Ask, Reason, Assist: Decentralized Robot Collaboration via Language and Logic

Anonymous Author(s)

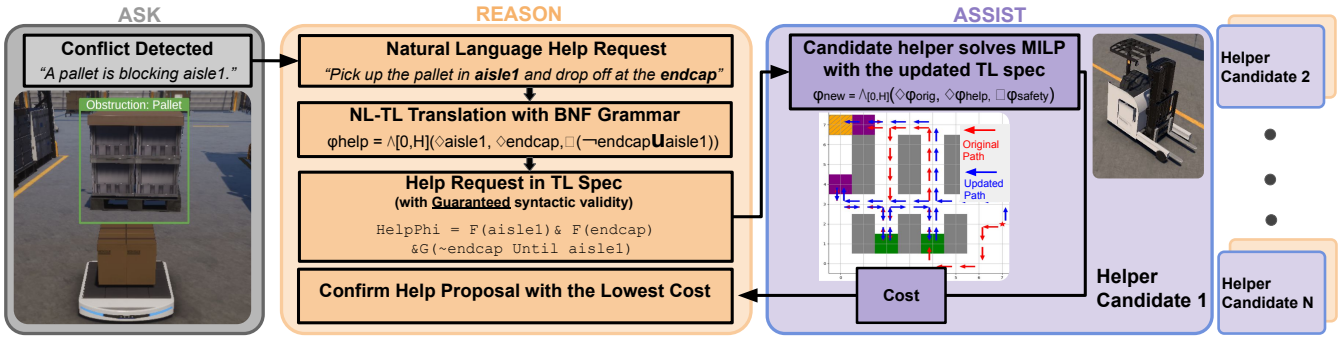

Fig. 1: **Overview of the Proposed Framework**: A requester broadcasts a natural language help request, which helpers translate into syntactically valid temporal logic (TL) via a BNF grammar. Then each helper independently solves an updated optimal path via MILP to assess the cost associated with the help task and proposes help. Finally, the requester selects and confirms the best offer that minimizes the overall impact to the system. Our framework demonstrates how natural language (NL) can serve as a flexible medium for heterogeneous multi-robot help-request forum. Using constrained generation with BNF grammar guarantees valid TL translations, while solving the decentralized MILP optimization problem achieves performance close to a centralized "Oracle" baseline.

***Abstract*— Increased robot deployment, such as in warehousing, has revealed a need for seamless collaboration among heterogeneous robot teams to resolve unforeseen conflicts. To address this challenge, we propose a novel decentralized framework that enables robots to request and provide help. The process begins when a robot detects a conflict and uses a Large Language Model (LLM) to decide whether external assistance is required. If so, it crafts and broadcasts a natural language (NL) help request. Potential helper robots reason over the request and respond with offers of assistance, including information about the effect on their ongoing tasks. Helper reasoning is implemented via an LLM grounded in Signal Temporal Logic (STL) using a Backus–Naur Form (BNF) grammar, ensuring syntactically valid NL-to-STL translations, which are then solved as a Mixed Integer Linear Program (MILP). Finally, the requester robot selects a helper by reasoning over the expected increase in system-level total task completion time. We evaluated our framework through experiments comparing different helper-selection strategies and found that considering multiple offers allows the requester to minimize added makespan. Our approach significantly outperforms heuristics such as selecting the nearest available candidate helper robot, and achieves performance comparable to a centralized "Oracle" baseline but without heavy information demands.**

I. Introduction

Modern warehouses and manufacturing sites are complex ecosystems where heterogeneous multi-robot teams must coordinate to meet demanding throughput targets. However, the dynamic nature that makes these mixed fleets effective also renders them brittle to unforeseen physical and semantic conflicts. Given the scale of modern-day robot deployments, a decentralized yet efficient conflict-resolution process is essential to unlock the full potential of these teams.

Recent advances in foundation models, particularly multi-modal Large Language Models (LLMs), provides a promising tool in addressing the first step of this process - identifying and describing the conflict [1]. Paired with an embodied agent, by processing sensor inputs and an understanding of robot capabilities, these models can generate a concise Help Request in Natural Language [2]. Although methods have been presented to use multi modal LLMs for end-to-end conflict resolution in multi-robot settings [3], the plans generated by foundation models do not provide the safety guarantees required in environments where humans and robots coexist. Furthermore, industrial robots frequently operate under strict time constraints, such as in order fulfillment scenarios where shipments must be staged at outbound docks by designated pickup times. Without guarantees on the safety and feasibility of conflict recovery plans, LLMs alone are insufficient for ensuring reliable multi-agent conflict resolution in time-sensitive and safety-critical environments.

One path forward to solving this challenge is to apply formal methods, such as the verification of Signal Temporal Logic (STL) and Linear Temporal Logic (LTL) specifications. STL and LTL tools can guarantee that a robot's plan satisfies constraints and have been successfully applied to multi-robot coordination and reconfiguration tasks [4][5]. Recent work has also explored their integration with LLMs to enforce safety constraints during task planning [6]. However, it remains open how to integrate these formal methods approaches with foundation model reasoning in a way that enables seamless, safe multi-agent coordination.

A. Related Work

1) Formal Methods for Task and Motion Planning: Formal methods, such as Temporal Logic (TL), provide a precise language for specifying robot objectives with temporal and spatial grounding. In particular, Signal Temporal Logic (STL) is well-suited to robotics as it reasons over real-valued trajectories with explicit time bounds [7]. In task and motion planning (TAMP), these requirements are often formulated as a mixed-integer program (MIP) which unifies discrete task choices and continuous motion in one certifiable solver [8]. Our framework builds directly on this paradigm but differs on *where* and *how* formal reasoning is invoked. Once an LLM translates NL to STL on demand, each potential helper independently solves a compact MILP that (i) certifies feasibility and (ii) estimates the marginal cost of adding the help task given its existing task schedule that remains to be private. This parallelized approach maintains the rigorous guarantee provided by formal methods while avoiding the need for a centralized planner and full disclosure of potentially proprietary information.

