2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

From Language to Deployment: Offline Optimization and Ontology-Guided Behavior Tree Generation for Transparent Robot Applications

Ruichao Wu*, Jiwei Pan*, Mohamed Youssef‡, Björn Kahl§, Werner Kraus¶ and Andrey Morozov‖

***Abstract*— Automatically generating robot applications from natural language promises to lower the barrier to automation, but remains difficult in domains that demand reliability and transparency, such as industrial assembly or collaborative manipulation. End-to-end policies and large language model (LLM)-based planners can map instructions to robot behaviors, but they often lack interpretability and provide limited assurance of correctness. We present a framework that composes applications from modular, application-independent atomic skills expressed as Behavior Trees (BTs). BTs are constructed and validated against an ontology-level dual graph to enforce control-flow and data-flow consistency before execution, ensuring transparency and structural correctness. Application-level parameters are optimized offline in simulation using Monte Carlo Tree Search guided by LLM-derived priors. Rather than serving as a runtime optimizer, this process systematically explores interdependent parameters, producing a dataset of reliable parameterizations. This dataset could later be used to train gating mechanisms for online adaptation. The framework is validated in a physical robotic setup, demonstrating transparent and consistent offline generation of deployable applications, and laying the foundation for adaptive, real-time systems.**

I. Introduction

Robots are increasingly expected to perform complex tasks specified in a natural language. This capability is attractive for industrial and collaborative settings, where non-experts should be able to program robots without low-level coding. However, resilient execution remains a fundamental challenge: language-derived task representations must be valid, parameter choices must generalize across conditions, and execution must be robust to variation in instructions and sim-to-real transfer.

The research question we address is: *How can natural-language instructions be transformed into robotic task executions that are both* valid *and* reliable*, ensuring correctness of task structure, robust parameterization, and resilience across linguistic and environmental variation?*

Existing approaches address parts of this problem but leave important gaps. End-to-end policy learning achieves language grounding but sacrifices transparency and verifiability [1–3]. Program-synthesis methods guided by LLMs generate task structures but often lack systematic validation, leading to fragile executions [4–6]. Parameter optimization is frequently ad hoc and computationally demanding [7, 8], while robustness to paraphrasing and explicit evaluation of sim-to-real parameter transfer remain limited in language-driven pipelines [9–11]. As a result, current pipelines emphasize accessibility but fall short of the reliability required for deployment in industrial settings, where transparency and correctness are as important as accessibility.

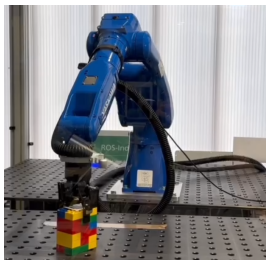

Fig. 1: Physical setup with a Yaskawa GP7 manipulator, Robotiq 2F-85 gripper, and Intel RealSense D435 camera.

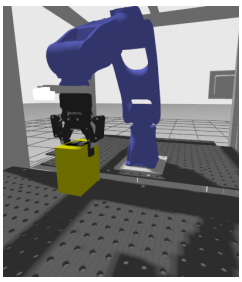

Fig. 2: Simulation setup in Gazebo Ignition with the same hardware setup as the physical system.

We address this gap with a modular framework that couples LLM-driven task generation with ontology-guided validation and offline search-based optimization (Fig. 3). Applications are represented as BTs, providing transparent, human-readable task logic. The dual-graph validation is conceptually inspired by the Dual Graph Error Propagation Model (`DEPM`), which models error propagation in mechatronic systems by separating control-flow and data-flow layers [12]. While `DEPM` targets fault activation and error propagation analysis, we extend the dual-graph principle to enforce structural correctness in language-driven task generation for robotics. A repair loop further ensures robustness against natural-language variation by automatically correcting invalid or ambiguous specifications. Finally, we address parameter optimization, which prior language-driven frameworks often handle heuristically. Our approach integrates Monte Carlo Tree Search (MCTS) with LLM-initialized priors. Rather than serving as a runtime optimizer, MCTS is applied offline in simulation to systematically explore interdependent parameters. This process both yields deployable BT parameterizations and generates a dataset of reliable parameter sets that can support future gating mechanisms for online adaptation.

To evaluate the framework, we conduct experiments in simulation (Fig. 2) and on a Yaskawa GP7 manipulator (Fig. 1), demonstrating that entire application generated

*Ruichao Wu, §Björn Kahl and ¶Werner Kraus are with Fraunhofer Institute for Manufacturing Engineering and Automation (IPA), Nobelstr. 12, 70569 Stuttgart, Germany `ruichao.wu@ipa.fraunhofer.de`, `bjöern.kahl@ipa.fraunhofer.de` and `werner.kraus@ipa.fraunhofer.de`

*Jiwei Pan and Andrey Morozov are with the Institute of Industrial Automation and Software Engineering, University of Stuttgart, Pfaffenwaldring 47, 70550 Stuttgart, Germany `st176838@stud.uni-stuttgart.de`, `andrey.morozov@ias.uni-stuttgart.de`

‡Mohamed Youssef is with Institute of Computer Science, Electrical Engineering a nd Information Technology, University of Stuttgart, Pfaffenwaldring 47, 70550 Stuttgart, Germany `st190193@stud.uni-stuttgart.de`

Ruichao Wu and Jiwei Pan contributed to this work equally

in simulation can be transferred successfully to real hardware. Another case study on a UR5e robot, conducted in simulation, further illustrates the portability and generality of the framework across different robotic platforms. While our current evaluation focuses on sequential BTs and offline optimization, these results demonstrate the feasibility of transparent, ontology-validated task generation and provide the foundation for runtime-adaptive extensions.

