GinSign: Grounding Natural Language Into System Signatures for Temporal Logic Translation

Anonymous ACL submission

Abstract

Natural language (NL) to temporal logic (TL) translation lets engineers specify and verify system behaviors without hand-crafting formal specifications. Existing frameworks lift NL into syntactically valid TL but leave atomic propositions abstract—not *grounded* to the concrete predicates and typed objects a system exposes (e.g., binding "find the bookbag" to `search(backpack)`). Ungrounded formulas cannot be checked over traces, yet current systems either omit grounding or ground inaccurately. We propose GinSign, a modular stage for Grounding Natural Language Into System Signatures that attaches to any NL-to-TL lifter. We cast grounding as signature-conditioned classification: enumerating the signature as an input prefix lets one encoder-only model assign each abstract AP a predicate and its typed arguments, generalize to unseen signature elements, and transfer across domains without retraining, while never hallucinating out-of-signature atoms. GinSign attains at least 99% joint grounding accuracy across seven domains; paired with a lifter, it produces executable specifications that track the oracle-grounding upper bound, improving grounded logical equivalence by 75% over the strongest competing grounder.

1 Introduction

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

Despite recent gains from sequence-to-sequence (seq2seq) models (Hsiung et al., 2022; Hahn et al., 2022; Pan et al., 2023) and large language models (LLMs) (Cosler et al., 2023; Fuggitti and Chakraborti, 2023; Chen et al., 2023; Xu et al., 2024), most NL-to-TL pipelines remain difficult to deploy in real-world systems. While modern lifting approaches often produce syntactically valid TL specifications, the atomic propositions (APs) are typically left undefined. Many state-of-the-art methods explicitly assume access to ground-truth atom groundings (Hsiung et al., 2022; Chen et al., 2023). However, without grounding APs in the target system and environment, the resulting TL formulas cannot be interpreted over traces or state machines, preventing deployment. Lang2LTL (Liu et al., 2023) addresses grounding by using embedding similarity to align lifted APs with semantic maps, but undershoots state-of-the-art lifted translation accuracy by more than 20%, highlighting the difficulty of jointly achieving accurate lifting and precise grounding.

To illustrate this challenge, consider the example in the top half of Figure 1. Existing systems may translate "The robot must find the bookbag and then deliver it to shipping" into $\Diamond(prop_1 \wedge \Diamond(prop_2))$, which is syntactically valid but semantically meaningless because the APs are not grounded in the system predicates and environment. As a result, the formula cannot be interpreted operationally on

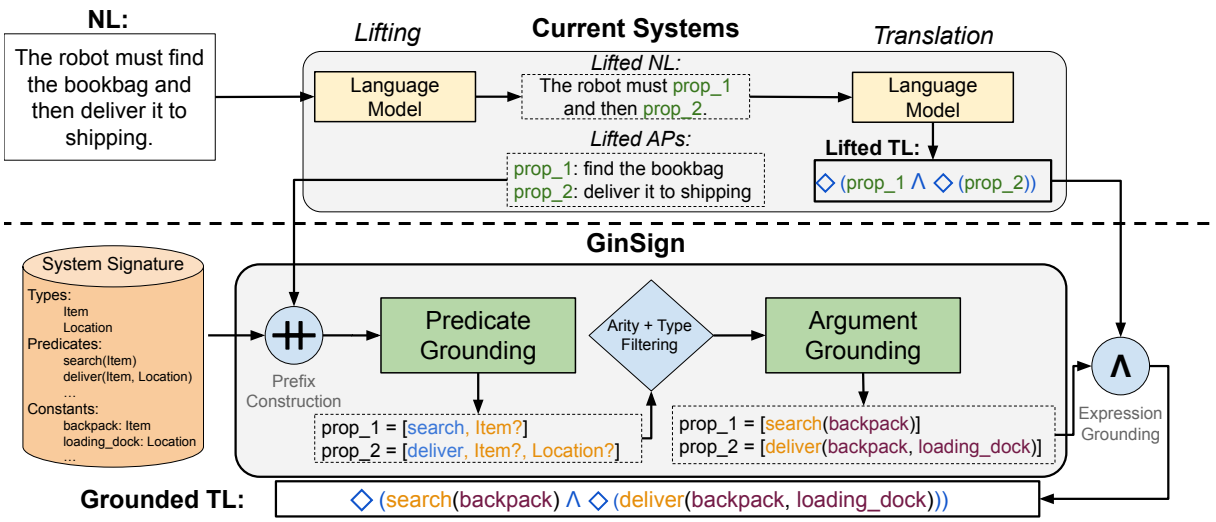

Figure 1: Overview of the GinSign framework.

traces or state machines. To address this limitation, we ground all APs during NL-to-TL translation, as illustrated in the bottom of Figure 1: GinSign maps $prop_1$ to `search(backpack)` and $prop_2$ to `deliver(backpack, loading_dock)`, yielding a fully grounded formula (Figure 1, bottom) that can now be evaluated over system traces. Note that the grounded constants (`backpack`, `loading_dock`) need not match the NL surface forms ("bookbag", "shipping"), which is precisely why grounding cannot be reduced to string matching.

