, large open-set decision into two smaller problems. A flat classifier would implicitly consider on the order of $\sum_{p \in P} \prod_{r=1}^{\text{arity}(p)} |C_{\tau_r}|$ labelings per instance (all fully-instantiated atoms), which quickly becomes intractable as $|C|$ grows. In contrast, the first stage selects among $|P|$ predicates; the second stage then solves at most $m = \text{arity}(p)$ independent choices, each over

$|C_{\tau_r}|$ typed constants. Concretely, the effective label budget per instance drops from $\Theta(\prod_r |C_{\tau_r}|)$ to $\Theta(|P| + \sum_r |C_{\tau_r}|)$, a dramatic reduction when $|C| \gg |P|$ and types partition C evenly. Moreover, type filtering eliminates invalid atoms by construction, ensuring any predicted $p(c_1, \dots, c_m)$ lies in \mathcal{P}_S . Coupled with windowed classification (Sec. 4.2), each sub-decision operates over at most m candidates at a time, further improving sample efficiency and calibration while preserving exact compatibility with the evaluation-time tournament.

Predicate Grounding: In Section 2, we framed grounding as an NL classification task. To ground an input specification, the classifier must treat every symbol in the signature as a potential label. Rather than allocating a fixed soft-max head with one output neuron *per* symbol—as typical token-classification pipelines would—we prepend a rigid, pseudo-natural prefix that *enumerates* the target signature and let the encoder attend over it. The BERT backbone, therefore, learns the abstract skill of scoring span–prefix alignments instead of memorizing a static label inventory. This process is seen in Figure 2 (a) where the input is lifted APs and the system signature predicates. Crucially, the class set is no longer baked into the model parameters: supplying a different prefix instantly defines a new label universe, so the same fine-tuned weights can be reused across domains with disjoint signatures. We later show (in Table 5) that this design yields promising out-of-distribution accuracy—comparable to in-domain performance—by simply swapping prefixes, demonstrating that the model has internalized grounding as a transferable reasoning operation rather than rote classification.

Filtering: After predicate grounding, we know which predicate(s) are present in the segment. Using this information, we query the system signature for the arity of the predicate (i.e., the type and number of required arguments). This knowledge filters the search space for argument grounding to the subset of arguments compatible with the predicted predicate’s types. We show the transition of information between predicate and argument grounding in Figure 2 (b), where predicate information from the system signature is used to map lifted APs to the known possible arguments.

Argument Grounding: Argument grounding is then framed as the further classification of typed natural language spans into specific domain arguments. Each argument is resolved independently against its type-filtered candidate set $L_c^{(r)}$ by the same BERT backbone as before, simply with a different prefix, as seen in Figure 2 (c). Thus, the final output always contains the correct number of arguments, with no need for an additional constraint or stopping criterion.

4.2 GROUNDING MODEL

We operationalize g_S as a classifier over an input-defined label set. For each lifted placeholder prop_i , the model receives (i) the local NL context corresponding to the lifted AP and (ii) a *prefix* that enumerates the relevant candidates from $\mathcal{S} = \langle T, P, C \rangle$ up to length maximum input length m . The model points to one element of the prefix; because the prefix is constructed from \mathcal{S} at input time, the label universe is domain-agnostic and requires no change to the classifier head across domains.

Prefix construction. Let $\text{enum}(\cdot)$ produce a fixed-order list of symbols as a token sequence. For predicate grounding, the candidate list is $L_p = \text{enum}(P)$. For argument grounding, once a predicate $\hat{p} \in P$ with arity a and type signature $(\tau_1, \dots, \tau_a) \in T^a$ is predicted, the r -th argument uses the type-filtered set,

$$L_c^{(r)} = \text{enum}(\{c \in C \mid \text{type}(c) = \tau_r\}).$$

We serialize the input as a pair $(x_{\text{AP}}, x_{\text{prefix}})$, where x_{prefix} is the tokenization of L (either L_p or $L_c^{(r)}$). Our exact implementation of this process is given in Appendix A.3, in Algorithm 1. When $N > m$, we perform prefix sharding and apply a two-stage *tournament* procedure. First, the model classifies within each shard W_j . The winning candidate from each shard is then re-assembled into a new prefix list, and the procedure repeats until a single element remains. This ensures scalability to arbitrarily long prefixes while keeping each classification head fixed in size.

Classification. Let $L = [\ell_1, \dots, \ell_N]$ be the candidate list for the current decision. We fix a shard size m and partition L into contiguous windows

$$W_j = [\ell_{(j-1)m+1}, \dots, \ell_{\min(jm, N)}], \quad j = 1, \dots, \lceil N/m \rceil.$$

The model h_θ maps a pair and a window to a discrete index in $\{1, \dots, |W_j|\}$:

$$h_\theta : (x_{\text{AP}}, W_j) \mapsto \hat{y} \in \{1, \dots, |W_j|\}.$$

For short lists ($N \leq m$), we pad L to length m with a `<pad>` token and classify once.

Training objective. For each training instance with gold label $\ell^* \in L$, we construct one or more *gold-in* shards W_j such that $\ell^* \in W_j$ (contiguous windows). The supervision is a single-label CrossEntropy over the shard positions:

$$\mathcal{L}_{\text{CE}}(\theta) = -\log p_{\theta}(y = \text{index}(\ell^* \in W_j) \mid x_{\text{AP}}, W_j).$$

Further implementation details can be found in Appendix A.7. Relevant code is available in the supplementary material and will be made publicly available upon acceptance.