Contribution: This paper introduces a framework for dependable and reliable language-driven robot programming with three main contributions:

1) *Generality:* Applications are composed from atomic, application-independent skills, enabling systematic reuse across robots and tasks rather than relying on task-specific libraries.
2) *Transparency and assurance:* Applications are represented as BTs, providing human-readable task logic. The dual-graph principle from mechatronic reliability analysis is adapted to an ontology-guided validation mechanism that enforces both control-flow and data-flow consistency, while supporting retrieval-based repair of invalid logic.
3) *Offline parameter optimization:* Application-level parameters are explored in simulation with MCTS guided by LLM-derived priors. While not used for runtime optimization, this offline process systematically captures long-horizon dependencies across skills and produces a dataset that supports both current deployment and future learning-based gating mechanisms.

II. State of the Art

Language-driven robot programming splits into three dominant strands, end-to-end vision-language-action (VLA) models, program-synthesis pipelines, and task and motion planning (TAMP). Each offers partial solutions but leaves gaps that our work addresses.

End-to-end approaches such as SayCan [1], RT-2 [13], and PaLM-E [14] demonstrate broad generalization from web-scale pre-training to control. However, their decision process is opaque and cannot be validated pre-execution, which limits their applicability in safety-critical settings. In contrast, our framework prioritizes transparency by generating BTs that can be automatically validated against ontological constraints before deployment, trading raw generality for interpretability and structural assurance.

TAMP integrates symbolic reasoning with geometric feasibility to produce theoretically correct task plans [15]. However, it typically requires extensive domain modeling and performs both symbolic and geometric planning offline [16], which makes adaptation to dynamic environments difficult. Our framework reduces this modeling burden by reusing typed skill manifests and ontology-guided validation instead of exhaustive symbolic- geometric models. A key distinction is that we retain geometric planning (trajectory generation) online, using MCTS to tune parameters such as joint constraints and planner selection offline to adapt application scenarios. This separation makes our approach more flexible than TAMP: geometric feasibility is handled by standard motion planners at runtime, while long-horizon dependencies across parameters are explored systematically offline. Nevertheless, similar to TAMP, our method still requires domain-specific manifests and currently lacks advanced robust online adaptation.

Recent systems map language to BT structure. *BT-GenBot* fine-tunes compact LLMs on command-BT pairs and releases code. But it explicitly notes that validating LLM-generated behaviors is challenging and their custom validator works only when a known solution exists. Our framework directly targets this gap by enforcing and repairing BT structure with an *ontology-guided dual graph* (CFG + DFG) *before* deployment, and by using *MCTS* offline to narrow the executability gap through systematic parameter search. *LLM-MARS* adapts Falcon-7B with LoRA to generate and discuss BTs for multi-robot tasks [17]; it emphasizes dialogue and multi-agent execution but does not provide manifest-wide typed validation, automated repair, or parameter optimization. *OBTEA* parses instructions into first-order goals and then constructs BTs via optimal expansion [18], offering strong goal-satisfaction guarantees tied to the domain's goal logic; however, its BT nodes are high-level APIs. In contrast, we compose *atomic, application-independent skills* from a manifest, validate both control and typed data dependencies across the entire library, and then optimize the many interdependent low-level parameters that determine execution quality. In principle, our pipeline could wrap around BTs generated by *BTGenBot*, *LLM-MARS*, or *OBTEA*; direct comparison on BT generation alone is therefore less informative for evaluating the assurance and optimization dimensions that are central to this work.

Another gap concerns parameter tuning. Most language-driven pipelines rely on default parameters or manual adjustment, which hinders portability across robots and environments [1, 6]. Search-based methods such as MCTS have shown strong performance in long-horizon decision making [7, 8], but have rarely been combined with language-derived priors for robotics. In our framework, MCTS is applied offline to refine LLM-initialized priors, systematically exploring interdependent parameters and capturing long-horizon dependencies across skills. Rather than serving as a runtime optimizer, this process produces a dataset of parameterizations that supports current deployment and lays the groundwork for lightweight gating mechanisms in future online adaptation.

A further limitation is that many LLM-based robot programming workflows retain humans in the loop for validation or correction, reflecting broader findings that LLMs struggle with autonomous long-horizon planning and benefit from external verification [5, 19, 20]. Our framework reduces this dependency by automating assurance through ontology-guided dual-graph validation, while still retaining Behavior Trees as a transparent representation that humans can inspect when necessary.

At the system level, this work extends upon and realizes the concept proposed in [21], which proposed a skill-based architecture for reliable robotic execution. That work introduced the idea of modular skill chains with local fault containment as a foundation for resilience. Meanwhile, we extend it toward language-driven programming by generating transparent BTs from natural language, validating and repairing them with a dual-graph mechanism, and systematically tuning their parameters with MCTS-guided search.

