Toward Reliable Robot Application Generation with Dual-Graph Validation and Simulation-in-the-Loop

Ruichao Wu[1], Jiwei Pan[2], Mohamed Youssef[3], Björn Kahl[4] and Andrey Morozov[5]

***Abstract*— Automatically generating reliable robot applications that can be deployed on real systems remains a major challenge. Existing approaches in task and motion planning, language-driven skill composition, demonstration-based learning, and ontology reasoning each address part of this problem, but none offer an integrated method that ensures both executability and runtime dependability. This paper introduces a framework that enables large language models (LLMs) to compose applications from dependable, application-independent atomic skills. Applications are expressed as Behavior Trees, which capture hierarchical task logic, and are subjected to ontology-based dual-graph validation to ensure control-flow and data-flow consistency before execution. Monte Carlo Tree Search (MCTS) is then applied in simulation to tune discrete and application-dependent parameters. The validated and tuned applications are demonstrated both in simulation and on a real robotic setup. Together, these contributions establish a principled framework for combining language-driven synthesis, knowledge-guided consistency, and MCTS-based tuning to advance the reliable generation of robot applications.**

I. Introduction

Deploying dependable robot applications remains a significant challenge. Building such applications requires domain expertise to design task logic, integrate heterogeneous components, and tune parameters. Despite progress in task and motion planning (TAMP) [1]–[6], motion planning libraries [7], and skill-based frameworks [8], [9], the problem of automatically creating robot applications that are both deployable and dependable remains open. At the same time, vision-language-action (VLA) models [10] and large language models (LLMs) have emerged as powerful approaches for generating robot behavior directly from natural language or multimodal prompts, raising new opportunities but also new challenges for dependability.

[1]Ruichao Wu and [4]Björn Kahl is with Fraunhofer Institute for Manufacturing Engineering and Automation (IPA), Nobelstr. 12, 70569 Stuttgart, Germany ruichao.wu@ipa.fraunhofer.de bjöern.kahl@ipa.fraunhofer.de

[2]Jiwei Pan is with the Institute of Industrial Automation and Software Engineering, University of Stuttgart, Pfaffenwaldring 47, 70550 Stuttgart, Germany st176838@stud.uni-stuttgart.de

[3]Mohamed Youssef is with the Institute of Computer Science, Electrical Engineering and Information Technology, University of Stuttgart, Pfaffenwaldring 47, 70550 Stuttgart, Germany st190193@stud.uni-stuttgart.de

[5]Andrey Morozov is with the Institute of Industrial Automation and Software Engineering, University of Stuttgart, Pfaffenwaldring 47, 70550 Stuttgart, Germany andrey.morozov@ias.uni-stuttgart.de

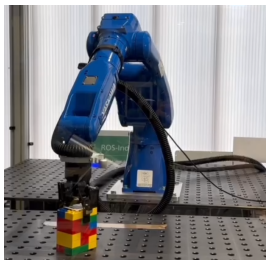


Fig. 1: Physical setup with a Yaskawa GP7 manipulator, Robotiq 2F-85 gripper, and Intel RealSense D435 camera.

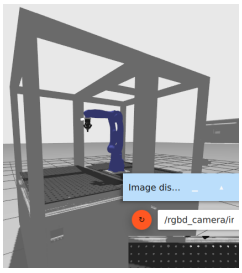


Fig. 2: Simulation setup in Ignition Gazebo with the same hardware configuration as the physical system.

The research question we address is: *How can robot applications be automatically generated in a way that ensures transparency, validity, and runtime dependability?* Existing approaches address parts of this problem but leave important gaps. TAMP integrates symbolic reasoning with geometric feasibility, producing plans that are often theoretically sound, but struggles with scalability in high-dimensional domains and with execution robustness in dynamic environments [6], [11], [12]. LLM-based planners reduce modeling effort by generating task logic from language, but executability depends on predefined skill libraries and heuristic validation [13]. VLAs enable end-to-end mapping from perception to action [10], but lack transparency, making their behavior difficult to inspect or guarantee before deployment. Other methods, such as demonstration- or ontology-based approaches [14], [15], provide semantic consistency or safety guarantees, but scale poorly or are applied post hoc. Together, these limitations highlight the need for a framework that unifies logic generation, validation, and parameter optimization.

We tackle this challenge from two complementary angles. At the system level, we develop a skill-based and resilience-oriented framework where robotic systems are decomposed into atomic skills that provide dependable services. Each skill is monitored to ensure local fault containment, forming a dependable substrate for higher-level application composition. At the application level, we introduce methods for automatically generating robot applications. An LLM first proposes candidate skill sequences, which are grounded in the skill manifest and iteratively refined through ontology-based dual-graph analysis. The dual graphs guide the construction of a Behavior Tree (BT) skeleton while enforcing control-flow and data-flow consistency, with retrieval-based repair ensuring convergence to a valid structure. Once a verified BT

skeleton is obtained, Monte Carlo Tree Search (MCTS) tunes *application-level parameters*. The search is guided by LLM-derived priors and evaluated in simulation, enabling long-horizon strategy tuning before deployment. Motion planners remain responsible for trajectory generation at runtime, while our focus is on ensuring that the generated application logic is transparent, valid, and dependable.

To evaluate this approach, we implement the framework in simulated robotic scenarios using Ignition and demonstrate generated applications on a real setup. An overview of both environments is shown in Fig. 2 and Fig. 1. The simulation provides a controlled testbed for parameter optimization, while the physical setup allows verification of deployability in real-world conditions. The results show that applications produced by our method are structurally consistent, can be automatically parameterized, and are closer to deployment readiness compared to existing baselines.