5 EXPERIMENTS

In this section, we present the results of our experiments and evaluations of our grounding framework, as well as its impact on end-to-end translation. This section is organized as follows. Subsection 5.1 discusses training and evaluation corpora information. The results of our isolated grounding evaluation are presented in Subsection 5.2. Finally, end-to-end translation results are presented in Subsection 5.3.

5.1 DATASETS AND METRICS

Table 2: Overview of the magnitude of each domain signature, by field.

Domain (\mathcal{S})	Types $ T $	Predicates $ P $	Arguments $ C $
Search and Rescue	2	7	44
Traffic Light	5	4	175
Warehouse	2	5	82

Datasets We use VLTL-Bench (English et al., 2025b) as the primary dataset for evaluation, as it is the only resource we found that grounds natural language specifications in a concrete state space. VLTL-Bench consists of three distinct domains (Search and Rescue, Traffic Light, and Warehouse), each providing lifted natural language specifications, grounded LTL formulas, and reference traces. Table 2 provides the magnitude of each element of the system signatures used in these datasets. We note here that while the Traffic Light domain has the greatest number of arguments, the Warehouse domain poses a distinct challenge to argument grounding: constants in this domain are not lexically consistent with their surface realizations in text, a difficulty reflected in our results. Because the grounding task—and particularly the grounded logical equivalence metric—requires access to both lifted and grounded APs, prior datasets such as Navigation (Wang et al., 2021), Cleanup World (MacGlashan et al., 2015), and GLTL (Gopalan et al., 2018) are not applicable and are therefore omitted from our evaluations.

Metrics For each evaluation, we report the mean, variance, and confidence of each metric. We compute the following metrics for our evaluations: **LE** (Logical Equivalence, and **GLE** (*Grounded* Logical Equivalence). For the Grounding tasks, we report F_1 scores over all APs. For the End-to-End Translation, **LE** is distinguished from **GLE** in that the former does not account for AP grounding in the Linear Temporal Logic, resulting in high scores for an expression such as “ $\text{prop}_1 \rightarrow \diamond(\text{prop}_2)$ ”, while *grounded* logical equivalence demands that prop_1 and prop_2 are properly defined in order to be scored as correct. To evaluate grounded logical equivalence, grounded TL is parsed by the pyModelChecking framework (Casagrande, 2024), which will extract the APs, allowing for comparison against the ground truth grounding.

5.2 GROUNDING EVALUATION

Here, we discuss the results of the isolated grounding evaluations. We perform three experiments to evaluate the performance of our proposed method against in-context LLM prompting baselines (provided in supplementary materials). First, we report the F_1 score achieved by each approach on the predicate and constant grounding task. In the predicate grounding task, each approach receives a lifted

natural language AP to be classified into one or more domain-specific predicates. In the argument grounding task, each approach receives a lifted natural language AP and all constant arguments of the appropriate type, with the goal of grounding into a specific domain constants.

Table 3: Evaluation of grounding approaches. We report F_1 (per AP) for both predicate and argument grounding. [†] Note that this framework does not distinguish between predicate and argument grounding. We therefore evaluate overall AP grounding in the **Argument Grounding** column. The prompt used by both GPT models is given in Appendix A.6.

Method	Predicate Grounding (F_1 , %)			Argument Grounding (F_1 , %)		
	Traffic Light	Search and Rescue	Warehouse	Traffic Light	Search and Rescue	Warehouse
GPT-3.5 Turbo	73.5	95.0	71.1	93.9	94.0	47.7
GPT-4.1 Mini	76.4	87.7	98.4	94.8	90.9	51.6
GPT-4o	85.9	94.9	82.4	87.0	95.1	70.3
Lang2LTL [†]	-	-	-	86.2	77.6	61.8
GinSign (proposed)	100.0	100.0	100.0	97.9	91.1	94.2

Predicate Grounding Evaluation GinSign performs perfectly (100%) in all domains, showing that prefix-enumerated classification on a label space of only 4-7 classes (as shown in 2) is solvable by lightweight BERT model. By contrast, GPT-3.5 Turbo and GPT-4.1 Mini lag substantially, especially in the Warehouse domain, where GPT-3.5 reaches only 71.1%. These results support our hierarchical decomposition: predicates are quite easily isolated, which promises to assist GinSign’s generalization through reliable filtering.

Argument Grounding Evaluation This task remains the key bottleneck, since each prediction must be drawn from dozens or hundreds of type-compatible constants. Table 2 shows that there are between 44 and 175 classes in the argument-space of each domain, making argument grounding significantly more difficult. GPT-4.1 Mini attains impressive performance in Traffic Light and Search-and-Rescue, but its performance collapses in Warehouse (51.6%). Lang2LTL struggles here as well (61.8% in Warehouse), reflecting the limitations of embedding-similarity when constants are lexically diverse. GinSign, in contrast, maintains robust performance across all domains ($\geq 90\%$), outperforming both LLM prompting and Lang2LTL by a large margin. We conclude that the reliable filtering information obtained by GinSign’s accurate predicate grounding enables notably higher performance on the argument grounding task by virtue of label space reduction.

5.3 END-TO-END TRANSLATION EVALUATION

Table 4 evaluates full NL-to-TL translation. We report both **Logical Equivalence (LE)**, which checks syntactic correctness of the temporal operators and lifted APs, while **Grounded Logical Equivalence (GLE)**, further requires that every AP is correctly grounded. As stated in Section 4, GinSign uses the same BERT lifting model and T5 lifted translation model employed in previous work Chen et al. (2023); English et al. (2025a).