2) Natural Language to Temporal Logic: While temporal logic is a powerful formal language for expressing constraints on task and motion plans, it requires significant effort and expertise to formulate. As such, a number of works have sought to automatically translate natural language (NL) instructions and specifications into temporal logic (TL) formulas, specifically using LLMs [9]–[13]. However, they lack guarantees on the accuracy and syntactic validity of generated temporal logic formulas. Recently, conformal prediction has been used to quantify the uncertainty of LLM outputs, and applied towards an accuracy guarantee for NL to TL [12]. Specifically, an LLM is prompted to incrementally translate an NL specification, with the LLM queried multiple times for each translation step. Then, the frequency of each unique response at each step is used as a measure of uncertainty, and triggers asking for help from a human when high uncertainty is detected. Syntactic validity in this work is addressed by repeatedly sampling from the LLM with the same prompt and filtering out responses which contain invalid operators or atomic propositions. We note that there is no guarantee that any of the sampled responses from the LLM will be syntactically valid, and that these filtering rules do not fully capture the syntax rules of temporal logic. We seek to address this problem in this work.

3) Meta-heuristics in Vehicle Routing Problems: As the centralized baseline for comparison, assigning a new help request within a fleet can be formulated as a dynamic Vehicle Routing Problem (VRP), a long studied subject within the operations research field [14]. Since exact optimization is intractable at realistic scales, constructive insertion heuristics (e.g sequential/greedy insertion [15]) provide a feasible solution promptly and meta-heuristics like Iterated Local Search (ILS) improve the optimality of the initial solution by introducing local moves with perturbations to escape local minima [16]. We implement this centralized hybrid VRP algorithm (insertion + ILS) as an "Oracle" baseline that can reshuffle tasks across all robots to minimize the fleet's overall makespan. In contrast to this centralized VRP approach, our framework only requires compact exchanges in NL $\rightarrow$ TL $\rightarrow$ MILP bids, minimizing the information demand without sacrificing scalability.

Contributions: We present a framework that enables heterogeneous warehouse robots, each subject to local task specifications, to request and offer assistance in natural language, while ensuring the safety and feasibility of the resulting recovery plan through formal verification. In particular, we make the following contributions:

1) We propose a method to transform natural language specifications into temporal logic specifications with a guarantee on validity (Secs. III-A and III-B).
2) We augment LLM agents with spatial and temporal reasoning capabilities, therefore making progress towards realizing guarantees from formal methods (Sec. III-C).
3) We rigorously evaluate our decentralized method against both simple heuristics and a near-optimal centralized "Oracle" baseline demonstrating a compelling trade-off between performance and information demand. (Sec. IV-B).

II. Preliminaries and Problem Statement

Consider robots operating in parallel on a variety of tasks; in this work we use warehousing as our canonical example. We investigate how a system can achieve decentralized conflict resolution while approaching the global efficiency of a centralized solution through peer-to-peer interactions that minimizes the information demand. **If a robot detects a conflict that it cannot resolve on its own, how can it request and receive help from a fellow robot while ensuring safety and minimizing impact on the overall system?** We now formalize these notions.

Notation: The reals are $\mathbb{R}$, natural numbers are $\mathbb{N}$, and integers are $\mathbb{Z}$. To describe functions operating on natural language, we denote $\mathbb{L}$ as the set of all natural language utterances. Subscripts are indices and superscripts are labels. We use $\leftarrow$ to denote the output of NL reasoning.

World and Robots: We consider a multi-robot system (MRS) with n robots, and $\mathcal{R} = \{0, 1, \cdots, n-1\}$ denoting the set of all robots. The robots operate in a discrete time grid world, so robot i is at $x_i(t) \in W \subset \mathbb{Z}^2$ at time $t = 0, 1, \cdots, H$, where $H \in \mathbb{N}$ is a finite time horizon. Robot i's trajectory is a sequence of its position from time $t \in [0, H]$, denoted by

$$X_i = \{x_i(t)\}_{t=0}^{H} \subseteq W. \tag{1}$$

We index a trajectory as $X_i(j) = x_i(j)$ at a specific timestep and $X_i(j:k) = \{x_i(t)\}_{t=j}^{k}$ for a range. Given trajectory X_i of duration H, its Manhattan distance is

$$D(X_i) = \sum_{t=0}^{H-1} \|x_i(t+1) - x_i(t)\|_1. \tag{2}$$

Each cell is either free space or a static obstacle, a common setup in modeling multi-robot systems [17]; we denote all free cells as $F \subseteq W$. Multiple robots can occupy a free cell concurrently ($x_i(t) = x_j(t) \in F$ is allowed); we assume local

coordination and collision avoidance are handled by lower-level planning and control. Finally, each robot has an NL representation of its own capabilities $c_i \in \mathbb{L}$ (e.g., "can lift pallets" or "max speed 1 m/s")

Tasks and Schedules: Let $\Psi = \{\psi_1, \psi_2, \ldots, \psi_K\}$ be the set of K initial tasks to be completed by the system. A task schedule is a partition of these tasks among the robots, denoted by $S = \{\Psi_0, \Psi_1, \ldots, \Psi_{n-1}\}$, where $\Psi_i \subset \Psi$ is the set of tasks assigned to robot i, $\bigcup_{i \in \mathcal{R}} \Psi_i = \Psi$. Finally, $\psi_i \cap \psi_j = \emptyset$ for $i \neq j$, i.e., no two robots are assigned the same task.

For each robot i, its assigned tasks Ψ_i are represented as an STL specification φ_i and path X_i for robot i. All robots must also obey a global specification[1] φ^{g}. We assume each robot's tasks are feasible while, we certify the syntactic validity of the STL via BNF constrained generation (Sec. III-B).