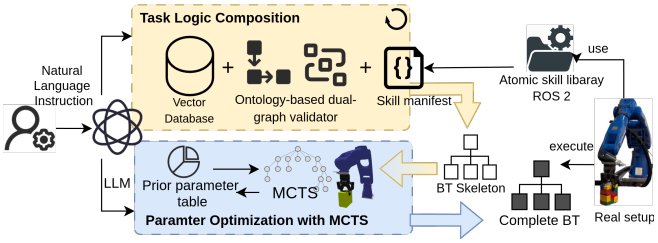

Fig. 3: Pipeline of the proposed framework. Natural-language instructions are parsed by an LLM, whose proposed skills are grounded through skill-manifest-based retrieval to ensure consistency with the skill manifest. The resulting BT skeleton is validated against ontology-based dual graphs, optimized by MCTS with backpropagation updates, and deployed for execution with fault containment at the atomic skill level.

III. System Design

The proposed framework transforms natural-language task descriptions into verified and deployable robot applications. As illustrated in Fig. 3, the process consists of two main stages: (1) task-logic composition, in which an LLM proposes a BT skeleton grounded to the skill manifest and validated with ontology-based dual graphs, and (2) offline parameter optimization, where MCTS explores discrete parameters in simulation to identify robust configurations and generate data for deployment. We note that optimization is performed offline; runtime adaptation remains future work.

A. Task Logic Composition

A prerequisite for automated application generation is a formal description of the system's functional capabilities. These are encoded in a *skill manifest*, which serves as the machine-readable presentation of our skill library. The skill library is the core component of the base framework introduced in Sec. II, providing reusable atomic skills that can be executed both in simulation and on the physical robot. Each atomic skill defines a self-contained capability (e.g., `ComputePathToPose`, `DetectArucoMarker`) and the manifest specifies its interface through a natural-language description, required parameters, and input/output types. The manifest therefore provides the contract between natural-language instructions and the low-level primitives available to the robot. While this requires upfront modeling effort, it is substantially lighter than full symbolic or geometric domain models required by TAMP.

Fig. 4 illustrates the four stages of task-logic composition.

1) Task Decomposition (Step 1 in Fig. 4): Given a natural-language task description, the LLM proposes a sequence of subtasks, each associated with candidate skills and tentative connections. The LLM is guided by a few-shot prompt constructed from the skill manifest and historical BT examples, which improves plausibility but does not guarantee correctness. The proposed sequence is therefore treated strictly as a hypothesis to be checked in subsequent stages.

2) Skill Selection Validation (Step 2 in Fig. 4): This step maps LLM-proposed skill references to the most similar manifest entries, ensuring that only available skills are used. Because natural language may contain synonyms, abbreviations, or minor mismatches, retrieval-augmented generation (RAG) is employed. Each manifest entry is embedded into a vector database (at atomic-skill granularity) by using an E5-family model [22, 23]

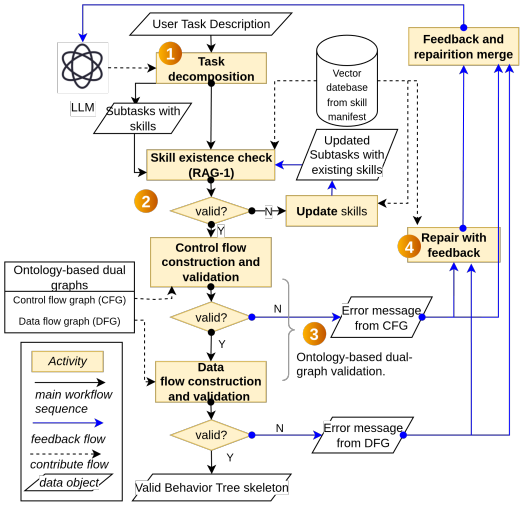

Fig. 4: Workflow for Behavior Tree skeleton creation. The four numbered stages correspond to the four steps described in Sec. III-A

and indexed with FAISS [24, 25]. At query time, LLM-proposed skills are embedded and compared against the database using cosine similarity:

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

The Top-k most similar entries are returned, while a similarity threshold θ filters out low-confidence matches [26–28]. This mapping prevents hallucinations and ensures that every node is bound to an actual skill entry.

3) Ontology-Based Dual-Graph Validation (Step 3 in Fig. 4): The ontology-based dual graphs consist of a CFG and DFG, both derived from the ontology encoded in the skill manifest. The CFG captures admissible ordering and BT hierarchy, while the DFG captures typed producer-consumer bindings between ports. Fig. 5 shows a partial excerpt of these ontology-based dual graphs; the full skill library contains many more nodes and edges.

For a given proposal, task-specific subgraphs are induced by querying the ontology-based dual graphs in `Neo4j`* with the retrieved skills and ports. The task-specific CFG is checked for reachability, ordering consistency, and acyclicity, ensuring compliance with ontology-defined constraints (e.g., a path must be planned before execution). The task-specific DFG is checked for type- and arity-correct producer-consumer bindings. Because the DFG captures all feasible data bindings, the induced subgraph may consist of multiple independent flows, each starting at an instance that consumes only external inputs and ending at an instance whose outputs are not consumed by subsequent skills. This combined validation ensures both logical consistency and complete variable binding before parameter optimization. In the present work, we restrict to sequential BTs; extending validation to full BT expressiveness (e.g., fallbacks, retries, parallel nodes) is left for future work.