Contribution: This paper introduces a framework for reliable robot application generation with three main contributions: 1. *Generality:* By formulating applications over atomic, application-independent skills, the framework supports reuse across different robots and tasks rather than being tied to task-specific libraries. 2. *Transparency and assurance:* An ontology-based dual-graph method that guides the construction of BT skeletons, enforces control-flow and data-flow consistency, and provides structured feedback for repairing invalid logic. 3. *Long-horizon robustness:* A parameter optimization scheme that leverages MCTS guided by LLM-derived priors to efficiently explore interdependent application-level parameter spaces, supporting long-horizon strategy search.

II. State of the Art

The long-term goal of autonomous robotics is to generate reliable robot applications that can be deployed directly on physical systems. Achieving this requires not only sequencing actions, but also representing them in a transparent structure, validating their interfaces and logic, tuning execution parameters, and ensuring runtime dependability. Existing research addresses these aspects through several distinct approaches.

Task and Motion Planning. Classical TAMP integrates symbolic task planning with geometric motion planning. Symbolic tasks are often formulated in STRIPS-like models and standardized in the Planning Domain Definition Language (PDDL) [1]–[3], enabling domain-independent planners to generate task-level plans. These high-level plans are then refined by motion planners to ensure feasibility in continuous space, including kinematics and collision constraints [4]–[6], [11], [12], [16]. The integration allows provably correct plans to be created, but typically requires extensive domain modeling and is often performed offline, with incremental variants interleaving symbolic and geometric search. In contrast, our framework leaves trajectory generation to runtime planners and instead focuses on validating application logic and tuning discrete parameters such as planner selection.

LLM-based planners. Recent work explores foundation models to generate robot applications directly from natural language or multimodal prompts. LLMs are often used to sequence predefined high-level skills such as *pick*, *place*, or *navigate*, producing code [13], [17] or BTs [18]–[20]. This brings flexibility and interpretability, but executability still depends on the coverage and fidelity of the high-level library; correctness typically relies on practical checks rather than formal guarantees [21]–[23]. Our framework instead relies on reusable atomic skills and ontology-backed interface validation to strengthen dependability.

VisionLanguageAction policies. Another direction is to bypass symbolic representations and directly map perception and language to actions via large-scale policies. Systems such as *RT-1/RT-2* [24], [25] are trained on paired language-action datasets; *PaLM-E* [26] fuses visual and language tokens for embodied reasoning; and generalist imitation-learned policies like *Diffusion Policy* [27], *PerAct* [28], *BC-Z* [29], *Open X-Embodiment* [30], and *RoboCat* [31] achieve strong cross-task and cross-robot generalization. These systems often enable zero-shot transfer without intermediate symbolic structures, but their decision process is opaque, making pre-execution validation difficult. Our focus remains on explicit symbolic structures that can be inspected and repaired when necessary.

Demonstration-, specification-, and ontology-based methods. Several approaches synthesize applications from symbolic knowledge or expert input. Demonstration-based methods lift expert trajectories into BTs or other hierarchical structures [32], [33], while temporal-logic specifications (e.g., LTL_f) can be compiled into correct-by- construction BTs [34]. Search-based synthesis explores BT spaces with grammar guidance or formal feedback [35], [36]. Ontologies and knowledge graphs, as in KnowRob, CRAM, or RoboBrain [37]–[39], encode expert knowledge for reasoning and interface consistency, and complementary verification methods provide formal safety or liveness guarantees on BTs [40], [41]. These approaches improve reusability, transparency, or correctness, but typically rely on manual authoring, scale poorly, and rarely address parameter tuning or runtime adaptation.

Parameter tuning. Another challenge in generating dependable robot applications is setting *application-level parameters* that influence task execution. These parameters are essential for long-horizon strategies but are typically left to manual adjustment. In contrast, much of the literature on automatic tuning has focused on low-level controllers or simulator calibration: Bayesian optimization methods such as SafeOpt [42] adjust controller gains while ensuring safety, causal approaches like CURE [43] identify important simulator configurations for sim-to-real transfer, and search-based methods including Diff-

Tune [44] or evolutionary strategies [45] optimize dynamics and locomotion policies. These contributions improve robustness at the control or simulation level, but do not address the symbolic application parameters that determine how robot skills are composed and executed.

Our framework tackles this gap by employing MCTS to tune application-level parameters within Behavior Trees. MCTS is particularly suitable because it excels at long-horizon planning and combinatorial search [46]–[48], allowing us to efficiently explore alternative parameterization of complex task strategies. By combining LLM-derived priors with MCTS search, our approach tunes symbolic logic in simulation before deployment, a capability not offered by existing parameter optimization pipelines.

III. System Design

The proposed framework transforms natural-language task descriptions into verified and deployable robot applications. The process has two stages: (i) application logic is composed from atomic skills using skill-manifest-grounded retrieval and ontology-guided dual-graph analysis, producing a verified BT skeleton; and (ii) application-level parameters of the BT are optimized in simulation via MCTS guided by LLM-derived priors.

A. Skill-Manifest-Grounded Composition with Ontology-Guided Dual Graphs

A prerequisite for automated application generation is a formal description of the system's functional capabilities. In this framework, available capabilities are encoded in a *skill manifest*, a machine-readable specification that enumerates atomic skills together with their semantics. Each entry (atomic skill) defines a skill name, a natural-language description, its parameters, and input/output (I/O) types. Ruichao: Shall I add an example here of a skill manifest entry? The skill manifest thus provides an explicit contract for how skills can be composed into higher-level logic.