Makespan: Our goal is for robots to minimize time to complete their own tasks plus help tasks, which is the *makespan* of each robot and the overall system. Consider robot i, with initial position $x_i(0) \in F$ and corresponding STL specification φ_i. If the robot's trajectory X_i satisfies φ_i, we write $X_i \models \varphi_i$. Note, this expression is a shorthand for $(X_i, 0) \models \varphi_i$, meaning we always evaluate the satisfaction of the trajectory starting from $t = 0$. We minimize makespan by finding X_i to take the least time to satisfy φ_i. To aid in defining the makespan we first define the *time to first satisfaction* $T(X_i, \varphi)_i$ as

$$T(X_i, \varphi_i) = \min_t \{t \mid X_i(0:t) \models \varphi_i\}. \tag{3}$$

Note that $T(X_i, \varphi_i) = \infty$ if $X_i \not\models \varphi_i$. Then we compute the makespan $M(x_i(0), \varphi_i) \in \mathbb{N}$ as

$$M(x_i(0), \varphi_i) = \min_{X_i} \{T(X_i, \varphi_i) \mid X_i(0) = x_i(0)\}. \tag{4}$$

That is, the makespan finds a trajectory that minimizes the time to first satisfaction of φ_i while obeying the robot's initial condition. We implement (4) as an MILP solved with Gurobi [18]. Further details are in Sec. IV.

Communications: We assume robots can send and receive natural language messages $m \in \mathbb{L}$ instantaneously and error-free. We denote $m_{i \to j}$ as a message from robot i to robot j, and a broadcast message as $m_{i \to \mathcal{R}}$.

Conflicts: Each robot can detect a conflict (e.g., using a VLM based conflict detector as in Fig. 1), which we denote as a tuple $(m^{\mathrm{c}}, x^{\mathrm{c}}, \tau^{\mathrm{c}}) \in \mathbb{L} \times W \times \mathbb{N}$ containing a natural language description, a location, and a duration required to resolve it. We assume x^{c} is resolved when a helper robot occupies the same cell as the requester for at least τ^{c} time steps; we leave low-level coordination to future work.

Problem. *Suppose robot i detects a conflict $(m^{\mathrm{c}}, x^{\mathrm{c}}, \tau^{\mathrm{c}})$. Without using centralized task assignment, we seek to identify another robot $j \in \mathcal{R}_{-i} = \mathcal{R} \setminus \{i\}$ is the set of candidate helper robots, and plan its motion such that it resolves the conflict while minimizing the increase in the total makespan across all robots. That is, create φ_j such that $x_j(t) = x^{\mathrm{c}}$ for all $t' \in [t, t + \tau^{\mathrm{c}}]$ while minimizing the sum of individual makespans $\sum_{i \in \mathcal{R}} M(x_i(0), \varphi_i)$.*

[1]See example in Sec. -A.1 for actuation limits and obstacles

III. Proposed Method

In this section, we introduce the main components of our framework, as depicted in Fig. 1. Concretely, Sec. III-A describes how a *requester* robot that requires help generates a help request and broadcasts it to the multi-robot system. Each robot in the system then evaluates its capabilities and availability, and any *helper* robot with the potential to assist formulates a help offer in natural language (NL). Then, Sec. III-B details how each helper translates its help request from NL to STL, while Sec. III-C describes how each helper computes an optimal path plan that minimizes total time impact to the system. Each potential *helper* then replies to the requester in natural language with its help offer and the associated, predicted time impact to the system. Finally, the requester chooses the helper with the lowest impact and affirms its selection via a help confirmation message.

A. Generating Help Requests, Offers, and Confirmations

We implement a multi-robot communication protocol for collaborative conflict resolution using LLMs. This protocol involves three capabilities: send, receive and broadcast. We define three types of messages: help requests, help offers, and help confirmations[2].

Help requests are broadcast messages sent by a robot facing a conflict after determining that it requires help, by reasoning over the conflict $(m^{\mathrm{c}}, x^{\mathrm{c}}, \tau^{\mathrm{c}})$ and its own capabilities c_i, conditioned on a prompt $p^{\mathrm{c}} \in \mathbb{L}$:

$$m^{\mathrm{r}}_{i \to \mathcal{R}_{-i}} \leftarrow \mathrm{NLR}(m^{\mathrm{c}}, x^{\mathrm{c}}, \tau^{\mathrm{c}}, c_i \mid p^{\mathrm{c}}), \tag{5}$$

where NLR abstractly represents NL reasoning, implemented with foundation models. Help requests describe the scene, location, what is required for the conflict to be resolved, and why the robot cannot resolve the conflict. To ensure conflict location and requester capabilities are included in help requests, we use constrained generation [19].

Help offers are sent by potential helpers in response to a help request. To generate help offers, each robot j reasons over the help request, its own location, and its capabilities:

$$m^{\mathrm{o}}_{j \to i} \leftarrow \mathrm{NLR}(m^{\mathrm{r}}_{i \to \mathcal{R}_{-i}}, x_j, c_j \mid p^{\mathrm{h}}). \tag{6}$$

We again use constrained generation to ensure help offers include information about capabilities. In addition, the help offer includes the duration it will take robot j to provide help τ^{h}_j (i.e., how long robot i needs to wait), and the additional time to help relative to completing its original tasks τ^{new}_j (i.e., how much robot j is impacted by having to complete the help request). We compute these durations below in Sec. III-C (see (12)).

Finally, the requester confirms the lowest-cost help offer by finding

$$j^\star = \arg\min_{j \in \mathcal{R}_{-i}} \left(\tau^{\mathrm{h}}_j + \tau^{\mathrm{new}}_j \right). \tag{7}$$

The selected helper $j^\star$ then receives the message $m^{\mathrm{s}}_{i \to j^\star} =$ "accept" while all other candidate helpers receive $m^{\mathrm{s}}_{i \to j} =$ "reject" for all $j \neq j^\star$.

[2]An example help request is in Sec. -B.1 and help offer in Sec. -B.2.