Logical equivalence. Prior seq2seq frameworks (NL2TL and Lang2LTL) all achieve near-perfect LE scores (95-100%). This suggests that lifting-based pipelines reliably capture operator structure. NL2LTL, which relies purely on prompting, lags at $\approx 42\%$.

Grounded logical equivalence. Here the differences are stark. No prior work except Lang2LTL attempts grounding, so GLE cannot be measured on these frameworks. Lang2LTL, which does

Table 4: **End-to-end Translation** evaluation results.

Baseline	Traffic Light		Search and Rescue		Warehouse	
	LE (%)	GLE (%)	LE (%)	GLE (%)	LE (%)	GLE (%)
NL2LTL (GPT-4.1) [†]	43.6	38.4	41.8	35.4	42.6	26.2
NL2TL [†]	98.7	60.1	95.0	54.4	99.0	46.2
Lang2LTL	100.0	73.6	100.0	59.0	100.0	38.8
GinSign (Proposed)	100.0	98.3	100.0	93.4	100.0	95.0

attempt grounding, suffers a significant reduction in accuracy, achieving GLE (73.6% in Traffic Light, 59.0% in S&R, 38.8% in Warehouse). In contrast, GinSign achieves $\geq 93\%$ in all domains, including 95.0% in Warehouse, representing more than a $2.4\times$ absolute gain over the strongest prior method in this domain. By stress testing the grounding components of these two frameworks, we reveal the high cost incurred by inaccurate AP grounding.

5.4 GROUNDING ABLATION

Here we ablate the BERT grounding model used for predicate and argument grounding in GinSign. We perform two ablations. First, in Table 5, we evaluate three models: one trained only on predicate grounding, one trained only on argument grounding, and one trained jointly on both tasks. All three models are trained on all domains but with partial domain signatures, and they are tested on the portions of the signatures held out during training. The list of held-out predicates and arguments for each signature is provided in Appendix A.5. Next, in Table 6, we evaluate the same three models, but each is trained on only two of the three domains at a time, using the full domain signatures, to assess how grounding generalizes to unseen domains.

Our results in Table 5 show that grounding maps natural language into elements of a system signature in a way that is not only generalizable by a single model but also improves out-of-distribution performance. Despite having no exposure to the predicates and arguments included in this evaluation during training, the jointly trained model matches or exceeds the performance of the specialized (pred or arg only) models. In Table 6, the diagonal entries show how GinSign generalizes to unseen domains. We find that GinSign consistently achieves high accuracy and respectable F_1 scores across all out-of-domain evaluations. Unsurprisingly, under this training regime, GinSign performs better on in-distribution domains than in Table 5, because the latter evaluates only on out-of-distribution data, whereas here the entire signature is in-domain. Overall, these two ablation studies demonstrate the robustness of the proposed GinSign approach, highlighting its ability to generalize to both unseen grounding tasks (predicates or arguments) and unseen domains.

Table 5: Evaluation of Intra-Domain OOD performance on the Predicate and Argument grounding task.

Model	Task	Traffic Light		Search & Rescue		Warehouse	
		Acc	F_1	Acc	F_1	Acc	F_1
Pred Only	Predicate	75.0	69.7	92.7	83.9	83.9	71.6
Joint	Predicate	83.1	80.3	92.7	85.5	85.1	73.0
Arg Only	Argument	99.1	94.7	97.0	88.4	93.2	62.6
Joint	Argument	99.9	99.5	99.1	96.2	94.2	66.7

Error Analyses Here we discuss the most prevalent failure modes that arose during our evaluations. Firstly, we observe the significantly lower performance exhibited by almost all grounding approaches on the Warehouse domain. In the argument grounding evaluation, the mean F_1 across all approaches is 63.8%, compared to a mean F_1 of over 90% over the Traffic Light and Search and Rescue domains. We found that the diverse natural-language references to the `item` arguments make them difficult to ground, leading to frequent errors. Additionally, the LLM-based approaches often failed to identify correct constants on the signature given in the input. This difficulty motivated our development of the hierarchical filtering approach, described in 4.1.

6 CONCLUSION

In this paper, we introduce GinSign, the first end-to-end grounded NL-to-TL framework that anchors every atomic proposition to a system signature. Treating grounding as an open-set, hierarchical span-classification task cleanly separates syntactic translation from semantic anchoring. Experiments on VLTL-Bench show that adding the signature prefix essentially solves both predicate and argument grounding, closing the accuracy gap with larger language models and even outperforms them on visually oriented domains. Crucially, explicit grounding enables model-checking evaluation, exposing

Table 6: Evaluation of Cross-Domain OOD performance on the Predicate and Argument grounding task.

Holdout Domain	Model	Task	Traffic Light		Search & Rescue		Warehouse	
			Acc	F ₁	Acc	F ₁	Acc	F ₁
Traffic Light	Pred Only	Predicate	100.0	100.0	100.0	100.0	100.0	100.0
	Joint	Predicate	98.9	82.0	100.0	100.0	100.0	100.0
	Arg Only	Argument	95.2	66.8	100.0	100.0	100.0	99.8
	Joint	Argument	95.6	68.2	100.0	100.0	99.8	97.8
Search & Rescue	Pred Only	Predicate	100.0	100.0	97.2	55.4	100.0	100.0
	Joint	Predicate	100.0	100.0	95.7	50.7	100.0	100.0
	Arg Only	Argument	100.0	100.0	96.4	72.9	100.0	99.7
	Joint	Argument	100.0	100.0	94.0	61.4	99.8	98.2
Warehouse	Pred Only	Predicate	100.0	100.0	100.0	100.0	99.2	86.8
	Joint	Predicate	100.0	100.0	100.0	100.0	99.2	87.0
	Arg Only	Argument	100.0	100.0	100.0	100.0	96.9	63.6
	Joint	Argument	100.0	100.0	100.0	99.9	97.1	65.4

semantic errors that remain invisible to purely string-based metrics and pushing NL-to-TL translation from seemingly plausible output toward truly verifiable specifications. We hope this work sparks broader adoption of grounded translation benchmarks and inspires future research on scalable grounded translation for richer temporal logics and larger system vocabularies.