Directly constraining LLM outputs to this manifest is challenging: language models often produce skill names that do not exactly match the manifest (e.g., synonyms, abbreviations, or hallucinated names), which would otherwise invalidate the generated logic. Retrieval-augmented generation (RAG) is introduced to bridge this gap. The skill manifest is transformed into a vector database. It is chunked at the granularity of atomic skills, such that each entry forms a self-contained document containing semantics and interface constraints, which avoids long-document embedding dilution and ensures that retrieval remains interpretable at the granularity of executable units in a BT. Each document is embedded using an E5-family model and indexed with FAISS for efficient dense retrieval [49]–[52]. At query time, LLM-proposed skills are embedded and compared against the index using cosine similarity [53]:

$$\mathrm{sim}(q, d) = \frac{\langle \mathbf{e}(q), \mathbf{e}(d) \rangle}{\|\mathbf{e}(q)\| \, \|\mathbf{e}(d)\|}.$$

Following standard dense-retrieval practice [54], [55], the Top-k most similar entries are returned, while a similarity threshold θ filters out low-confidence matches [56]. This enables semantic matching rather than exact string matching, preventing hallucinations and ensuring that all generated skills map to manifest-compliant atomic skills.

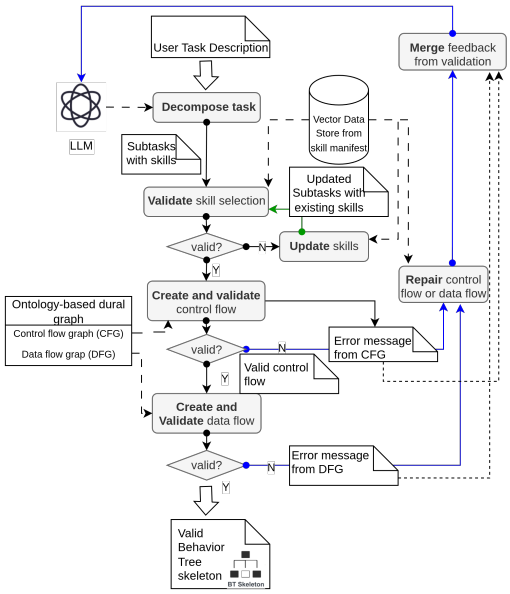


Fig. 3: Workflow for Behavior Tree skeleton creation. The LLM decomposes a natural-language task into subtasks, which are grounded to the skill manifest via a FAISS-based vector database. The corrected sequence is validated against ontology-derived control-flow and data-flow graphs. Validation errors trigger retrieval-based repair and re-generation until a consistent BT skeleton is obtained.

The workflow of BT skeleton generation is illustrated in Fig. 3 and formalized in Alg. 1. It proceeds in four steps:

1) **Task decomposition.** Given a natural-language task description and exemplar cases, the LLM decomposes the task into a sequence of subtasks, each specifying an execution order, a short description, and a recommended atomic skill.
2) **Existence check (RAG-1).** Each recommended skill is verified against the manifest. If the name does not exist or falls below the similarity threshold, Top-1 retrieval from the vector database replaces it with the closest valid manifest entry. This ensures that the sequence contains only valid skills before validation begins.
3) **Ontology-based construction and validation.** The grounded sequence is interpreted as a BT skeleton and validated with ontology-derived dual graphs. The control-flow graph (CFG) captures ordering constraints and enforces logical consistency as illustrated in Fig. 4 (bottom), where execution order is checked. Based on this ontology-derived CFG, the corrected sequence is transformed into a single valid control flow. The data-flow graph (DFG)

Algorithm 1 BT Skeleton Construction and Validation with RAG and Ontology-Based Dual Graphs

Require: Task description, skill manifest vector database, exemplar cases
Ensure: Verified BT skeleton

repeat
  $skills \leftarrow$ LLM_Decompose(task, examples)
  for each $skill$ in $skills$ **do**
    $cand, score \leftarrow$ RAG_Retrieve(skill, database, Top=1)
    if $score < \theta$ **then**
      $skill \leftarrow cand$
    end if
  end for
  $CFG \leftarrow$ ApplyOntologyCFG(skills)
  $DFG \leftarrow$ ApplyOntologyDFG(CFG)
  $feedback \leftarrow$ Validate(CFG, DFG)
  if $feedback == NOK$ **then**
    $candidates \leftarrow \{\}$
    for each $error$ in $feedback$ **do**
      $cand\_err \leftarrow$ RAG_Retrieve(error.skill, database, Top=k)
      $candidates[error.skill] \leftarrow cand\_err$
    end for
    $task \leftarrow$ Update_Task(task, candidates, feedback)
  end if
until $feedback == OK$
return BT_Skeleton(skills)

represents producer-consumer relations of typed inputs and outputs as shown in Fig. 4 (top). By applying the ontology-derived DFG, the single control flow is split into multiple consistent data flows. Together, the dual graphs not only construct the BT skeleton but also validate that it satisfies both typing and logical constraints.

4) **Repair with feedback (RAG-2).** If the dual-graph checks detect violations, structured error messages are converted into retrieval queries. The vector database then provides Top-k candidate replacements, which are combined with the error feedback and task context and re-injected into the LLM prompt. A corrected sequence is generated, which re-enters the construction and validation loop. This repair cycle continues until both CFG and DFG checks succeed.