In this paper, we propose GinSign, a framework for **G**rounding Natural Language **In**to System **Sign**atures for Temporal Logic. GinSign is a drop-in grounding stage for existing NL-to-TL pipelines: an upstream lifter infers the logical structure (temporal operators, boolean connectives) to produce an abstract formula, and GinSign then grounds each atomic proposition to the predicates and typed constants of a given system signature, turning that formula into an executable specification. This modularity lets GinSign improve the deployability of any lifting frontend without retraining it. Our main contributions are as follows:

- We cast AP grounding as a multi-step classification problem: given a lifted AP and a system signature (possibly with unseen classes), classify the NL span into a predicate and its typed arguments drawn from the signature.
- We solve this with a hierarchical framework that first grounds the predicate, then its typed arguments, using a single encoder-only model. Rather than fixing a per-domain label space, we enumerate the signature as an input prefix and train the classifier *jointly* on both subtasks, so the same weights transfer to unseen predicates, arguments, and even domains by swapping the prefix.
- GinSign attains $\geq$99% joint grounding accuracy across all seven domains; composed with an off-the-shelf lifter, it yields end-to-end specifications that track the oracle-grounding upper bound and are executable on real traces, improving grounded logical equivalence by 75% over the strongest competing grounder.

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 framework. Section 5 reports our experimental evaluation. Section 6 concludes this work with a discussion of limitations and directions for future work.

2 Background

2.1 Linear Temporal Logic

The syntax of LTL is given by the following grammar:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

where each atomic proposition belongs to a known set $\pi \in \mathcal{P}$ and $\Sigma = 2^{\mathcal{P}}$ is the finite alphabet of observations. LTL semantics are defined inductively over infinite traces $\sigma \in \Sigma^{\omega}$; we give the full definition, together with the Kripke structure

$\mathcal{M} = (S, S_0, R, \mathcal{K})$ used for verification, in Appendix A.1. Because every AP in φ is interpreted via the labeling function $\mathcal{K} : S \to 2^{\mathcal{P}}$, *grounding* the APs is a prerequisite to model checking: the formula becomes semantically testable only once its APs are linked to concrete predicates over system states (Hsiung et al., 2022).

2.2 System Signatures

Planners and automated systems are typically grounded in a *planning domain definition language* (Ghallab et al., 1998) or similar domain-specific library (Zhang et al., 2024; Oswald et al., 2024), which realize many-sorted logical systems. This motivates using a many-sorted system signature $\mathcal{S} = \langle T, P, C \rangle$ of *type*, *predicate*, and *constant* symbols (Finkbeiner and Zarba, 2006) as the formal vocabulary tying a grounded TL specification to the structure it specifies. A predicate $p(t_1, \ldots, t_m)$ binds typed constant arguments into well-formed atoms such as `deliver(apple, loading_dock)`; we expand on each component in Appendix A.1.

2.3 NL-to-TL Translation

Following prior work, NL-to-TL translation is divided into three phases: *lifting*, *translation*, and *grounding*, which we illustrate with "`Eventually pick up the package from room A.`"

Lifting identifies which NL tokens refer to atomic propositions via a function $\lambda : \{1, \ldots, |\mathbb{S}|\} \to \mathbb{Z}_{\geq 0}$ that maps each token of a sequence $\mathbb{S}$ to 0 (not part of any AP, including operators and connectives) or to $n > 0$ (the n-th AP). Here, "`pick up the package from room A`" becomes a single placeholder, yielding "`Eventually` $\texttt{prop}_1$`.`"

Translation maps the lifted specification s to a TL formula $f : s \to \varphi$ built from the AP set $\mathcal{P}$ and the boolean and temporal operators ($\Diamond, \Box, \bigcirc, \mathcal{U}$), giving $\varphi = \Diamond\, \texttt{prop}_1$.

Grounding resolves each placeholder to a concrete atom. For a lifted specification with k placeholders, grounding is the total function

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

which assigns to every placeholder either (i) a nullary predicate $p \in P$ or (ii) a fully instantiated atom $p(c_1, \ldots, c_m)$ whose arguments c_i are constants of the appropriate type. The image $\mathcal{P}_{\mathcal{S}}$ is the *grounded* AP vocabulary permitted by $\mathcal{S}$; hereafter we refer to constants as *arguments*. For $\mathcal{S} = \langle \{\texttt{room}, \texttt{object}\}, \{\texttt{pick_up(obj, room)}\}, \{\texttt{roomA}, \texttt{package1}\} \rangle$, grounding resolves $\texttt{prop}_1 \mapsto$ `pick_up(package1, roomA)`, giving $\varphi = \Diamond$ `pick_up(package1, roomA)`.

3 Related Work

Neural NL-to-TL methods either translate in one shot via LLM prompting or decompose the task into lifting, translation, and verification stages (Table 1). We organize our discussion around the grounding capability that distinguishes our work.

Methods without grounding. NL2LTL (Fuggitti and Chakraborti, 2023) provides a few-shot, template-based prompting package requiring user-supplied LTL templates and examples. NL2TL (Chen et al., 2023) decomposes translation into GPT-based lifting and seq2seq translation. GraFT (English et al., 2025a) applies grammar-constrained decoding to guarantee syntactically valid outputs. These methods, however, produce only lifted formulas whose APs are never grounded, so their output cannot be verified on traces or state machines.

Methods with grounding. Lang2LTL (Liu et al., 2023) is the closest prior work that grounds APs, separating LLM-based referring-expression recognition from grounding by embedding similarity; however, it presumes proposition embeddings from a pre-built semantic map and does not extend to arbitrary system signatures. Work on *visual grounding* instead resolves perceptual references in scenes (Qiao et al., 2020), rather than the formal symbolic grounding we address. In contrast, GinSign grounds APs against *arbitrary* system signatures and, as we show, attains precise grounding across diverse domains.