Limitations and Future Work GinSign was tested only on VLTL-Bench, whose signatures may not reflect larger or evolving systems. The framework handles propositional LTL; extending it to metric or first-order variants will require grounding for numbers, time bounds, and quantifiers. Constant-level grounding remains the accuracy bottleneck, especially when names are ambiguous, and the method assumes the signature is fixed at inference. Future work should introduce richer benchmarks, add retrieval- or interaction-based grounding to tackle large constant sets, and develop mechanisms that adapt to signature updates without retraining.

REFERENCES

- 540
541
542 Houssam Abbas, Georgios Fainekos, Sriram Sankaranarayanan, Franjo Ivančić, and Aarti Gupta.
543 Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embed-*
544 *ded Computing Systems (TECS)*, 12(2s):1–30, 2013.
- 545 Rajeev Alur. *Principles of Cyber-Physical Systems*. The MIT Press, 2015. ISBN 0262029111.
- 546
547 Pierfrancesco Bellini, Riccardo Mattolini, and Paolo Nesi. Temporal logics for real-time system
548 specification. *ACM Computing Surveys (CSUR)*, 32(1):12–42, 2000.
- 549 Jonathan Bowen and Victoria Stavridou. Safety-critical systems, formal methods and standards.
550 *Software engineering journal*, 8(4):189–209, 1993.
- 551
552 Rafael C Cardoso, Georgios Kourtis, Louise A Dennis, Clare Dixon, Marie Farrell, Michael Fisher,
553 and Matt Webster. A review of verification and validation for space autonomous systems. *Current*
554 *Robotics Reports*, 2(3):273–283, 2021.
- 555 Alberto Casagrande. `pymodelchecking`. [https://github.com/albertocasagrande/](https://github.com/albertocasagrande/pyModelChecking)
556 `pyModelChecking`, 2024.
- 557
558 Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. NI2tl: Transforming natural languages
559 to temporal logics using large language models. In *Proceedings of the 2023 Conference on*
560 *Empirical Methods in Natural Language Processing*, 2023.
- 561 Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. `nl2spec`:
562 Interactively translating unstructured natural language to temporal logics with large language
563 models. In *Computer Aided Verification: 35th International Conference, CAV 2023, Paris, France,*
564 *July 17–22, 2023, Proceedings, Part II*, pp. 383–396, Berlin, Heidelberg, 2023. Springer-Verlag.
565 ISBN 978-3-031-37702-0. doi: 10.1007/978-3-031-37703-7_18. URL [https://doi.org/](https://doi.org/10.1007/978-3-031-37703-7_18)
566 [10.1007/978-3-031-37703-7_18](https://doi.org/10.1007/978-3-031-37703-7_18).
- 567
568 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep
569 bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL
570 <http://arxiv.org/abs/1810.04805>.
- 571 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep
572 bidirectional transformers for language understanding, 2019. URL [https://arxiv.org/](https://arxiv.org/abs/1810.04805)
573 [abs/1810.04805](https://arxiv.org/abs/1810.04805).
- 574
575 William H English, Dominic Simon, Sumit Kumar Jha, and Rickard Ewetz. Grammar-forced
576 translation of natural language to temporal logic using LLMs. In *Forty-second International*
577 *Conference on Machine Learning*, 2025a. URL [https://openreview.net/forum?id=](https://openreview.net/forum?id=p411a7WHox)
578 [p411a7WHox](https://openreview.net/forum?id=p411a7WHox).
- 579 William H English, Chase Walker, Dominic Simon, Sumit Kumar Jha, and Rickard Ewetz. Verifiable
580 natural language to linear temporal logic translation: A benchmark dataset and evaluation suite,
581 2025b. URL <https://arxiv.org/abs/2507.00877>.
- 582 Bernd Finkbeiner and Calogero G. Zarba. Many-sorted logic. In *Decision Procedures for Verification*,
583 chapter 1. 2006. Lecture notes, Reactive Systems Group, CISPA.
- 584
585 Francesco Fuggitti and Tathagata Chakraborti. NI2l - a python package for converting natural lan-
586 guage (nl) instructions to linear temporal logic (ltl) formulas. In *AAAI Conference on Artificial Intel-*
587 *ligence*, 2023. URL <https://api.semanticscholar.org/CorpusID:259726762>.
- 588 M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins.
589 PDDL—The Planning Domain Definition Language, 1998. URL [http://citeseerx.ist.](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212)
590 [psu.edu/viewdoc/summary?doi=10.1.1.37.212](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212).
- 591
592 Nakul Gopalan, Dilip Arumugam, Lawson L. S. Wong, and Stefanie Tellex. Sequence-to-sequence
593 language grounding of non-markovian task specifications. *Robotics: Science and Systems XIV*,
2018. URL <https://api.semanticscholar.org/CorpusID:46994194>.