The result of this iterative process is a manifest-grounded and ontology-guided BT skeleton that is not only checked for correctness but also systematically constructed to be executable, providing a reliable basis for parameter optimization.

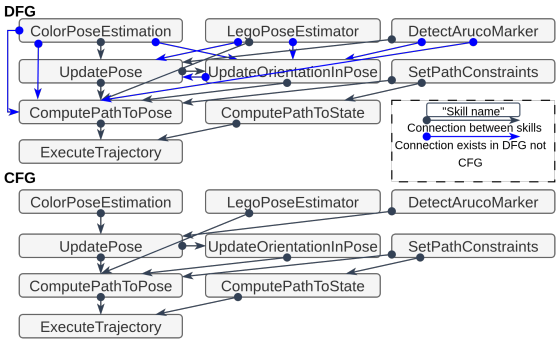


Fig. 4: Dual-graph used for construction and validation of application logic. Top: Data-flow graph ensures that outputs of one skill match inputs of subsequent skills. Bottom: Control-flow graph enforces correct ordering and BT hierarchy. Both graphs are derived from the skill manifest and used to validate generated BT skeletons.

B. Parameter Optimization with LLM Priors and MCTS-Based Search

After structural validation, the BT skeleton specifies a sequence of atomic skills, but many of these skills require parameters to be instantiated before execution. In the skill manifest, each atomic skill declares its required inputs, which fall into three categories. (i) Parameters fixed by the physical setup, such as robot identifiers, workspace limits, or gripper geometry, can be directly assigned. (ii) Parameters that depend on outputs of preceding skills are defined by the DFG and automatically bound to propagated variables. (iii) Remaining parameters are left unspecified at generation time; these include discrete design choices such as planner selection, constraint modes, or grasp offsets. This third category is critical for execution quality but cannot be set deterministically, and is therefore optimized in simulation.

Parameter optimization is formulated as a sequential decision problem over the BT skeleton, where each parameter choice influences subsequent feasibility and performance. MCTS is adopted due to its proven efficiency in long-horizon combinatorial spaces and its ability to balance exploration and exploitation [46]–[48]. The root of Monte Carlo Tree encodes the untuned BT, each depth corresponds to one parameter slot; edges represent candidate values; and a root-to-leaf path yields a fully instantiated BT to be executed in simulation. The overall optimization loop is illustrated in Fig. 5, consisting of *Selection* (traverse the Monte Carlo search tree to identify a promising branch), *Expansion* (add unexplored parameter candidates to the search tree), *Rollout* (simulate the corresponding parameterized BT to obtain a reward), and *Backpropagation* (propagate the rollout result to update statistics along the search path).

To reduce branching and focus exploration, a prior over candidate values is introduced for each parameter. Priors are obtained from a LLM conditioned on the validated BT skeleton, the skill manifest, and historical execution cases. The model produces a categorical distri-

bution $P(s,a)$ over candidate values together with rationales via Chain-of-Thought prompting [57], improving stability and consistency. The resulting *parameter prior table* serves as an informative heuristic for search.

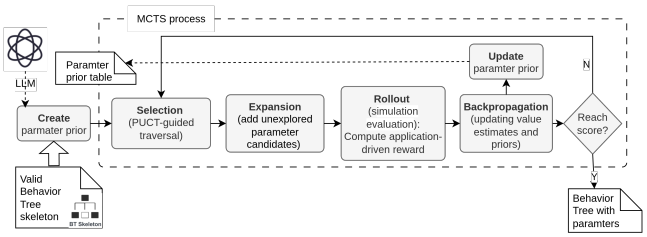


Fig. 5: Parameter optimization workflow. Starting from a validated BT skeleton and an LLM-derived prior table, MCTS iterates over *Selection*, *Expansion*, *Rollout*, and *Backpropagation*.

Selection uses the PUCT rule [46], [58] to balance exploitation and exploration:

$$\mathrm{PUCT}(s,a) = Q(s,a) + c \cdot P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)},$$

where $Q(s,a)$ is the average return from rollouts, $N(s,a)$ is the visit count, $P(s,a)$ is the prior probability of selecting action a in state s, and $c > 0$ is an exploration constant. In early stages of search, when visit counts $N(s,a)$ are small and empirical estimates $Q(s,a)$ are unreliable, branches with high prior probability $P(s,a)$ are favored. This focuses computation on semantically plausible configurations rather than spending effort uniformly. As rollouts accumulate, visit counts grow, reducing the influence of $P(s,a)$ and shifting the balance toward the empirical statistics $Q(s,a)$. PUCT thus provides a smooth transition: early exploration is guided by prior knowledge, while convergence in later stages is driven by statistically validated evidence. This property makes it well suited for coupled, long-horizon parameterization problems where initial guidance is critical but final robustness must be empirically verified.

Each candidate configuration (BT skeleton with parameters) is evaluated through a full rollout in simulation. The outcome is assessed by a reward function that separates binary task success from graded requirement satisfaction:

$$R = \begin{cases} r_{\text{fail}}, & \text{if task fails,} \\ R_{\max} - \sum_{i=1}^{m} \lambda_i \, \delta_i - \lambda_v \, \mathcal{V}, & \text{if task succeeds,} \end{cases}$$

where r_{fail} is a low penalty score discouraging infeasible parameterizations, $R_{\max}$ is the maximum attainable reward, δ_i denotes normalized deviations from requirement i (e.g., execution time, accuracy), λ_i is its weighting factor, $\mathcal{V}$ represents aggregated violations of hard constraints, and λ_v specifies the associated penalty weight. This design ensures that only configurations achieving success are rewarded, while among successful instances, those with fewer deviations and constraint violations receive higher scores. The formulation is application-agnostic and can be instantiated with domain-specific metrics.