4 Methodology

In this section, we introduce GinSign, a *grounding* module that augments any NL-to-TL lifting-

Framework	Lifting	Translation	Grounding
LLM-Baseline	×	✓	×
NL2LTL	×	✓	×
NL2TL	✓	✓	×
GraFT	✓	✓	×
Lang2LTL	✓	✓	✓
GinSign (ours)	✓	✓	✓

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

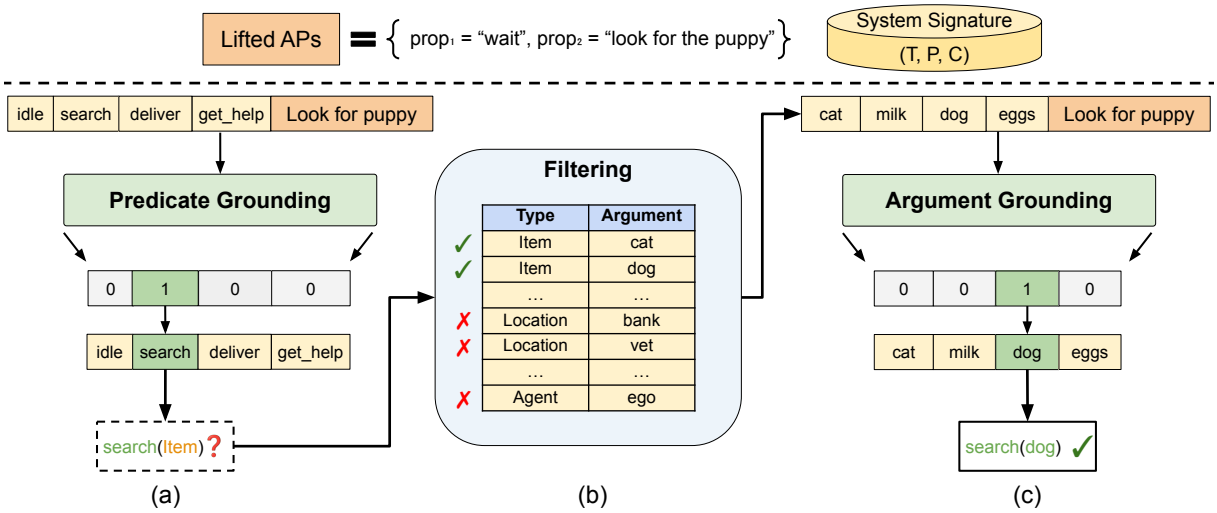

Figure 2: An overview of our grounding components. Given n lifted NP APs, we convert the system signature into a prefix using Algorithm 1. The lifted NL is first combined with the predicates prefix to ground the predicate to a known action (a). Since each predicate requires an argument, we filter out 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 encoder model, which processes any prefix and lifted NL.

and-translation framework—such as LLM prompting, NL2TL (Chen et al., 2023), or GraFT (English et al., 2025a)—to produce grounded, model-checkable specifications. GinSign accepts two input components: a system signature, and a natural language specification. Following the standard compositional decomposition into lifting, translation, and grounding, we treat lifting and translation as provided by the upstream backbone and focus on grounding the APs it produces into the state space defined by the signature. The grounded APs are then inserted into the temporal-logic formula produced by the backbone's translation stage.

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 single encoder-only 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. Next, we provide the details of the predicate grounding, the argument grounding, and the grounding model.

Hierarchical strategy: We design a hierarchical grounding strategy because of the logic 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 $p(c_1, \ldots, c_m)$, thereby completing $g_{\mathcal{S}}$ (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 $\Theta\big(\prod_r |C_{\tau_r}|\big)$ to $\Theta\big(|P| + \sum_r |C_{\tau_r}|\big)$, a dramatic reduction when $|C| \gg |P|$ and types partition C evenly. Moreover, type filtering eliminates invalid atoms by construction, ensuring any

predicted $p(c_1, \ldots, c_m)$ lies in $\mathcal{P}_{\mathcal{S}}$. Coupled with windowed classification (Section 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 encoder 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 (Table 5) that this design yields promising out-of-distribution accuracy on unseen signature elements within a domain 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 encoder 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 $g_{\mathcal{S}}$ as a classifier over an input-defined label set. For each lifted placeholder $\texttt{prop}_i$, 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 $\mathrm{enum}(\cdot)$ produce a fixed-order list of symbols as a token sequence. For predicate grounding, the candidate list is $L_p = \mathrm{enum}(P)$. For argument grounding, once a predicate $\hat{p} \in P$ with arity a and type signature $(\tau_1, \ldots, \tau_a) \in T^a$ is predicted, the r-th argument uses the type-filtered set,

$$L_c^{(r)} \;=\; \mathrm{enum}\big(\{\, c \in C \mid \mathrm{type}(c) = \tau_r \,\}\big).$$

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

$$W_j \;=\; [\,\ell_{(j-1)m+1}, \ldots, \ell_{\min(jm,N)}\,]$$

$$j = 1, \ldots, \lceil N/m \rceil.$$

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

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

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 $\ell^\star \in L$, we construct one or more

gold-in shards W_j such that $\ell^\star \in W_j$ (contiguous windows). The supervision is a single-label CrossEntropy over the shard positions; when multiple gold-in shards exist for a single instance, the loss is averaged over all such shards:

$$\mathcal{L}_{\text{CE}}(\theta) = \frac{1}{|\mathcal{J}^\star|} \sum_{j \in \mathcal{J}^\star} \Big[-\log\, p_\theta\big(y = \text{index}(\ell^\star \in W_j) \mid x_{\text{AP}}, W_j\big) \Big],$$

where $\mathcal{J}^\star = \{j \mid \ell^\star \in W_j\}$ is the set of shard indices containing the gold label. Further implementation details can be found in Appendix A.8. 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. Section 5.1 discusses training and evaluation corpora information. The results of our isolated grounding evaluation are presented in Section 5.2. Finally, end-to-end translation results are presented in Section 5.3.