- 594 Christopher Hahn, Frederik Schmitt, Julia J Tillman, Niklas Metzger, Julian Siber, and Bernd
595 Finkbeiner. Formal specifications from natural language. *arXiv preprint arXiv:2206.01962*, 2022.
596
- 597 Suman Harapanahalli, Niall O Mahony, Gustavo Velasco Hernandez, Sean Campbell, Daniel Riordan,
598 and Joseph Walsh. Autonomous navigation of mobile robots in factory environment. *Procedia*
599 *Manufacturing*, 38:1524–1531, 2019.
- 600 Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. Mining parametric temporal logic properties
601 in model-based design for cyber-physical systems. *International Journal on Software Tools for*
602 *Technology Transfer*, 20(1):79–93, 2018.
- 603 Eric Hsiung, Hiloni Mehta, Junchi Chu, Xinyu Liu, Roma Patel, Stefanie Tellex, and George
604 Konidaris. Generalizing to new domains by mapping natural language to lifted LTL. *CoRR*,
605 abs/2110.05603, 2021. URL <https://arxiv.org/abs/2110.05603>.
- 607 Eric Hsiung, Hiloni Mehta, Junchi Chu, Xinyu Liu, Roma Patel, Stefanie Tellex, and George
608 Konidaris. Generalizing to new domains by mapping natural language to lifted ltl. In *2022*
609 *International Conference on Robotics and Automation (ICRA)*, pp. 3624–3630. IEEE, 2022.
- 610 Savas Konur. A survey on temporal logics for specifying and verifying real-time systems. *Frontiers*
611 *of Computer Science*, 7(3):370, 2013. doi: 10.1007/s11704-013-2195-2. URL [https://](https://journal.hep.com.cn/fcs/EN/abstract/article_4956.shtml)
612 journal.hep.com.cn/fcs/EN/abstract/article_4956.shtml.
- 613 Mohammed Lafi, Bilal Hawashin, and Shadi AlZu’bi. Eliciting requirements from stakeholders’
614 responses using natural language processing. *Computer Modeling In Engineering & Sciences*, 127
615 (1):99–116, 2021.
- 617 Carl Lamar. Linguistic analysis of natural language engineering requirements. Master’s thesis,
618 Clemson University, 2009.
- 619 Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and
620 Ankit Shah. Grounding complex natural language commands for temporal tasks in unseen
621 environments. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th*
622 *Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp.
623 1084–1110. PMLR, 06–09 Nov 2023. URL [https://proceedings.mlr.press/v229/](https://proceedings.mlr.press/v229/liu23d.html)
624 [liu23d.html](https://proceedings.mlr.press/v229/liu23d.html).
- 625 James MacGlashan, Monica Babes-Vroman, Marie desJardins, Michael L. Littman, Smaranda
626 Muresan, S. Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. Grounding english
627 commands to reward functions. In *Robotics: Science and Systems*, 2015. URL [https:](https://api.semanticscholar.org/CorpusID:1709515)
628 [//api.semanticscholar.org/CorpusID:1709515](https://api.semanticscholar.org/CorpusID:1709515).
- 630 Piergiuseppe Mallozzi, Patrizio Pelliccione, Alessia Knauss, Christian Berger, and Nassar Moham-
631 madiha. Autonomous vehicles: state of the art, future trends, and challenges. *Automotive systems*
632 *and software engineering: State of the art and future trends*, pp. 347–367, 2019.
- 633 James Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi.
634 Large language models as planning domain generators. In *34th International Conference on*
635 *Automated Planning and Scheduling*, 2024. URL [https://openreview.net/forum?id=](https://openreview.net/forum?id=C88wQIv0aJ)
636 [C88wQIv0aJ](https://openreview.net/forum?id=C88wQIv0aJ).
- 637 Jiayi Pan, Glen Chou, and Dmitry Berenson. Data-efficient learning of natural language to linear
638 temporal logic translators for robot task specification. In *2023 IEEE International Conference on*
639 *Robotics and Automation (ICRA)*, pp. 11554–11561. IEEE, 2023.
- 641 Yanyuan Qiao, Chaorui Deng, and Qi Wu. Referring expression comprehension: A survey of methods
642 and datasets. *IEEE Transactions on Multimedia*, PP:1–1, 12 2020. doi: 10.1109/TMM.2020.
643 3042066.
- 644 Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena,
645 Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified
646 text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL
647 <http://jmlr.org/papers/v21/20-074.html>.