If both validations succeed, the sequence of instantiated atomic skills is accepted as a BT skeleton (without parameters) and forwarded to the parameter optimization stage. Otherwise, the repair loop is triggered.

*`https://github.com/neo4j/neo4j?tab=readme-ov-file`

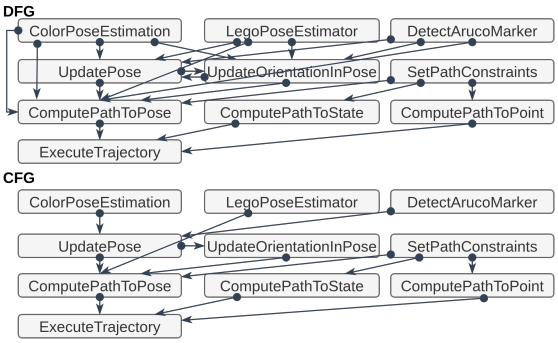

Fig. 5: Partial ontology-based dual graphs derived from the skill manifest. **Top:** data-flow graph (DFG) with arrows indicating producer→consumer bindings between typed ports. **Bottom:** control-flow graph (CFG) with arrows indicating admissible ordering/hierarchy relations between skills.

4) Repair with Feedback (Step 4 in Fig. 4): When dual-graph validation fails, structured error messages are returned (e.g., cycle detected, undefined variable, or type mismatch). These errors are transformed into retrieval queries, and candidate replacements are obtained from the manifest and ontology. The candidates, combined with the error context, are injected into a new LLM prompt. The revised proposal is then re-grounded and re-validated. This cycle repeats until both CFG and DFG checks succeed, yielding a consistent BT skeleton.

B. MCTS-based Parameter Optimization

After dual-graph validation, the BT skeleton specifies a sequence of atomic skills, but many of these require parameters before execution. In the skill manifest, parameters fall into three categories: (i) constants fixed by the physical setup, such as robot identifiers, workspace limits, or gripper geometry, (ii) parameters automatically bound via the DFG to outputs of preceding skills, and (iii) task-dependent parameters left unspecified at generation time, such as trajectory planner selection, constraint modes, or grasp offsets. The third category is critical for execution quality and is optimized in simulation.

Parameter optimization in our framework is a sequential decision problem over the BT skeleton: each parameter choice not only affects immediate feasibility but also constrains the options available for subsequent skills. This creates a long-horizon, combinatorial search space with strong dependencies between parameters. MCTS is particularly suitable for this setting, as it represents the search space explicitly as a tree and balances exploration and exploitation through rollouts [29–31]. Unlike greedy or random strategies, which may converge prematurely, MCTS builds statistics across rollouts to progressively refine exploration toward promising parameter regions [7, 31], while still allowing novel configurations to be explored. This makes it well matched to BT parameterization, where the space is too large for exhaustive enumeration but too structured for purely stochastic search.

The search space is represented as a Monte Carlo tree. The root corresponds to the validated but unconfigured BT skeleton. Each depth level represents a parameter slot of an atomic skill instance, edges encode candidate values, and a root-to-leaf path yields a fully instantiated BT configuration to be executed in simulation. When a skill instance requires multiple parameters, they are expanded sequentially as consecutive layers of the tree, ensuring that dependencies are respected while allowing systematic exploration.

Fig. 6 illustrates the four stages of the optimization loop.

1) `Selection` (Step 1 in Fig. 4): `Selection` determines which parameterization path (a sequence of values for unresolved parameters of atomic skill instances) should be explored next. This is a long-horizon, combinatorial problem: early parameter choices strongly constrain downstream feasibility, making naive exploration inefficient. To address this, we adopt the PUCT rule, originally developed in AlphaGo and AlphaZero to integrate policy priors into tree search [8, 32]. PUCT combines prior knowledge with empirical rollout statistics, making it well suited for our scenario where LLM-derived priors provide semantic guidance at the start of search but final parameterizations must be validated empirically.

Formally, each partially parameterized BT corresponds to a state s, and each outgoing edge represents a candidate value for the next parameter. `Selection` traverses the tree by choosing one candidate at each level according to:

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

Here $Q(s,a)$ is the mean reward from previous rollouts, $N(s,a)$ is the visit count, and $\sum_b N(s,b)$ is the total visits to s. The prior $P(s,a)$ is obtained once from an LLM conditioned on the validated BT skeleton, the skill manifest, and historical execution cases. To improve stability and consistency, the LLM produces both categorical probabilities and rationales via Chain-of-Thought prompting [33]. These priors bias early exploration toward semantically plausible values, while empirical statistics (Q, N) dominate as rollouts accumulate.

2) `Expansion` (Step 2 in Fig. 4): When `Selection` reaches a state s with unexplored parameters, new edges are added to represent candidate values for that atomic skill instance. `Expansion` must balance between exploiting values that appear promising and ensuring that rarely chosen options are not ignored. For this reason, we employ an ϵ-greedy strategy inspired by recent work in MCTS [34]: with probability $1-\epsilon$ the most promising unexplored candidate is expanded, while with probability ϵ another candidate is chosen at random. This explicit randomization complements the PUCT-driven `Selection` step by ensuring diversity at the frontier of the tree, which is particularly important in our scenario where each skill may expose multiple discrete parameters and unvisited branches could contain valid configurations.