5.1 Datasets and Metrics

Datasets We evaluate on seven domains spanning two benchmark families. From VLTL-Bench (English et al., 2025b) we use the three grounded domains; Search and Rescue, Traffic Light, and Warehouse, each providing natural language specifications with fully grounded LTL formulas over an explicit system signature. Table 6 in Appendix A.5 reports the size of each domain signature: its number of types, predicates, and constant arguments. We additionally evaluate on four prior-work datasets: Navigation (Wang et al., 2021), Cleanup World (MacGlashan et al., 2015), GLTL (Gopalan et al., 2018), and Conformal (Wang et al., 2025). While these datasets were originally released without system signatures, we construct signatures for each by extracting the predicate–argument structure from the ground-truth grounded formulas, enabling grounded evaluation on all seven domains.

Metrics For the isolated grounding evaluation (Table 2), we report per-AP *joint accuracy*: the fraction of atomic propositions for which the predicted predicate and all arguments exactly match the ground truth. For the end-to-end evaluation (Table 3), we report **GLE** (Grounded Logical Equivalence). A prediction is GLE-correct when (i) its masked temporal structure matches the ground-truth lifted formula, and (ii) every atomic proposition is correctly grounded in the system signature. Condition (i) depends only on the lifting and translation method (and corresponds to the *Upper Bound* row in Table 3); condition (ii) isolates the grounding method's contribution. We evaluate grounding correctness by comparing predicted (predicate, arguments) tuples against the ground truth, which avoids dependence on domain-specific formula notation conventions.

5.2 Grounding Evaluation

We evaluate grounding in isolation: each method receives the ground-truth lifted APs and system signature and must predict the correct predicate and all arguments for every atomic proposition. We report per-AP joint accuracy (predicate and all arguments exactly correct) across all seven domains.

LLM Prompting. Table 2 reports joint grounding accuracy for every baseline. Frontier LLMs ground reliably on the VLTL-Bench domains—the strongest, GPT-oss-120B, reaches 98.8% on Traffic Light and 97.9% on Search & Rescue—but the GPT grounders collapse on the prior-work domains: GPT-oss-120B falls to 13.2% (Cleanup World) and 15.6% (GLTL), and GPT-4o to just 4.6% (Cleanup World). Claude Sonnet is more robust on these (71.0% and 89.7%) yet weakest on the VLTL-Bench domains, so no single LLM grounds reliably across the board. Warehouse is hard for every LLM (44–72%). The failure mode is consistent: despite being given the signature, the models hallucinate predicate names from the input text (e.g., "move_to", "enter") instead of selecting from it—precisely the error that GinSign 's signature-restricted classification rules out by construction.

Lang2LTL. Embedding-based cosine similarity performs reliably when the signature is small: Lang2LTL achieves 100% on Cleanup World and GLTL, where the enumeration of all grounded atoms is compact enough for nearest-neighbor retrieval to be unambiguous. On domains with larger argument spaces, however, performance drops from 75.5% on Warehouse and 72.3% on Navi, as lexically similar constants become difficult to disambiguate by embedding distance alone.

GinSign. Our method achieves $\geq$99% joint ac-

Table 2: Isolated grounding evaluation. We report per-AP joint accuracy (%): the fraction of atomic propositions whose predicate and all arguments are predicted correctly, when each method is given the ground-truth lifted APs and the system signature. Best per column in bold; the prompt used by all LLMs is given in Appendix A.7.

Grounder	TL	S&R	WH	CW	CF	GLTL	Navi
GPT-4o	83.4	94.3	44.6	4.6	72.8	13.2	94.6
GPT-oss-120B	98.8	97.9	72.0	13.2	55.4	15.6	93.2
Claude Sonnet	63.7	87.7	44.9	71.0	73.6	89.7	92.4
Lang2LTL	85.0	88.5	75.5	**100.0**	97.8	**100.0**	72.3
GinSign (ours)	**100.0**	**100.0**	**99.3**	**100.0**	**100.0**	**100.0**	**99.0**

Table 3: Grounded Logical Equivalence (GLE, %) across lifted translation frameworks and grounding methods. GLE requires that the predicted formula is both structurally correct and that every atomic proposition is correctly grounded in the system signature. Upper Bound uses ground-truth AP instances.