- 648 Vasumathi Raman, Constantine Lignos, Cameron Finucane, Kenton Lee, Mitch Marcus, and Hadas
649 Kress-Gazit. Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using
650 natural language, 06 2013.
- 651 Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots
652 that use language. *Annual Review of Control, Robotics, and Autonomous Systems*,
653 3(Volume 3, 2020):25–55, 2020. ISSN 2573-5144. doi: [https://doi.org/10.1146/](https://doi.org/10.1146/annurev-control-101119-071628)
654 [annurev-control-101119-071628](https://doi.org/10.1146/annurev-control-101119-071628). URL [https://www.annualreviews.org/content/](https://www.annualreviews.org/content/journals/10.1146/annurev-control-101119-071628)
655 [journals/10.1146/annurev-control-101119-071628](https://www.annualreviews.org/content/journals/10.1146/annurev-control-101119-071628).
- 656 JG Thistle and WM Wonham. Control problems in a temporal logic framework. *International Journal*
657 *of Control*, 44(4):943–976, 1986.
- 658 Alvaro Veizaga, Mauricio Alferez, Damiano Torre, Mehrdad Sabetzadeh, and Lionel Briand. On
659 systematically building a controlled natural language for functional requirements. *Empirical*
660 *Software Engineering*, 26(4):79, 2021.
- 661 Christopher Wang, Candace Ross, Yen-Ling Kuo, Boris Katz, and Andrei Barbu. Learning a natural-
662 language to ltl executable semantic parser for grounded robotics. In Jens Kober, Fabio Ramos,
663 and Claire Tomlin (eds.), *Proceedings of the 2020 Conference on Robot Learning*, volume 155
664 of *Proceedings of Machine Learning Research*, pp. 1706–1718. PMLR, 16–18 Nov 2021. URL
665 <https://proceedings.mlr.press/v155/wang21g.html>.
- 666 David P Watson and David H Scheidt. Autonomous systems. *Johns Hopkins APL technical digest*,
667 26(4):368–376, 2005.
- 668 Yilongfei Xu, Jincao Feng, and Weikai Miao. Learning from failures: Translation of natural
669 language requirements into linear temporal logic with large language models. In *2024 IEEE 24th*
670 *International Conference on Software Quality, Reliability and Security (QRS)*, pp. 204–215. IEEE,
671 2024.
- 672 Xiang Yin, Bingzhao Gao, and Xiao Yu. Formal synthesis of controllers for safety-critical autonomous
673 systems: Developments and challenges. *Annual Reviews in Control*, 57:100940, 2024.
- 674 Junbeom Yoo, Eunkyong Jee, and Sungdeok Cha. Formal modeling and verification of safety-critical
675 software. *IEEE software*, 26(3):42–49, 2009.
- 676 Tianyi Zhang, Li Zhang, Zhaoyi Hou, Ziyu Wang, Yuling Gu, Peter Clark, Chris Callison-Burch,
677 and Niket Tandon. PROC2PDDL: Open-domain planning representations from texts. In Bhavana
678 Dalvi Mishra, Greg Durrett, Peter Jansen, Ben Lipkin, Danilo Neves Ribeiro, Lionel Wong, Xi Ye,
679 and Wenting Zhao (eds.), *Proceedings of the 2nd Workshop on Natural Language Reasoning and*
680 *Structured Explanations (@ACL 2024)*, pp. 13–24, Bangkok, Thailand, August 2024. Association
681 for Computational Linguistics. URL <https://aclanthology.org/2024.nlrse-1.2/>.
- 682
- 683
- 684
- 685
- 686
- 687
- 688
- 689
- 690
- 691
- 692
- 693
- 694
- 695
- 696
- 697
- 698
- 699
- 700
- 701

702 A APPENDIX

703 A.1 NOTATION

704 To improve readability and clarity, we summarize the main symbols and notation used throughout the
705 paper.

706 Languages, traces, and temporal logic.

- 707 • Σ : finite alphabet of observations (e.g., sets of atomic propositions that hold at a time step).
- 708 • Σ^ω : set of infinite traces over Σ (infinite sequences of observations).
- 709 • φ : LTL formulas.
- 710 • $\pi \in \mathcal{P}$: atomic proposition symbol.
- 711 • \mathcal{P} : (lifted) atomic proposition vocabulary used in LTL formulas.
- 712 • $\bigcirc, \diamond, \square, \mathcal{U}$: “next”, “eventually”, “always”, and “until” temporal operators, respectively.
- 713 • $\mathcal{M} = (S, S_0, R, L)$: Kripke structure modeling the implementation, where S is the set of
714 states, $S_0 \subseteq S$ the initial states, $R \subseteq S \times S$ the transition relation, and $L : S \rightarrow 2^{\mathcal{P}}$ the
715 labeling function.
- 716 • $\llbracket \varphi \rrbracket \subseteq \Sigma^\omega$: set of traces that satisfy the LTL formula φ .

717 System signatures and grounded atoms.

- 718 • $\mathcal{S} = \langle T, P, C \rangle$: many-sorted system signature.
- 719 • T : finite set of *type* symbols (e.g., location, agent, item).
- 720 • P : finite set of *predicate* symbols.
- 721 • C : finite set of *constant* symbols.
- 722 • $\tau_1, \dots, \tau_m \in T$: types in the type signature of a predicate.
- 723 • $p \in P$: predicate symbol with arity $\text{arity}(p)$ and type signature $(\tau_1, \dots, \tau_m) \in T^m$.
- 724 • $c \in C$: constant symbol; we also refer to constants as *arguments*.
- 725 • $\text{type}(c) \in T$: type of constant c .
- 726 • $p(c_1, \dots, c_m)$: grounded atomic proposition obtained by instantiating p with typed constants
727 $c_i \in C$ of appropriate types.
- 728 • \mathcal{P}_S : grounded atomic-proposition vocabulary induced by \mathcal{S} ,

$$729 \mathcal{P}_S = \{p \mid p \in P\} \cup \{p(c_1, \dots, c_m) \mid p \in P, c_i \in C\}.$$

730 NL-to-TL translation and grounding.

- 731 • s : natural-language (NL) specification.
- 732 • \mathbb{S} : token sequence of an NL specification.
- 733 • $f : s \rightarrow \varphi$: NL-to-TL translation function.
- 734 • $\lambda : \mathbb{S} \rightarrow \mathbb{Z}_{\geq 0}$: lifting function that maps each token to 0 (non-AP) or to an AP index.
- 735 • prop_i : i -th lifted AP appearing in the lifted NL or LTL (e.g., `prop_1`).
- 736 • k : number of lifted AP placeholders in a specification.
- 737 • g_S : grounding function

$$738 g_S : \{\text{prop}_1, \dots, \text{prop}_k\} \rightarrow \mathcal{P}_S,$$

739 which maps each lifted placeholder to a grounded atom in \mathcal{P}_S .