After each rollout, node statistics are updated (Backpropagation) [59]:

$$W(s,a) \leftarrow W(s,a) + R, \quad N(s,a) \leftarrow N(s,a) + 1,$$
$$Q(s,a) = \frac{W(s,a)}{N(s,a)}.$$

Early-stopping criteria (e.g., stability of the incumbent score or a budget on simulations) can be used to terminate the search when improvements saturate.

A distinctive feature of this framework is that the prior distribution $P(s,a)$ is not fixed but refined continuously. Initial priors are provided by the LLM based on the validated BT skeleton, skill manifest, and historical cases. During search, rollout feedback is incorporated to adjust priors online, enabling a gradual transition from static, model-based beliefs to simulation-informed distributions Two update mechanisms can be integrated into the backpropagation stage:

- *Exponential Moving Average (EMA).* EMA updates the prior distribution smoothly by combining the existing prior with recent rollout outcomes. It emphasizes stability and fast adaptation:
$$P_{\text{new}}(a) = \alpha \cdot P_{\text{old}}(a) + (1-\alpha) \cdot \hat{p}(a),$$
where $\alpha \in [0,1]$ is the smoothing factor and $\hat{p}(a)$ reflects the empirical success frequency of parameter a. EMA is widely applied in reinforcement learning as a lightweight way to stabilize training dynamics [60].
- *Bayesian Sampling via Beta Posterior Updates.* When modeling success/failure outcomes, Bayesian updating with a Beta distribution provides principled, uncertainty-aware reasoning. After each rollout, successes and failures for parameter a are added to counts:
$$\alpha \leftarrow \alpha + \mathbb{I}[\text{success}], \quad \beta \leftarrow \beta + \mathbb{I}[\text{failure}],$$
yielding an updated estimate
$$P(a) = \frac{\alpha}{\alpha + \beta}.$$
This approach has been studied extensively in Bayesian reinforcement learning [61] and has been shown to improve search efficiency in tree-based planning via Bayesian Monte Carlo Tree Search (Bayes-MCTS) [62].

Both mechanisms enable a dynamic transition from static LLM-derived priors to simulation-informed distributions. A systematic comparison of EMA and Bayesian updates, including convergence behavior and robustness, is provided in Sec. V-C.

IV. Case Study: Pick-and-Place

To demonstrate the end-to-end workflow, a case study is conducted in both Ignition simulation (Fig. 2) and on a physical setup (Fig. 1). Both environments consist of a Yaskawa Motoman GP7 manipulator, a Robotiq 2F-85 gripper, and an Intel RealSense D435 depth camera.

A. BT Skeleton Generation and Validation

The user provides the following natural-language task description:

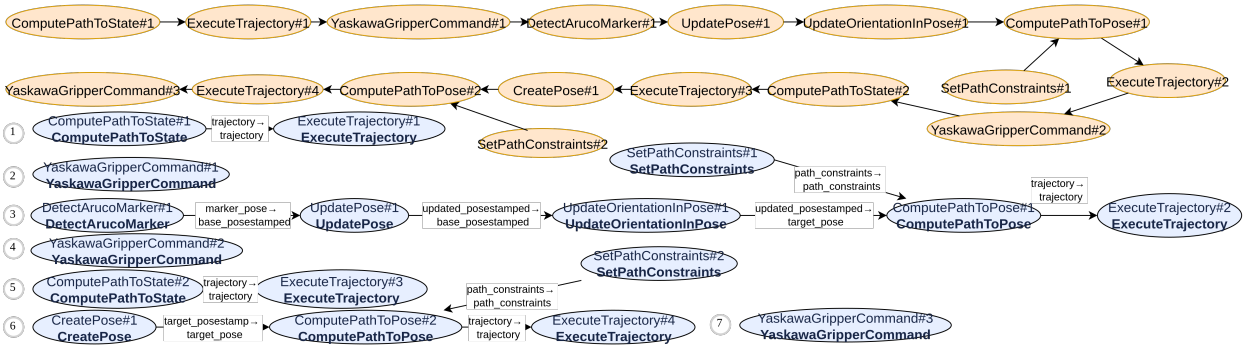


Fig. 6: Combined control-flow and data-flow representation of the case study. Orange oval nodes with arrows represent the control flow, defining valid execution order among atomic skills. Blue oval nodes with arrows represent the data flow, capturing producer-consumer relations of typed inputs and outputs. Each oval node corresponds to an instance of an atomic skill: the **bold text** indicates the skill type, and the suffix `#number` denotes its index in the order of appearance during execution. Arrows between orange nodes show control dependencies, arrows between blue nodes show data dependencies.

> "Use a Yaskawa GP7 robot named `gp7_arm` to pick up the cube labeled with ArUco marker ID 42 and place it at $(x = 0.6, y = 1.7, z = 1.3)$ in the world frame, with the gripper controlled by `YaskawaGripperCommand`."