Lift + Translate	Grounding	TL	S&R	WH	CW	CF	GLTL	Navi
LLM Prompting	GPT-4o	29.0	44.4	10.4	0.0	4.8	0.0	1.6
	GPT-oss-120B	42.2	45.8	21.5	0.2	4.3	0.0	**2.3**
	Claude Sonnet	16.3	38.3	9.5	5.0	**5.3**	10.1	2.1
	Lang2LTL	31.3	38.1	24.6	**23.0**	4.8	**10.7**	1.1
	GinSign (ours)	**43.0**	**48.1**	**42.7**	**23.0**	**5.3**	**10.7**	**2.3**
	Upper Bound	43.0	48.1	42.9	23.0	5.3	10.7	2.8
NL2TL	GPT-4o	56.2	66.2	11.6	1.0	11.0	2.0	55.2
	GPT-oss-120B	77.4	73.0	33.8	5.8	9.0	6.2	76.4
	Claude Sonnet	34.4	58.2	11.8	30.0	13.7	34.4	73.8
	Lang2LTL	56.8	57.6	35.8	**35.4**	21.3	**37.6**	42.4
	GinSign (ours)	**79.6**	**73.6**	**65.6**	**35.4**	**23.3**	**37.6**	**85.8**
	Upper Bound	79.6	73.6	66.6	35.4	23.3	37.6	87.2
GraFT	GPT-4o	71.0	90.0	22.6	2.0	24.7	4.4	61.6
	GPT-oss-120B	97.6	96.2	53.8	9.8	21.0	10.8	85.0
	Claude Sonnet	44.6	79.4	23.2	59.8	30.7	84.4	82.2
	Lang2LTL	71.4	77.8	56.2	**99.8**	57.0	**99.6**	48.6
	GinSign (ours)	**100.0**	**100.0**	**98.6**	**99.8**	**62.3**	**99.6**	**95.0**
	Upper Bound	100.0	100.0	100.0	99.8	62.3	99.6	97.0

curacy across all seven domains. The hierarchical decomposition eliminates the failure modes of both LLM prompting (predicate hallucination) and embedding similarity (argument confusion in large constant sets). Because the predicate prefix redefines the label space at inference time, the same model generalizes across domains with no retraining, producing the consistent performance visible in the prior-work columns.

5.3 End-to-End Translation Evaluation

We run a full factorial over lift+translate backbones and grounding modules. Three backbones—LLM prompting, NL2TL (Chen et al., 2023), and GraFT (English et al., 2025a)—are paired with five grounders: three LLM-prompting grounders (GPT-4o, GPT-oss-120B, Claude Sonnet), the embedding-similarity grounder of Lang2LTL (Liu et al., 2023), and our GinSign grounder. Each pair is scored end-to-end across seven domains (Table 3) under **Grounded Logical Equivalence (GLE)**, which requires both correct temporal/lifted-AP structure and correct grounding of every AP, yielding a model-checkable output. GinSign additionally receives the system signature, which defines the target state space and is a prerequisite for grounded output. Per backbone we report an oracle-grounding **Upper Bound**: the GLE ceiling fixed by the backbone's lifted-formula accuracy.

GinSign tracks this ceiling almost exactly. Under

Table 4: Leave-one-out grounding evaluation: joint accuracy (%) on the held-out domain. Each row trains on all domains within the family except the listed domain, then evaluates on that held-out domain.

Held-out domain	Joint Acc.
Cleanup World	100.0
Conformal	24.3
GLTL	100.0
Navi	33.5
Traffic Light	38.6
Search & Rescue	78.8
Warehouse	36.3

GraFT it equals the bound in five of seven domains and lands within two points in the rest (WH 98.6 vs. 100.0; Navi 95.0 vs. 97.0), with the same saturation under NL2TL and LLM prompting. It is the best or tied-best grounder in all 3×7 cells, with the largest margins in lexically diverse domains—on Warehouse under GraFT it reaches 98.6 versus 56.2 for the next best (Lang2LTL), a 75% gain. Where embedding similarity already saturates (CW, GLTL), GinSign simply matches it, while LLM-prompting grounders are erratic (e.g., GPT-oss-120B drops to 9.8 on CW). Grounding is therefore no longer the bottleneck: with GinSign , end-to-end error is governed by lifting+translation, and pairing GinSign with the strongest backbone yields the best specifications overall.

5.4 Grounding Ablation

We probe generalization with a leave-one-out study (Table 4): for each domain we train on the rest of its family with full signatures and evaluate on the held-out domain, so every score is strictly out-of-domain. Transfer is strong where the held-out vocabulary overlaps structurally with training (Cleanup World and GLTL at 100.0, Search & Rescue at 78.8) but weak where it is lexically and structurally distinct (Conformal 24.3, Navi 33.5, Warehouse 36.3). Cross-domain transfer to genuinely novel vocabularies thus remains an open challenge.

Held-out signature elements. We further test whether the jointly-trained design generalizes to signature elements unseen during training. Within each domain we hold out a subset of predicates and arguments (Appendix A.6) for evaluation, comparing our single joint model against models trained on only predicate or only argument grounding (Table 5). The joint model matches or exceeds the specialized models on every domain and metric: predicate accuracy on Traffic Light improves from 75.0 to 83.1, and argument F_1 on Search & Rescue from 88.4 to 96.2, showing that sharing one encoder across both subtasks costs no accuracy.

Table 5: Intra-domain held-out generalization. Predicate and argument grounding accuracy (Acc) and F_1 (%) on signature elements held out from training (Appendix A.6), comparing our single model (Joint) against paired task-specialized models (Pred-Only, Arg-Only).

		Traffic Light		Search & Rescue		Warehouse	
Model	Task	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 Section 4.1.