3) `Rollout` (Step 3 in Fig. 4): When `Selection` reaches a leaf node (a fully parameterized BT), The instantiated BT is executed in simulation to measure its performance. The reward is computed as:

$$R = \begin{cases} r_{\text{fail}}, & \text{if task fails,} \\ R_{\max} - \sum_{i=1}^{m} \lambda_i\,\delta_i - \lambda_v\,\mathcal{V}, & \text{if task succeeds,} \end{cases} \quad (2)$$

where r_{fail} penalizes infeasible configurations, δ_i are normalized deviations from requirement i with weights λ_i, and $\mathcal{V}$ denotes hard-constraint violations with penalty λ_v. This ensures that only successful executions are rewarded, and among them, those better satisfying requirements receive higher scores.

4) `Backpropagation` (Step 4 in Fig. 4): The `Rollout` result is propagated back along the path to update

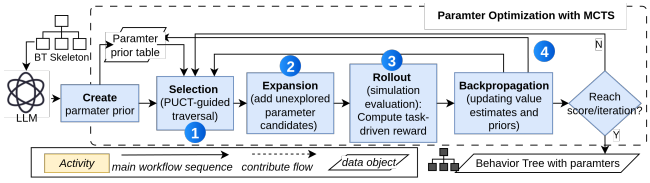

Fig. 6: Parameter optimization workflow. Starting from a validated BT skeleton and an LLM-derived prior table, MCTS iterates over *Selection*, *Expansion*, *Rollout*, and *Backpropagation*.

cumulative rewards $W(s,a)$, visit counts $N(s,a)$, and mean returns $Q(s,a)$. In addition, the prior distribution $P(s,a)$ is initialized once from the LLM and then refined continuously during search. Two alternative mechanisms are evaluated for refining the prior distribution $P(s,a)$ during search: (1) an *Exponential Moving Average (EMA)*, which smoothly integrates empirical outcomes into the prior while maintaining stability, and (2) a *Bayesian update* with a Beta posterior, providing an uncertainty-aware estimate of success probabilities. This continuous refinement enables a gradual transition from semantic guidance at the beginning to simulation-grounded evidence as rollouts accumulate. These mechanisms are evaluated in Sec. V.

IV. Case Study: Pick-and-Place

The physical evaluation is intended as a proof-of-concept validation rather than a large-scale statistical study.

A. Industrial Cell with Randomized Object Placement (Primary)

We instantiate the pipeline on a Yaskawa GP7 cell where a cube attached with an ArUco marker is randomly placed in front of the robot at the start of each run. The LLM model used is GPT-4o-mini, accessed via the Azure OpenAI service. Pick-and-place is selected as a canonical benchmark: while simple, it stresses both task-logic correctness (grasping vs. placing order) and parameter sensitivity (trajectory planners, constraints, offsets), making it a representative test for sim-to-real dependability.

1) BT Skeleton Generation and Validation: The user provides the following natural-language task description:

> "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. 5). 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. 7, orange nodes). And then DFG expands it into multiple data flows (Fig. 5, blue nodes). The validated BT skeleton is automatically exported to XML format.

Importantly, while the skeleton enforces structural correctness, it does not guarantee robust execution. Many skills expose interdependent parameters whose choice determines whether the task succeeds, highlighting the need for the MCTS optimization stage.

2) 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. Rather than relying on manual selection, MCTS explores these parameters systematically, guided by LLM-derived priors. `Selection` in MCTS follows the PUCT rule (Eq. 1) introduced in Sec. III-B.

Setup. In our experiments, the exploration constant is set to $c = 1.5$ (with $c = 3.0$ also tested). For EMA, we use a budget of $N = 300$ iterations per round over 5 rounds (1500 root visits total), reusing cached results across rounds to stabilize the moving average. Each selection step corresponds to traversing the tree and choosing a parameter branch; when the branch has been explored before, the cached evaluation is reused. As a result, 802 *unique rollouts* are actually executed in Ignition, while the remaining iterations reused cached results. For Bayesian updates, the posterior distribution is explicitly uncertainty-aware and does not require multiple rounds for stability. We therefore run a single round with $N = 500$ iterations, which suffices to demonstrate rapid convergence while keeping the number of unique Ignition rollouts ($\sim$342) comparable to the EMA setting. This difference reflects the characteristics of the two update mechanisms: EMA benefits from repeated passes to smooth fluctuations, whereas Bayesian updating concentrates probability mass effectively within a shorter search horizon.

During each rollout, a candidate parameterization of the BT skeleton is executed in Ignition, and feedback is collected. Based on the design of base framework, 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 Sec. III-B, which separates binary task success from graded requirement satisfaction. In this case study, the reward is instantiated for the success run as:

$$R = 100 - \min(50, 5\Delta d) - \min(30, 2\Delta t), \qquad (3)$$

where 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.