B. Translating Help Proposals to STL Specifications

To translate help proposals from NL to STL, We draw from a body of recent work on the transformation of natural language tasks and specifications into temporal logic formulas using large language models [9]–[11]. We enhance these natural language to temporal logic methods by defining a Backus-Naur form (BNF) grammar, a notation system for defining formal languages that can be used to constrain the output of LLMs [20]. Additionally, we finetune the LLM model using LoRA [21]. Specifically, defining a BNF grammar for STL involves defining unary and binary temporal and boolean operators, allowed predicates, and text representations of temporal logic relations. BNF constrained generation enforces the syntactic validity of generated temporal logic formulas, allowing us to directly feed LLM outputs into an STL solver[3]. However, constrained generation has been shown to degrade task performance in many cases, i.e. not "minimally invasive" [22]. As such, we carefully designed the BNF grammar[4] to be sufficiently relaxed as to be "minimally invasive", for example including optional whitespace and parentheses, and allowing for arbitrary nesting and recursion, while maintaining the guarantee that the output can be processed using standard STL parsers. Ultimately, we generate an STL specification by reasoning over the help offer message and conflict location, conditioned on a prompt p^{STL} and subject to the BNF grammar:

$$\varphi_j^{\text{h}} \leftarrow \text{NLR}(m^{\text{o}}_{j\to i}, x^{\text{c}} \mid p^{\text{STL}}, \text{BNF}). \tag{8}$$

C. Solving for Optimal Robot Paths

Suppose an initial, globally near-optimal task schedule $S^{\text{orig}} = \{\Psi_0, \ldots, \Psi_{n-1}\}$ has been determined. Each candidate helper robot $j \in \mathcal{R}_{-i}$ has an original task specification φ_j^{orig} and has found a corresponding optimal path

$$X_j^{\text{orig}} = \{x_j(t) \mid t = 0, \cdots, M(x_i(0), \varphi_j^{\text{orig}})\} \tag{9}$$

by minimizing Manhattan distance[5] and time to satisfy the task specification (i.e., makespan) while always obeying global specifications:

$$X_j^{\text{orig}} = \arg\min_X \quad D(X) + T(X, \varphi^{\text{orig}}) \tag{10a}$$

$$\text{s.t.} \quad X \models \Diamond\varphi_j^{\text{orig}} \wedge \Box\varphi^{\text{g}}. \tag{10b}$$

Once the help offer $m^{\text{o}}_{j\to i}$ is translated into φ_j^{h} as in (8), robot j computes an updated path that both helps and completes the original tasks:

$$X_j^{\text{new}} = \arg\min_X \quad D(X) + T(X, \Diamond\varphi_j^{\text{h}}) + T(X, \varphi_j^{\text{new}}) \tag{11a}$$

$$\text{s.t.} \quad X \models \varphi^{\text{new}} \tag{11b}$$

$$\varphi_j^{\text{new}} = \Diamond\varphi_j^{\text{h}} \wedge \Diamond\varphi_j^{\text{orig}} \wedge \Box\varphi^{\text{g}}. \tag{11c}$$

[3]Details on how we encode STL specifications can found in the appendix.

[4]An example BNF grammar can be found in Sec. -A.3.

[5]We use Manhattan distance to capture the number of movements in a grid world and improve computation time over solving a Mixed Integer Quadratic Program (MIQP) with Euclidean Distances.

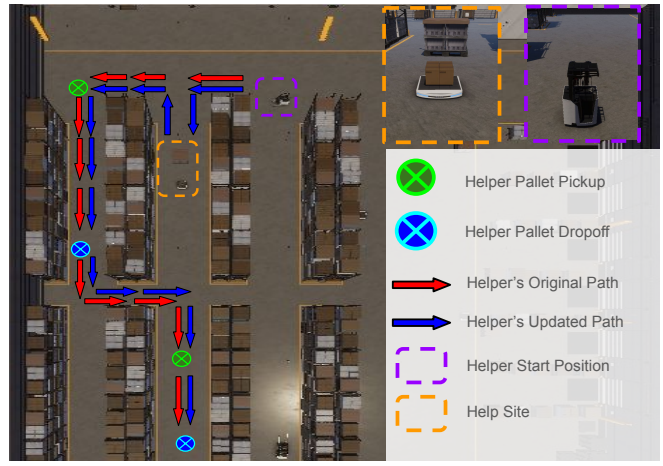

Fig. 2: An example of a reconfigured path. The help site is reached in 2 time-steps, extending the original path by 2 time-steps for a total cost $\tau_j^{\text{h}} + \tau_j^{\text{new}} = 4$ time-steps.

The term $T(X, \Diamond\varphi_j^{\text{h}})$ represents how long the requester must wait for help, whereas the term $T(X, \varphi_j^{\text{new}})$ represents the helper's time required to complete the help task plus all other tasks. This encourages the helper to minimize how long the requester must wait.

We formulate (10) and (11) as Mixed Integer Linear Programs (MILPs) by defining predicates in the STL formula as binary variables over discrete time steps, and solve the formulated MILPs with Gurobi [18].

Finally, when crafting a help offer, robot j reports the total time it will take to help *and* the additional time to help relative to its original tasks:

$$\tau_j^{\text{h}} = M(x_j(0), \varphi_j^{\text{h}}) \tag{12}$$

$$\tau_j^{\text{new}} = M(x_j(0), \varphi_j^{\text{new}}) - M(x_j(0), \varphi_j^{\text{orig}}). \tag{13}$$

Note that, in our implementation, τ_j^{h} and τ_j^{new} are computed as a byproduct of solving (11), instead of solving new MILPs to compute these makespans.

Importantly, our method does not alter the original task schedule S^{orig}; only the chosen helper j augments its plan with the help task. This contrasts with an approach using a centralized Oracle (c.f., [15], [16]), which is free to reallocate any task from any ψ_k to any other robot to minimize the total system cost which is the sum of the individual makespan, $\sum_{i\in\mathcal{R}} M(x_i(0), \varphi_i)$. This means that our method sacrifices optimality for decentralization, as we assess next.