6 Conclusion

We introduced GinSign, a modular grounding stage that anchors every atomic proposition to a system signature and, composed with any lifting-and-translation backbone, yields end-to-end grounded NL-to-TL translation. Casting grounding as an open-set, hierarchical span-classification task cleanly separates syntactic translation from semantic anchoring, and adding the signature prefix essentially solves both predicate and argument grounding, matching or exceeding far larger language models. Crucially, explicit grounding makes specifications model-checkable, exposing semantic errors that string-based metrics miss and pushing NL-to-TL translation from plausible output toward verifiable specifications.

Limitations

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.

Ethical Considerations

This paper presents work whose goal is to advance the field of Machine Learning, specifically in the area of formal verification and autonomous systems. The ability to automatically translate natural language specifications into temporal logic has potential societal benefits in making formal verification more accessible and enabling safer autonomous systems. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Houssam Abbas, Georgios Fainekos, Sriram Sankaranarayanan, Franjo Ivančić, and Aarti Gupta. 2013. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(2s):1–30.

Rajeev Alur. 2015. *Principles of Cyber-Physical Systems*. The MIT Press.

Pierfrancesco Bellini, Riccardo Mattolini, and Paolo Nesi. 2000. Temporal logics for real-time system specification. *ACM Computing Surveys (CSUR)*, 32(1):12–42.

Jonathan Bowen and Victoria Stavridou. 1993. Safety-critical systems, formal methods and standards. *Software engineering journal*, 8(4):189–209.

Rafael C Cardoso, Georgios Kourtis, Louise A Dennis, Clare Dixon, Marie Farrell, Michael Fisher, and Matt Webster. 2021. A review of verification and validation for space autonomous systems. *Current Robotics Reports*, 2(3):273–283.

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

Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. 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*, page 383–396, Berlin, Heidelberg. Springer-Verlag.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

William H English, Dominic Simon, Sumit Kumar Jha, and Rickard Ewetz. 2025a. Grammar-forced translation of natural language to temporal logic using LLMs. In *Forty-second International Conference on Machine Learning*.

William H English, Chase Walker, Dominic Simon, Sumit Kumar Jha, and Rickard Ewetz. 2025b. Verifiable natural language to linear temporal logic translation: A benchmark dataset and evaluation suite. *Preprint*, arXiv:2507.00877.

Bernd Finkbeiner and Calogero G. Zarba. 2006. Many-sorted logic. In *Decision Procedures for Verification*, chapter 1. Lecture notes, Reactive Systems Group, CISPA.

Francesco Fuggitti and Tathagata Chakraborti. 2023. Nl2ltl - a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas. In *AAAI Conference on Artificial Intelligence*.

M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. 1998. PDDL—The Planning Domain Definition Language.

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

Christopher Hahn, Frederik Schmitt, Julia J Tillman, Niklas Metzger, Julian Siber, and Bernd Finkbeiner. 2022. Formal specifications from natural language. *arXiv preprint arXiv:2206.01962*.

Suman Harapanahalli, Niall O Mahony, Gustavo Velasco Hernandez, Sean Campbell, Daniel Riordan, and Joseph Walsh. 2019. Autonomous navigation of mobile robots in factory environment. *Procedia Manufacturing*, 38:1524–1531.

Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. 2018. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer*, 20(1):79–93.

Eric Hsiung, Hiloni Mehta, Junchi Chu, Xinyu Liu, Roma Patel, Stefanie Tellex, and George Konidaris. 2022. Generalizing to new domains by mapping natural language to lifted ltl. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3624–3630. IEEE.

Savas Konur. 2013. A survey on temporal logics for specifying and verifying real-time systems. *Frontiers of Computer Science*, 7(3):370.

Mohammed Lafi, Bilal Hawashin, and Shadi AlZu'bi. 2021. Eliciting requirements from stakeholders' responses using natural language processing. *Computer Modeling In Engineering & Sciences*, 127(1):99–116.

Carl Lamar. 2009. Linguistic analysis of natural language engineering requirements. Master's thesis, Clemson University.

Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. 2023. Grounding complex natural language commands for temporal tasks in unseen environments. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 1084–1110. PMLR.

James MacGlashan, Monica Babes-Vroman, Marie desJardins, Michael L. Littman, Smaranda Muresan, S. Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. 2015. Grounding english commands to reward functions. In *Robotics: Science and Systems*.

Piergiuseppe Mallozzi, Patrizio Pelliccione, Alessia Knauss, Christian Berger, and Nassar Mohammadiha. 2019. Autonomous vehicles: state of the art, future trends, and challenges. *Automotive systems and software engineering: State of the art and future trends*, pages 347–367.

James Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. 2024. Large language models as planning domain generators. In *34th International Conference on Automated Planning and Scheduling*.

Jiayi Pan, Glen Chou, and Dmitry Berenson. 2023. Data-efficient learning of natural language to linear temporal logic translators for robot task specification. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11554–11561. IEEE.

Yanyuan Qiao, Chaorui Deng, and Qi Wu. 2020. Referring expression comprehension: A survey of methods and datasets. *IEEE Transactions on Multimedia*, PP:1–1.

Vasumathi Raman, Constantine Lignos, Cameron Finucane, Kenton Lee, Mitch Marcus, and Hadas Kress-Gazit. 2013. Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language.

Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. 2020. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(Volume 3, 2020):25–55.

JG Thistle and WM Wonham. 1986. Control problems in a temporal logic framework. *International Journal of Control*, 44(4):943–976.