756 **Grounding model and prefix-based classification.**

- 757 • x_{AP} : NL span corresponding to a lifted atomic proposition (local AP context).
- 758 • x_{prefix} : tokenized prefix enumerating candidate symbols from the signature.
- 759 • $L = [\ell_1, \dots, \ell_N]$: ordered list of candidate labels (predicates or type-filtered constants).
- 760 • N : number of candidates in L .
- 761 • $L_p = \text{enum}(P)$: enumerated predicate candidate list.
- 762 • $L_c^{(r)} = \text{enum}(\{c \in C \mid \text{type}(c) = \tau_r\})$: candidate list for the r -th argument of a predicate, filtered by type.
- 763 • m : maximum prefix window (shard) size used by the classifier.
- 764 • W_j : j -th shard (window) of L ,

$$765 \quad W_j = [\ell_{(j-1)m+1}, \dots, \ell_{\min(jm, N)}], \quad j = 1, \dots, \lceil N/m \rceil.$$

- 771 • h_θ : BERT-based grounding model that maps (x_{AP}, W_j) to a discrete index $\hat{y} \in \{1, \dots, |W_j|\}$.
- 772 • ℓ^* : gold (correct) label in L for a given training instance.
- 773 • $\mathcal{L}_{\text{CE}}(\theta)$: cross-entropy loss over shard positions,

$$774 \quad \mathcal{L}_{\text{CE}}(\theta) = -\log p_\theta(y = \text{index}(\ell^* \in W_j) \mid x_{\text{AP}}, W_j).$$

- 775 • R : number of tournament rounds when $N > m$; scales as $R = O(\log_m N)$ in our analysis.

780 **A.2 COMPUTATION SCALING COMPARISON**

781 In this section, we compare the cost scaling behavior of GinSign to a generative LLM that grounds
782 by conditioning on an enumerated domain signature in the prompt. Let N denote the number
783 of candidate symbols (predicates or type-filtered constants) that must be considered for a single
784 grounding decision.

785 A prompt-based generative LLM must serialize all N candidates into a single input sequence. Because
786 transformer self-attention scales quadratically with sequence length, the incremental cost of including
787 N candidates in the prompt is at least

$$788 \quad O(N^2),$$

789 ignoring constant context terms. Thus, increasing the size of the domain signature directly incurs a
790 quadratic increase in compute and memory.

791 By contrast, GinSign uses a BERT encoder that never attends over more than a fixed window of m
792 prefix tokens at a time. When $N > m$, we shard the prefix into windows W_j of size at most m and
793 perform a tournament reduction. In each round, the N candidates are partitioned into $\lceil N/m \rceil$ shards,
794 all of which are processed in a single batched BERT forward pass. The per-round sequence length is
795 therefore bounded by $O(m)$, and the per-round cost is $O(N)$ (with m treated as a constant factor).

796 Each round reduces the candidate set by a factor of approximately m , so the number of rounds is

$$797 \quad R = O(\log_m N).$$

798 Consequently, the overall token-level complexity of GinSign for a fixed shard size m is

$$799 \quad O(N \log_m N) = O(N \log N)$$

800 that is, *near-linear* in the size of the domain signature and requiring only a logarithmic number
801 of fixed-length BERT passes. In contrast, a generative LLM must process a single, increasingly
802 long prompt whose self-attention cost grows quadratically in N , making GinSign substantially more
803 scalable for large signatures.

A.3 SYSTEM SIGNATURE PREFIX

Algorithm 1 Prefix Generation Algorithm with Optional Type

```
1: Input: Signature  $\mathcal{S} = \langle T, P, C \rangle$ , optional parameter type
2: Initialize prefix as empty list
3: if type is not provided then
4:   for each predicate  $p \in P$  do
5:     prefix.append(p)
6:   end for
7: else
8:   for each constant  $c \in C$  do
9:     if type(c) = type then
10:      prefix.append(c)
11:    end if
12:   end for
13: end if
14: Output: prefix
```

810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

864 A.4 SYSTEM SIGNATURES
865866 In this section, we provide the entire system signatures of each domain in VLTL-Bench (English
867 et al., 2025b).868 **Search and Rescue**869 **Types:**
870

- 871
- 872 • **Person:** injured_civilian, injured_hostile, injured_person,
873 injured_rescuer, injured_victim, safe_civilian,
874 safe_hostile, safe_person, safe_rescuer, safe_victim,
875 unsafe_civilian, unsafe_person, unsafe_rescuer,
876 unsafe_victim
 - 877 • **Hazard:** debris, fire_source, flood, gas_leak,
878 unstable_beam, active_debris, active_fire_source,
879 active_flood, active_gas_leak, active_unstable_beam,
880 inactive_debris, inactive_fire_source, inactive_flood,
881 inactive_gas_leak, inactive_unstable_beam, impending_debris,
882 impending_fire_source, impending_flood, impending_gas_leak,
883 impending_unstable_beam, probable_debris,
884 probable_fire_source, probable_flood, probable_gas_leak,
885 probable_unstable_beam, nearest_debris, nearest_fire_source,
886 nearest_flood, nearest_gas_leak, nearest_unstable_beam

887 **Predicates:**

- 888
- 889 • avoid(Hazard)
 - 890 • communicate(Person)
 - 891 • deliver_aid(Person)
 - 892 • get_help(Person)
 - 893 • go_home()
 - 894 • photo(Hazard)
 - 895 • record(Hazard)