The task is decomposed into subtasks by the LLM and grounded to the skill manifest through RAG. Each proposed atomic skill is checked against the skill manifest, and out-of-vocabulary names are replaced by retrieved entries. The corrected sequence is then validated through ontology-derived control-flow and data-flow graphs (Fig. 4). The CFG encodes application-independent ordering relations among skills, while checks producer-consumer relations among typed inputs and outputs. When violations are detected, error messages trigger retrieval of Top-k candidates and regeneration of the sequence until a consistent BT skeleton is produced. After CFG construction and validation, the sequence of subtasks is mapped into a single valid control flow (Fig. 6, orange nodes). And then DFG expands it into multiple data flows (Fig. 4, blue nodes). The validated BT skeleton is automatically exported to XML format.

B. Parameter Tuning with MCTS

After obtaining a verified BT skeleton, application-level parameters are optimized using the MCTS-based method described in Sec. III-B. In this case study, four trajectory planning instances are included: two `ComputePathToState` and two `ComputePathToPose` nodes. Each instance requires the choice of a planning pipeline, a specific planner identifier, and path constraints. The pipelines include the *OMPL* framework with multiple sampling-based planners (e.g. RRT, RRT*, PRM, etc.) and the *Pilz Industrial Motion Planner* with the PTP option. Instead of relying on manual selection, MCTS explores these parameters systematically, guided by LLM-derived priors. During each rollout, a candidate parameterization of the BT skeleton is executed in Gazebo simulation, and feedback is collected. Thanks to our skill-based and resilience-oriented framework design [63], failures are detected by built-in containment mechanisms within each atomic skill. For example, `ExecuteTrajectory` performs continuous collision checking and halts BT execution if a collision occurs, while `ComputePathToState` and `ComputePathToPose` terminate execution if no feasible trajectory is found.

The evaluation of each rollout follows the reward function introduced in Section III-B, which separates binary task success from graded requirement satisfaction. In this case study, the reward is instantiated as:

$$R = \begin{cases} r_{\text{fail}}, & \text{if } s = 0, \\ 100 - \min(50, \Delta d \times 5) - \min(30, \Delta t \times 2), & \text{if } s = 1, \end{cases}$$

where $s \in \{0, 1\}$ is the task success signal, $r_{\text{fail}} \in [0, 20]$ is a low penalty score for failed rollouts, Δd is the Euclidean placement error (cm), and Δt is the overtime in seconds. Penalties are capped at 50 and 30 points, respectively. For example, an offset of 6 cm and a delay of 12 s yield $R = 46$. This design ensures that only successful configurations receive high rewards, while among successes, those achieving higher accuracy and faster execution are preferred. An early stopping criterion is introduced: if a configuration achieved $R > 95$ for five consecutive rollouts, it is accepted and the search terminated. Backpropagation then updated the search tree statistics as described in Section III-B. A total of 300 iterations per round for 10 rounds are specified, yielding 3000 rollouts in principle, but due to caching of duplicate parameterizations only 354 unique rollouts are executed.

The evolution of planner priors under EMA updates is shown in Fig. 7 for `ComputePathToPose` and in Fig. 8 for `ComputePathToState`. For `ComputePathToPose`, both instances (#1 and #2) quickly converged to stable solutions. For both, the priors concentrated on `PTP`, with the *Pilz Industrial Motion Planner* clearly preferred over *OMPL* after ~ 5000 iterations. The planner distributions stabilized with high confidence (> 0.9 prior), indicating consistent preference for planner *PTP* from *Pilz Industrial Motion Planner* across independent runs. For `ComputePathToState`, the priors converge to *OMPL*. In our runs, #1 stabilizes on `BKPIECEkConfigDefault`, while

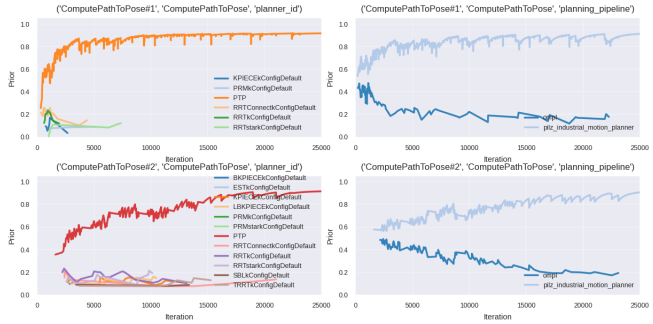
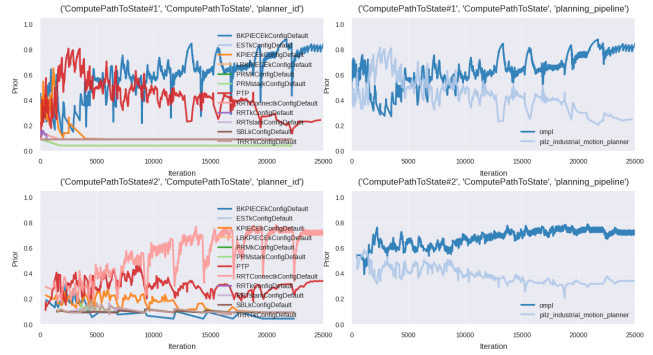
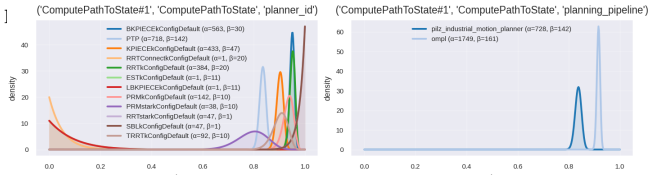


Fig. 7: Evolution of planner priors for `ComputePathToPose#1` and `ComputePathToPose#2` under EMA updates. Convergence occurs within ∼5k iterations, with OMPL pipelines dominating.