Alvaro Veizaga, Mauricio Alferez, Damiano Torre, Mehrdad Sabetzadeh, and Lionel Briand. 2021. On systematically building a controlled natural language for functional requirements. *Empirical Software Engineering*, 26(4):79.

Christopher Wang, Candace Ross, Yen-Ling Kuo, Boris Katz, and Andrei Barbu. 2021. Learning a natural-language to ltl executable semantic parser for grounded robotics. In *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 1706–1718. PMLR.

Jun Wang, David Smith Sundarsingh, Jyotirmoy V. Deshmukh, and Yiannis Kantaros. 2025. Conformalnl2ltl: Translating natural language instructions into temporal logic formulas with conformal correctness guarantees. *Preprint*, arXiv:2504.21022.

David P Watson and David H Scheidt. 2005. Autonomous systems. *Johns Hopkins APL technical digest*, 26(4):368–376.

Yilongfei Xu, Jincao Feng, and Weikai Miao. 2024. 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)*, pages 204–215. IEEE.

Xiang Yin, Bingzhao Gao, and Xiao Yu. 2024. Formal synthesis of controllers for safety-critical autonomous systems: Developments and challenges. *Annual Reviews in Control*, 57:100940.

Junbeom Yoo, Eunkyoung Jee, and Sungdeok Cha. 2009. Formal modeling and verification of safety-critical software. *IEEE software*, 26(3):42–49.

Tianyi Zhang, Li Zhang, Zhaoyi Hou, Ziyu Wang, Yuling Gu, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024. PROC2PDDL: Open-domain planning representations from texts. In *Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ACL 2024)*, pages 13–24, Bangkok, Thailand. Association for Computational Linguistics.

A Appendix

A.1 Extended Preliminaries

This section provides the expanded preliminaries condensed in Section 2.

LTL semantics. Recall the LTL syntax introduced in Section 2, where each atomic proposition belongs to a set of known symbols $\pi \in \mathcal{P}$ and $\Sigma = 2^{\mathcal{P}}$ is the finite alphabet of observations. The semantics of LTL are defined inductively over infinite traces $\sigma = \sigma_0\sigma_1\sigma_2 \cdots \in \Sigma^\omega$. We write $\sigma, i \models \varphi$ to denote that φ holds at position i of σ:

$$
\begin{aligned}
&\sigma, i \models \pi && \text{iff } \pi \in \sigma_i \\
&\sigma, i \models \neg\varphi && \text{iff } \sigma, i \not\models \varphi \\
&\sigma, i \models \varphi_1 \wedge \varphi_2 && \text{iff } \sigma, i \models \varphi_1 \text{ and } \sigma, i \models \varphi_2 \\
&\sigma, i \models \bigcirc\varphi && \text{iff } \sigma, i{+}1 \models \varphi \\
&\sigma, i \models \varphi_1 \mathcal{U} \varphi_2 && \text{iff } \exists\, j \geq i.\ \sigma, j \models \varphi_2 \\
& && \quad \text{and } \forall\, i \leq k < j.\ \sigma, k \models \varphi_1
\end{aligned}
$$

The derived operators are $\Diamond\varphi \equiv \mathit{true}\ \mathcal{U}\ \varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. A trace σ satisfies φ, written $\sigma \models \varphi$, iff $\sigma, 0 \models \varphi$, and the language of φ is $[\![\varphi]\!] = \{\sigma \in \Sigma^\omega \mid \sigma \models \varphi\}$. To verify a real system against an LTL requirement, one typically models the implementation as a finite-state Kripke structure $\mathcal{M} = (S, S_0, R, \mathcal{K})$, where S is the state set, $S_0 \subseteq S$ the initial states, $R \subseteq S \times S$ the total transition relation, and $\mathcal{K} : S \to 2^{\mathcal{P}}$ the labeling function. Because every atomic proposition in φ is interpreted via $\mathcal{K}$, the syntactic formula only becomes semantically testable once its APs are linked to concrete predicates over system states (Hsiung et al., 2022).

System signatures. Each component of $\mathcal{S} = \langle T, P, C\rangle$ 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: $p(t_1, \ldots, t_m)$ 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. System signatures describe all of the non-logical terminals in a formal language (Finkbeiner and Zarba, 2006).

A.2 Notation

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

Languages, traces, and temporal logic.

- Σ : finite alphabet of observations (e.g., sets of atomic propositions that hold at a time step).
- Σ^ω : set of infinite traces over Σ (infinite sequences of observations).
- φ : LTL formulas.
- $\pi \in \mathcal{P}$: atomic proposition symbol.
- $\mathcal{P}$: (lifted) atomic proposition vocabulary used in LTL formulas.
- $\bigcirc, \Diamond, \Box, \mathcal{U}$: "next", "eventually", "always", and "until" temporal operators, respectively.
- $\mathcal{M} = (S, S_0, R, \mathcal{K})$: Kripke structure modeling the implementation, where S is the set of states, $S_0 \subseteq S$ the initial states, $R \subseteq S \times S$ the transition relation, and $\mathcal{K} : S \to 2^{\mathcal{P}}$ the labeling function.
- $[\![\varphi]\!] \subseteq \Sigma^\omega$: set of traces that satisfy the LTL formula φ.

System signatures and grounded atoms.