Results. To illustrate parameter convergence, we high-

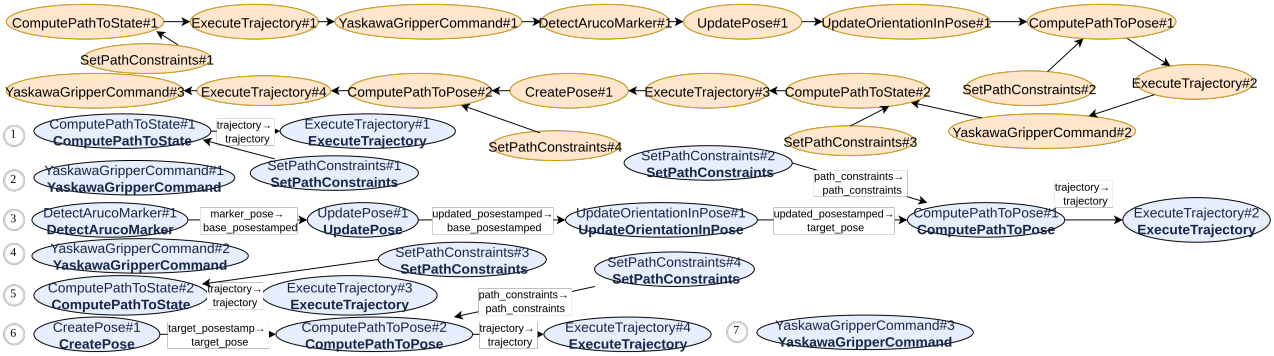

Fig. 7: Combined control-flow and data-flow representation of the case study. Orange oval nodes with arrows represent the control flow, defining valid execution order among atomic skills. Blue oval nodes with arrows represent the data flow, capturing producer-consumer relations of typed inputs and outputs. 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.

light two representative instances: `ComputePathToPose#1` (planning the trajectory toward the pick position) and `ComputePathToState#2` (planning the trajectory after grasping, when the robot carries the object). These motions are critical: the first requires avoiding large unnecessary movements when approaching the cube, while the second requires smooth and stable motion to prevent the grasped cube from falling.

Fig. 8 shows the posterior evolution across iterations. In both cases, Bayesian updating rapidly concentrates probability mass on planners with consistent performance, while gradually eliminating poor candidates. For `ComputePathToPose#1`, the distribution converges to favor the Pilz PTP option, reflecting its reliability in generating short, direct trajectories toward the grasp. For `ComputePathToState#2`, the distribution stabilizes on OMPL-based planners (notably RRTConnect), reflecting their ability to generate feasible and smooth paths when carrying the cube. The comparison against EMA updates (Fig. 9) shows that EMA requires more iterations to discard low-performing candidates and plateaus more slowly, whereas Bayesian updating achieves sharper convergence with fewer samples.

B. UR5e Case Study (Portability)

To assess portability, we replicate the same pick-and-place task on a UR5e in Ignition, using a skill manifest adapted to UR5e-specific kinematics and drivers. No changes to the pipeline or algorithms are required. As in the GP7 case, the pipeline generated a valid BT skeleton and optimized parameters with Bayesian MCTS. The posterior dynamics shows the same qualitative trends:

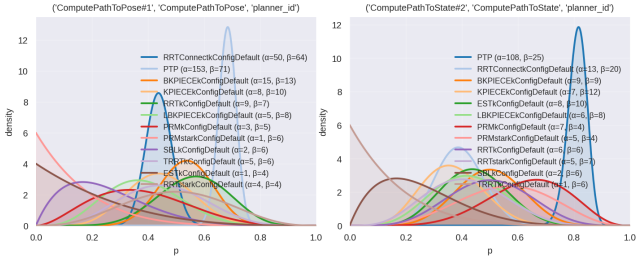

Fig. 8: Bayesian posterior updates for `ComputePathToPose#1` and `ComputePathToState#2`. The posterior concentrates rapidly on Pilz PTP for pose motions and RRTConnect for state motions, discarding low-performing planners efficiently.

Pilz PTP favored for pose motions, and OMPL RRT-Connect or BKPIECE favored for state motions with more strict path constraints. This consistency across platforms demonstrates portability in simulation to a different robot arm, with only the manifest swapped.

V. Ablation Study

To assess the contribution of individual components of our framework, we conduct a set of ablation experiments across simulation and real-robot tasks. These studies isolate the effect of `Backpropagation` strategies, language robustness, prior initialization, and transferability.

A. Backpropagation Strategies

We compare two strategies for updating priors during MCTS `Backpropagation`: EMA and Bayesian sampling with Beta posteriors. Fig. 10 shows the best-so-far reward against rollout index for both methods. Both strategies eventually approach high rewards, but Bayesian sampling converges more rapidly and stabilizes earlier. Specifically, Bayesian sampling reached $R = 0.99$ within 41 rollouts, while EMA required over 473 rollouts. For clarity, only the first 342 rollouts of EMA are plotted, during which it plateaued around $R = 0.98$.

Results. Bayesian sampling consistently concentrated probability mass on feasible parameter values, enabling much faster improvement than EMA. The order-of-magnitude difference in rollouts-to-threshold (41 vs. 473) demonstrates its efficiency advantage.

Conclusion. Bayesian updating is more sample-efficient than EMA in our case study, providing faster convergence and more stable performance. We therefore adopt Bayesian sampling as the primary backpropagation strategy for subsequent experiments.