IV. Experiments

To demonstrate the efficacy of the proposed end-to-end framework for multi-robot collaboration, we isolate and evaluate its two constituent modules: the natural language–to–temporal logic translation component and the temporal logic–to–task plan generation component. Each module is quantitatively compared against state-of-the-art methods in natural language–to–temporal logic translation and vehicle routing, respectively.

All experiments were performed using Python[6] on a desktop computer with a 32-core i9 CPU, 32 GB RAM, and an NVIDIA RTX 4090 GPU.

A. Experiment 1: Natural Language to Temporal Logic

In this experiment, we evaluate our method for transforming natural language (NL) to temporal logic (TL) using BNF constrained generation. We implement our method with Gemma 3 12B LLM using the `llama.cpp` library for BNF constrained generation and LLaMA-Factory [23] for finetuning. The BNF grammar is included in the LLM prompt and as a constraint.

Hypothesis: We hypothesize that our method demonstrates comparable translation accuracy to existing baselines on a large and diverse data set of NL-TL pairs, with a strict guarantee on the validity of TL formulas, while running on a significantly smaller LLM (Gemma 3) which can be deployed onboard a robot with a single consumer GPU.

Experiment Design: To benchmark NL to TL translation performance, we use the dataset presented in [9], which consists of 7,500 pairs of natural language sentences and corresponding signal temporal logic (STL) formulas in the context of navigation tasks. 6,000 pairs are used for finetuning, and 1,500 pairs are used for evaluation.

We compare our method to several ablated variants and to a GPT-4 baseline presented in [9]. The ablations we consider include variants without the grammar constraint, without the grammar prompt, and with combinations thereof, allowing us to isolate the contributions of each component of our method. Our method and each ablation are evaluated with 5 and 20 "few-shot" examples of natural language tasks with accurate temporal logic translations included in the prompt to the LLM, with 500 NL-TL pairs randomly sampled from the dataset. Each evaluation is rerun 3 times, with resampled few-shot examples and NL-TL pairs. In our approach, both the BNF grammar and few shot examples are included in the prompt, and the BNF grammar is enforced during decoding using `llama.cpp`'s constrained generation. Results with constrained generation are not presented for GPT-4, as this model can only be accessed through the OpenAI API, which does not allow for a BNF grammar constraint to be specified.

Metrics: We measure Validity as the percentage of generated formulas that are syntactically correct and Accuracy as percentage of valid generated STL formulas logically equivalent to true STL formulas from the dataset. Accuracy is computed on the mutually valid set of generated formulas across all ablations. The logical equivalence and syntactic correctness of STL formulas are checked using the `Spot` library [24]. Note we also measure Inference Time as average runtime per NL-STL pair (ms), in Appendix -C.

Results and Discussion: The results are summarized in Table I. Our method achieves 100% formula validity in all cases with no loss in accuracy due to constrained generation, confirming our hypothesized guarantee. Additionally, we show improved translation accuracy in the navigation dataset with a significantly smaller LLM.

For each STL formula which was incorrectly translated from natural language by the LLM, we check containment to determine the severity of the failure. In other words, if the true STL formula is contained in the STL formula generated by the LLM, then every TAMP that satisfies the LLM formula will also satisfy the true formula, and therefore the constraint expressed in the natural language will not be violated. Formally,

$$\varphi^{\text{LLM}} \implies \varphi^{\text{true}} \iff \forall \psi \in \Psi : \psi \models \varphi^{\text{LLM}} \Rightarrow \psi \models \varphi^{\text{true}}. \tag{14}$$

We report the percentage of LLM generated STL formulas which contain the true STL formula in Table II.

See Table III for the inference time of our method and each ablation.

# Ex.	Method Variant	Validity (%)	Accuracy (%)
5	Gemma F + P + C (Ours)	**100.0** ± 0.00	**98.98** ± 0.17
	F + P	99.8 ± 0.45	98.98 ± 0.17
	F	99.0 ± 1.22	84.97 ± 8.65
	GPT-4 F + P	100.0 ± 0.00	62.82 ± 3.66
20	Gemma F + P + C (Ours)	**100.0** ± 0.00	**97.58** ± 1.16
	F + P	99.67 ± 0.01	97.58 ± 1.16
	F	99.60 ± 0.04	93.73 ± 2.40
	GPT-4 F + P	100.0 ± 0.00	92.73 ± 3.00

TABLE I: Ablation study on NL to TL with varying number of few-shot examples. F, P, and C refer to few shot prompting, inclusion of the BNF grammar in the prompt, and the BNF grammar constrained generation respectively.

# Ex.	Method	LLM $\implies$ True (%)
5	Gemma F + P + C (Ours)	99.45 ± 0.24
	F + P	99.45 ± 0.24
	F	87.02 ± 9.19
	GPT-4 F + P	74.13 ± 6.62
20	Gemma F + P + C (Ours)	99.25 ± 0.23
	F + P	99.25 ± 0.23
	F	97.52 ± 0.89
	GPT-4 F + P	95.20 ± 6.64

TABLE II: Containment checking of LLM generated STL formulas for each NL to TL ablation. If the true STL formula is contained in the LLM-generated STL formula, the constraint specified in the natural language will always be satisfied.

B. Experiment 2: Mobile Robot Blocked by a Pallet

In our second experiment, we compare our decentralized framework to a centralized "Oracle" in a scenario where forklift robots respond to a help request from a mobile robot prompted by the scene description m^{c} = "A pallet is blocking the entrance to the picking aisle." Each forklift robot $j \in \mathcal{H}$ then evaluates whether it can handle this additional task (moving the obstructing pallet) on top of its existing pick-and-place (PNP)[7] jobs by updating its STL specification φ_j^{orig}. The help task, φ_j^{h}, is similarly encoded as a PNP job where the forklift must travel to the help site, pick up the

[6]Code and Fine-tuned LLM model available at: `https://github.com/ask-reason-assist/ask-reason-assist-codebase`

[7]Our STL specification of a PNP job is in Sec. -A.2.

obstructing pallet, and place the pallet in the nearest free cell $x^{\text{adj}} = \arg\min_x \{\|x - x^c\|_1\}$.