(a) Evolution of planner priors for `ComputePathToState#1` and `ComputePathToState#2` under EMA updates. Stable convergence is reached by ∼15k iterations, with consistent preference for OMPL



(b) Evolution of planner priors for `ComputePathToState#1` under Bayesian Sampling updates. Stable convergence is reached by ∼15k iterations, with consistent preference for OMPL planners.

Fig. 8: Two pictures

#2 converges to `RRTConnectkConfigDefault`. In both cases the *OMPL* pipeline is preferred over *Pilz Industrial Motion Planner*. The posteriors become sharply peaked after the early phase and remain stable, showing that EMA drives rapid and consistent settling on a small subset of planners.

V. Experiments

Beyond the case study, additional experiments are conducted to evaluate specific properties of the framework. All runs were executed in Ignition simulation with an iteration budget of 25 000 rollouts using the same reward function as defined in Section IV. Performance was assessed by the ability to produce executable trajectories and by the convergence of the reward over iterations.

A. LLM-only vs. MCTS

To examine the necessity of parameter tuning, we compared direct use of LLM-generated parameters against MCTS-optimized parameters. In this experiment, the BT skeleton was instantiated directly with parameters proposed by the LLM without further search, and execution was tested in Ignition simulation. For the MCTS baseline, the same skeleton was tuned over 25 000 rollouts using EMA updates.

Result: The LLM-only configuration consistently failed to produce valid trajectories, as parameter combinations violated feasibility constraints. In contrast, MCTS tuning identified feasible parameters, yielding executable motion plans. This demonstrates that search is required to bridge the gap between language generation and reliable execution.

B. Effect of LLM Priors

To evaluate the role of priors, we compared two initialization strategies for MCTS: (i) initializing with LLM-derived prior distributions stored in the parameter prior table, and (ii) initializing with uniform weights over all candidate parameters. Both were run for 25 000 rollouts under otherwise identical conditions.

Result: Using LLM priors accelerated convergence by more than 40% compared to uniform initialization. Without priors, exploration wasted rollouts on infeasible planners, while priors guided the search toward semantically plausible options.

C. EMA vs. BS Updates

We compared two update strategies for refining priors during Backpropagation. In both cases, the same BT skeleton and reward function were used, and each method was run for 25 000 rollouts. The Exponential Moving Average (EMA) update smoothed probabilities using recent outcomes, while Bayesian Sampling (BS) updated probabilities via a Beta distribution over successes and failures.

Result: EMA reached the desired reward threshold faster, highlighting its bias toward rapid convergence. However, BS maintained broader exploration and achieved higher final rewards in cases where EMA converged prematurely. This reveals a trade-off between convergence speed (EMA) and robustness against local optima (BS).

EMA is computationally efficient and effectively smooths prior estimates in dynamic contexts [60]. In contrast, Bayesian updating offers statistical rigor and preserves uncertainty, enabling robust exploration in high-dimensional and noisy environments [61], [62]. By comparing EMA and Bayesian updating, the framework assesses a trade-off between convergence speed and statistical robustness. EMA can rapidly refine priors in near-stationary settings, while Bayesian sampling provides calibrated uncertainty modeling, which is especially valuable during early search when evidence is sparse.

D. Different LLMs

To test whether the framework depends on a specific language model, we replaced GPT-5 with GPT-4.0-mini when generating parameter priors. Both models were used to initialize MCTS runs with the same BT skeleton and reward function.

Result: Both LLMs produced nearly identical prior distributions and converged to equivalent parameterizations after MCTS tuning. This indicates that the framework is robust to the choice of LLM backbone. At the current stage of development, the output is a linear sequence of skills rather than a fully hierarchical Behavior Tree.

A remaining challenge is bridging sim-to-real discrepancies in parameterized strategies, such as grasp offsets that depend on object size and geometry. Future work will explore adaptive tuning mechanisms that adjust application-level parameters online, enabling robust transfer across varying physical conditions.

ACKNOWLEDGMENT

This work has received funding from the European Unions Horizon Europe research and innovation programme under grant agreement No. 101070254 CORESENSE. Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union or the Horizon Europe programme. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *IJCAI*, 1971.

[2] Drew McDermott et al. Pddl: The planning domain definition language. Technical report, AIPS-98 Planning Competition Committee, 1998.

[3] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[4] Caelan R. Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:265–293, 2021.

[5] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32(9-10):1194–1227, 2013.

[6] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*, 37(10):1134–1151, 2018.

[7] Michael Görner, Robert Haschke, Helge Ritter, and Jianwei Zhang. Moveit! task constructor for task-level motion planning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 190–196, 2019.

[8] Kirill Dorofeev. Skill-based engineering in industrial automation domain: Skills modeling and orchestration. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 158–161, 2020.

[9] Taneli Lohi, Samuli Soutukorva, and Tapio Heikkilä. Programming of skill-based robots. In *2024 IEEE 19th Conference on Industrial Electronics and Applications (ICIEA)*, page 17. IEEE, August 2024.

[10] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances. 2022.

[11] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Stripstream: Integrating symbolic planners and blackbox samplers. *CoRR*, abs/1802.08705, 2018.

[12] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 639–646, 2014.