897 **Traffic Light**898 **Types:**
899

- 900
- 901 • **Light:** light_north, light_south, light_east, light_west
 - 902 • **Color:** red, yellow, green
 - 903 • **Road:** (all enumerated roads, e.g., east_1st_avenue, east_1st_street, ...,
904 west_10th_street)
 - 905 • **Vehicle:** vehicle, car, bus, truck, motorcycle, motorbike,
906 bicycle
 - 907 • **Person:** person, pedestrian, jaywalker, cyclist

909 **Predicates:**

- 910
- 911 • change(Light, Color)
 - 912 • record(Event)
 - 913 • photo(Person), photo(Vehicle)
 - 914 • get_help(Person)
- 915
-
- 916
-
- 917

918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971

Warehouse

Types:

- **Item:** aeroplane, apple, backpack, banana, baseball_bat, baseball_glove, bear, bed, bench, bicycle, bird, boat, book, bottle, bowl, broccoli, bus, cake, car, carrot, cat, cell_phone, chair, clock, cow, cup, dining_table, dog, donut, elephant, fire_hydrant, fork, frisbee, giraffe, hair-drier, handbag, horse, hot_dog, keyboard, kite, knife, laptop, microwave, motorbike, mouse, orange, oven, parking_meter, person, pizza, potted_plant, refrigerator, remote, sandwich, scissors, sheep, sink, skateboard, skis, snowboard, sofa, spoon, sports_ball, stop_sign, suitcase, surfboard, teddy-bear, tennis_racket, tie, toaster, toilet, toothbrush, traffic_light, train, truck, tv_monitor, umbrella, vase, wine_glass, zebra
- **Location:** shelf, loading_dock

Predicates:

- deliver(Item, Location)
- pickup(Item)
- search(Item)
- get_help()
- idle()

A.5 HELD-OUT SYSTEM SIGNATURE ELEMENTS

Warehouse

Predicate Holdouts:

- get_help()
- deliver(Item, Location)

Argument Holdouts:

- loading_dock, apple, banana, bench, bicycle, book, bottle, bowl, bus, car, chair, cup, dog, donut, elephant, fork, frisbee, giraffe, keyboard, kite, knife, motorbike, remote

Search and Rescue

Predicate Holdouts:

- communicate(Person)
- record(Person/Threat)

Argument Holdouts:

- debris, flood, probable_flood, active_flood, gas_leak, injured_victim, injured_rescuer, safe_victim, unsafe_victim, active_fire_source, inactive_fire_source, nearest_flood, probable_debris, unstable_beam, active_gas_leak

972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025

Traffic Light

Predicate Holdouts:

- photo(Traffic_Target, Road)

Argument Holdouts:

- pedestrian, motorcycle, collision, cyclist, jaywalker, yellow
- north_1st_street, north_2nd_street, ..., north_10th_street
- south_1st_street, south_2nd_street, south_3rd_street, ..., south_6th_street
- east_1st_avenue, east_2nd_avenue, east_3rd_avenue, east_4th_avenue, east_5th_avenue
- west_1st_avenue, west_2nd_avenue, west_3rd_avenue, west_4th_avenue, west_5th_avenue, west_6th_avenue
- northeast_1st_street, northeast_2nd_street
- northwest_1st_street, northwest_2nd_street
- southeast_1st_street, southeast_2nd_street
- southwest_1st_street, southwest_2nd_street
- north_1st_avenue, north_2nd_avenue
- south_1st_avenue, south_2nd_avenue

A.6 LLM GROUNDING PROMPT

```
Scenario Configuration: {each scenario configuration given in Appendix A.4.}
Sentence: {sentence}
Lifted Sentence: {lifted_sentence}

Return a dictionary of the types, predicates, and constants for each prop_n in the lifted sentence.
The dictionary should be in this form:
prop_dict: {
  "prop_1": {
    "action_canon": *string*,
    "args_canon": *list of strings*,
  },
  "prop_2": {
    "action_canon": *string*,
    "args_canon": *list of strings*,
  }
}
Now, predict:
prop_dict:
```

1026 A.7 GINSIGN BERT GROUNDER HYPERPARAMETERS
1027

1028 We use the `bert-base-uncased` checkpoint hosted on HuggingFace (Devlin et al., 2018). We
1029 apply the following training protocol and hyperparameters to all (Predicate-only, Argument-only, and
1030 Joint) grounding models used in our evaluation:

- 1031 • Learning Rate: $5e^{-5}$
- 1032 • Epochs: 3
- 1033 • Batch Size: 16
- 1034 • Weight Decay: 0.01
- 1035 • Early Stopping Threshold: $1e - 6$
- 1036 • Early Stopping Patience: 3
- 1037 • Shard size $m = 20$

1038 Our training code is available in the supplementary materials, and will be made publicly available
1039 upon acceptance. The shard size m could be optimized experimentally, but we find that $m = 20$ is
1040 large enough to capture all predicate prefixes, and requires at most 5 tournaments in the case of the
1041 largest constant set (Traffic Light street names, see A.4).

1042 A.8 LARGE LANGUAGE MODEL DISCLOSURE
1043

1044 During the preparation of this paper, the authors employed large language models (LLMs) as assistive
1045 tools for limited tasks including proof-reading, text summarization, and the discovery of related work.
1046 All substantive research contributions, analyses, and claims presented in this paper were conceived,
1047 developed, and verified by the authors. The authors maintain full ownership and responsibility for the
1048 content of the paper, including its technical correctness, originality, and scholarly contributions.

1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079