Hypothesis: We hypothesize that our decentralized framework, where each potential helper solves for its updated TL specification via MILP in parallel, will achieve system efficiency comparable to a near-optimal "Oracle" with full system knowledge. We expect our method to significantly outperform myopic, distance-based heuristics, even when such heuristics are coupled with a central optimizer.

Experiment Design and Baseline: We conducted 100 simulation trials with randomly chosen help-site locations. To ensure a fair comparison within each trial, a single, consistent starting scenario was established for the fleet.

This involved spawning six forklift robots at random initial positions and distributing total of twelve PNP tasks among the fleet according to a global schedule, S^{orig}. We precompute the schedule using the "Oracle" baseline to find a near-optimal solution to the MRTA problem. When a help task φ^{h} arises, the Oracle finds a new via ILS metaheuristics, as detailed[8] in [16].

Our method and the three distinct baselines were then evaluated against this identical initial setup: (B1) The ILS-based global planner "Oracle", (B2) the closet forklift to the help site solves for the optimal path via MILP, and (B3) a hybrid approach where help task is assigned to the closet agent, but the Oracle is allowed to re-optimize all other tasks.

Results and Discussion: Fig. 3 summarizes our findings. The Centralized Oracle (B1) added an average of 4.49 time-steps to the system by inserting the help task, establishing a strong performance benchmark. Our *decentralized* method achieved a highly competitive result of 5.47 steps on average with a standard deviation of 3.32 time steps, supporting that informed local optimization captures the majority of the performance gains of a centralized system without the need for global task reassignment or heavy information demand. Importantly, the average `Gurobi` model solve time was 5.3 seconds with 30 discretized time horizon, demonstrating computational practicality for real-time deployment.

On the other hand, our *optimal cost* approach on average provides 46% and 53% efficiency gains over B2 and B3 respectively. We observe that the inter-dependencies in PNP jobs make distance-based heuristics (B2-B3) suboptimal, consistent with previous literature in operations research [25]. Furthermore, an initial myopic decision (B3) can constrain the overall system's optimality from which even a powerful optimizer cannot recover efficiently [26]. In contrast, our method explicitly avoids such myopia by reasoning over future task interactions.

V. Demonstrations

We now demonstrate the utility of NL in conflict resolution on three complex warehouse tasks. These complement our experiments, which stress-test components of our method. Here, we illustrate the practicality of our overall framework by implementing it in a high-fidelity physics-based Unity simulator. In particular, we integrate our NL-to-TL translation module with a MILP-based TAMP mechanism for executing sequentially nested tasks, alongside local collision avoidance and an A* path planner for realistic motion plans. Full videos of each demonstration are available on our project website `https://ask-reason-assist.github.io`.

[8]The algorithm table can be found in Appendix -D

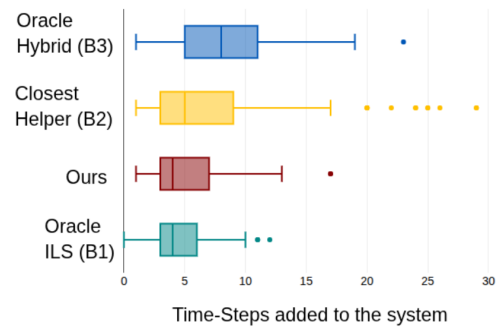

Fig. 3: Comparison of total time-steps added to the system under different methods tested in Sec. IV-B. Our decentralized method tracks the Oracle solution within 18% while significantly outperforming the distance based heuristics (B2) and hybrid approach (B3).

Demo 1: Pallet Cleanup: First, we revisit the pallet-blocking task from the second experiment, where forklift robots must help a mobile robot. The NL help request, "A pallet is blocking the aisle", is translated into a syntactically valid TL specification for the MILP solver based TAMP module updates the optimal forklift agent's task schedule. This demo showcases a complete rollout of the framework to complement the experiment in Sec. IV-B.

Demo 2: Warehouse Kitting: Our second scenario is a warehouse kitting task, a common workflow in fulfillment operations. Here the NL input is, "Eventually pick up an item from Aisle A and Aisle B and Aisle C." Our NL-TL module successfully translates this into a semantically correct and syntactically valid TL formula:

$$\varphi^{\text{h}} = \bigwedge_{[0,H]} \left(\Diamond x^{\text{pick A}}, \Diamond x^{\text{pick B}}, \Diamond x^{\text{pick C}} \right) \tag{15}$$

Our system uses this formula to compute the optimal sequence of visits to minimize the total makespan added to the system. This demo showcases our system's ability to handle unordered conjunctive goals (i.e., tasks that can be done in any order), as the LLM correctly translates the semantics that the first three APs can become true in any order.

Demo 3: Sequential Tool Retrieval: In our final, most complex scenario, the NL help request is, "First visit the front office to pick up the high fidelity scanner, then travel to the help site to scan the damaged good, and finally navigate back to the front office to return the scanner." This is translated into the nested STL formula:

$$\varphi^1 = \bigwedge_{[0,H]} \left(\Diamond x^{\text{pickup Scanner}}, \Diamond x^{\text{scan}}, \Box(\neg x^{\text{scan}} \mathbf{U} x^{\text{pickup Scanner}}) \right)$$

$$\varphi^{\text{h}} = \bigwedge_{[0,H]} \left(\Diamond \varphi^1, \Diamond x^{\text{return Scanner}}, \Box(\neg x^{\text{return Scanner}} \mathbf{U} \varphi^1) \right)$$