- $\mathcal{S} = \langle T, P, C\rangle$: many-sorted system signature.
- T : finite set of *type* symbols (e.g., `location`, `agent`, `item`).
- P : finite set of *predicate* symbols.
- C : finite set of *constant* symbols.
- $\tau_1, \ldots, \tau_m \in T$: types in the type signature of a predicate.
- $p \in P$: predicate symbol with arity $\mathrm{arity}(p)$ and type signature $(\tau_1, \ldots, \tau_m) \in T^m$.
- $c \in C$: constant symbol; we also refer to constants as *arguments*.

- $\mathrm{type}(c) \in T$: type of constant c.
- $p(c_1, \ldots, c_m)$: grounded atomic proposition obtained by instantiating p with typed constants $c_i \in C$ of appropriate types.
- $\mathcal{P}_{\mathcal{S}}$: grounded atomic-proposition vocabulary induced by $\mathcal{S}$,

$$\mathcal{P}_{\mathcal{S}} = \{\, p \mid p \in P \} \cup \{\, p(c_1, \ldots, c_m) \mid p \in P,\ c_i \in C \}.$$

NL-to-TL translation and grounding.

- s : natural-language (NL) specification.
- $\mathbb{S}$: token sequence of an NL specification.
- $f : s \to \varphi$: NL-to-TL translation function.
- $\lambda : \mathbb{S} \to \mathbb{Z}_{\geq 0}$: lifting function that maps each token to 0 (non-AP) or to an AP index.
- $\mathtt{prop}_i$: i-th lifted AP appearing in the lifted NL or LTL (e.g., `prop_1`).
- k : number of lifted AP placeholders in a specification.
- $g_{\mathcal{S}}$: grounding function

$$g_{\mathcal{S}} : \{\mathtt{prop}_1, \ldots, \mathtt{prop}_k\} \to \mathcal{P}_{\mathcal{S}},$$

  which maps each lifted placeholder to a grounded atom in $\mathcal{P}_{\mathcal{S}}$.

Grounding model and prefix-based classification.

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

$$W_j = [\ell_{(j-1)m+1}, \ldots, \ell_{\min(jm,N)}]$$

$$j = 1, \ldots, \lceil N/m \rceil.$$

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

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

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

A.3 Computation Scaling Comparison

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

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

$$O(N^2),$$

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

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

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

$$R \;=\; O(\log_m N)\,.$$

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

$$O(N \log_m N) \;=\; O(N \log N)$$

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

A.4 System Signature Prefix

Algorithm 1 Prefix Generation Algorithm with Optional Type

```
Input: Signature S = ⟨T, P, C⟩, optional parameter type
Initialize prefix as empty list
if type is not provided then
    for each predicate p ∈ P do
        prefix.append(p)
    end for
else
    for each constant c ∈ C do
        if type(c) = type then
            prefix.append(c)
        end if
    end for
end if
Output: prefix
```

A.5 System Signatures

In this section, we provide the entire system signatures of each domain in VLTL-Bench (English et al., 2025b). **Search and Rescue**

Table 6: Size of each domain signature: the number of types ($|T|$), predicates ($|P|$), and constant arguments ($|C|$).

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

Types:

- Person: `injured_civilian`, `injured_hostile`, `injured_person`, `injured_rescuer`, `injured_victim`, `safe_civilian`, `safe_hostile`, `safe_person`, `safe_rescuer`, `safe_victim`, `unsafe_civilian`, `unsafe_person`, `unsafe_rescuer`, `unsafe_victim`
- Hazard: `debris`, `fire_source`, `flood`, `gas_leak`, `unstable_beam`, `active_debris`, `active_fire_source`, `active_flood`, `active_gas_leak`, `active_unstable_beam`, `inactive_debris`, `inactive_fire_source`, `inactive_flood`, `inactive_gas_leak`, `inactive_unstable_beam`, `impending_debris`, `impending_fire_source`, `impending_flood`, `impending_gas_leak`, `impending_unstable_beam`, `probable_debris`, `probable_fire_source`, `probable_flood`, `probable_gas_leak`, `probable_unstable_beam`, `nearest_debris`, `nearest_fire_source`, `nearest_flood`, `nearest_gas_leak`, `nearest_unstable_beam`

Predicates:

- `avoid(Hazard)`
- `communicate(Person)`
- `deliver_aid(Person)`
- `get_help(Person)`
- `go_home()`
- `photo(Hazard)`
- `record(Hazard)`

Traffic Light
Types:

- Light: `light_north`, `light_south`, `light_east`, `light_west`
- Color: `red`, `yellow`, `green`
- Road: (all enumerated roads, e.g., `east_1st_avenue`, `east_1st_street`, `. . . ,` `west_10th_street`)
- Vehicle: `vehicle`, `car`, `bus`, `truck`, `motorcycle`, `motorbike`, `bicycle`

- Person: person, pedestrian, jaywalker, cyclist

Predicates:

- change(Light, Color)
- record(Event)
- photo(Person), photo(Vehicle)
- get_help(Person)

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.6 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

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.7 LLM Grounding Prompt

Scenario Configuration: {each scenario configuration given in Appendix A.5.}
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:

A.8 GinSign BERT Grounder Hyperparameters

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

- Learning Rate: $5e^{-5}$
- Epochs: 3
- Batch Size: 16
- Weight Decay: 0.01
- Early Stopping Threshold: $1e-6$
- Early Stopping Patience: 3
- Shard size $m = 20$

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

A.9 Large Language Model Disclosure

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