[13] Wenlong Huang et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[14] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[15] Matthias Mayr, Francesco Rovida, and Volker Krueger. Skiros2: A skill-based robot control platform for ros. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6273–6280. IEEE, 2023.

[16] Neil T. Dantam, Zachary Kingston, and Lydia E. Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems*, 2016.

[17] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. 2023.

[18] Riccardo Andrea Izzo, Gianluca Bardaro, and Matteo Matteucci. Btgenbot: Behavior tree generation for robotic tasks with lightweight llms. *CoRR*, abs/2403.12761, 2024.

[19] Xiaolei Chen, Junjie Shen, Haiyan Liu, and Zhejun Jin. Integrating intent understanding and optimal behavior tree execution architecture. In *IJCAI*, 2024.

[20] Jonathan Styrud, Matteo Iovino, Mikael Norrlöf, Mårten Björkman, and Christian Smith. Automatic behavior tree expansion with llms for robotic manipulation. *CoRR*, abs/2409.13356, 2024.

[21] Subbarao Kambhampati. Large language models are not zero-shot planners. *arXiv preprint arXiv:2402.01817*, 2024.

[22] Karthik Valmeekam et al. Large language models still cant plan: A benchmark for llms on planning. In *NeurIPS Datasets and Benchmarks*, 2023.

[23] Karthik Valmeekam and Subbarao Kambhampati. On the planning abilities of large language models: A critical evaluation. *arXiv preprint arXiv:2401.01923*, 2024.

[24] Anthony Brohan et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[25] Anthony Brohan et al. Rt-2: Vision-language-action models transfer web knowledge to robotics. *arXiv preprint arXiv:2307.15818*, 2023.

[26] Daniel Driess et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[27] Yilun Ma et al. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[28] Mohit Shridhar et al. Peract: Perceiver-actor for language-conditioned manipulation. In *CoRL*, 2022.

[29] Eric Jang et al. Bc-z: Zero-shot task generalization with robotic imitation learning. In *CoRL*, 2021.

[30] RT-X Team. Open x-embodiment: Robotic learning with diverse datasets and robots. *arXiv preprint arXiv:2310.08864*, 2023.

[31] Konstantinos Bousmalis et al. Robocat: A self-improving robotic agent. *arXiv preprint arXiv:2306.11706*, 2023.

[32] Russell French et al. Learning behavior trees from demonstration. In *ICRA*, 2019.

[33] Gregory Suddrey et al. Learning and executing re-usable behaviour trees from demonstrations. *arXiv preprint arXiv:2106.01650*, 2021.

[34] Utsav Neupane et al. Designing behavior trees from goal-oriented ltl_f formulas. In *ICRA*, 2023.

[35] Erik Scheide et al. Behavior tree learning for robotic task planning through monte carlo dag search. In *ICRA*, 2021.

[36] Weijia Hong et al. Formal verification based synthesis for behavior trees. In *SETTA*, 2023.

[37] Moritz Tenorth and Michael Beetz. Knowrob: Knowledge processing for autonomous robots. *IROS*, 2009.

[38] Michael Beetz et al. Cram 2.0: A cognitive architecture for everyday manipulation. *arXiv preprint arXiv:2306.07073*, 2023.

[39] Ashutosh Saxena et al. Robobrain: Large-scale knowledge engine for robots. *arXiv preprint arXiv:1412.0691*, 2014.

[40] Oliver Biggar and Mohammad Zamani. A framework for formal verification of behavior trees with linear temporal logic. In *ICRA*, 2020.

[41] Benjamin Serbinowski et al. Verifying temporal logic specifications for behavior trees. In *SEFM*, 2022.

[42] Felix Berkenkamp, Andreas Krause, and Angela P. Schoellig. Bayesian optimization with safety constraints: Safe and automatic parameter tuning in robotics, 2020.

[43] Md Abir Hossen, Sonam Kharade, Jason M. O'Kane, Bradley Schmerl, David Garlan, and Pooyan Jamshidi. Cure: Simulation-augmented auto-tuning in robotics, 2025.

[44] Sheng Cheng, Minkyung Kim, Lin Song, Chengyu Yang, Yiquan Jin, Shenlong Wang, and Naira Hovakimyan. Difftune: Auto-tuning through auto-differentiation, 2024.

[45] Carlotta Sartore, Marco Rando, Giulio Romualdi, Cesare Molinari, Lorenzo Rosasco, and Daniele Pucci. Automatic gain tuning for humanoid robots walking architectures using gradient-free optimization techniques, 2024.

[46] Levente Kocsis and Csaba Szepesvári. Bandit-based monte carlo planning. In *ECML*, 2006.

[47] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *NeurIPS*, 2010.

[48] Cameron B. Browne et al. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[49] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training, 2024.

[50] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Multilingual e5 text embeddings: A technical report, 2024.

[51] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[52] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

[53] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613620, November 1975.

[54] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[55] Vladimir Karpukhin, Barlas Ouz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering, 2020.

[56] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024.

[57] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837, 2022.

[58] David Silver, Aja Huang, Chris J. Maddison, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[59] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[60] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2nd edition, 2018.

[61] Mohammed Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Convex optimization: Algorithms and complexity. *Foundations and Trendső in Machine Learning*, 8(56):359483, 2015.

[62] Gerald Tesauro, V T Rajan, and Richard Segal. Bayesian inference in monte-carlo tree search, 2012.

[63] Ruichao Wu, Andrey Morozov, and Björn Kahl. Enhancing resilience in robotic systems through self-awareness and adaptive recovery.