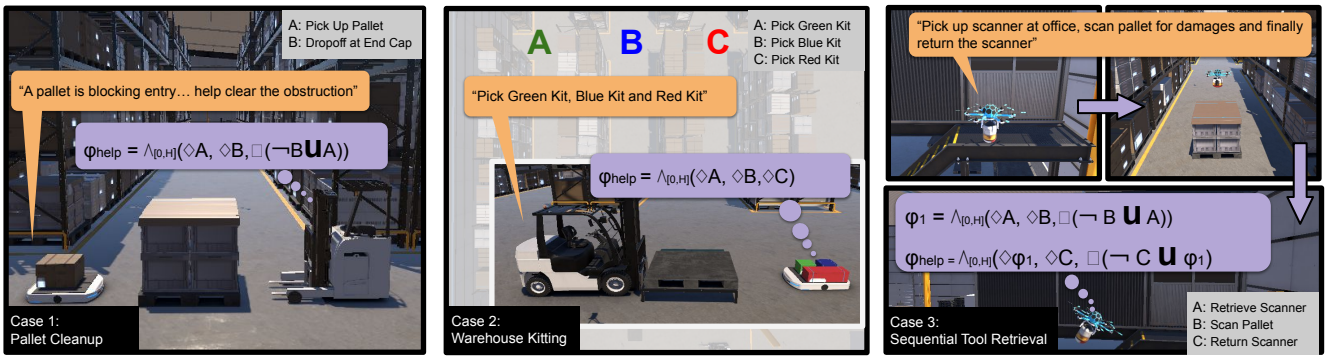

Fig. 4: Unity-based demonstrations of our NL-to-TL decentralized framework. (Left) **Pallet Cleanup**: a forklift responds to an NL request to clear a blocked aisle, updating its MILP plan to integrate the help task. (Center) **Warehouse Kitting**: unordered conjunctive goals ("pick A,B,C") are translated into a concise TL formula allowing flexible task sequencing. (Right) **Sequential Tool Retrieval**: strict temporal ordering ("first-then-finally") is captured by the nested TL specification, enabling execution of complex multi-step tasks. These demonstrations showcase our end-to-end framework—where each helper robot can translate natural language into valid temporal logic and update its MILP plan—running in a realistic Unity warehouse simulator.

Our NL-to-TL translation module correctly interprets the keywords "First, then and finally" to enforce a strict order in which each AP holds true, highlighting how our framework enables a robot to autonomously leverage the expressiveness of complex TL specifications. This demo showcases the framework's practicality for capturing strict sequential and nested temporal dependencies from natural language.

VI. Conclusion and Future Work

This paper presents a novel framework for robots to request and receive help for decentralized conflict resolution. Our method lets robots coordinate and reason over their own capabilities in natural language (NL) while reasoning over their motion plans and tasks in formal temporal logic (TL). We achieve this via a method for converting between NL and TL with guaranteed validity, thereby giving LLM agents spatial and temporal reasoning. Through experimental evaluation, we find that our method can resolve conflicts while maintaining overall system performance similar to a centralized Oracle baseline. Furthermore, demonstrations across several robot morphologies and capabilities show how our method successfully combines the flexibility of NL with the formality of TL. That said, our method still has limitations to address in future work. First, we assume conflicts are detected correctly. Second, we have no considered low-level motion planning or physics in helper or requester capabilities. Third, we have only used NL in a specific help request problem setting, but it remains open how NL can integrate into broader multirobot operations.

References

[1] R. Sinha, A. Elhafsi, C. Agia, M. Foutter, E. Schmerling, and M. Pavone, "Real-time Anomaly Detection and Reactive Planning with Large Language Models," *arXiv preprint arXiv:2407.08735*, 2024.

[2] J. Chen, C. Yu, X. Zhou, *et al.*, "Emos: Embodiment-aware heterogeneous multi-robot operating system with llm agents," *arXiv preprint arXiv:2410.22662*, 2024.

[3] Y. Kato, T. Yoshida, Y. Sueoka, *et al.*, "Design of a Multi-robot Coordination System based on Functional Expressions using Large Language Models," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2024, pp. 3447–3454.

[4] M. Guo and D. V. Dimarogonas, "Multi-agent Plan Reconfiguration under Local LTL Specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[5] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3451–3458, 2022.

[6] Z. Yang, S. S. Raman, A. Shah, and S. Tellex, "Plug in the Safety Chip: Enforcing Constraints for LLM-driven Robot Agents," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2024, pp. 14 435–14 442.

[7] C. Belta and S. Sadraddini, "Formal Methods for Control Synthesis: An Optimization Perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 115–140, 2019.

[8] V. Kurtz and H. Lin, "Mixed-integer Programming for Signal Temporal Logic with Fewer Binary Variables," *IEEE Control Systems Letters*, vol. 6, pp. 2635–2640, 2022.

[9] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, "Nl2tl: Transforming Natural Languages to Temporal Logics using Large Language Models," *arXiv preprint arXiv:2305.07766*, 2023.

[10] F. Fuggitti and T. Chakraborti, "NL2LTL–a Python Package for Converting Natural Language (NL) Instructions to Linear Temporal Logic (LTL) Formulas," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 16 428–16 430.

[11] A. Mavrogiannis, C. Mavrogiannis, and Y. Aloimonos, "Cook2LTL: Translating Cooking Recipes to LTL Formulae using Large Language Models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2024, pp. 17 679–17 686.

[12] J. Wang, D. S. Sundarsingh, J. V. Deshmukh, and Y. Kantaros, "ConformalNL2LTL: Translating Natural Language Instructions into Temporal Logic Formulas with Conformal Correctness Guarantees," *arXiv preprint arXiv:2504.21022*, 2025.

[13] Y. Fang, Z. Jin, J. An, H. Chen, X. Chen, and N. Zhan, "Enhancing transformation from natural language to signal temporal logic using llms with diverse external knowledge," *arXiv preprint arXiv:2505.20658*, 2025.

[14] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[15] M. M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.

[16] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," in *Handbook of metaheuristics*, Springer, 2003, pp. 320–353.