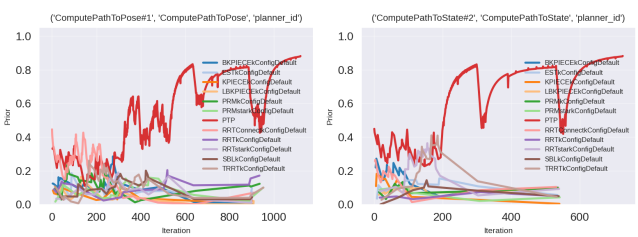

Fig. 9: EMA updates for `ComputePathToPose#1` and `ComputePathToState#2`. EMA shows slower convergence, requiring more iterations to filter out poor candidates and stabilize preferences.

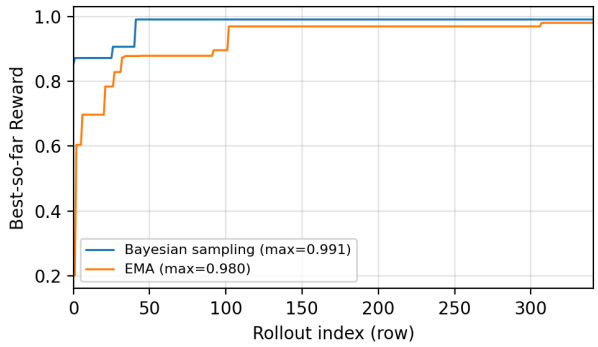

Fig. 10: Learning curves comparing EMA and Bayesian sampling as `Backpropagation` strategies in MCTS. Bayesian sampling converges faster and stabilizes earlier, while EMA is slower and plateaus at a slightly lower reward.

B. Sim-to-Real Transfer

To evaluate sim-to-real credibility, we compare parameter sets optimized by Bayesian and EMA backpropagation. For each method, we select the top-3 parameterized BT configurations and run them for 20 rollouts each in simulation to estimate success rates. The best-performing configuration among these is then deployed on the physical Yaskawa GP7. Tbl. I summarizes the success rates in simulation. The top-1 configuration from Bayesian sampling achieves the highest success rate and is selected for real execution.

Results. When transferring to the real system, two small adjustments are required due to differences between simulation and hardware: (i) the open/close positions in `YaskawaGripperCommand` skill returned to account for driver-level differences, and (ii) a +0.02 m offset is applied to the z-axis in `UpdatePose` skill, compensating for systematic bias in the ArUco marker detection algorithm, which estimates depth from 2D image geometry and marker size. With these adjustments, the selected configuration is executed 10 times on the robot, resulting in 9 successes and 1 failure. The single failure is caused by `DetectArucoMarker` skill not returning a marker pose within the timeout, rather than parameterization error.

Conclusion. Simulation-optimized BT parameters transfer reliably to the physical robot with minimal adjustments. This validates the credibility of our sim-to-real pipeline.

C. Language Robustness

We evaluate robustness to linguistic variation by generating 20 semantically equivalent paraphrases of a base task (Yaskawa GP7; target cube ArUco = 42; target pose $(0.6, 1.7, 1.3)$; gripper via `YaskawaGripperCommand`). Each paraphrase is processed through the same four-stage pipeline described in Sec. III-A: *Task Decomposition* → *Skill Selection Validation* → *Ontology-Based Dual-Graph Validation* → *Repair with Feedback*. When validation failed, structured error messages (e.g., type mismatch, missing port binding) are transformed into retrieval queries, and the candidates plus error context are injected into the next LLM prompt, exactly as in the repair loop of Sec. III-A.4. A BT skeleton is deemed valid once it passes both CFG and DFG checks. Equivalence across paraphrases is measured by graph isomorphism after grounding (node types and ports canonicalized).

TABLE I: Sim-to-real evaluation of top-3 configurations per method. Values show success rate in simulation (20 runs each). The highest-scoring configuration, obtained from Bayesian sampling, is deployed to the real robot (9/10 successes).

Method	Top-1	Top-2	Top-3
Bayesian	[87.25%]	[85.77%]	[80.06%]
EMA	[77.31%]	[81.24%]	[77.67%]

Results. 2/20 paraphrases produce valid BT skeletons by Round 2 (one repair), 15/20 by Round 3 (two repairs), and 3/20 require a single restart (≈3–4 repairs total). The mean time-to-valid is 19.45 s (min 12.44 s). All paraphrases ultimately produce BT skeletons that are graph-isomorphic to the grounded base, ensuring executable and structurally consistent task logic. The dominant error class is `SetPathConstraints` argument omission or type mismatch, indicating that constraint-node parameter parsing remains the main failure surface.

Conclusion. With strict multi-round repair and ontology-based dual-graph validation, the pipeline preserves semantic fidelity and structural consistency across paraphrases. Rare local non-convergence is resolved with a one-time restart. These findings support the convergence and repeatability of our closed loop (LLM generation → manifest-grounded retrieval → ontology-based validation → repair with feedback) and motivate future work on disambiguation strategies and stronger priors for constraint nodes such as `SetPathConstraints` to further reduce repair effort and latency.