[17] R. Stern, N. Sturtevant, A. Felner, *et al.*, "Multi-agent Pathfinding: Definitions, Variants, and Benchmarks," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, 2019, pp. 151–158.

[18] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2024.

[19] L. Beurer-Kellner, M. Fischer, and M. Vechev, "Prompting is Programming: A Query Language for Large Language Models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 1946–1969, 2023.

[20] B. Wang, Z. Wang, X. Wang, Y. Cao, R. A Saurous, and Y. Kim, "Grammar Prompting for Domain-specific Language Generation with Large Language Models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 65 030–65 055, 2023.

[21] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models.," *ICLR*, vol. 1, no. 2, p. 3, 2022.
[22] L. Beurer-Kellner, M. Fischer, and M. Vechev, "Guiding LLMs the Right Way: Fast, Non-invasive Constrained Generation," *arXiv preprint arXiv:2403.06988*, 2024.
[23] Y. Zheng, R. Zhang, J. Zhang, *et al.*, "Llamafactory: Unified efficient fine-tuning of 100+ language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand: Association for Computational Linguistics, 2024.
[24] Alexandre Duret-Lutz, E. Renault, M. Colange, *et al.*, "From Spot 2.0 to Spot 2.10: What's New?" In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, ser. Lecture Notes in Computer Science, vol. 13372, Springer, Aug. 2022, pp. 174–187.
[25] R. De Koster, T. Le-Duc, and K. J. Roodbergen, "Design and Control of Warehouse Order Picking: A Literature Review," *European Journal of Operational Research*, vol. 182, no. 2, pp. 481–501, 2007.
[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

APPENDIX

A. Encoding STL Specifications

We propose a lightweight interface to express navigational tasks as STL specifications. Inspired by `stlpy` [8], our method emphasizes extracting *spatial* propositions and decoupling *temporal* constraints. We provide an illustrative example here. Consider encoding "*Visit Aisle 1 at least once before time T while avoiding obstacles*," in STL:

$$\varphi^{\text{spec}} = \Box_{[0,H]} \neg \text{Obstacle} \wedge \Diamond_{[0,T]} \text{Aisle1} \tag{16}$$

where Aisle1 and Obstacle are atomic propositions (AP) tied to the corresponding world coordinates (true when the agent is in the region). We specify the time horizon as

```
model.spec = F(aisle1) & G(NOT(obstacle))
model.T = T
```

1) Global STL Specification: Each robot is subject to global safety and actuation constraint. For example, all ground vehicles must avoid obstacle cells and can only move one grid space in any cardinal direction):

$$\varphi^{\text{global}} = \Box(\neg \text{Obstacle} \wedge \mathcal{L}^{\text{actuation}}) \tag{17}$$

2) Pick-and-Place (PNP) Tasks as STL Specifications: We create a PNP specification as

$\varphi_1 = (\neg x^{\text{place}}\, \mathbf{U}_{[0,H]}\, x^{\text{pick}})$

(Robot must pickup the pallet before placing it)

$\varphi_2 = \Box_{[0,H]}(x^{\text{pick}} \rightarrow (\neg x^{\text{others}} \mathbf{U}_{[0,H]} x^{\text{place}})$

(Pallet picked up must be placed before starting other tasks)

$\varphi_3 = \Diamond_{[0,H]} x^{\text{place}}$

(Pallet is eventually placed within time horizon H)

$\varphi_{\text{pnp}}(x^{\text{place}}, x^{\text{pick}}) = \varphi_1 \wedge \varphi_2 \wedge \varphi_3$

3) Example STL BNF Grammar:

```
root ::= ws expr ws
expr ::= term (binary-op term)*

term ::= atomic-formula | unary-op ws "(" ws expr
     ws ")" | unary-op ws atomic-formula | "~"
    ws term | "(" ws expr ws ")"

predicate-name ::= "go_to_charger" | "
    go_to_rack_A"
atomic-formula ::= predicate-name | "(" ws
    predicate-name ws ")"
ws ::= [ \t\n]*

binary-op ::= ws ("&" | "|" | "->" | "U") ws
# '&' (and): both propositions must be true
# '|' (or): at least one predicate must be true
# '->' (implies): if first predicate is true,
    then second predicate must be true
# 'U' (until): first predicate must be true at
    least until second predicate is true

unary-op ::= "G" | "F"
# G (globally): Predicate must always be true at
    every timestep
# F (eventually): Predicate must be true at some
    time in the future
```

B. Natural Language

1) Example Help Request:

Mobile Robot ID 1:
"A pallet is blocking the aisle at location (1, 4). Assistance is required to move the pallet. Pick up pallet at (1,4) and drop it off at the closest free drop zone."

2) Example Help Offer:

Forklift ID 3:
"I can help you in 5 minutes, but it will add 8 minutes to my overall makespan."

C. Experiment I Inference Time Results

# Examples	Method Variant	Inference Time (ms)
5	Gemma F + P + C (Ours)	801.6 ± 36.2
	F + P	583.4 ± 28.2
	F	410.2 ± 9.2
	GPT-4 F + P	803.7 ± 30.3
20	Gemma F + P + C (Ours)	1191.7 ± 14.1
	F + P	974.9 ± 2.5
	F	414.9 ± 3.9
	GPT-4 F + P	2736.9 ± 25.2

TABLE III: Inference times for NL-to-TL method variants with varying numbers of few-shot examples.

D. Oracle Baseline with the ILS Algorithm

Algorithm: Oracle ILS
Input: Cost function J, Neighborhood $\mathcal{N}$
Output: Near-optimal schedule S^*

```
S_0, S, S* ← GreedyInsertion(J)
for k = 1 to max_iterations do:
    S^cand ← Perturb(S) // Perturbation Step
    S' ← LocalSearch(S^cand, N) // Local Search Step
    if J(S') < J(S) or rand() < P(S', S) then:
        S ← S' // Acceptance Criterion
    if J(S) < J(S*) then:
        S* ← S
return S*
```