VI. Conclusion

We have presented a framework that automatically generates reliable robot applications from natural-language descriptions by combining skill-manifest-grounded logic construction, ontology-level validation, and MCTS-driven parameter optimization. The case study and experiments demonstrate that our approach enforces structural consistency in the generated logic while systematically tuning interdependent parameters, bridging the gap between language-driven generation and executable robotic behavior. Together, these elements provide a unified pipeline from high-level task description to deployable application, with transparency and structured assurance built into each step. Despite these advances, our current framework has limitations. Our current synthesis supports sequential BT skeletons. We adopted this scope deliberately to keep validation and parameter optimization tractable in this first study, since sequential plans already expose challenging long-horizon dependencies. The framework is not inherently limited to sequentiality, and future work can extend it to richer BT constructs, potentially leveraging advances from systems such as *BTGenBot* and *OBTEA*. We view this as a natural progression rather than a fundamental limitation. In addition, parameter tuning assumes close alignment between simulation and the physical setup, limiting robustness under variation in object properties or environments. Future work will extend ontology-guided synthesis to richer BT structures for adaptive execution, improve MCTS with hybrid rollout strategies that balance fairness and interdependency exploration, and incorporate branch-cutting techniques such as RAVE [35] and progressive widening [36] to reduce search time. We also aim to reduce reliance on exact sim-to-real alignment by incorporating perception-driven adaptation, enabling transfer to a wider range of tasks and environments.

ACKNOWLEDGMENT

This work has received funding from the European Union's 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] M. Ahn et al. *Do As I Can, Not As I Say: Grounding Language in Robotic Affordances*. 2022.

[2] J. Liang et al. "Code as Policies: Language Model Programs for Embodied Control". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 9493–9500.

[3] S. Yang et al. "Foundation Models for Decision Making: Problems, Methods, and Opportunities". In: *ArXiv* abs/2303.04129 (2023).

[4] K. Valmeekam et al. "Large Language Models Still Can't Plan (A Benchmark for LLMs on Planning and Reasoning about Change)". In: *NeurIPS 2022 Foundation Models for Decision Making Workshop*. 2022.

[5] K. Valmeekam et al. *On the Planning Abilities of Large Language Models (A Critical Investigation with a Proposed Benchmark)*. 2023.

[6] W. Huang et al. "InnerMonologue: Embodied Reasoning through Planning with Language Models". In: CoRL 2022 (to appear). 2022.

[7] R. Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *Computers and Games*. Ed. by H. J. van den Herik et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 72–83.

[8] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[9] J. Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 23–30.

[10] N. Funk et al. "Learn2Assemble with Structured Representations and Search for Robotic Architectural Construction". In: *Proceedings of the 5th Conference on Robot Learning*. Ed. by A. Faust et al. Vol. 164. Proceedings of Machine Learning Research. PMLR, Aug. 2022, pp. 1401–1411.

[11] S. James et al. "RLBench: The Robot Learning Benchmark & Learning Environment". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3019–3026.

[12] A. Morozov and K. Janschek. "Probabilistic error propagation model for mechatronic systems". In: *Mechatronics* 24.8 (2014), pp. 1189–1202.

[13] A. Brohan et al. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*. 2023.

[14] D. Driess et al. *PaLM-E: An Embodied Multimodal Language Model*. 2023.

[15] C. R. Garrett et al. "Integrated Task and Motion Planning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 4 (2021), pp. 265–293.

[16] C. R. Garrett et al. *PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning*. 2020.

[17] A. Lykov et al. *LLM-MARS: Large Language Model for Behavior Tree Generation and NLP-enhanced Dialogue in Multi-Agent Robot Systems*. 2023.

[18] X. Chen et al. *Integrating Intent Understanding and Optimal Behavior Planning for Behavior Tree Generation from Human Instructions*. 2024.

[19] K. Valmeekam et al. "PlanBench: an extensible benchmark for evaluating large language models on planning and reasoning about change". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS '23. New Orleans, LA, USA: Curran Associates Inc., 2023.

[20] S. Kambhampati et al. *LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks*. 2024.

[21] R. Wu et al. *Enhancing Resilience in Robotic Systems through Self-Awareness and Adaptive Recovery*.

[22] L. Wang et al. *Text Embeddings by Weakly-Supervised Contrastive Pre-training*. 2024.

[23] L. Wang et al. *Multilingual E5 Text Embeddings: A Technical Report*. 2024.

[24] J. Johnson et al. "Billion-scale similarity search with GPUs". In: *IEEE Transactions on Big Data* 7.3 (2019), pp. 535–547.

[25] M. Douze et al. "The Faiss library". In: (2024).

[26] P. Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021.

[27] V. Karpukhin et al. *Dense Passage Retrieval for Open-Domain Question Answering*. 2020.

[28] Y. Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024.

[29] L. Kocsis and C. Szepesvári. "Bandit Based Monte-Carlo Planning". In: *Machine Learning: ECML 2006*. Ed. by J. Fürnkranz et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293.

[30] D. Silver and J. Veness. "Monte-Carlo Planning in Large POMDPs". In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010.

[31] C. B. Browne et al. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43.

[32] D. Silver et al. "Mastering the game of go without human knowledge". In: *nature* 550.7676 (2017), pp. 354–359.

[33] J. Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. 2022, pp. 24824–24837.

[34] C. Tian. "Monte-Carlo tree search with Epsilon-Greedy for game of amazons". In: *Applied and Computational Engineering* 6 (June 2023), pp. 904–909.

[35] S. Gelly and D. Silver. "Monte-Carlo tree search and rapid action value estimation in computer Go". In: *Artificial Intelligence* 175.11 (2011), pp. 1856–1875.

[36] A. Couëtoux et al. "Continuous Upper Confidence Trees". In: *Learning and Intelligent Optimization*. Ed. by C. A. C. Coello. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 433–445.